

11/15

Missed
Lecture

Lecture 18:

Dealing with an Uncertain Future

- Amortized Analysis
 - Table Doubling
- Competitive Analysis
 - Move To Front

Amortized Analysis

- Setting-
sequence of operations

e.g. Binary search tree
insert
delete
lookup

- efficiency?

so far: time per operation worst case ← worst case
over all inputs
+ all operation sequences
+ all ops within the sequence
e.g. $O(\log n)$ insert, $O(\log n)$ delete, ...

today: average of sequence of ops $S = \langle B_1, B_2, \dots, B_m \rangle$

- still worst case over choice of ops
- but some ops in sequence can take a while as long as average remains low

operation B_i has cost C_i ,

total cost of $S = \sum C_i$

amortized

$$\text{cost of } S = \frac{\sum C_i}{m}$$

↖ worst case over
all inputs,
all operation sequences
average cost of op
is same over
the sequence → average cost of op

The difficulty

- C_i 's can vary widely
- algorithm may try to optimize data structure

Table Doubling

Table of items - array of slots in contiguous memory \leftarrow no particular order

allowable operation: insert

(delete) \leftarrow not now!

How many slots to allocate? too many is a waste, not enough is an obvious problem!

Notation:

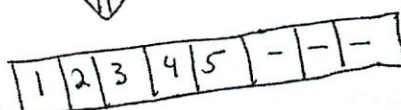
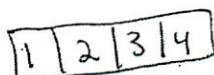
D_0 = empty table

D_i = table with i elements inserted

Plan:

What if table fills up? ($i = 2^k + 1$)

- Allocate new table $2 \times$ size (size 2^{k+1})
- copy elements over (cost 2^k)
- insert new element



$$\text{Cost } c_i = \begin{cases} 1 & \text{if } i \neq 2^k + 1 \text{ for any } k \\ 2^{k+1} & \text{if } i = 2^k + 1 \text{ for some } k \end{cases}$$

just insert
double, copy, + insert

What is $C(S) = \sum_{i=1}^m c_i$? Sum of costs

$\Theta(m)$ easy to see (you probably saw it in 6.006!)

3 Book methods:

- aggregate accounting
- potential

today

Potential Method:

- Assign costs to ops which include extra charges to cover later big things:
(similar to paying constant monthly fee instead of sums of unpredictable fees that might be very small or sometimes very large)

- Let $\Phi_i = \Phi(D_i) =$ "potential" or "bank balance" associated with D_i
= amt of prepaid work

$$\Phi_0 = 0$$

$$\Phi_i \geq 0$$

- Given Φ , define "amortized cost" \hat{C}_i of i^{th} op

as $\hat{C}_i = C_i + \underbrace{(\Phi_i - \Phi_{i-1})}_{\text{change in potential} = \Delta\Phi_i}$

Sum over $k \leq i$

if $\Delta\Phi_i > 0$: prepaying for later work $\Phi \uparrow$
if $\Delta\Phi_i < 0$: use saved work, withdrawal $\Phi \downarrow$

$$\hat{C}(s) = C(s) + \Phi(m) - \Phi(0) \quad (\text{since telescopes})$$

$$\geq C(s) \quad (\text{since } \Phi_0 = 0, \Phi(m) \geq 0)$$

so \hat{C} is upper bound on C !!

Apply to table doubling:

Let $\Phi_i = 2 \cdot X$ (# items that have never been moved)

$\leftarrow \begin{matrix} = 1 \text{ right after create a table} \\ = j \text{ when } j \text{ inserts after a creation} \end{matrix}$

$i =$	1	2	3	4	5	6	7	8	9	
new table?	✓	✓	✓		✓				✓	
$\Phi_i =$	2	2	2	4	2	4	6	8	2	$\Phi = 2^{k-1}$ when $i = 2^k$ (right before a table expand)
$C_i =$	1	2	3	1	5	1	1	1	9	
$\hat{C}_i =$	3	2	3	3	3	3	3	3	3	

When table expands, all items moved, so
 $\Delta \Phi = 2 \cdot (1 - \frac{i-1}{2}) = 3 - i$
 $C_i = i$
 $\Rightarrow \hat{C}_i = 3$

e.g. insert 5 prepays for moving 5 \$1
 " 6 " " 6 \$2
 " 7 " " 7 \$3
 " 8 " " 8 \$4

When table not expanded there is one new item that hasn't been moved, so $\Delta \Phi = 2 \cdot 1$
 & since $C_i = 1$
 $\Rightarrow \hat{C}_i = 3$

when 9 comes along, we have \$8 in bank

$$\text{so, } \hat{C}(s) \leq 3 \cdot m$$

$$\text{+ } C(s) \leq \hat{C}(s) \leq 3m$$

total actual cost for any m is $\Theta(m)$

Note: amortized cost defined relative to Φ
 different choices of Φ yields different bounds

Move to Front via Competitive Analysis

• amortized analysis is a tool

Given n elements in unordered list

e.g. hash table with chaining

Operation: search (x) $x = \text{key}$

Cost: # elements in list examined

given sequence $S = \langle x_1, \dots, x_m \rangle$ of keys

cost is
$$C(S) = \sum_{i=1}^m c_i$$

where $c_i = \text{posn of } x_i \text{ in list}$

ORDER OF LIST MATTERS !!!

What can we do?

• want most frequent request up front

• some options:

• if know statistics in advance,
can order elements in list according
to usage (i.e. most accessed, 2nd most accessed, ...)

"Dynamic Updates" • could keep counts + reorder list according to counts

• MOVE ELEMENT SEARCHED FOR TO FRONT OF LIST

"Move Forward Algorithms"
 $A \equiv$ Algorithms that move element just searched
for closer to front.

we'll
study this
class, what is
best member of \mathcal{A} ?

AA.6
6046 F2012

- assume cost of movement = 0

since just need to adjust a few pointers
& will be dominated by other costs

(for sanity check, come back later &
check that if cost = $\Theta(1)$ then analysis
still works!)

- Move-to-front (MTF)

- Move-up-one

- Let $C_A(s) =$ cost of running alg A on sequence s

↑

can compare algs A & B

via $C_A(s) + C_B(s)$

- Algorithm is online

if doesn't know future:

ie. when processing x_i

doesn't know requests $\langle x_{i+1} \dots x_n \rangle$

- Algorithm is offline

if knows all requests before processing

Call best offline algorithm "OPT" ← but still assume
 $OPT \in \mathcal{A}$

How much does it help to know the future?

is C_{OPT} much better than C_A for any online A ?

or is there a really good online algorithm?

Surprising theorem by [Sleator Tarjan 1985]:

$$\text{Thm } \forall s \quad C_{MTF}(s) \leq 2 C_{OPT}(s)$$

\uparrow
 Competitive
 ratio

i.e. MTF is never worse than twice as bad
 as any algorithm, on or offline!
 (useful in practice too)

Proof will use amortized analysis via potential
 fctn, but amortized analysis depends on OPT!

i.e. Φ_i compares MTF list order
 to opt list order

Notation

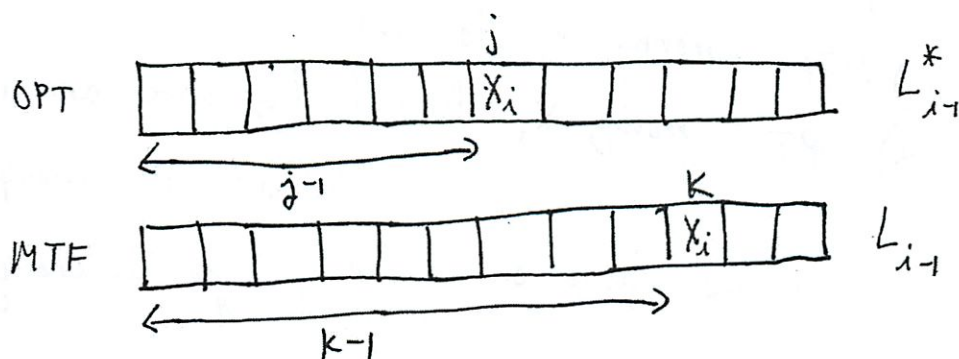
Let $L_i =$ MTF's list after i opns of S

$L_i^* =$ OPT's list after i opns of S

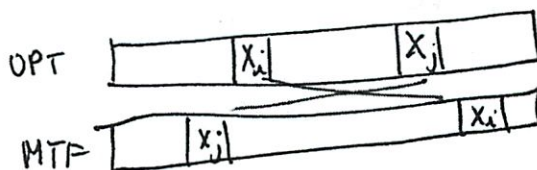
(assume $L_0 = L_0^*$ i.e. they start with same order)

i^{th} step ($i = 1 \dots m$ where $|S| = m$)

Suppose x_i in posn j in OPT's list L_{i-1}^*
 " " " k " MTF's " L_{i-1}



Let $\phi_{i-1} = \#$ inversions in these lists
 i.e. $\#$ pairs $\{x_i, x_j\}$ in different relative orders



$$0 \leq \phi_{i-1} \leq \binom{n}{2} \quad + \quad \phi_0 = 0$$

$$\hat{C}_i = C_i + \underbrace{\phi_i - \phi_{i-1}}_{\text{what is this?}}$$

↑
this is
K

Main Observation:

Since only X_i is moving in both OPT & MTF
(this is because we are looking only
at OPT $\in \mathcal{A}$, our special class of MF algs!)

only pairs including X_i affect ϕ .

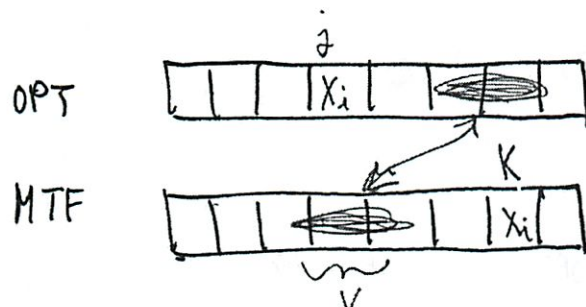
Two types of change:

- 1) MTF moving X_i to front
- 2) OPT moving X_i forward by some amount }
this only decreases ϕ !

So we focus on bounding type 1 change...

(Think of MTF moving 1st & OPT moving 2nd
& we calculate $\Delta \text{Potential}$ after each
move ...)

Type 1 Change MTF moves x_i to front
OPT stays fixed



Let $V =$ # items after x_i in OPT's list

but before x_i in MTF's list

note $K-1-V =$ # items before x_i in OPT + before x_i in MTF

When MTF moves,

$$\text{new inversions} = K-1-V$$

$$\text{removed inversions} = V$$

$$\text{Change} = K-1-V-V$$

$$\hat{C}_i \leq K + \Delta \phi$$

$$= 2(K-V)-1$$

Lemma $K - V \leq j$

PF.

there are $K - 1 - V$ in MTF list which are
also before X_i in OPT's list. \blacksquare

So $\hat{C}_i \leq 2j - 1$ j is OPT's cost for
 i th search

$$\begin{aligned} C_{\text{MTF}}(s) &= \sum_{i=1}^m C_i \\ &\leq \sum_{i=1}^m \hat{C}_i \\ &\leq 2 C_{\text{opt}}(s) - m \\ &\leq 2 C_{\text{opt}}(s) \quad \blacksquare \end{aligned}$$

So, "Knowing future" helps by factor of at most 2.

Similar results even if OPT can be more flexible.

Recitation

11/16

Missed recitation

Notes not yet posted

11/16

Recitation 9: Competitive analysis and amortized analysis

1 Competitive analysis of online algorithms

An Online algorithm is an algorithm that can process inputs in a serial fashion without knowing all the inputs in advance. Because online algorithms are forced to make decisions solely based on the input they have received so far, it may be impossible to achieve optimal result.

Competitive analysis compares performance of an online algorithm with an offline algorithm that knows all the inputs.

A deterministic online algorithm A has *competitive ratio* k if for all inputs the expected cost from algorithm A is $O(\text{cost from optimal algorithm})$. In other words,

$$E[C_A(\delta)] \leq \alpha * C_{MIN}(\delta)$$

where α is a constant and δ is any possible input. Note that we are focusing on the cost, or the quality of the solution itself, not the running time or space requirement.

We will look at examples of online algorithms.

1.1 Ski rental

After finals week, suppose that you head to a ski resort. You have the entire vacation as well as the Independent Activities Period to ski. Unfortunately, you know from past experience that, at some point, the fun will come to a premature end when fate steps in and breaks your leg. On each day until then, you have to make an important decision: should you rent ski equipment for 1 dollar or buy your own for T dollars? If you keep renting long enough, you will eventually find that you have spent more than T dollars, so it would have been cheaper to buy your own equipment at the beginning. However, if you buy your own, then you might break your leg that very day, wasting $T - 1$ dollars.

One idea would be to always buy on the first day. However, if you break your leg that day, then you spent T dollars while the optimum algorithm would have rented and spent only 1 dollar, so this algorithm is only $T - \text{competitive}$. A better idea is to rent for T days and then buy on day $T + 1$. To analyze this algorithm, suppose that you break your leg on day d . If $d \leq T$, then we always rented, which was the optimal decision. If $d > T$, then we will pay $2T$. The optimal decision would have been to buy on the first day, which would cost T dollars. But we only spent twice that, so this algorithm is 2-competitive.

1.2 Paging

Paging is an important problem in computer systems design. We model a machine's memory as consisting of two parts: an unlimited number of pages of *slow* memory, and a

cache consisting of k pages of *fast* memory. On a page request, if the requested page is not in the cache (*cache miss* or a *fault*), a page in the cache must be *evicted* to allow a space for the requested page. A *paging strategy* specifies the choice of which page to evict on a cache miss.

Some of the commonly used paging strategies are:

- LRU: evict the least recently used page.
- Random: evict a random page.
- FIFO: evict the earliest fetched page.
- Frequency counts: evict the least frequently used page.

We will show that LRU strategy is $k - \text{competitive}$. (LRU guarantees less than k times minimal number of cache misses.)

First partition input sequences into phases. The first phase begins immediately after LRU first faults. A phase ends immediately after LRU has faulted k times since the start of the phase, and the next phase begins at this point. In other words, a phase contains k faults.

Now by proving that an optimal algorithm (OPT) would fault at least once per phase, we prove $k - \text{competitiveness}$.

Consider any phase such that LRU faults *twice* on some page p in this phase. We know that at least k other distinct pages must have been requested in between the two requests of p (because otherwise p would not have been evicted by LRU). Hence, there are at least $k + 1$ distinct pages requested in this phase, and thus OPT faults at least once in this phase. On the other hand, consider any phase such that LRU faults on k distinct pages in this phase. Let p be the last fault of the previous phase. Note that even if p is one of the k faults in this phase, at least k other distinct pages must have been requested in this phase (because otherwise p would not have been evicted by LRU). Since p was in OPT's cache at the start of this phase, OPT faults at least once in this phase.

Therefore, LRU is $k - \text{competitive}$.

2 Amortized analysis: binary counter

We are dealing with three different flavors of amortized analysis in this class. We're going to apply each of them to a simple algorithm of keeping binary counter to see how they work.

Consider a binary counter composed of b bits that represent an integer. We'll consider one operation on the counter, which is *Increment*. The integer in the binary counter will increase by one every time *Increment* is called. Find the amortized cost for *Increment* operation given that switching one bit (0-to-1, 1-to-0) has cost of 1.

2.1 Aggregate analysis

In aggregate analysis, we show that for all n , a sequence of n operations takes worst-case cost $\text{Cost}(n)$ in total, leading to amortized cost of $\text{Cost}(n)/n$. All operations have same amortized cost in aggregate analysis.

In the binary counter example, we make an observation that i -th bit from the right is switched every 2^{i-1} times. (Rightmost bit is switched on every call, 2nd bit is switched every 2 calls, 3rd bit is switched every 4 calls, and so on.)

When *Increment* is called n times, the total cost of bit flipping is as follows:

$$Cost(n) = \lfloor n/2^{b-1} \rfloor + \lfloor n/2^{b-2} \rfloor + \dots + \lfloor n/2 \rfloor + n < 2 * n$$

Therefore the amortized cost is 2, $O(1)$.

2.2 Accounting method

In accounting method, we assign amortized cost to different operations, which could be more or less than actual costs. When an operation's amortized cost exceeds its actual cost, we assign the difference as *credit* associated with specific objects within the data structure.

In the binary counter example, since we start out with all 0s and 1-to-0 flip can happen only when a bit is 1, it is guaranteed that a 1-to-0 flip should follow a 0-to-1 flip. We can assign cost of 2 to each 0-to-1 flip and 'store' the prepaid cost of 1 to be used for later 1-to-0 flips.

0-to-1 flips happen once per every *Increment* operation, so the amortized cost is 2.

2.3 Potential method

Potential method is similar to accounting method, but the prepaid work is called *potential* and is associated with the data structure as a whole rather than with specific objects within the data structure.

Potential function Φ maps data structure after i th operation to its real number potential.

In the binary counter example, the potential method is very similar to the accounting method in spirit in that you pay extra cost for 0-to-1 flips to account for 1-to-0 flips later. Potential of the data structure (the bit counters) is defined as number of 1s among the bits. The amortized cost is 2 again.

6.046 Amortized Analysis Study

6.046
(LRS)

avg the time to perform a seq of vents
every few ops is expensive,
(we did this in 6.006) so we spread out the cost

Not like probabilistic average case

Since we avg performance is guaranteed ^{in the} worst case

1. Aggregate analysis

$T(n)$ for n ops

so $T(n)/n$ avg each op

2. Accounting

Some "prepaid credit"

3. Potential

"potential energy of the whole"

7

In class: potential method

- Can be used to pay for future ops
- w/ data structure as a whole
 - not a specific object

What was it before w/ accounting?
→ get credits to pay for future ops

D_0 = initial data structure

C_i = cost of i th op

$$D_i = D_{i-1} + C_i$$

Φ = potential function

maps $\Phi(D_i)$ to a real #

the potential associate w/ D_i

$$\begin{aligned}\hat{C}_i &= \text{amortized cost of } i\text{th operation} \\ &= C_i + \Phi(D_i) - \Phi(D_{i-1})\end{aligned}$$

③ why do we subtract D_i old?
Amortized cost

$$\begin{aligned}\sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \phi(D_i) - \phi(D_{i-1})) \\ &= \sum_{i=1}^n c_i + \phi(D_n) - \phi(D_0)\end{aligned}$$

If we define ϕ as ~~upper bound~~ $\phi(D_n) \geq \phi(D_0)$
Then the total amortized cost $\sum_{i=1}^n \hat{c}_i$ gives
an upper bound on the total actual cost $\sum_{i=1}^n c_i$

So pay in advance

If $\phi(D_i) > \phi(D_{i-1})$ then \hat{c}_i is an overcharge
and potential \uparrow
also undercharges

* So must choose potential fn ϕ

(4)

So is for MultiPop Φ is the #
of objects on the stack.

$$\Phi(D_0) = 0$$

So total amortized cost is upper bound
skipped proof

So Push

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= (s+1) - s \\ &= 1\end{aligned}$$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1+1 \\ &= 2 \\ &= \text{amortized cost}\end{aligned}$$

I don't get the fancy notation...

⑤

So amortized cost is $O(1)$

So total $O(n)$

which is upper bound

That didn't seem special...

Incrementing Binary Counter

Notes

Φ = bank balance = amt of prepaid work

$$\Phi_0 = 0$$

$$\Phi_i \geq 0$$

$\Phi_i > 0$ = prepaying for work; $\Phi \uparrow$

$\Phi_i < 0$ = using saved work; $\Phi \downarrow$

\hat{C} is upper bound on C

6

Table doubling example

This is double table when full

When $i = 2^k + 1$

$i = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9$
new table $\checkmark \ \checkmark \ \checkmark \ \checkmark \ \checkmark \ \checkmark \ \checkmark \ \checkmark \ \checkmark$

$\Phi_i = 2 \ 2 \ 2 \ 4 \ 2 \ 4 \ 6 \ 8 \ 2$

$C_i = 1 \ 2 \ 3 \ 1 \ 5 \ 1 \ 1 \ 1 \ 4$

$\hat{C}_i = 3 \ 2 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3$

So let me understand this...

$\Phi = \$$ in the bank

$\hat{C}_i = \text{max cost}$

$C_i = \text{actual cost of } i\text{th operation}$

Oh either $\begin{cases} n & \text{to copy size } n \text{ table} \\ 1 & \text{if not copying table} \end{cases}$

⑦

So we start w/ C_i

~~D_i = data structure from i th op~~
Does not matter

Φ = potential

note defined rel to Φ

diff choices of Φ lead to diff bounds

Ohhh $\Phi := 2^{\text{multiply}} \times (\text{items that have never been moved})$
= 1 right after create table
= j after j unmoved inserts
+= 2^{k-1} (right before a table expands)

So we define that

8

When table expands, all items have moved

$$\Delta \phi = 2 \cdot \left(1 - \frac{(i-1)}{2}\right) = 3-i$$

$$c_i = i$$

$$\hat{c}_i = 3$$

When table not expanded, there is 1 item that hasn't been moved, so

$$\Delta \phi = 2 \cdot 1 + \text{since } c_i = 1$$

$$\hat{c}_i = 3$$

\hat{c}_i is amortized cost

I think I am starting to see it!!...

$$\hat{c}(s) \leq 3 \cdot m$$

$$+ c(s) \leq \hat{c}(s) \leq 3m$$

So actual cost for any m is $\Theta(m)$

9

Move to front example

$$C_{\text{MTF}}(s) \leq 2 C_{\text{OPT}}(s)$$

Never worse than twice as bad as any
alg

Online it doesn't know future

offline knows all requests upfront

will call opt (a black box optimal
alg)

Then proof of that
Should review ...

Φ_{i-1} = # of inversions in the lists



$$0 \leq \Phi_{i-1} \leq \binom{n}{2}$$

10

$$\hat{C}_i = C_i + \phi_i - \phi_{i-1}$$

T_k

Only pairs including x_i affect ϕ

1. MTF moving x_i to front

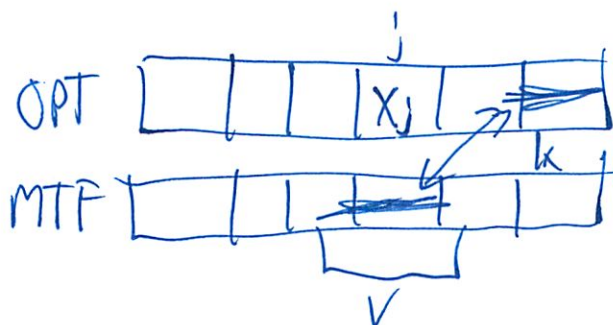
2. OPT moving x_i forward by same amt
 \hookrightarrow decreasing ϕ

So only bounding ^{Type} 1 change

Type 1 change

MTF moves x_i to front

OPT stays fixed



$V = \#$ items after x_i in OPT
 and $\#$ items before x_i in MTF

⑪

So $k-1-v = \# \text{ items } \left\{ \begin{array}{l} \text{before } X_i \text{ in Opt} \\ \text{before } X_i \text{ in MTF} \end{array} \right.$

So when MTF moves

$$\begin{aligned} \text{new inversions} &= k-1-v \\ \text{removed inversions} &= v \end{aligned}$$

$$\text{Change} = k-1-v-v$$

$$\begin{aligned} \hat{C}_i &\leq k + \Delta\phi \\ &= 2(k-v) - 1 \end{aligned}$$

$$\text{Show } k-v \leq j$$

$$\text{So } \hat{C}_i \leq 2j - 1$$

? opts cost for i th search

$$C_{\text{MTF}}(s) \leq 2 C_{\text{opt}}(s)$$

I kinda got that --

Wikipedia is non-profit, but it's the #5 website in the world with costs like any top site. To protect our independence, we'll never run ads. We take no government funds. We run on reader donations.

If everyone reading this gave \$10, our fundraiser would be done in an hour. Not everyone donates, and that's ok because every year just enough pitch in however much they want.

When we've raised enough, we stop asking. We're not there yet. Please help us forget fundraising and get back to Wikipedia.

Ad on printout

Please help

Potential method

From Wikipedia, the free encyclopedia

In computational complexity theory, the **potential method** is a method used to analyze the amortized time and space complexity of a data structure, a measure of its performance over sequences of operations that smooths out the cost of infrequent but expensive operations.^{[1][2]}

Contents

- 1 Definition of amortized time
- 2 Relation between amortized and actual time
- 3 Amortized analysis of worst-case inputs
- 4 Example
- 5 Applications
- 6 References

Read 11/18 opt

Definition of amortized time

In the potential method, a function Φ is chosen that maps states of the data structure to non-negative numbers. If S is a state of the data structure, $\Phi(S)$ may be thought of intuitively as an amount of potential energy stored in that state;^{[1][2]} alternatively, $\Phi(S)$ may be thought of as representing the amount of disorder in state S or its distance from an ideal state. The potential value prior to the operation of initializing a data structure is defined to be zero.

I like this

Let o be any individual operation within a sequence of operations on some data structure, with S_{before} denoting the state of the data structure prior to operation o and S_{after} denoting its state after operation o has completed. Then, once Φ has been chosen, the amortized time for operation o is defined to be

$$T_{\text{amortized}}(o) = T_{\text{actual}}(o) + C \cdot (\Phi(S_{\text{after}}) - \Phi(S_{\text{before}})),$$

Φ = amt of disorder

where C is a non-negative constant of proportionality (in units of time) that must remain fixed throughout the analysis. That is, the amortized time is defined to be the actual time taken by the operation plus C times the difference in potential caused by the operation.^{[1][2]}

Relation between amortized and actual time

Despite its artificial appearance, the total amortized time of a sequence of operations provides a valid upper bound on the actual time for the same sequence of operations. That is, for any sequence of operations o_0, o_1, \dots , the total amortized time $\sum_i T_{\text{amortized}}(o_i)$ is always at least as large as the total actual time $\sum_i T_{\text{actual}}(o_i)$. In more detail,

$$\sum_i T_{\text{amortized}}(o_i) = \sum_i (T_{\text{actual}}(o_i) + C \cdot (\Phi(S_{i+1}) - \Phi(S_i))) = \left(\sum_i T_{\text{actual}}(o_i)\right) + C \cdot (\Phi(S_{\text{final}}) - \Phi(S_{\text{initial}})) \geq \sum_i T_{\text{actual}}(o_i),$$

where the sequence of potential function values forms a telescoping series in which all terms other than the initial and final potential function values cancel in pairs, and where the final inequality arises from the assumptions that $\Phi(S_{\text{final}}) \geq 0$ and $\Phi(S_{\text{initial}}) = 0$. Therefore, amortized time can be used to provide accurate predictions about the actual time of sequences of operations, even though the amortized time for an individual operation may vary widely from its actual time.

Amortized analysis of worst-case inputs

Typically, amortized analysis is used in combination with a worst case assumption about the input sequence. With this assumption, if X is a type of operation that may be performed by the data structure, and n is an integer defining the size of the given data structure (for instance, the number of items that it contains), then the amortized time for operations of type X is defined to be the maximum, among all possible sequences of operations on data structures of size n and all operations o_i of type X within the sequence, of the amortized time for operation o_i .

With this definition, the time to perform a sequence of operations may be estimated by multiplying the amortized time for each type of operation in the sequence by the number of operations of that type.

Example

A dynamic array is a data structure for maintaining an array of items, allowing both random access to positions within the array and the ability to increase the array size by one. It is available in Java as the "ArrayList" type and in Python as the "list" type. A dynamic array may be implemented by a data structure consisting of an array A of items, of some length N , together with a number $n \leq N$ representing the positions within the array that have been used so far. With this structure, random accesses to the dynamic array may be implemented by accessing the same cell of the internal array A , and when $n < N$ an operation that increases the dynamic array size may be

implemented simply by incrementing n . However, when $n = N$, it is necessary to resize A , and a common strategy for doing so is to double its size, replacing A by a new array of length $2n$.^[3]

This structure may be analyzed using a potential function $\Phi = 2n - N$. Since the resizing strategy always causes A to be at least half-full, this potential function is always non-negative, as desired. When an increase-size operation does not lead to a resize operation, Φ increases by 2, a constant. Therefore, the constant actual time of the operation and the constant increase in potential combine to give a constant amortized time for an operation of this type. However, when an increase-size operation causes a resize, the potential value of n prior to the resize decreases to zero after the resize. Allocating a new internal array A and copying all of the values from the old internal array to the new one takes $O(n)$ actual time, but (with an appropriate choice of the constant of proportionality C) this is entirely cancelled by the decrease of n in the potential function, leaving again a constant total amortized time for the operation. The other operations of the data structure (reading and writing array cells without changing the array size) do not cause the potential function to change and have the same constant amortized time as their actual time.^[2]

Therefore, with this choice of resizing strategy and potential function, the potential method shows that all dynamic array operations take constant amortized time. Combining this with the inequality relating amortized time and actual time over sequences of operations, this shows that any sequence of n dynamic array operations takes $O(n)$ actual time in the worst case, despite the fact that some of the individual operations may themselves take a linear amount of time.^[2]

Applications

The potential function method is commonly used to analyze Fibonacci heaps, a form of priority queue in which removing an item takes logarithmic amortized time, and all other operations take constant amortized time.^[4] It may also be used to analyze splay trees, a self-adjusting form of binary search tree with logarithmic amortized time per operation.^[5]

References

- ^{a b c} Goodrich, Michael T.; Tamassia, Roberto (2002), "1.5.1 Amortization Techniques", *Algorithm Design: Foundations, Analysis and Internet Examples*, Wiley, pp. 36–38.
- ^{a b c d e} Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. "17.3 The potential method". *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 412–416. ISBN 0-262-03293-7.
- ^a Goodrich and Tamassia, 1.5.2 Analyzing an Extendable Array Implementation, pp. 139–141; Cormen et al., 17.4 Dynamic tables, pp. 416–424.
- ^a Cormen et al., Chapter 20, "Fibonacci Heaps", pp. 476–497.
- ^a Goodrich and Tamassia, Section 3.4, "Splay Trees", pp. 185–194.

Retrieved from "http://en.wikipedia.org/w/index.php?title=Potential_method&oldid=494806379"

Categories: Analysis of algorithms

- This page was last modified on 28 May 2012 at 17:22.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

How does it compare to accounting method?

Lemailed in

I think I kinda see diff...

Seems more semantic than anything...

Michael Plasmeier

From: Annie I-An Chen <anniecia@MIT.EDU>
Sent: Tuesday, November 20, 2012 9:33 AM
To: Michael Plasmeier
Cc: 6046-tas@mit.edu
Subject: Re: [6046-tas] Potential vs accounting method?

Hi Michael,

Both methods seek to use amortized costs to bound the total work on the sequence of operations. The amortized cost is defined differently in each case.

The accounting method keeps an account of work credits. The balance in the account is always kept nonnegative (and therefore can be used as an upper bound of the total work done). This is achieved by "depositing" some work credit for certain operations (this is the amortized cost defined for these operations), so they can be "withdrawn" and used for other operations later on (these "other operations" may have zero amortized cost). In other words, it prepays for future operations.

On the other hand, the potential method uses a potential function to "balance out" the work done in each operation, in order to make the amortized cost easy to calculate. The potential function only depends on the data structure.

Not sure if this helps. I think my best suggestion would actually be to go over the examples in CLRS to see how different methods solve the same problem (we also covered one example, incrementing a binary counter, in recitation last week).

Best,
Annie

Was not there, emailed a reply asking

On Sun, Nov 18, 2012 at 5:36 PM, Michael Plasmeier <plaz@theplaz.com> wrote:

(I tried posting on Piazza but the class was inactive)

What are the differences between the potential and the accounting methods?

The book says "we associate the potential with the whole data structure rather than specific objects" Is this how accounting works? Could you elaborate on this difference?

Thanks!

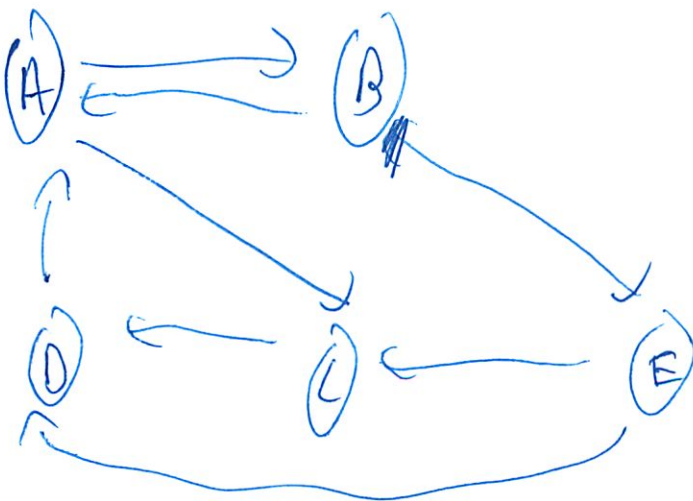
6046-tas mailing list
6046-tas@lists.csail.mit.edu
<https://lists.csail.mit.edu/mailman/listinfo/6046-tas>

- Model
- Two problems
 - Leader election
 - Maximal ind set

(missed) could improve running time

Many diff models
invented every day

Synchronous Network



$G(V, E)$
 $n = |V|$

(2)

Can send messages only along the ~~right~~ links

Can get in # of steps \leq to diameter of network

Strongly connected - usually
this is not it

Execute in rounds

Each node knows its neighbors but not ~~global~~ global topology

async network model lots of places in life

→ execute in rounds

- send + collect messages

- process

- be ready for next round

Sync = common clock

③

Can take dist. alg class to hear about diff types of models.

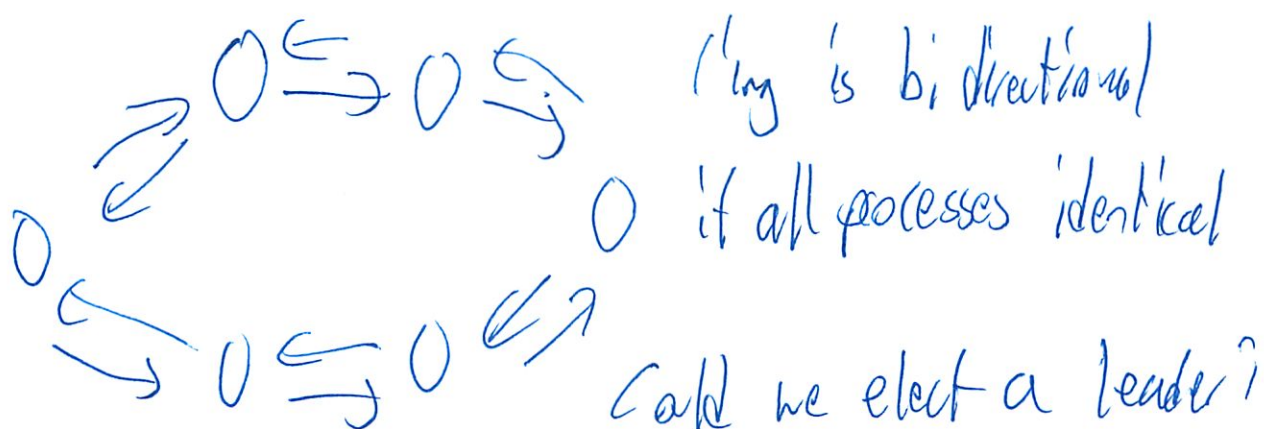
Leader Election

Want to distinguish 1 process as leader

Leader at end of process "I am leader"

Others "I am the loser"

Maximal ind set / Ring Network



No if each is completely identical

Either all are leaders or none are

Must add something - could be randomness

④

Unique Identifier (UID)

processes have UIDs

algorithm: choose max value UID as leader

But that is randomness

Could have each compare w/ neighbors + ripple outwards;

Send biggest UID we've seen so far

Send clockwise around the ring

Round 1: Own UID

Round 2+: Running tally of max UID seen so far

Termination: When you get your own UID back
then you are the leader!

(Since only moving clockwise)

(5)

Complexity:

$$\text{Rounds} = n$$

$$\begin{aligned} \text{Communication} &= \# \text{ of single hop messages} \\ &\leq n^2 \quad O(n^2) \end{aligned}$$

Can we do better?
Of course!

How? Can we reduce to $\leq n^2$

Divide + Conquer - Send messages both ways
Solves subproblems - want leaders of sections!

Initially everyone is a leader

~~But~~ Then divide up into sections

Pick a leader of that

Then have all the leaders communicate

6)

BiDirectional ring

Round i (start $i=1$)

Elect leaders in sections of length 2^i

Send message at to distance 2^{i-1}

on either side w/ your ID

When messages come back, they tell you if
you have the largest ID in your section
of the ring.

Still counting # of hops

Want reduction in message complexity taking
that into account

"dead" nodes are not sending their own messages

"live" nodes that ~~won~~ won in round i
go on to $i+1$

(7)

When your own message reaches everyone (and returns)
you can stop

Determination of

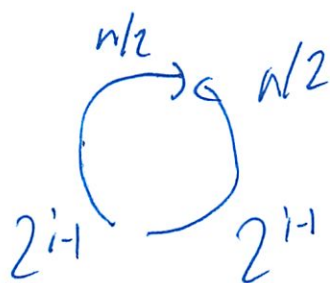
depends on if ~~the~~ known # of processes in network

But gets own vid back \rightarrow you are the leader

- if you know the size of the network

if known n

$2^{i-1} \geq n/2$ is sufficient



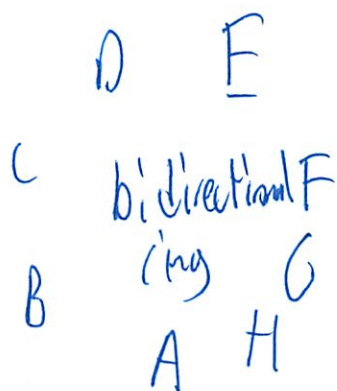
(not following)

if don't know n

back to situation where must wait
for message to return to you.

8

$2^{i-1} \geq n$ for message to come back to 2^{i-1}



Round 1

$A \rightarrow B \rightarrow A$

$A \rightarrow C \rightarrow A$

if A was leader

B+C ~~and~~ will not be leaders

Since adjacent

Now we actually have a

reduction/divide + conquer algo

[if A is a leader, B, C can't be leaders

$i \geq 2$ b/w any two participating processes

There are at least 2^{i-2} ^{electd leader nodes} non-participating processes
lower nodes

each leader has a bigger id than distance 2^{i-2}
on each side

⑨

2^{i-2} is the increment

How far one can reach

Complexity $O(n \lg n)$

↳ like Merge Sort

Number of participating processes $\leq \frac{n}{2^{i-2}}$

Number of messages = 2^{i-1} in each direction (2)

$$= 2^{i-1} \times \underset{\substack{\uparrow \\ \text{2 direction}}}{2} \times \underset{\substack{\uparrow \\ \text{there and back}}}{2}$$

$$= 2^{i+1}$$

$$\text{So } \frac{n}{2^{i-1}} \cdot 2^{i+1} = 8n \text{ messages per round}$$

$\lg n \text{ rounds}$

So $8n \lg n$ messages total $\rightarrow O(n \lg n)$

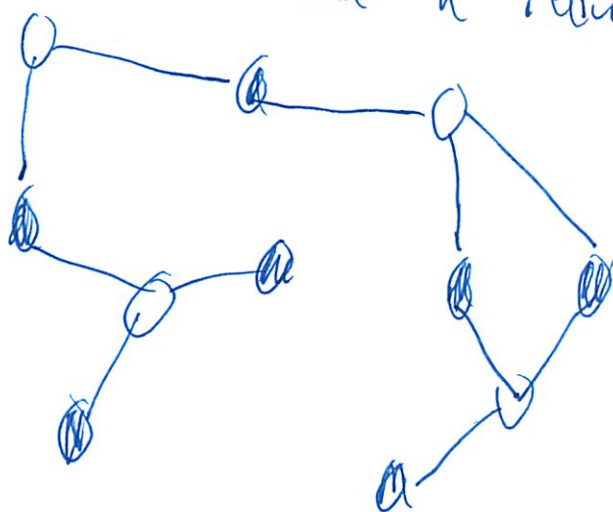
(10)

Will ignore messy detail like blahy
at most 1 msg per link in clock cycle
Sync network model
Some constant factor for messages to not collide

bidirectional $A \leftrightarrow B$

Maximal ind set

could be matching or coloring problem
~~the~~ via a reduction



want subset I of vertices V
of $G(V, E)$ is ind it.
no two vertices in I are
neighbors based on E .

(11)

Maximal vs maximum ind set
↑
here
greed

Pick node

Delete neighbors

Shrink graph

Want to parallelize this like w/ leader election

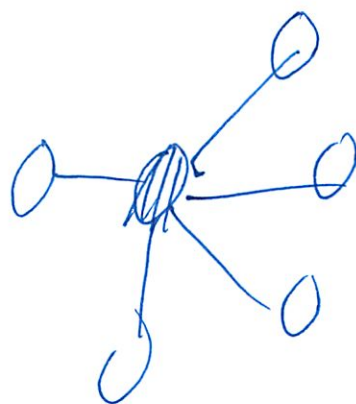
Comm w/ neighbors

Multiple processes that declare themselves winners

Then process those

Maximal \neq maximum

↑
no strict superset
of I is an MIS



(12)

Trying to speed up this alg

Could also do for approx. algs

For general graph - processes know neighbors
but don't know topology

Can do it in $O(\lg n)$

Greedy ^{was} $O(n)$

We're doing a lesser bound for a more specific network
~~max~~ max degree d is known to all processes
here in our example $= 3$

(Simplified) Luby's Algorithm

1. Initialize: All processes added to live set
(current graph)

⑬ Then ^{live} set shrinks ~~off~~ as nodes realize they are not in MIS

1. Each Live node marks itself w/ prob $\frac{1}{2d}$

2. Each marked node ~~if~~ checks its neighbors.

If any marked, V unmarks itself.

Could do tiebreaker in a more sophisticated alg

Could problem w/ optimality \leftarrow we don't guarantee though
but not w/ maximality

3. Each remaining marked node ~~the~~ will add itself to MIS, \leftarrow winner

- removes itself + all neighbors from Live set
 \uparrow losers

1, 2, 3 = Round

4. Termination When $\text{live} = \emptyset$, process terminates

(14)

there as well we have a seq of cards
(misses)

Corner case

taking nodes and making them go dark
2 vertices themselves



but stay till next round
don't declare losers

next time only I might generate the # to mark

Theorem Prob that # of cards $> 8d \lg n$
is at most $\frac{1}{n}$ degree
of vertices

(5)

Expected # of rounds is $O(d \ln(n))$

Since we have prob.

Main Lemma Prob [live V adds self to MIS in a round]
Maximal ind set

$$\leq \frac{1}{4d}$$

To show it converges

Must show shrinkage of the network

Proof Given a live V

What is prob we add a self to live set.

$$P(V \text{ marks itself in step } i) = \frac{1}{2d}$$

$$P(\text{any neighbor of } V \text{ marks self in step } i)$$

$$\leq \sum_{\text{Union bound}} \frac{1}{2d} \quad \text{a sum of prob.}$$

(16)

$$So \leq \frac{1}{2d} = \frac{1}{2}$$

remember d is upper bound on # neighbors

$$\# \text{ neighbors} \leq d$$

P[v marks itself and stays marked after step 2)
and no neighbors are marked

$$\geq \frac{1}{2d} \cdot \frac{1}{2} = \frac{1}{4d} \quad \text{[scribble]}$$

So Lemma proved (✓)

Now prove theorem


Show how many nodes deleted in each round
using the lemma.

(must review!)

(17)

$$\begin{aligned} \text{Prob}(v \text{ stays LIVE after } c \cdot 4d \cdot \ln n \text{ rounds}) \\ \leq \left(1 - \frac{1}{4d}\right)^{4d \cdot c \cdot \ln n} \quad \text{Since rounds are ind} \\ \leq e^{-c \ln n} \\ = n^{-c} \end{aligned}$$

Now set $c=2$ and show theorem

$$\begin{aligned} P\{\text{any } v \text{ stays LIVE after } 8d \ln n \text{ rounds}\} \\ \leq n \cdot n^{-2} \quad (\text{union bound}) \quad c=2 \\ = \frac{1}{n} \end{aligned}$$


Here more involved proof since probabilistic analysis

log result that depends on max degree

18

More general on takes away d

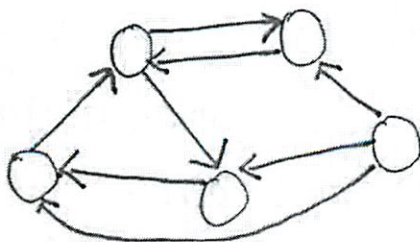
↳ Notes are online

Distributed Algorithms

- Model
- Two problems
 - Leader election
 - Maximal Independent Set

Synchronous Network Model

- Processes at nodes of a network digraph $G(V, E)$ $n = |V|$
- Links connect some process pairs
 - Send messages only along links
 - Each node knows its in-neighbors and out-neighbors
- Execute in rounds: in each round
 - send and collect messages
 - assume common clock



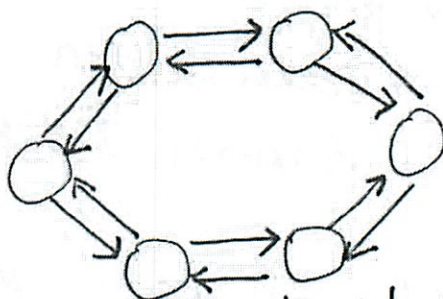
Processes know immediate neighbors but not global topology. Algorithm may assume a topology.

(2)

Leader Election

Want to distinguish exactly one process as leader
Leader outputs "I am the leader" and no
one else outputs anything (or outputs "loser")

Ring network:
- bidirectional links



Thm: If all processes are identical, it is
impossible to elect a leader.
because processes are always in identical states

Unique Identifier (UID)

Processes have UID's

Algorithm: choose max-valued UID as leader

At each round:

Send value of biggest UID seen so far
clockwise (1st round process sends its UID)

When a process receives its own UID,
it declares itself the leader

Process UID made it all the way clockwise
around the ring and hence is max valued.

Note: unidirectional ring suffices here.

Complexity

Number of rounds : n

Communication : number of single-hop messages $\leq n^2$

Can we do better?

First elect local leaders who will compete with one another for larger and larger sections of the ring.

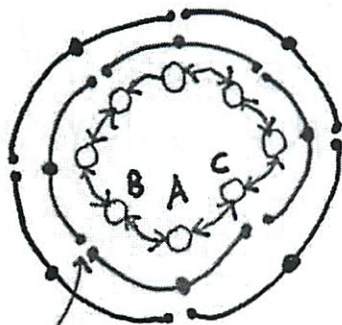
Hierarchical Leader Election

Assume bidirectional ring.

Round i : send messages 2^{i-1} hops on either side with your id.
When messages come back they tell you whether you have largest ID in your section.

Round 1:

Sections shown



Overlap!

8 initial nodes, down to
4 leaders after Round 1

If A is a leader, B
and C cannot be leaders.

ANALYSIS / ALGORITHM

(4)

Round i : elect leaders in sections of length 2^i .
Send message out to distance 2^{i-1} on either side with your id.
When messages come back, they'll tell you whether you have the largest id in your section of the ring.
Only processes who won in round i go on to round $i+1$.

When your own message reaches everyone, you can stop.

- if n is unknown, then need message to go all the way around the ring $2^{i-1} \geq n$
- for known n , $2^{i-1} \geq n/2$ is sufficient

Any process that loses in any round can output 0.

Round i : number of messages per participating process is 2^{i+1}

$i \geq 2$: Between any two participating processes, there are at least 2^{i-2} non-participating processes, since each participating process has bigger id than distance 2^{i-2} on either side.

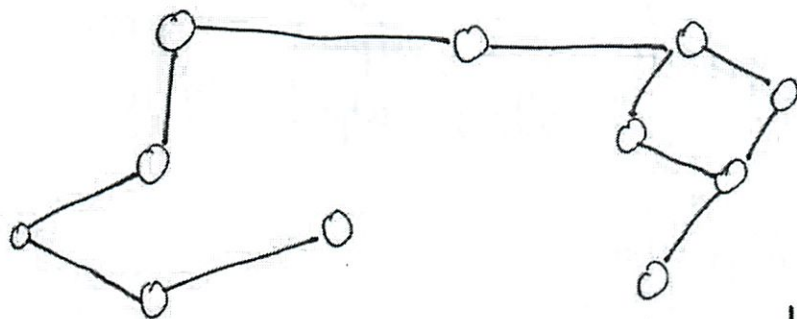
$$\text{Number of participating processes} \leq \frac{n}{2^{i-2}+1} \leq \frac{n}{2^{i-2}}$$

$$\# \text{ messages} \leq \frac{n}{2^{i-2}} \times 2^{i+1} \leq 8n \text{ for each round.}$$
$$\# \text{ rounds} = \log n.$$

Maximal Independent Set

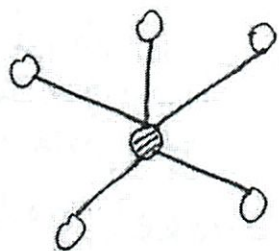
(5)

Undirected graph \Rightarrow bidirectional communication



Subset I of vertices V of undirected graph $G(V, E)$ is independent if no two vertex neighbors are in I .

Independent set I is maximal if no strict superset of I is independent. Maximal \nRightarrow Maximum



⊗ maximal independent set (MIS)
Not maximum!

Greedy sequential algorithm:

Pick a vertex. Put into MIS.

Remove vertex and its neighbors from G .

Repeat till G is empty.

HARD TO PARALLELIZE

(6)

SETUP & ASSUMPTION

Max degree d known to all processes.

OUTPUT: MIS I of network graph.

Each process in I outputs "winner"
others output "loser"

(Simplified) Luby's Algorithm

Initialize: All processes added to "LIVE" set.

$MIS \leftarrow \emptyset$

Round: (1) Each live node/process "marks" parallel itself with probability $\frac{1}{2d}$

local comm: (2) Each marked node v checks neighbors. If any marked, v unmarks itself. Note: two adjacent marked nodes could both unmark themselves.

(3) Each remaining marked node
winner \leftarrow - adds itself to MIS
- removes itself and all neighbors from LIVE set.

unmarked node removed from LIVE loser

(4) When "LIVE" = \emptyset , process terminates.

(7)

Theorem

Probability that number of rounds $> 8d \ln n$
is at most $1/n$.

\Rightarrow Expected number of rounds is $O(d \ln n)$

Main Lemma

$\Pr [\text{LIVE } v \text{ adds self to MIS in one round}] \geq \frac{1}{4d}$

Proof: v LIVE

$\Pr [v \text{ marks itself in Step 1}] = \frac{1}{2d}$

$\Pr [\text{any neighbor of } v \text{ marks self in Step 1}]$
(union bound)

$\leq \sum_{w \text{ neighbor of } v} \frac{1}{2d}$

$\leq \frac{d}{2d} = \frac{1}{2}$

$\therefore \Pr [v \text{ marks itself and stays marked after Step 2}]$
 $\geq \frac{1}{2d} \cdot \frac{1}{2} = \frac{1}{4d}$ \square

(8)

Proof of Theorem

basic idea

(Lemma \Rightarrow at each round $\geq \frac{1}{4d}$ fraction of nodes deleted.)

$$\begin{aligned} \Pr[v \text{ stays LIVE after } c \cdot 4d \cdot \ln n \text{ rounds}] \\ &\leq \left(1 - \frac{1}{4d}\right)^{4d \cdot c \cdot \ln n} \quad \text{since rounds are indep.} \\ &\leq e^{-c \ln n} = n^{-c} \end{aligned}$$

$$\begin{aligned} \Pr[\text{any } v \text{ stays alive after } 8d \ln n \text{ rounds}] \\ &\leq n \cdot n^{-2} \quad (\text{union bound}) \\ &= \frac{1}{n} \quad \square \end{aligned}$$

Luby's actual algorithm is a slight modification
(includes tiebreaker in Step 2)

Significantly more complex analysis.

Does not make max-degree d assumption.

$O(\log n)$ rounds for general graphs.

Applications to maximal matching and vertex coloring.

11/25

(LDS)
Binary Counter

Potential Method

potential after i th operation

$b_i = \# \text{ of } 1\text{s after } i\text{th operation}$

i th op resets x_i bits

So max cost $x_i + 1$

if $b_i = 0$ then i th op resets all k bits

$$\text{So } b_{i-1} = x_i = k$$

$$b_i \geq 0 \quad b_i = b_{i-1} - x_i + 1$$

both cases $b_i \leq b_{i-1} - x_i + 1$

$$\begin{aligned} \phi(D_i) - \phi(D_{i-1}) &\leq (b_{i-1} - x_i + 1) - b_{i-1} \\ &= 1 - x_i \end{aligned}$$

②

Amortized cost

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &\leq (x_i + 1) + (1 - x_i) \\ &= 2\end{aligned}$$

So at 0 $\phi(D_0) = 0$

Worst case is $O(n)$

After n increments at b_n

$$0 \leq b_0, \quad b_n \leq k$$

? # of bits in counter

$$\sum_{i=1}^n c_i = \sum_{i=1}^n \hat{c}_i - \phi(D_n) + \phi(D_0)$$

$$\hat{c}_i \leq 2 \quad \text{for} \quad 1 \leq i \leq n$$

$$\begin{aligned}\phi(D_0) &= b_0 \\ \phi(D_n) &= b_n\end{aligned}$$

(3)

So actual cost is

$$\sum_{i=1}^n c_i \leq \sum_{i=1}^n 2 - b_n + b_0$$
$$= 2n - b_n + b_0$$

Since $b_0 \leq k$ as long as $k = O(n)$
total cost is $O(n)$

I didn't get that

Need to go back & study counter example closer...

I think the problem is I don't get
it conceptually

↳ It's just some other way to represent
amortized cost

(4)

In class Amortized table example

What is m_i

Each says amortized cost at most 3

Problem Set 5

This problem set is due at **11:59pm** on **Tuesday, November 27, 2012**.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Each problem must be turned in separately to stellar.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. A proof (or indication) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluting and obtuse descriptions will receive low marks.

Exercise 5-1. Do Exercise 11.3-6 in CLRS on page 269.

Exercise 5-2. Do Exercise 11.5-1 in CLRS on page 282.

Exercise 5-3. Do Exercise 35.1-4 in CLRS on page 1111.

Exercise 5-4. Do Exercise 35.2-2 in CLRS on page 1116.

Exercise 5-5. Do Exercise 17.1-3 in CLRS on page 456.

Exercise 5-6. Do Exercise 17.2-2 in CLRS on page 459.

Exercise 5-7. Do Exercise 17.3-2 in CLRS on page 462.

Problem 5-1. Task Scheduling Approximation

There are n tasks t_1, t_2, \dots, t_n , each of which has an associated cost $c_i \in \mathbb{R}$, such that

$$0 < c_i < 1$$

A task t_i consumes c_i fraction of the resources when run on a “standard” computer. A set of tasks T can be run on the same computer simultaneously only if

$$\sum_{i \in T} c_i \leq 1$$

i.e. there are enough resources to run all tasks in T .

You would like to run all tasks simultaneously on identical “standard” computers. Multiple tasks can run on a single computer if the constraint given above is satisfied. Each task must be run on a single computer, i.e. you cannot run task t_i partially on different computers. Since computers cost money, you would like to use the minimum number of computers to run all tasks. Sadly, it turns out that finding the minimum number of computers you need to run all tasks is an NP-hard problem.

In desperation, you decide to pursue the greedy approach. You iterate over the tasks and for each task, you assign the task to run on the first computer that can accommodate the task. If there is no computer on which you can run the task, you add a computer and assign the task to run on the added computer. You keep the computers in the order they were added.

Prove that the greedy approach yields a 2-approximation.

Problem 5-2. Queue

We would like to implement a FIFO (first in, first out) queue that supports the following operations:

- **ENQUEUE(item)**: appends the item to the back of the queue. The operation must always succeed.
- **DEQUEUE(k)**: k must be a positive integer. The operation pops and returns k elements from the front of the queue. If the total number of the elements in the queue is less than k , then the operation returns all items in the queue. If the queue is empty, it returns *NONE*.

Suppose that to implement the queue you can use two LIFO (last in, first out) stacks, each of which supports the following operations:

- **PUSH(item)**: pushes/inserts the item to the stack. The operation always succeeds.
- **POP()**: removes and returns the most recently pushed item in the stack. If the stack is empty, it returns *NONE*.

Assume that for a stack, each operation costs 1 unit in the running time.

Give an implementation of the queue using two stacks such that both operations *ENQUEUE* and *DEQUEUE* have an amortized cost of $O(1)$. The amortized cost of *DEQUEUE* should be $O(1)$ regardless of the input k .

Problem 5-3. Distributed Median

Alice has a list of n numbers and Bob has another list of n numbers. Jointly, they have $2n$ numbers that are distinct from each other. They would like to know the median element of their combined arrays. Since $2n$ is even, let the median be the n th smallest number.

They have limited communication bandwidth and would like to minimize the communication cost. They can send each other messages and each message can contain one of the following: an integer in range $[0, n]$, one of the numbers they have, or an English word that contains no more than five letters.

- (a) Give a deterministic algorithm where the number of messages used is $O(\log n)$ and the running time is $O(n \log n)$. Alice and Bob can follow different protocols.
- (b) Modify your algorithm to have the running time of $O(n)$ while keeping the number of messages to be $O(\log n)$. The resulting algorithm should also be deterministic.

11/25

5-1 n tasks $\tau_1, \tau_2, \dots, \tau_n$ each w/ a cost $c_i \in \mathbb{R}$

$$0 < c_i < 1$$

b/w 0 and 1

 τ_i consumes c_i fraction of resource when run on
std computer

$$\sum_{i \in T} c_i \leq 1$$

↳ so c_i is % of CPU it takes up

Want to run all tasks on std computers

Can't divide up tasks b/w PCs

ie $\frac{1}{2}$ task on each computer

This is NP-hard

②

So instead approximate!

Greedy approach

assign to 1st PC which can run the task
if no PC, you add a PC

Note: Not least constrained
The 1st PC that can accommodate it
Prove this is a 2-approximation

Didn't I do this before?

$$\max\left(\frac{c}{c^*}, \frac{c^*}{c}\right) \leq \rho(n)$$

Show max size is ≤ 2 ~~normal~~ ^{optimal} size (c^*)

③

So optimal is a perfect arrangement

Worst case is all tasks $\leq 1/2$

Then n tasks = n pcs

But optimally, the same!

So what is another worst case?

A declining order 7, 5, 3

vs

7	5
3	

optimal \rightarrow 2 machines — same

Do we need to find an example?

Or just the bounds ---?

(wish I had the recitation notes...)

(9)

Want to show $S \leq 2S^*$

Traveling salesman
Showed it visited each node twice
So $C(W) = 2 C(T)$ $C(T) \leq C(H^*)$
So $C(W) \leq 2C(H^*)$ Given
Plus some details

We want to make a similar claim here,
But I'm not seeing what we can do.

Cost of our approx will not be worse than 2

Could say double computers means $\geq 50\%$
empty on each machine (if we spread avg and tasks)

So we can combine machines

$$\leq 50 + \leq 50 = \leq 100$$

⑤

Is that mathematically rigorous enough?

I've never fully understood how to tell...

So say we have our optimal $\leq 100\%$

We double the # of servers
and distribute each task

now avg is $\leq 50\%$

but we can still have 75, 25

Can fit it back together

Can we say since avg is ≤ 50 it could
fit together

I think we can...

Some rule about avgs that more than half
the #s are $<$

(6)

Actually not necessarily

1, 100, 100

Avg = 67

But that is not what we are claiming

~~Adding~~ Can add machines avg ≤ 50
To consolidate

Since we have a cap at 100 I think we can

But I am at a loss for formal proof...

On the right the track

$$\leq 50 + \leq 50 = \leq 100$$

~~Need to do the algorithm~~

~~If design~~ Just prove

11/26
OH

⑦

Show

~~Alg~~ Alg in a) will not result in too many processors being used $\leq 50\%$

Since they go for lot

Least constrained \rightarrow would still be 2-approx
Have to think

~~Show~~

Ok is that what I've done?

How to show ^{approx} alg will not result in too many processors being used $\leq 50\%$

But this is basically what I've shown already!

I think what I have works...

8

11/26
11:30P
Dmy

(write up a)

✓ according to approx

Need to show natural ordering leads to
that particular ordering of items

But wait what am I really trying to show
than does my $\leq 50\%$ thing help?

That we can consolidate

I don't think I've shown anything yet

Something about how we allocate machines

If $\leq 50\%$ something will come and fill it
up

$$\leq 50 + \leq 50 = \leq 100\%$$

④ (I think I got the big picture - but I'm not putting it all together...)

It's not avg - it's something else

Or when have ~~all~~ avg $\leq 50\%$ can combine enough below the average

Since cap is 100% - no extreme outliers
will combine

So since avg will always > 50

and ~~opt~~ 2 optimal is ~~at~~ 50

we are always better! QED!

I think I solved it, but am missing some
subtlety or other

5-2 Queue

FIFO queue



enqueue(item)

↳ must always succeed

dequeue(k)

↳ pops k elements from front

So use 2 LIFO queues
each supports



Push(item)

↳ inserts item, always succeeds

$O(1)$

POP()

removes most recent item

$O(1)$

amortized cost for both $O(1)$
regardless of k

②

hmm I think I've seen something like this before

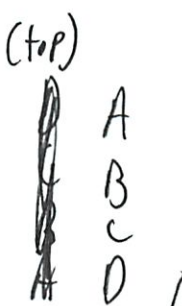
Stack Overflow

One ~~stack~~ stack needed to insert &
other stack to output/remove

When empty load input in reverse

So push A, B, C, D

Then pop()

So loads  (top) A
B
C
D, clears queue!

and can pop

But then push E

leave and queue until its empty!

③

So we have

E top

D C ~~A~~ B A top

pop(4)

E top

— top

then

— top

E top

push F G H

F G H top

E top

pop(1)

F G H top

— top

(4)

pop(3)

— top
H G F top
└─┬─┘
└─┘
3

or

Ah nice!

How would you write the code for that

~~while~~ queue not empty

~~pop~~
~~while~~ $< k$
~~while~~ if queue empty
transfer

3

pop(1)

3

or try up to k or ~~max~~ left

$\max(k, \text{left})$

? then it would transfer

⑤

I just don't think I could do the potential analysis
unless looked at it closely
which prob should do

(write up A)

11/26
12A

So do we need to show analysis?

Prob should

even ~~you~~ just good for me to try as an exercise

~~TA~~

Overhead TA talking about it

'More complex than it looks?'

I'm pretty sure what I have now
just need to prove it

(6)

Analysis

Book

Oh there is binary counter introduced

Accounting method

Enqueue \rightarrow ~~when blank~~ ^{always} $O(1)$
just push

dequeue $O(1)$ when full

but sometimes have to add which is $O(n)$

at most for 1 item

so $O(2)$

which is $O(1)$

But is this the accounting method?

①
So it appears here we always pay the amortized rate

We deduct actual cost

and call rest a credit

So push costs $O(n)$ amortized

- deduct 1 on actual push cost

- deduct 2 on transfer

- deduct 1 on dequeue

Potential Method

There is the big confusion from me

See if I got it

8

$i = 1 \quad 2 \quad 3 \quad 4$

↙ but it depends when we reverse

$G = \#$ of items un moved
= 1 right after

We don't really need - do later

(how did I forget all this already?)

11/25

5-3 | Distributed Median

Alice has list of n #
 Bob has list of n #) distinct

Want to know median of combined array

Note n th #

So if n is 5

1 2 3 4 5 6 7 8 9 10
 └──────────┘
 n th # = the median

Limited communications b/w - expensive!

So can send each other messages

- an integer $[0, n]$
- one of the #s they have
- an English word < 5 letters

②

a) Give a deterministic time alg where

$$\# \text{ of messages} = O(\lg n)$$

$$\text{running time} = O(n \lg n)$$

Alice + Bob can have diff protocols.

Distributed alg

classi example on leader election

Then splitting into sections

Median

Middle # in a list

half larger, have bigger
smaller

(3)

Hmm usually it's mixed in

1 3 5 7 9
2 4 6 8 10
 ↑
 median

(Median of 1 then median of other - No!

Hmm at a loss here

But I generally understand the setup of what is required...

How much comm is allowed?

$O(\log n)$

Ok so not just 1 message ...

$\log n$ is

$\frac{n}{2}$ $\frac{\log(n)}{1}$

$\frac{3}{4} \mid 2$

$\frac{8}{3}$

(4)

So that tells us something

↳ But I am not immediately thinking of what - grr

local median

global median

1/200
0H

Find median

Review the going the median w/ 5 / plot

$n \log n$ here

Sort

each agent can sort (parallel)

ith #

* Alice ith is median of all global
if it beats $n-i$ #s in Bob

5
If both sorted list

given ith # \rightarrow can check

b) is pivot } $\log n \rightarrow$ binary search
take half

Ok got a good hint here!

Sort $O(n \log n)$

Then binary search

Binary search

Finds key inside a sorted array

Looks at middle

match \rightarrow return

$<$ \rightarrow left

$>$ \rightarrow right

(course lang

time $O(\log n)$

6

I should be better at seeing $\lg n \rightarrow$ binary search

Log time

$$T(n) = O(\lg n)$$

base is 2 since binary $\lg_2 n$

Often binary trees or binary search

remember n is # of items!

So $n = 48$ means

$$\log_2(48) = 5.55$$

So takes 5

48

24

12

6

3

1

← 5 times!

$$n = 20$$

$$\log_2(20) = 4.3$$

20

10

5

2.5

1.25

← 4

⑦ $\log 100$ is 6.64

100

50

25

12.5

6.25

3.125 (6 times)

1.5

Anyway how many messages are sent?

So 1. Each sorts $O(n \lg n)$

2. Then Alice sends her median
which is $\frac{n}{2}$ -th #

Then Bob compares where it fits
↳ $O(\lg n)$ Binary search to find

I think I forgot $O(\lg n)$ sorted array $O(n)$ unsorted

8

Bob sends back ~~$\frac{2n}{2} - 1$~~ th pos

the position it is the

it should be the n th number in the combined list

So $\underbrace{12345}_{5th}$ $\underbrace{678910}_{0th}$ $n = 5$
 $\underbrace{\hspace{1.5cm}}_{= 5th = n} \textcircled{1}$

$\underbrace{1\ 3\ 5\ 7\ 9}_{\text{median is } 5}$ $\underbrace{2\ 4\ 6\ 8\ 10}_{\text{that would be 3rd here}}$
 $3th$

$3rd + 3rd = 6th \textcircled{x}$

but diff by 1

yeah subtract one from right
did that implicitly above
 $1st - 1 = 0th$

9

So that is in good case

1 2 3 4 5 6 7 8 9 10

3 is 3rd

3 would be $1st - 1 = 0th$

$$3 + 0 < 5 \quad \textcircled{x}$$

So binary search on 4.5 now?

4 is 4th?

\textcircled{x}

5th is 5th \textcircled{v}

Could we also try to jump $\frac{n}{2}$ of the difference

$$\text{So } (5 - 3 + 0) = \frac{2}{1} = 1 \text{ jump}$$

Yeah that is smarter

In this case they are the same

But I am confused if general statement ---

(10)

though ^{my} way (#2) also takes into account 2nd part
? so better?

b) Median Finding/Pivot alg

(search text book for median :))

Whole ~~the~~ chap on Medians I should have read!

Median chap in CLRS

Minimum(A)

Can scan through $O(n) \rightarrow n-1$

~~Now~~ find both in $2n-2$ by doing this twice

but we could also $3 \lceil \frac{n}{2} \rceil$ comparisons

by maintaining min, max items seen so far
process els in pairs w/ each other

3 compars for every 2 elements

Randomized-Select()

like Randomized-partition()

16

Partition (A, p, r) rearranges subarray $A[p \dots r]$ in place

Selects pivot

4 (possibly empty region)

i p j

2	8	7	1	3	5	6	4
---	---	---	---	---	---	---	---

used in
quicksort

p i j

2	8	7
---	---	---

p i j

2	8	7
---	---	---

p i j

2	8	7	1
---	---	---	---

p i j

2	1	7	8	3
---	---	---	---	---

p i j

2	1	3	8	7	5
---	---	---	---	---	---

$A[i]$ is pivot

1st section $\leq r$

2nd $> r$

^ values swapped ^ what is actually going on

(12)

Randomize-Select()

returns the smallest el of array $A[p \dots r]$

Go recursively ~~selecting~~ narrows $O(\lg n)$

Actually $\Theta(n^2)$ since partitioning $\Theta(n)$
if pick worst case

but $\Theta(n)$ expected

don't see why...
On each partitioning $\Theta(n)$
max n of those worst case

Some complex recurrence stuff I should know...

Selection in worst case lin time

(now on to main event...)

recursively partitions input array

but we guarantee a good split here

determines the smallest of input array $n \geq 1$

1. Divide n elements into $\lfloor n/5 \rfloor$ groups
of 5 el each with remainder
in a group

(13)
ie 26 into 5 grps 5 each
1 grp 1 ~~Ans~~

2. Find the median of each group
by insertion sorting each group

3. Use select recursively to find median of $\{n/5\}$ medians

4. Partition median of medians using
modified partition

k on low side

x is ~~the~~ k th smallest

$n-k$ on high side

5. If $i = k$, return x

Otherwise use select recursively to find

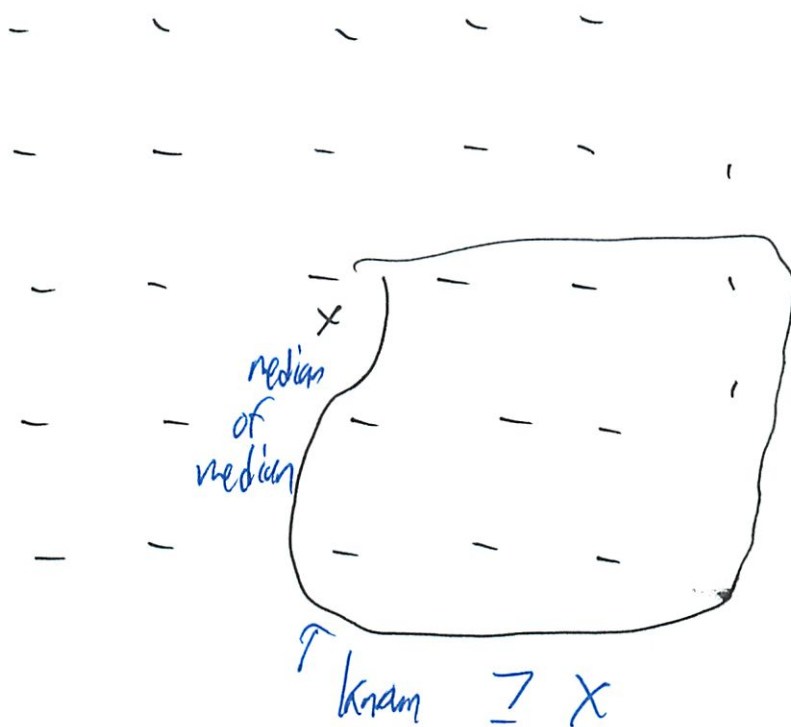
i th smallest el on low side if $i < k$

or ~~the~~ $(i-k)$ th smallest el on high side
if $i > k$

(skipping explanation of running time)
should have though

(14)

So



Run select on this

(but doesn't that mean it's now not 'red'?)

At least $\frac{3n}{10} - 6$ less than x

So select on at most

$$\frac{7n}{10} + 6 \text{ el}$$

So step 5 takes at most

$$T\left(\frac{7n}{10} + 6\right) \text{ time}$$

Show this is linear by substitution

15

Anyway back to our problem

it is deterministic

$O(n)$ to compute its median

But what do we do w/ messages?

Before we needed fast access to i th element
This appears the same linear search

Select(i th smallest e)

But the extra steps ~~a~~ can be done
and are still linear

according to fancy ~~para~~ math p 222

CLRS 3

(16)

(write up a)

but what is answer?

| 1 3 5 7 9 | 2 4 6 8 10 |
 ↑

5 would be

$$3rd + 2nd = 5 = n$$

What to return?

(here value = n \rightarrow confusing)

| 11 13 15 17 19 | 12 14 16 18 20 |
 ↑ ↑
 3rd 2nd

$$3 + 2 = 5 = n$$

return 3rd in Alice
but if Median in Bob?

⑥

11 13 17 19 20 | 12 14 15 16 18

17 is 3rd
would be 5th - 1 = 4

$$3 + 4 > 5$$

$$\frac{5-7}{2} = 1$$

13 is 2nd
would be 2nd - 1 = 1

$$2 + 1 < 5$$

Now know median is ~~A~~ Bob

So Bob takes over

know b/w ~~2nd~~ 2nd and 4th

Try that \rightarrow median 15th
is 3rd

that would be 2nd ~~AM~~ - 5th
? no - 1
but why?

No say 15th is ~~2nd~~ 2nd
then this is 1) no!

(18)

15 is 3rd + 1 = 4th

then other is 2nd - 1 = 1st

I hmm IDK

Oh I mis read 15 would be 3rd

- 1 = 2 (✓)

it is symmetrical

(write up b)

11/26

Can we find the i th?

10P

None

Well we do find the

Then see what i th on other list (i th)

$i + j = n$

we wiggle i and see what j does...

19

No the thing is how do we know what
it is w/o sorting!

* key TA hint Alice is median
if it beats $n-i$ th item in Bob

So example

1 3 5 7 4 | 2 4 6 8 10

↑
5 is median if it beats $n-i$ th item in Bob

or

5-3rd

2nd is 4 beats 0th

But 7 - or must start w/ median!

So another example

70

| 1 2 3 4 5 | | 6 7 8 9 10 |

↑
3

1 - 1

5 - 3 = 2nd item

⊗ No

| 7 4

5 - 4 = 1st item

⊗ No

5 5 - 5 = 0th item

'I guess then it works?
always

Hmm seems to work - why?

| 1 2 3 6 7 | | 4 5 8 9 10 |

3 beats 2nd ⊗

6 beats 1st ⊙ yes

So switch?

2nd beats 3rd? ⊙ yes

(21)

Weird - I don't get why it works

And couldn't prove --

How do we adjust?

If don't beat, lower

But what if other way? Can it ever be?

1 2 6 7 8 → 3 4 5 9 10 ✓

6 beat 4 (X) No

7 3 (X) No

8 3 (O) always

↑ clearly not right

Then try other way

5 beat 6 (X) No

(22)

So if we alternate

A:	6	beat	4	⊗ No
B:	5		6	⊗ No
A	7		9	⊗ No
B	9		2	

No mistake

5 2 ⊙ yes

So 5 walked

| 11 12 17 18 19 | 13 14 15 16 20 |

A 17 beat 14 ⊙ yes
A 18 beat 13 ⊙ yes
A 19 beat —
B 15 beat 12 ⊙ yes

(23)

What do we do if doesn't work?

Some sort of binary search

Try to get median m in Bob's not middle

11 17 18 19 20 | 12 13 14 15 16

A 18 beat 13 ✓

B 14 beat 17 x Then what

A 14 beats 12 ✓

A 20 beats 1

A 17 beats 14 ✓

A 11 beats 16 (x)

~~B~~ B 15 beats 11 (✓)

Hmm should have asked

Thought I knew...

24

Offset

$$\frac{18-13}{2} \quad (\text{these are \# not positions})$$

~~(1 - long i)~~
is always

$$(n-i) - i$$

is always $n - 2i$

~~18-13~~ $\frac{3-2}{2} = .5$

~~19-14~~ $\frac{4-1}{2} = 1.5$

Or just make a binary jump in A
just say that

(25)

So

| 11 17 18 19 20 | | 12 13 14 15 16 |

A 18 beat 13 ✓

B 14 beat 17 ✗

So ~~20 - 18~~ position wise
~~19~~

18 beat 12

Also I think I'm doing offset wrong

~~11 17 18 19~~

11 16

17 15

18 14

19 13

20 12

18 beat 14 ✓

~~14 beat 13~~ flip

19 beat 13 ✓

14 beat 18 ✗

13 beat 19 ✗

(26)

of course not!

15 beat 17 (X) No now!

So was I doing offset right?

Rethink hint

Alice's ith is Bob's median if it beats
 $n-i$ #s in Bob

So $n-i$ #s

So $\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}$

3 beats $5-3=2$ No

4 1 No

5 0 (V) yes

$\{1, 3, 5, 7, 9\}, \{2, 4, 6, 8, 10\}$

5 beats 2 yes (V)

$\{11, 12, 17, 18, 19\}, \{1, 3, 14, 15, 16\}$ ²⁰

17 beats 2 # in B (V) yes

but not median

(27)

or it beats exactly 2 # in Bob

where no 17 beats 4 #s

15 beats exactly 2 in Alice!

Should have read closer earlier!

Do I need to fix a?

LI used a different rule

But how do we get primitive of counting $<$ then

Do a subset on

Then we know how many $<$

~~Look up $(n-i)$ th~~

~~Must~~ No for last example must be exactly $n-i$
 $= 2$

$<$ 17 in B

So look up 2nd 3rd \leftarrow 2nd \leftarrow 1
all must $<$ 17
except 3rd

(28)

That is a vedy

If doesn't work regular binary search on A

If exist \rightarrow do on B

That makes so much more sense!

I think I've got it!

if too many on B

Alice moves smaller

too few on B

Alice moves Bigger

⑦

We can look that up w/ our $\text{SELECT}(n-i)$ on Bob.

We also try this starting at the median of Bob
and running $\text{SELECT}(n-i)$ on Alice.

When they both work (local i th ~~value~~ & remote $(n-i)$ th)
We have found the median at the local i th value.

If it does not work, we binary search Alice
as before where we jump half the distance remaining.

Michael Plasmeier
6.046 R07

P-Set 5 #1
OHI TA

Task Scheduling Approximation

The approximation algorithm outlined in the P-set instructions is a 2-approximation algorithm to the NP-hard optimal algorithm.

$$C \leq 2C^*$$

ans ↑ approximation ↑ optimal-sol answer

To do this we need to show that our approximation algorithm will not result in ~~more~~ too many processors being used at $\leq 50\%$.

②

Lets consider a ^{optimal} ~~perfect~~ arrangement of tasks on machines. All are $\leq 100\%$ utilized.

Now lets double the number of machines.

Now the average utilization of the machines is $\leq 50\%$ ~~can~~ (at maximum 50%)

Now we could consolidate machines back so we are at the original number of machines, (since we came from that)

So for our approximation algorithm we need to show that the average ^{utilization} of machines is always $\geq 50\%$ ~~when we have 2x # of optimal machines.~~
to show that we have a 2-approximation algorithm.

③ Back to the approximation algorithm:

As we assign tasks to machines, we will never be at a point where the average $\leq 50\%$.

If the average is less than ^{or =} 50% then we have some machines that we can combine since

$$\leq 50 + \leq 50 = \leq 100$$

But we will never get to this point, since we add ~~various~~ tasks to the first free machine.

If a machine had $\leq 50\%$, a $\leq 50\%$ task would be added to it. Thus our average ^{utilization} never falls below 50% and

We saw 2 * optimal is at most 50% so

we are always better off than 2 * optimal # of machines



Michael Plasmele
6.046 R09

P-Set 5 #2

Queue

We can simulate a FIFO queue using two LIFO queues.

We have one LIFO queue used to insert elements.
We have another LIFO queue used to remove elements.
When our ~~LIFO~~ removal queue is empty, we fill it from our insertion queue.

class FIFO from LIFO:

```
def __init__(self):
```

```
    self.insertq = new LIFO queue();  
    self.removeq = new LIFO queue();
```

②

```
def enqueue(item):  
    self.insertq, appendpush(item)
```

```
def dequeue(k):  
    answer = []  
    while (len(answer) < k):  $\leftarrow \text{len(answer)} < k$   
        if (self.removeq, lenght() == 0):  
            while (self.insertq, length() != 0):  
                self.removeq, appendpush(self.insertq, pop())  
            answer.append(self.removeq, pop())
```

^ Though dequeue(k) could be more cleverly written
to pop(k) if our LIFO queues support ~~pop(k)~~
and reloads when needed
ie pop(len(removeq))
reload()
pop(rest)

(3)

Now proof that it meets our timing constraints.

An enqueue() is always $O(1)$

A dequeue() can take $O(1)$

but if it must "reload" it can take
up to $O(n)$

But let us think about this. An item
is only "reloaded" once. So each
item is only touched exactly 4 times,

- added to insert $q \in \text{enqueue}()$
- removed from insert q
- added to removal $q \in \text{reload}()$
- removed from removal $q \in \text{dequeue}()$

So by the accounting method we have $O(4)$

which is $O(1)$,

↑ does not depend
on # of items
in list.

Michael Plawalec
6.046 R07

P-set 5 #3
Tol
04/14

Distributed-Median

a) Niere method.

We can have Alice and Bob each
independently sort their lists $O(n \lg n)$
Now Alice sends her median to Bob. ^(and quicksort)

This is the $\frac{n}{2}$ -th number

Then Bob compares where it would fit in his list
 $O(\lg n)$ using binary search

He returns the position where the item will
fit in his list $- 1$.
_{subtract}

ie start of list = 1st position \rightarrow return 0
after 1st element = 2nd position \rightarrow return 1
at end of list = $n+1$ th position \rightarrow return n

⑦

Then Alice adds the position of the median in her list with the result from Bob.

If that value is $= n$, return answer.

If that value is $< n$, we are too low
" " " " $> n$ " " " high

We then calculate our offset

$$\left[\frac{\begin{array}{cc} \text{Bob} & - \text{Alice} \\ \text{ith} & \text{ith} \\ \text{returned} & \\ \text{value} & \end{array}}{2} \right]$$

and move that many in Alice's list

Repeat till solve

↳ will be binary search because each time we go to median of our offset

We return the 'ith' value in Alice

(see later for if no 'ith' value in Alice works)

(3)

Example



Median of Alice = 3 in 3rd position

Alice sends 3 to Bob

3 would be in 6th position w/ Bob

Bob returns 0

$3 + 0 < 5 \rightarrow$ too low

Offset is $\frac{5-3}{2} = 1$

So try 4th item of Alice

4 sent to Bob

Bob returns 0

$4 + 0 < 5 \rightarrow$ too low

Offset $\left\lceil \frac{5-4}{2} \right\rceil = 1$

⓪

④

So 5th of Alice

Send 5 to Bob

Bob returns 0

$$5 + 0 = 5 \rightarrow \text{return answer} = 5!$$

If the ~~Median~~ ~~is in Bob~~ linear search does not converge

Then the median is in Bob!

Alice sends Bob "check you" = "cy"

We know the highest "too low" value in Bob
lowest "too high" " " "

So we can take the median of the position.

Then Bob checks that and confirms with Alice

What position the number would be in her list
in the same way

(5)

Example 2

11 13 17 19 20 12 14 15 16 18
Alice Bob

we know answer is 15
↓

A sends B 17

B returns 4

$$3 + 4 > 5$$

$$1 \left\lceil \frac{5-7}{2} \right\rceil = 1$$

A sends B 13

B returns 1

$$2 + 1 < 5$$

A sends B "cy"

B knows b/w 4th and 2nd

B checks 3rd position

B sends A 15

A returns 2

$$3 + 2 = 5$$

B returns answer = 15

6)

b) Running Time $O(n)$

Here we substitute the ~~merge~~ sorting of each list with the $\text{SELECT}(i)$ algorithm in (LRS v3 Section 9.3).

This algorithm divides our set into $\lfloor n/5 \rfloor$ groups

We then follow the same procedure as before.

Alice starts at her median $O(n)$

Checks location w/ Bob $O(1)$

~~Who search for the~~

We are good if this is bigger than ~~the~~ exactly

$n - i$ values ~~at~~ in Bob.
↑ ↑
of items position of
in each Alice's #.

⑦
We can check this by $\text{SELECT}(\text{Bob}, n - i + 1)$

We should get back a value that is \geq Alice's median
with ~~all~~ all ^{others} values $<$ Alice's median. There should
be $n - i$ of those values.

If it doesn't work, we binary search on Alice
moving the median we check half of the
remaining value.

if too many on Bob \rightarrow Alice moves smaller
if too ~~many~~ ^{few} on Bob \rightarrow " " bigger

We can run multiple SELECT s all within
 $O(n)$ by the analysis in CLRS v3 p222.

If we run out of values on Alice, we check
values on Bob. by the same procedure.

(8)

This is $O(n)$ ^{On each side} ~~to each~~ since

SELECT takes $O(n)$, including narrowing down the result set further.

$O(\lg n)$ messages are sent back + forth because of Binary search,

If we need to go to Bob to start checking, we end up doing double the work, but this is asymptotically the same.

Problem Set 5 Solutions

This problem set is due at **11:59pm** on **Tuesday, November 27, 2012.**

- Exercise 5-1.** Do Exercise 11.3-6 in CLRS on page 269.
- Exercise 5-2.** Do Exercise 11.5-1 in CLRS on page 282.
- Exercise 5-3.** Do Exercise 35.1-4 in CLRS on page 1111.
- Exercise 5-4.** Do Exercise 35.2-2 in CLRS on page 1116.
- Exercise 5-5.** Do Exercise 17.1-3 in CLRS on page 456.
- Exercise 5-6.** Do Exercise 17.2-2 in CLRS on page 459.
- Exercise 5-7.** Do Exercise 17.3-2 in CLRS on page 462.

Problem 5-1. Task Scheduling Approximation

There are n tasks t_1, t_2, \dots, t_n , each of which has an associated cost $c_i \in \mathbb{R}$, such that

$$0 < c_i < 1$$

A task t_i consumes c_i fraction of the resources when run on a “standard” computer. A set of tasks T can be run on the same computer simultaneously only if

$$\sum_{i \in T} c_i \leq 1$$

i.e. there are enough resources to run all tasks in T .

You would like to run all tasks simultaneously on identical “standard” computers. Multiple tasks can run on a single computer if the constraint given above is satisfied. Each task must be run on a single computer, i.e. you cannot run task t_i partially on different computers. Since computers cost money, you would like to use the minimum number of computers to run all tasks. Sadly, it turns out that finding the minimum number of computers you need to run all tasks is an NP-hard problem.

In desperation, you decide to pursue the greedy approach. You iterate over the tasks and for each task, you assign the task to run on the first computer that can accommodate the task. If there is no computer on which you can run the task, you add a computer and assign the task to run on the added computer. You keep the computers in the order they were added.

Prove that the greedy approach yields a 2-approximation.

Solution:

Let

$$S = \sum_{i=1}^n c_i$$

Since each computer has the limit of 1, the optimal number of computers m^* is at least $\lceil S \rceil$.

Let m be the number of computers obtained by the greedy approach. It's clear that $m \geq m^*$ because m^* is the minimum possible number of computers. Let's show that $m \leq 2m^*$.

In the greedy solution, let T_j denote the set of tasks run on the computer j and let a_j denote the total load on the computer j , i.e.

$$a_j = \sum_{i \in T_j} c_i$$

There can be at most one j such that $a_j < .5$. We can prove by contradiction. Suppose there are $j_1 < j_2$ such that $a_{j_1} < .5$ and $a_{j_2} < .5$. Notice that

$$a_{j_1} + a_{j_2} < 1$$

This means that we would have scheduled tasks in T_{j_2} to run on j_1 (or some other computer) because the computer j_1 could actually accommodate the tasks and there was no need to add the computer j_2 . This contradicts the greedy approach solution.

Since there can be at most one j such that $a_j < .5$, we have at least $m - 1$ computers where the total load is at least $.5$. Therefore,

$$S = \sum_{j=1}^m a_j > .5 * (m - 1)$$

This yields $2S + 1 > m$. Since m is integer, $m \leq \lceil 2S \rceil$. Finally, we can show that

$$m \leq \lceil 2S \rceil \leq 2 \lceil S \rceil \leq 2m^*$$

This completes the proof that the greedy approach yields a 2-approximation.

Problem 5-2. Queue

We would like to implement a FIFO (first in, first out) queue that supports the following operations:

- **ENQUEUE(item)**: appends the item to the back of the queue. The operation must always succeed.
- **DEQUEUE(k)**: k must be a positive integer. The operation pops and returns k elements from the front of the queue. If the total number of the elements in the queue is less than k , then the operation returns all items in the queue. If the queue is empty, it returns *NONE*.

Suppose that to implement the queue you can use two LIFO (last in, first out) stacks, each of which supports the following operations:

- **PUSH(item)**: pushes/inserts the item to the stack. The operation always succeeds.
- **POP()**: removes and returns the most recently pushed item in the stack. If the stack is empty, it returns *NONE*.

Assume that for a stack, each operation costs 1 unit in the running time.

Give an implementation of the queue using two stacks such that both operations **ENQUEUE** and **DEQUEUE** have an amortized cost of $O(1)$. The amortized cost of **DEQUEUE** should be $O(1)$ regardless of the input k .

Solution: Consider the following implementation of the queue using stacks S_1 and S_2 .

- **ENQUEUE(item)**: takes the item and pushes the item to the first stack S_1 .

- **DEQUEUE(k)**: first, it starts popping items from S_2 . If it successfully pops k items, then it returns those items. If the stack S_2 becomes empty, then we “dump” the contents of S_1 into S_2 by popping an element from S_1 and immediately pushing it into S_2 until S_1 becomes empty. Now pop the remaining items from S_2 until S_2 is empty or the total of k items were popped out. Return *NONE* if no elements were found, otherwise return all popped items.

The correctness follows from the fact that when we “dump” S_1 into S_2 , the item order gets reversed and the oldest item now is at the top of the stack S_2 . When both S_2 and S_1 are not empty, stack S_2 contains the oldest items in the correct order to be popped out and S_1 contains the newest items in LIFO order.

We can use the accounting method to prove $O(1)$ amortized cost. Let’s charge 3 for *DEQUEUE* operations and 5 for *ENQUEUE* operations. The actual cost of *ENQUEUE* is 1, thus each item in S_1 has a surplus of 4. Existing elements in S_2 were popped from S_1 and pushed into S_2 , thus elements in S_2 have surplus of 2. This surplus can be used to remove the element from S_2 when doing *DEQUEUE*. 1 unit can be used to pop and another unit for the additional computations *DEQUEUE* does when removing elements in a batch.

3 units in *DEQUEUE* can be used to pay for pop operations that result in *NONE* when doing *DEQUEUE*. There are three possible *NONE*s when popping: one *NONE* comes from reaching the bottom when doing initial popping from S_2 ; the second *NONE* comes when reaching the bottom of S_1 upon dumping; and the last *NONE* comes from possibly reaching the bottom of S_2 when popping after dumping.

The accounting method shows that both operations have an amortized cost of $O(1)$ and the amortized cost of *DEQUEUE* doesn’t depend on k .

Problem 5-3. Distributed Median

Alice has a list of n numbers and Bob has another list of n numbers. Jointly, they have $2n$ numbers that are distinct from each other. They would like to know the median element of their combined arrays. Since $2n$ is even, let the median be the n th smallest number.

They have limited communication bandwidth and would like to minimize the communication cost. They can send each other messages and each message can contain one of the following: an integer in range $[0, n]$, one of the numbers they have, or an English word that contains no more than five letters.

- Give a deterministic algorithm where the number of messages used is $O(\log n)$ and the running time is $O(n \log n)$. Alice and Bob can follow different protocols.

Solution: Let Alice and Bob sort their numbers using an $O(n \log n)$ sorting algorithm. Suppose that Alice has numbers $A[1], A[2], \dots, A[n]$ and Bob has numbers $B[1], \dots, B[n]$ sorted in the increasing order.

For now, assume that Alice has the median number. Notice that $A[i]$ is median if and only if

$$B[n - i] < A[i] < B[n - i + 1]$$

because all numbers are distinct and the median is the n th smallest number.

Alice does binary search on her own sorted list. Alice chooses i to be the middle of the remaining numbers and asks Bob to send her $B[n - i]$ and $B[n - i + 1]$. If $A[i] < B[n - i]$, then $A[i]$ is too small and she recurses into the top half of the current range. If $B[n - i] < A[i] < B[n - i + 1]$, then $A[i]$ is the median and she sends the success message to Bob and let's him know what the median is. If $A[i] > B[n - i + 1]$, then $A[i]$ is too big and she recurses into the bottom half of the current range.

If Alice starts from the range $[1, n]$, then each time she will cut the size of the range in half. If she doesn't have the median, she will have no range to recurse on and will terminate in $O(\log n)$ steps. If she has the median, then she is guaranteed to find the median in $O(\log n)$ steps.

If Alice fails to find the median, let Alice and Bob switch their roles. Since Alice doesn't have the median, Bob must have the median and he is guaranteed to succeed in $O(\log n)$ steps. The total number of messages is still $O(\log n)$. The running time is dominated by sorting, which is $O(n \log n)$.

- (b) Modify your algorithm to have the running time of $O(n)$ while keeping the number of messages to be $O(\log n)$. The resulting algorithm should also be deterministic.

Solution:

Instead of sorting, let both Alice and Bob use the linear order statistic algorithm whenever they need to find a certain element. Each time the recursion is done, both Alice and Bob can reduce the number of elements they need to consider by half. If the range has size m , then the order statistic takes $O(m)$ time to complete because some order information is already known from the previous steps. Thus the running time becomes

$$T(n) = T(\lceil n/2 \rceil) + O(n)$$

This solves to $O(n)$ running time.

Binary Counter Intro

$A [0 \dots k-1]$ of bits
 \uparrow low order \uparrow high order

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i = \text{'integer value'}$$

To Increment():

$i = 0$

while $i < A.\text{len}$ and $A[i] = 1$

$A[i] = 0$

$i = i + 1$

if $i < A.\text{len}$

$A[i] = 1$

} just add 1

Carry
add 1
to pos i
Carry

So $O(k)$ worst case when all 1s

0111 \rightarrow 1000

So $O(n \log n)$ for n times

②

But not all bits flip each time

Total # of flips

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} \\ = 2n$$

So bit 0 flips every time
1 every other ie $\lfloor n/2 \rfloor$
2 $\lfloor n/4 \rfloor$ times, every fourth time
:
i $\lfloor n/2^i \rfloor$

So $O(n)$ total

or $O(n)/n = O(1)$ each time

③

Accounting method

amortized cost

get credit

$$\sum_{i=1}^n \hat{C}_i \geq \sum_{i=1}^n C_i$$

↑
amortized cost

$$\text{Credit} = \hat{C}_i - C_i \text{ must be non neg}$$

STUDENT DETAIL

Gradebook for /course/6/fa12/6.046J


 Michael Plasmeier
 Cumulative Grade: 101

R07

Comment:

View More

Sort By:

Filter...

	Due Date	Points	Max Points	Weight
Problem 1-1	09-25-2012	3.00	3	1
Problem 1-2	09-25-2012	3.00	3	1
Problem 1-3	09-25-2012	3.00	3	1
Problem 1-4	09-25-2012	2.00	3	1
Problem 2-3	10-05-2012	2.00	3	1
Problem 2-1	10-05-2012	2.00	3	1
Problem 2-2	10-05-2012	2.00	3	1
Problem 2-4	10-05-2012	1.00	3	1
Quiz 1	10-11-2012	32.00	80	80
Problem 3-1	10-25-2012	1.00	3	1
Problem 3-2	10-25-2012	1.00	3	1
Problem 3-3	10-25-2012	3.00	3	1
Problem 3-4	10-25-2012	2.00	3	1
Problem 4-1	11-06-2012		6	1
Problem 4-2	11-06-2012		6	1
Quiz 2-1	11-14-2012	10.00	15	0
Quiz 2-2	11-14-2012	10.00	20	0
Quiz 2-3	11-14-2012	2.00	20	0
Quiz 2-4	11-14-2012	7.00	25	0
Quiz 2-5	11-14-2012	15.00	20	0

As of
11/27

6.046

11/27

Intro to Crypto Pt 1

(3 min late)

Hash fn

Random Oracle Model

Desirable properties

Applications to Security

Crypto is much harder to calculate
↳ more intensive

Hash Fn

(review to get terms right)

A hash fn maps arbitrary strings of data to
fixed length

- output deterministic (same each time)
- public (unless keyed - which we sometimes do)
↳ same ans each time

②

- "random" manner

- inputs uncorrelated w/ output

- Small change in input = large change in output

- ∞ ^{input} domain and ^{output} fixed range

want domain spread evenly among the range
any ~~input~~ output is equally likely for any input

- ^{input} $h\{0,1\}^*$ arb length

\uparrow encode characters in binary

- output $\{0,1\}^d$

\uparrow length of hash

256
1024

- no secret key today

~~but~~ - anyone can compute h in poly time
(mixed)

6.006 \rightarrow Simple mod math

(3)

- No Sorting \rightarrow that is $O(n \lg n)$
- must be linear in length of key
 - assumed that length is constant
 - so constant time look up
- but we need much more complex hash w/ crypts
so poly time in length of input

- MD4 and MD5, SHA1, SHA-256, SHA3

<u>MD4</u>	and	<u>MD5</u>	<u>SHA1</u>	<u>SHA-256</u>	<u>SHA3</u>
128 bit		128 bit	160 bit	256	recently completed

Secure hash functions
Some extra properties

- hard to invert
- hard to find collisions
- etc

MD4, MD5 ~~are~~ broken now - can find collisions
SHA-1 - find collisions 2^{69} so on edge of broken

④

SHA-256 still good
SHA-3 just established

What properties should they have?

Ideal: Random Oracle Model

(not achievable in practice)

On input $x \in \{0, 1\}^*$

has a book anyone can read

When time starts \rightarrow book empty

People ask Oracle for hash of x

If not in book, Oracle flips coin d times

Oracle records value in book

$(x, h(x))$

If in book, return y where (x, y) in book

⑤

Anyone has access to book

book grows infinitely

(unbounded space)

↳ non bounded space

look ups in books are ~~instant~~ non poly time

↳ since input len is ~~unbounded~~ unbounded

So not feasible in practice

- 'if previous' → returns 'it

- 'if new' → returns random

Desirable Properties

- one way (OW)

infeasible given $y \in_R \{0, 1\}^d$

to find any x s.t. $h(x) = y$

Can't ~~take~~^{use} output to get input

(that created that output)

trivial example: $h(x) = x \bmod 11$

takes ^{infeasible =} exponential time in d

6

2. Collision Resistance (CR) Infeasible to find x, x'
s.t., $x \neq x'$ and $h(x) = h(x')$

Want no collisions - breaks security

~~can't~~

like identity theft

a single collision bad

2b. Target Collision Resistance (TCR) Infeasible given
 x , to find $x' \neq x$ s.t., $h(x) = h(x')$
finding a specific x that collides.

If have CR then have TCR
but not $TCR \rightarrow CR$

3. Pseudo-randomness () Want same thing back for
Same key, but should look random
- behavior indistinguishable from Random Oracle

9

4, Non Malleability (NM) Infeasible given $h(x)$
to produce $h(x')$ where x and x' are related.

eg. $x' = x + 1$

Small changes input \rightarrow large changes output
large " " \rightarrow " " "

Most subtle property

Must have all these properties

Applications to Security

Passwords

Canonical example

to store passwords on your disk

20 ~~bits~~ characters

8 bits

= ~~160~~ 160 bits

⑧

If you lose your laptop, don't want your other accounts to be hacked

→ don't want /etc/passwd to store actual password
instead they are hashed!

(1) Store $h(PW)$, not PW on computer

Each time login, compute $h(\text{typed } PW)$

Compare w/ $h(\text{stored } PW)$

Must =

Disclosure of $h(PW)$ should not reveal ~~the~~ PW
One-way

TCA would be nice to have → So some other
 PW can't also unlatch the door

9

h is CR \rightarrow h is TCR

but not other way

h is OW \nleftrightarrow h is CR, TCR
'not at all related!'

'inversion' in $O(2^d)$ just through exhaustive enumeration
ie brute force

but how long to find a collision?

this is the birthday problem

if n people

some $\sqrt{\quad}$ property since look at
all pairs (n^2)

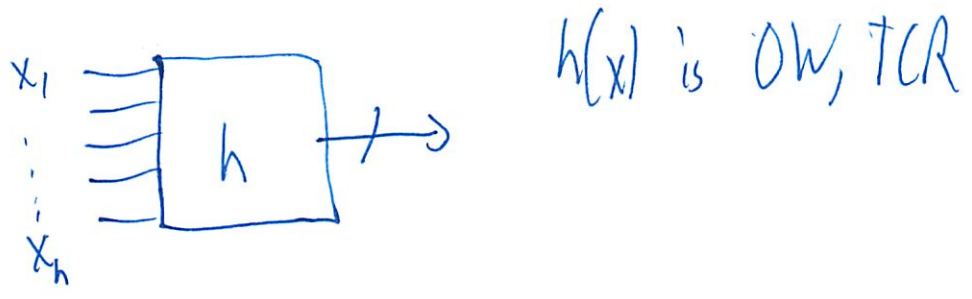
any pair could collide

each map to random pt w/ hash-fn

so collisions $O(2^{d/2})$ time

10

One way and Collision Resistance



If want to break TCR property (OW does not imply TCR)



↑ XOR of 2 new inputs a, b is x_1 "input"

$h'(a, b, x_2, \dots, x_n)$ ~~is still OW~~

~~but~~ is still OW

but swapping a, b is ~~not~~ targeted collision

So not TCR

$x \rightarrow a=1, b=0$ vs $a=0, b=1 \leftarrow x'$

we find a x' for a x

s.t. $h(x) = h(x')$

①
TCR does not imply OW

h is OW, TCR

$$h'(x) = \begin{cases} 0 \parallel x & \text{if } |x| \leq d-1 \\ 1 \parallel h(x) & \text{otherwise} \end{cases}$$

\downarrow concat

But what about OW-ness of $h'(x)$

We can invert for a lot of values w/ 0 in front

$$h'(x) = \underbrace{0 \parallel 0 \parallel 0 \dots}_{d=\text{len}}$$

Says we have trivial inversion \rightarrow use the rest to invert

Hard to break TCR of this

if have y and $h'(y)$

Can we find y' so that $h'(x) = h'(y')$?

Hard to do!

top line \rightarrow just the value

bottom line $\rightarrow h(x)$ is collision resistant as well

So properties are distinct!

(12)

File Corruption

ie on web server as you download
or when give file to backup provider

So for each F , store $h(F)$ yourself
↳ Can't allow corruption
- 160 bits

Check if file modified by recomputing $h(F_{retrieved})$

So need TCR

$F \rightarrow F'$

s.t. $h(F)$ stays $h(F')$

(13)

Digital Signatures

⚡ (not the schemes today)

Pk_A = Alice Public key

Sk_A = Alice Private (Secret) key

Signing | $\overset{\downarrow \text{Signature}}{\sigma} = \text{sign}(Sk_A, M) \leftarrow \text{Alice}$

Once Alice has done this,

Verify | $\text{verify}(M, \sigma, Pk_A) = \text{true or false} \leftarrow \text{anyone}$

Can authenticate Alice as the only person who signed this document

↳ well anyone who knows Sk_A which is hopefully only Alice

For large M , to sign is $O(m)$
↳ # of bits

So instead sign $h(m)$
 $\sigma = \text{sign}(Sk_A, h(m))$

(14)

~~Adversary wants to forge Alice's sig~~
not today, instead

Bob gets Alice to sign X , then claims she signed X'

$$\text{if } h(x) = h(x')$$

Collision resistance

Bob wants to write a 2nd contract

Alice signs hash of original

Bob says you signed 2nd contract

Bob can choose X, X'

↳ so not TCR

OW does not ~~matter~~ matter here
You have the contract

(15)

Actions + Commitments

Promise to do things

Are beholdent to those

Person who made commitment is forced to honor
that commitment

Sealed bids

Alice has value x

Alice computes $C(x)$

^{commitment}
and submits it as a ^{sealed} bid

When bidding is over, Alice opens sealed bid $C(x)$
to reveal x

(6)

Properties of Commitment

$C(x)$ is hash fn $H(x)$

How can Alice dupe auctioneer

Bob " "

auctioneer " Alice or Bob

We don't know who is bad guy?

What can ~~the~~ go wrong?

Secrecy \rightarrow don't want $C(x) \rightarrow x$
Otherwise know what bid is
So One-way

CA - otherwise Alice could claim lower value
ie if 10, 10 million collide

\uparrow \uparrow Claim bid lot
Then later claim 10

Alice should be bound to commitment

(17)

Non malleability can't modify so slightly bigger

Given $C(x)$ shouldn't be possible to
produce $C(x+1)$

(slightly diff than (R, OW))

↳ Don't know x

but can't make x' related to x

if Alice bids $C(x)$

Bob can't bid $C(x+1)$ even w/o knowing x

Bob doesn't know how much he bid

Alice would need to unsal lot

Bob knows he is close, guaranteed higher

Tries all the values nearby

Must be careful w/ range of x values

$$C(x) = h(r || x)$$

↑ random value added to x
Since x may have small range

(18)

get people to prepend random values
then when open bid \rightarrow drop random values

①

1/27

6.046

L20

Intro to Cryptography : Hashing (Part I of II)

Hash functions

Random oracle model

Desirable properties

Applications to Security

Hash Functions

A hash function maps arbitrary strings of data to fixed length output in deterministic, public, "random" manner.

$$h : \{0,1\}^* \rightarrow \{0,1\}^d$$

strings of arbitrary length ≥ 0 strings of length d

Hash Functions

(2)

No secret key. All operations public.

Anyone can compute h , poly time computation

Examples: $\underbrace{MD4, MD5}_{128}$, $\underbrace{SHA-1}_{160}$, $\underbrace{SHA-256}_{256}$, $\underbrace{SHA-512}_{512}$

d : 2^6 ✓ 2^{37} ✓? 2^{64}

Ideal: Random Oracle

(not achievable in practice)

Oracle: on input $x \in \{0, 1\}^*$

if x not in book

flip coin d times to determine $h(x)$

record $(x, h(x))$ in book

else: return y where $(x, y) \in \text{book}$

Gives random answer every time, except as required for consistency with previous answers. (h must be deterministic)

In practice, \nexists RO so need something "pseudo random"

Desirable Properties

OW ① "one-way" (pre-image resistance)
 Infeasible, given $y \in_R \{0, 1\}^d$ to
 find any x s.t. $h(x) = y$
 ↑ "pre-image" of y

CR ② Collision-resistance (strong collision resistance)
 Infeasible to find x, x' , s.t. $x \neq x'$
 and $h(x) = h(x')$ (a "collision")

TCR ③ Weak collision resistance (target CR,
 2nd pre-image resistance)
 Infeasible given x , to find $x' \neq x$
 s.t. $h(x) = h(x')$

PRF ④ Pseudo-randomness
 Behavior indistinguishable from RO

NM ⑤ Non-malleability
 Infeasible, given $h(x)$, to produce
 $h(x')$ where x and x' are "related"
 (e.g. $x' = x + 1$)

Informal definitions. Formal requires family
 of hash functions

(4)

Facts

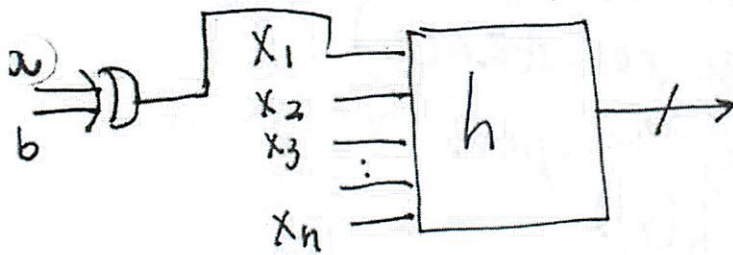
h is CR $\Rightarrow h$ is TCR (but not reverse)

h is OW $\nleftrightarrow h$ is CR, TCR (neither impl. holds)

Collisions can be found in $O(2^{d/2})$ - birthday attack

Inversion can be found in $O(2^d)$

Examples



OW \nRightarrow TCR

$h(x)$ is OW, CR
 $h'(a, b, x_2, \dots, x_n)$
 is still OW, but
 not TCR

$$h'(x) = \begin{cases} 0 \parallel x & \text{if } |x| \leq n \\ 1 \parallel h(x) & \text{otherwise} \end{cases}$$

h is OW, CR, but h' is TCR, not OW

TCR \nRightarrow OW

Applications (contd.)

④ Commitments

Alice has value x (e.g., auction bid)

Alice then computes $C(x)$ and submits it as her bid
"commitment to x "

$C(x)$ is her "sealed bid"

When bidding is over, Alice "opens" $C(x)$
to reveal x

Binding : Alice should not be able to open $C(x)$ in multiple ways.

Secrecy : Auctioneer seeing $C(x)$ should not learn anything about x

NM : Given $C(x)$ shouldn't be possible to produce $C(x-1)$

Need: NM, CR, OW (really need more for secrecy!)
 $h(x) = h(x) \parallel \text{msb}(x)$

How: $C(x) = h(r \parallel x)$ $r \in_R \{0,1\}^{256}$

to open reveal r & x

randomized

← This could be OW but
expose most significant
bit and break secrecy!

Applications

① Password storage

- Store $h(PW)$, not PW , on computer
- Use $h(PW)$ to compare against $h(PW')$ where PW' is the typed password
- Disclosure of $h(PW)$ should not reveal PW
- Need OW.

② File modification detector

- For each file F , store $h(F)$ securely (on DVD)
- check if F modified by recomputing $h(F)$
- need TCR (adversary wants to change F but not $h(F)$)

③ Digital signatures

PK_A : Alice's Public key
 SK_A : Alice's Private key

Signing: $\sigma = \text{sign}(SK_A, M)$

Verify: $\text{verify}(M, \sigma, PK_A) = \text{true/false}$

Adversary wants to forge a signature that verifies
 For large M , easier to sign $h(M)$ $\sigma = \text{sign}(SK_A, h(M))$
 Need CR, don't need OW. Alice gets Bob to sign x , then claims he signed x' , if $(h(x) = h(x'))$

Crypto Part 2

Encryption

1/28

- Symmetric key encryption
- Key exchange
- Asymmetric key exchange
 - RSA
- NPC problem + Crypto; knapsack crypto

→ 2 main crypto primitives: hashing + encryption
Today 2 varieties of encryption ↑
yesterday ↑
today

WW2 schemes: broke if had both cipher + plain text

We have better algorithms today

Even if you have a million cipher, plain text pairs

Should be hard to decrypt new messages

How do you set up the keys?

Want hard problems

People wanted crypto based on NP-complete

Efforts pretty much failed

Why?

②

Symmetric key Encryption

$$C = e_k(m)$$

Diagram illustrating the encryption process:

- C : cipher text
- e_k : encryption fn + key
- m : message / plain text

$$m = d_k(c)$$

Diagram illustrating the decryption process:

- m : get back same message
- d_k : decryption - same key
- c : cipher text

e, d must be inverses of each other

↳ end up being bijections

must know key k in both cases

~~the~~

Generally the algorithms have reversible operations
(unlike or hashes)

- XOR - Since doing twice cancels out

$$X \oplus Y = Z$$
$$Z \oplus Y = X$$

③

- + / -

- multiplication / division - though hard since fixed # of bits
- shift right / left (circular)

Proofs not interesting

having enough cards to scramble your message

want 2^k amt of work to decrypt w/o key

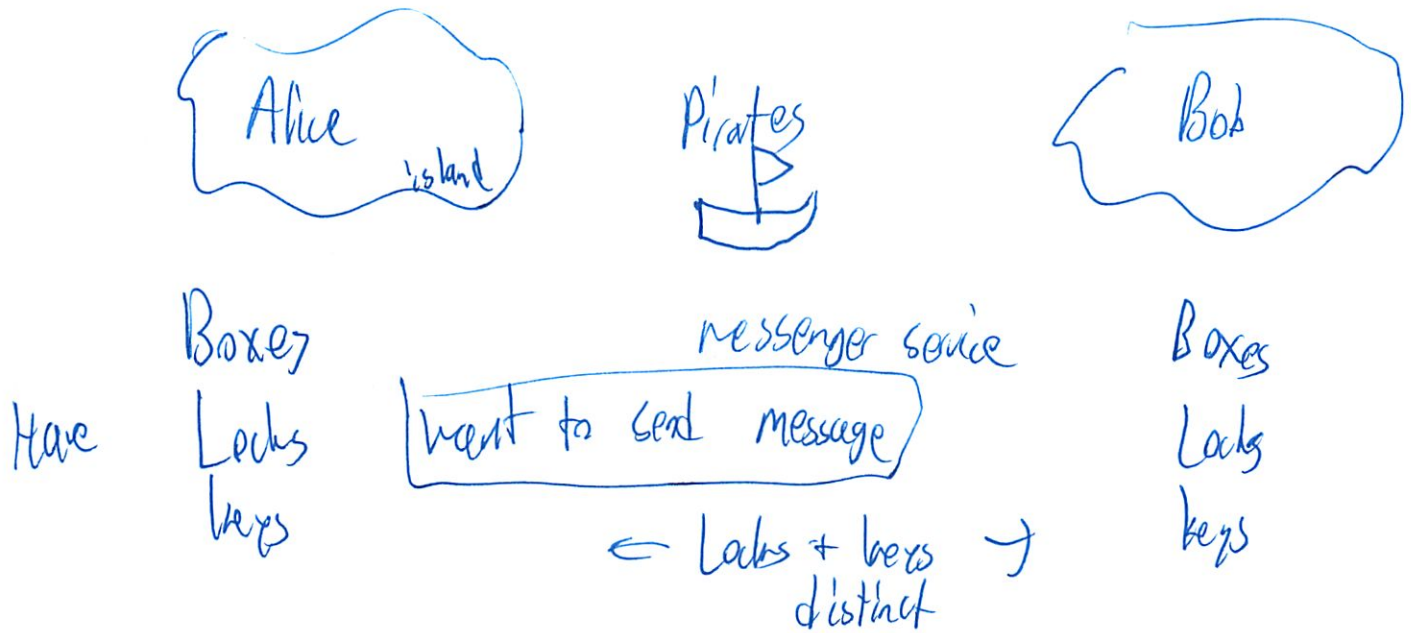
RC4, RC5 do this
are pretty simple

AES-256 is the state of the art

Key Exchange

How can you exchange keys in presence
of eavesdroppers

(4)



Can't send keys w/ messenger service

- pure messages
- unlocked boxes

Alice puts m into box, locks w/ k_A , keeps key

Alice sends box to Bob

Bob locks box w/ k_B , keeps key

Bob sends box to Alice

Alice unlocks w/ k_A

Alice sends box to Bob

Bob unlocks k_B , reads m !

5

Note we need commutivity of locks

Can't do w/ symmetric encryption

lock A

lock B

unlock A

unlock B

Diffe-Hellman key exchange

follows outline of riddle

$G = \mathbb{F}_p^*$ finite field (mod p , a prime)

* means only looking at invertible elements

$\{1, 2, \dots, p-1\}$

\uparrow No 0!

~~Prime~~ Init

$2 \leq g \leq p-2$

g, p public

Alice selects a $\begin{matrix} \perp \\ \lceil \end{matrix} 1 \leq a, b \leq p-2$

Alice computes $g^a \bmod p$ Sends to Bob
Bob computes $g^b \bmod p$ Sends to Alice

Alice computes $(g^b)^a \bmod p = k$

Bob computes $(g^a)^b \bmod p = k$

They both have k

Note all computation is mod - since fixed field
Relies on $(g^b)^a = (g^a)^b$

How could this scheme break?

Adversary sees $g^a \bmod p$
 \uparrow \uparrow
 knows since public

logs are continuous usually
but discrete here since wraps around

⑦
Shann to be NP, but not NPC

Very hard to do in practice

pretty much a 1-way function

The discrete log problem

↳ Diffie-Hellman requires

- Also man in the middle attack
~~Diffie-Hellman~~ ↳ Not a Diffie-Hellman problem

- Diffie-Hellman Problem If attacker could
Given $g^a \bmod p$, $g^b \bmod p$

Generate $g^{ab} \bmod p$

More specific than discrete log problem

- eavesdropper = passive
- man in the middle = active

⑧
man-in-the-middle

Active adversary - intercepts communication
tries to replace communications

Alice \leftrightarrow Mal \leftrightarrow Bob



they think they are
communicating directly

Mal decrypts messages and then reencrypts

Must somehow certify who someone is
using their public keys

More details in other classes

9

Public key encryption

Message + public key = Cipher text

Cipher text + private key = message

Two keys P_k , S_k
public secret-private

Are linked in some mathematical way
knowing P_k should tell you nothing about S_k

RSA

1978

Alice picks two large secret primes p, q

$$N = p * q$$

Chooses an encryption exponent e which

satisfies $\gcd(e, (p-1)(q-1)) = 1$ e rel. prime

(10)
Alice publishes public key (N, e)

\hookrightarrow encryption exponent

People can send any message to her w/
her public key, only she can decrypt

Note this whole thing relies on factoring

Decryption exponent d found by extended euclidean algorithm

$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$

$$\phi = (p-1)(q-1)$$

Alice's private key = d

Can throw away p, q - not needed anymore
but must remain secret

How do you know Alice is Alice?

Person must certify public key w/ trusted 3rd party
ie Verisign

②

Uses signing algorithm

Anyone can verify signature

$$C = M^e \pmod{N} \text{ encryption}$$

$$M = C^d \pmod{N} \text{ decryption}$$

If had p, q can speed up encryption

So must keep p, q instead of throwing away

$$(M^e \pmod{p}) (M^e \pmod{q}) = (M^e \pmod{N})$$

faster

Why it works:

Since $ed \equiv 1 \pmod{\phi} \Rightarrow$ an integer k

$$\text{s.t. } ed = 1 + k\phi$$

Two cases: $\gcd(m, p)$

$\begin{cases} \text{prime} \\ \text{and prime} \end{cases}$

$\begin{cases} \text{can be } 1 \text{ or } p \end{cases}$

$\begin{cases} m \text{ not divisible} \\ \text{by prime} \end{cases}$

$\begin{cases} m \text{ is a multiple of } p \end{cases}$

(12)

$\gcd(m, p) \rightarrow p$ $m \bmod p \equiv 0$ so everything 0
 $\gcd(m, p) \rightarrow 1$ m, p are rel. prime

By Fermat's Little Theorem

$$m^{p-1} \equiv 1 \pmod{p}$$

whole thing works out quite easily

$$(m^{p-1})^{k(q-1)} \equiv 1 \pmod{p}$$

multiply both sides by m

$$m \left[(m^{p-1})^{k(q-1)} \right] = [1 \pmod{p}] \cdot m$$

$$m^{1+k(p-1)(q-1)} = m \pmod{p}$$

$$m^{ed} = m \pmod{p}$$

Did for 1 prime p

Also do for q

$$m^{ed} = m \pmod{q}$$

(13)

$$N = p \cdot q$$

$$m^{ed} = m \pmod{N}$$

Assumptions

Factoring assumption

knowing P_k tells ya nothing about S_k

1. Given N hard to factor into p, q

2. RSA problem Given e
Adversary knows

$$\text{s.t., } \gcd(e, \underbrace{(p-1)(q-1)}_{\phi}) = 1$$

Adversary knows ϕ

Adversary doesn't know ϕ

Can they try to find N , ~~given~~ such that

$$m^e = c \pmod{N}$$

(14)

RSA has been secure and ^{stayed} ~~stand~~ secure

Must select p, q

- very subtle
- > 2048 bits
- various tests to check they are good

All of the stuff that can go wrong

Why do NP-complete problems make up poor assumptions

Is N composite?

$\in NP$ unknown if NPC ?

Knapsack

Why don't we use NP-hard problems since we know they are hard?

Given a pile of n items each w/ diff weights w_i
is it possible to put items in knapsack s.t.
we get a specific weight s ?

(15)

$$S = b_1 \cdot w_1 + b_2 \cdot w_2 + \dots + b_n w_n$$

$$b_i = \text{bits}$$

We showed these were NP-complete problems

Can build a crypto alg on this!

Hellman - Merkle

Prot: cool + efficient

General KP is NPC

But super-increasing knapsacks which are solvable
in linear time

$$w_j \geq \sum_{i=1}^{j-1} w_i$$

$$\{2, 3, 6, 13, 27, 52\}$$

? Prof! You should know this alg

Use the difficulty of general knapsack as adversary to solve
but ease of super-increasing knapsack to encrypt

6

Merkle-Hellman

Private key \rightarrow S. Inec. KP
 \uparrow
Private transform
Public key
 \checkmark

'hard' 'general' KP

Will use that (i.e. what) to encrypt/decrypt

Transform two private integers N, M s.t. rel prime
 $\gcd(N, M) = 1$

Multiply all values in set by N and then mod N ,

$N = 31$ private key $\{2, 3, 6, 13, 27, 53\}$

$M = 105$ public key $\{62, 93, 81, 88, 102, 37\}$

(17)

Short preview of what happens
longer example in notes

Message: 011000

Ciphertext: $93 + 81 = 174$

I need public key

Recipient knows $N = 31, N = 105$ $\{2, 3, 6, 13, 27, 52\}$

Multiplies each ciphertext block by $N^{-1} \pmod{m}$

$$N^{-1} = 61 \pmod{105}$$

Must do computation to solve s.t. in linear time

$$174 \cdot 61 = 4$$

ST
↑ Solve ~~M~~ Unpack in linear time
3+6

$$\text{So } B_i \text{ (bits)} = \boxed{011000}$$

That gets back message

But hopelessly broken
just means worst case exponential

(17)

Many cases of knapsack are efficient

Less low density
(small range)

So efficient is avg case

But transform produces only low density

So you dead

So in general NPC problems have
polynomial avg cases

CRYPTOGRAPHY AND COMPLEXITY: PART II of II

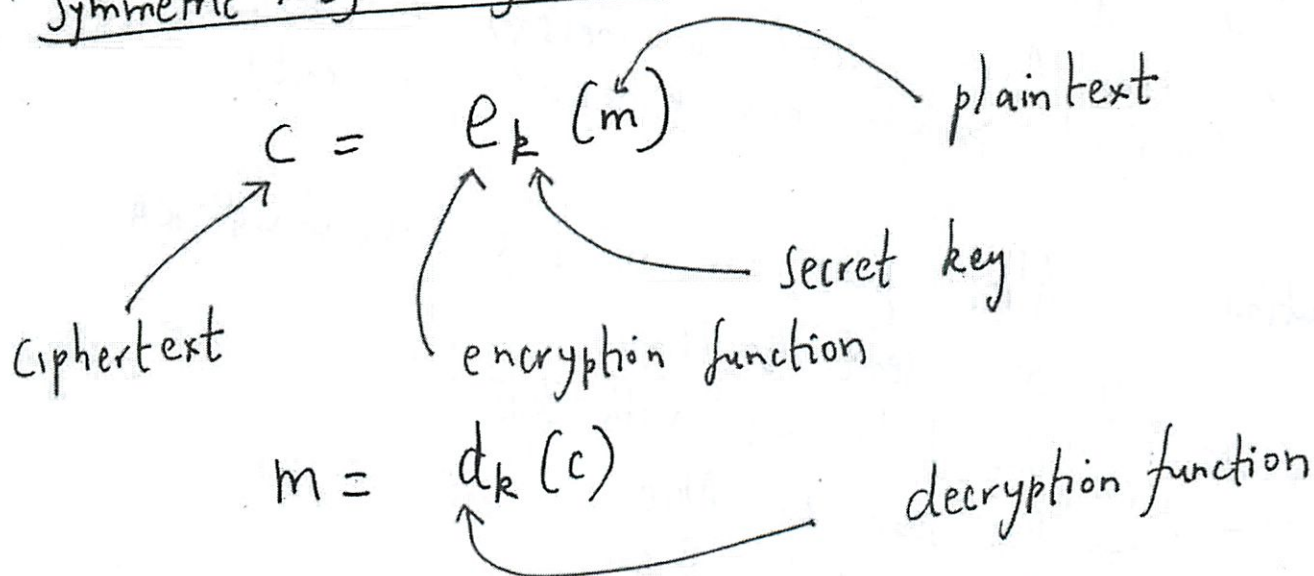
L21
6.046

①

11/29

- Symmetric Key Encryption
- Key Exchange
- Asymmetric key encryption
- RSA
- NP-complete problems & cryptography
 - graph coloring
 - knapsack.

Symmetric Key Encryption

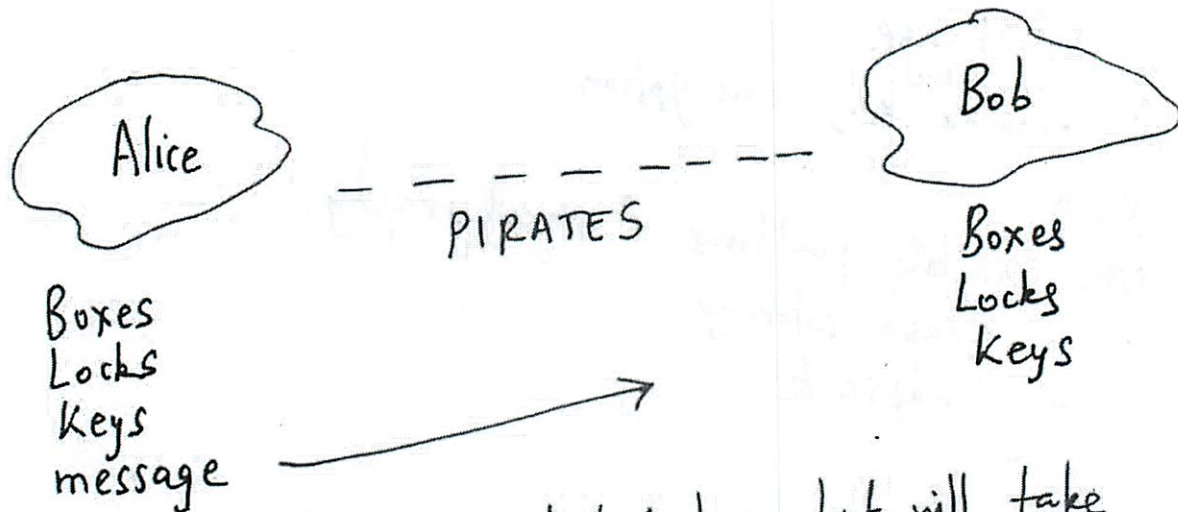


E, D permute & reverse-permute
reversible operations \oplus $+/-$, shift left/right
Symmetric algos: AES, RC5, DES

(2)

Key Management Question

How does secret key k get exchanged/shared?



Pirates won't touch locked box, but will take away keys, messages in unlocked box(es)

How does Alice send a message to Bob?
(without pirates knowing what was sent)

Solution:

- Alice puts m in box, locks it with K_A
- Box sent to Bob
- Bob locks box with K_B
- Box sent to Alice
- Alice unlocks K_A
- Box sent to Bob
- Bob unlocks K_B , reads m !

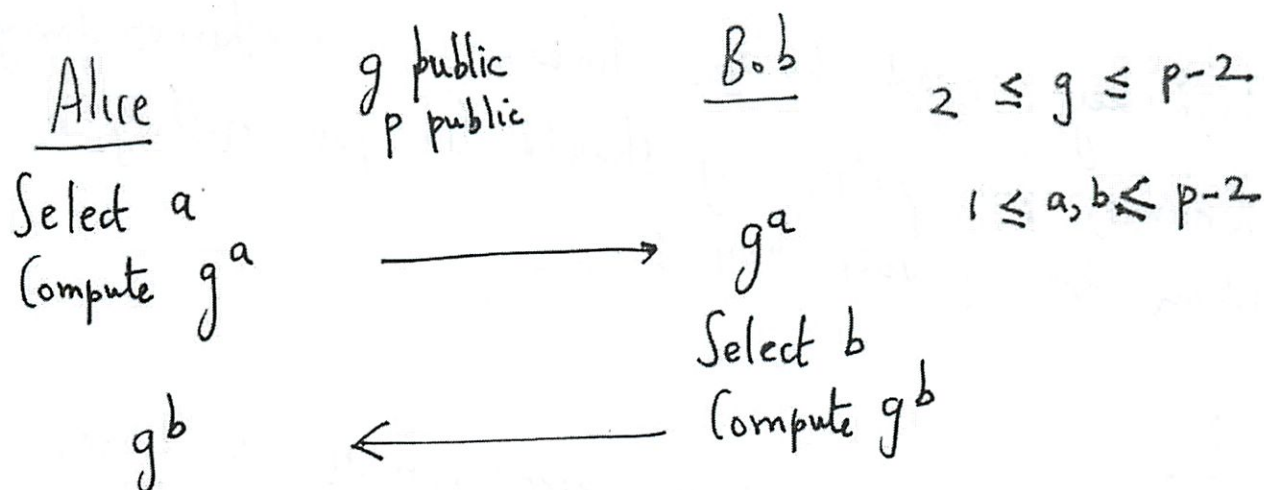
Commutative locks! Lock K_A , Lock K_B ,
remove K_A , remove K_B

(3)

Diffie-Hellman Key Exchange

$$G = F_p^*$$

finite field (mod p , a prime)
 * means invertible elements only
 $\{1, 2, \dots, p-1\}$



Assumes Discrete Log Problem is hard. Given g^a , compute a
 Diffie Hellman Problem is hard. Given g^a, g^b compute g^{ab}

Attack? Man-in-the-middle

Alice doesn't know she is communicating with Bob.
 Alice agrees to a key with Eve (thinks she is Bob)
 Bob agrees to a key with Eve (thinks she is Alice)
 Eve can see all communications

Public Key Encryption

Message + public key = Ciphertext

Ciphertext + private key = Message

Two keys need to be linked in a mathematical way
Knowing the public key should tell you nothing
about the private key.

RSA

Alice picks two large secret primes p & q .

Alice computes $N = p \cdot q$

Chooses an encryption exponent e which satisfies

$$\gcd(e, (p-1)(q-1)) = 1$$

$$e = 3, 17, 65537$$

Alice public key = (N, e)

Decryption exponent obtained using Extended Euclidean Algorithm
by Alice

$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$

Alice private key = (d, p, q)

not absolutely necessary,
only for efficiency

(5)

ENCRYPTION & DECRYPTION WITH RSA

$$c = m^e \pmod{N} \quad \text{encryption}$$

$$m = c^d \pmod{N} \quad \text{decryption}$$

Why it works

$$\phi = (p-1)(q-1)$$

Since $ed \equiv 1 \pmod{\phi}$ there exists
an integer k such that $ed = 1 + k\phi$

Two cases:

1) $\gcd(m, p) = 1$ By Fermat's theorem

$$m^{p-1} \equiv 1 \pmod{p}$$

$$(m^{p-1})^{k(q-1)} \cdot m \equiv m \pmod{p}$$

$$m^{1+k(p-1)(q-1)} = m^{ed} \equiv m \pmod{p}$$

2) $\gcd(m, p) = p$ $m \bmod p = 0$
trivial case $m^{ed} \equiv m$

So in both cases $m^{ed} \equiv m \pmod{p}$
III rly $m^{ed} \equiv m \pmod{q}$

Since p & q are distinct primes

$$m^{ed} \equiv m \pmod{N}$$

$$\text{So } c^d = (m^e)^d \equiv m \pmod{N}$$

HARDNESS OF RSA

1) Given N , hard to factor into p, q
Factoring

2) Given e such that
 $\gcd(e, (p-1)(q-1)) = 1$

RSA Problem

and c , find m such that
 $m^e = c \pmod{N}$

NP-Completeness

Is N composite? $\in NP$

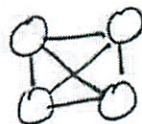
unknown if
NP-complete

Is a graph k -colorable?

NP-complete

Assign k colors to each vertex
 such that no two vertices connected
 by an edge share the same color

not 3-colorable



Given a pile of n items, each with
 different weights w_i , is it possible
 to put items in a knapsack such
 that we get a specific weight S ?
 $S = b_1 w_1 + b_2 w_2 + \dots + b_n w_n$?

NP-complete

NP-completeness & Cryptography

NP-completeness: about worst-case complexity

Cryptography: want a problem instance, with suitably chosen parameters that is hard on average.

Most Knapsack cryptosystems have failed.

Determining if a graph is 3-colorable is NP-complete

But very easy on average, because average graph, beyond a certain size, is not 3-colorable!

Consider standard backtracking search to determine 3-colorability.

Order vertices v_1, \dots, v_t . Colors = $\{1, 2, 3\}$

Traverse graph in order of vertices

On visiting a vertex, choose smallest possible color that "works".

If you get stuck, backtrack to previous choice, and try next choice

Run out of colors for 1st vertex \rightarrow NOT

Successfully color last vertex \rightarrow YES.

Random graph of t vertices, average number of vertices traveled < 197 , REGARDLESS of t !

KNAPSACK CRYPTOGRAPHY

(8)

General knapsack problem: NP-complete

Super-increasing knapsacks: linear time solvable

$$w_j \geq \sum_{i=1}^{j-1} w_i \quad \{2, 3, 6, 13, 27, 52\}$$

Merkle Hellman Cryptosystem:

Private key \rightarrow super increasing knapsack problem

\downarrow PRIVATE TRANSFORM

Public key \leftarrow "hard" general knapsack problem

Transform: two private integers N, M s.t. $\gcd(N, M) = 1$

Multiply all values in the sequence by N , and then mod M .

$N = 31, M = 105$ private key = $\{2, 3, 6, 13, 27, 52\}$
public key = $\{62, 93, 81, 88, 102, 37\}$.

(9)

MERKLE-HELLMAN EXAMPLE

Message = 011000 110101 101110

Ciphertext : 011000 $93 + 81 = 174$
 110101 $62 + 93 + 88 + 37 = 280$
 101110 $62 + 81 + 88 + 102 = 333$

= 174, 280, 333

Recipient knows $N = 31, M = 105$ $\{2, 3, 6, 13, 27, 52\}$
 Multiplies each ciphertext block by $N^{-1} \pmod{M}$

$$N^{-1} = 61 \pmod{105}$$

$$\begin{aligned} 174 \cdot 61 &= 9 = 3 + 6 = 011000 \\ 280 \cdot 61 &= 70 = 2 + 3 + 13 + 52 = 110101 \\ 333 \cdot 61 &= 48 = 2 + 6 + 13 + 27 = 101110 \end{aligned}$$

BEAUTIFUL BUT BROKEN

Lattice based techniques break this scheme.

Density of knapsack $d = \frac{n}{\max \{ \log_2 w_i : 1 \leq i \leq n \}}$

Lattice basis reduction can solve knapsacks of low density. Unfortunately, M-H scheme always produces knapsacks of low density!

↑ on average, easy to solve!

(I don't think I've been here in a month!)

- Travel - missed
- Quiz - canceled
- Travel - missed
- Thanksgiving - canceled
- Today - here
- Next week is last recitation - will miss

Today: Crypto

- Merkle tree
- Cha - Rivest Cryptosystem

Two: Hash function

Can use to verify function, data block, element = x

$h(x)$ = collision resistant hash function

Use to verify constance of x

⑦

Merkle tree

hash tree

That can be used to verify n elements at once

↳ now we have n elements

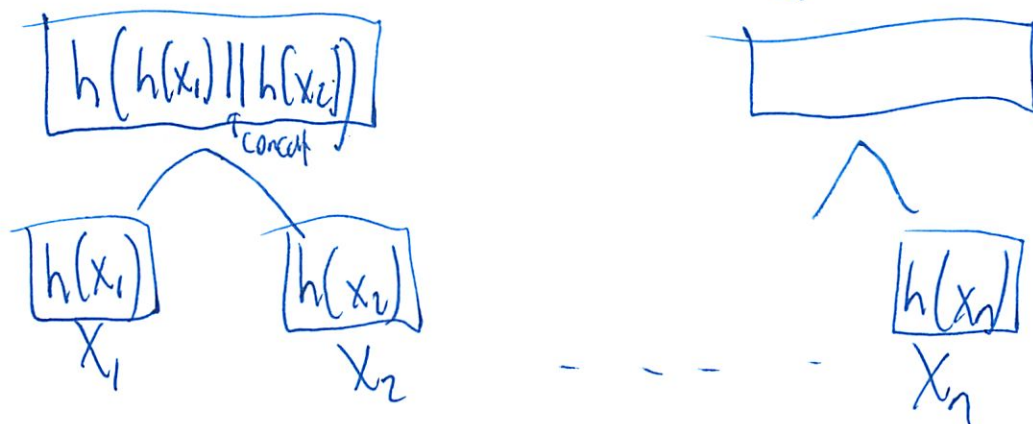
x_1, \dots, x_n
(before just 1 node)

- binary tree

- leaves are hashes of elements ($h(x_i)$)

- inner nodes are hashes of their children

$h(\text{children concatenated})$



(3)

Goal is that root is the same w/ what
you had before

Can't tell which leaf is wrong - only that a
leaf is bad

To verify a leaf, hash the leaf and use
value of its ancestors ~~the~~ and their signalings

(Other students not seeing it \rightarrow seems pretty intuitive to me)

Root used to verify everything

Chor-Riest

Saw Mohel-Helman crypto on Thur

NP-complete

but easy in many cases

This is another knapsack system from 1998

④

Only one that uses algebra to decrypt $\#s$.

~~Not secure, but hasn't been~~

Hasn't been broken, but not considered secure

3 parts

1. Key generation

2. Encryption

3. Decryption

1. Select prime p and positive integer h

s.t. $h \leq p$ and $p^h - 1$ has only small factors

Suppose we choose $p=7$ and $h=4$

$$p^h - 1 = 2400 = 2^5 \cdot 3 \cdot 5^2$$

τ prime factors
small

5

2. Select a ^{monic} irreducible random ~~poly monotonically~~ polynomial $f(x)$ of degree h over \mathbb{Z}_p
↑ everything modulo p

ex. $f(x) = x^4 + 3x^3 + 5x^2 + 6x + 2$

monic = coefficient of highest term is 1

irreducible = can't write as product of smaller #

Modulo
 $3x^3 + 6x^3 = 2x^3$ since modulo $7 = p$

why Consider a finite field F_q which consists of
 p polynomials in $\mathbb{Z}_p[x]$ of degree $< h$

So size of field = q

$$q = p^h$$

↑ h terms

must choose single coeff for each
from p choices

(6)

We can do multiplication of polynomials mod $f(x)$

Same ideas as w/ #s

$$(x^3 + 1)(x^5 + 1) \equiv \text{ (mod } f(x) \text{)}$$

We chose $p^h - 1$ to make discrete log problem easy
choosing p and h as in step 1 well feasible

3. Select a random primitive element $g(x)$ of F_q
↳ generator of the multiplicative group

Can take discrete log base $g(x)$

$$\forall f(x) \in F_q, \exists i \quad g(x)^i = f(x) \pmod{f(x)}$$

(what it means to be a primitive el)

$f(x)$ = any polynomial

⑦ All multiplication are mod $f(x)$

Example $g(x) = 3x^3 + 3x^2 + 6$

4) $\forall i \in \mathbb{Z}_p$

$$a_i = \log_{g(x)} (x+i)$$

↳ discrete log is feasible

$$a_0 = \log_{g(x)} X = 1028$$

↑ if raise this to that power mod $f(x)$
then you get the x

$$g(x)^{1028} = x \pmod{f(x)}$$

↳ polynomial multiplication
using mod

5) Permutation π of $\{0, \dots, p-1\}$

$$\text{ex } \{0, 1, \dots, 6\} \rightarrow \{6, 4, 0, 2, 1, 5, 3\}$$

Just pick a permutation

(Class pretty hostile today)

⑦

$$a) \forall i \in \{0, \dots, p-1\}$$

$$\text{Compute } c_i = (a_{\pi(i)} + d) \pmod{p^h - 1}$$

d is random integer

$$\text{s.t. } 0 \leq d \leq p^h - 2$$

$$d = \text{1702}$$

$$PK = (c_0, \dots, c_{p-1}), p, h$$

$$SK = (f(x), g(x), \pi, d)$$

Encryption

$$M = (m_0, \dots, m_p)$$

$\hookrightarrow p$ bits

$$m_i \in \{0, 1\}$$

$$\sum m_i = h$$

p 0/1 bits

h will be 7

9

of bits conveyed is $\underbrace{\left\lfloor \log_2 \left(\frac{p}{h} \right) \right\rfloor}_{\text{amt of info conveyed}}$

M of this length \geq

Can convert to M vector and backward

Correspondence b/w any message and contained message

m is any message

Can convert to a larger M

Encryption

$$C = \sum_{i=0}^{p-1} m_i c_i \pmod{p^h - 1}$$

?
Cipher text
target sum

⑩ Decryption

Or Co are special

Instead of solving knapsack we do computation w/ polynomials

- 1) $r = (c - hd) \bmod (p^h - 1)$
- 2) $u(x) = g(x)^r \bmod f(x)$
- 3) $s(x) = u(x) + f(x)$

$S(x)$ is monic polynomial of degree h
over \mathbb{Z}_p

s.t. it will have following form

$$S(x) = \prod_{j=1}^h (x + x_j)$$

↳ find roots by enumerating over \mathbb{Z}_p

②

$$5) M = (m_0, \dots, m_{p-1})$$

$$TP^{-1}(x_j) \quad 1 \leq j \leq h$$

to find which indices = 1

(compute reverse of permutation w/ x_j)

in m $TP^{-1}(x_j)$ are the 1s and the rest is 0s

Now we calculate n

Read recitation notes for more

To solve w/o key must solve knapsack problem

w/ key can solve it w/o knapsack problem

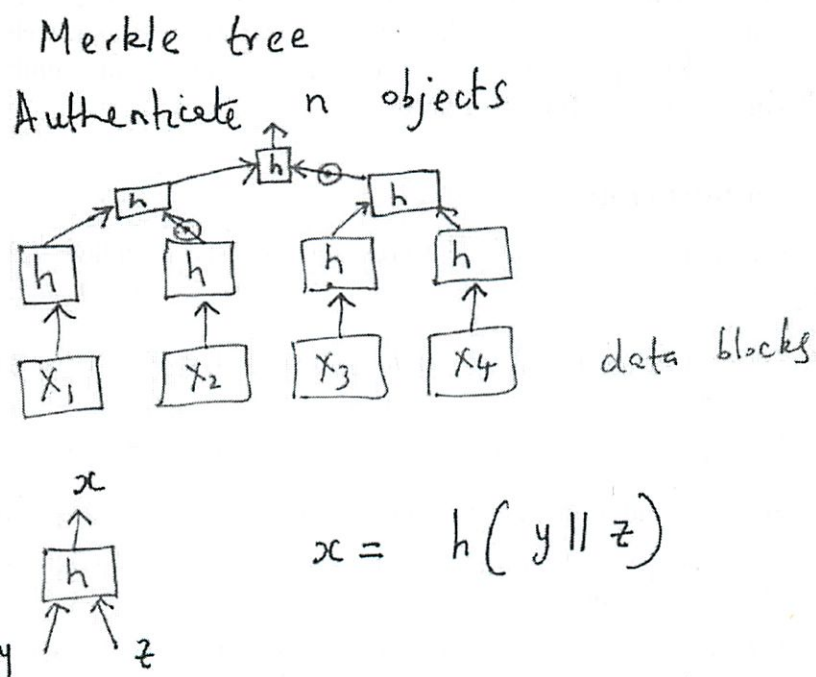
11/30

Recitation 10: Cryptography

1 Merkle Trees

In cryptography, Merkle trees or hash trees are a type of data structure that contains a tree summary information about a larger piece of data and used to verify its contents. Such trees are combination of hash lists and hash chaining. A Merkle tree can be used to verify (authenticate) n elements as follows. Construct a binary tree where

- the leaves of the tree are hashes of elements
- the inner nodes are the hashes of their children
- the root is the top hash that is used to verify the integrity of all elements



To verify a leaf, need to know the leaf and its ancestors and their siblings. Merkle trees are nice because a branch can be verified without having the whole tree. To guarantee the integrity, we need to use a hash function that is collision resistant. One can show that if an adversary manages to break the integrity, then a collision has been found.

In general, can use non-binary authentication tree, e.g. integrity of files in nested file directories.

2 Chor-Rivest Knapsack Cryptosystem

In class, we saw Merkle-Hellman's knapsack cryptosystem. There have been many attempts to utilize the NP-completeness of the knapsack problem to achieve a secure encryption, but most of the schemes were broken over time. The common feature among these schemes is that the decryption involves finding a subset of weights that sum to the target number.

The Chor-Rivest public key encryption scheme is also based on the knapsack problem. No feasible attack is known if the parameters are chosen well. It's also the only one that doesn't use some form of modular multiplication to disguise an easy subset sum problem.

Key generation

To generate a pair of public and secret keys, one should follow the following steps:

1. Select a prime p and integer $h \leq p$, such that $p^h - 1$ has only small factors.

Example: $p = 7$, $h = 4$, then $q = 2401$ and $p^h - 1 = 2400 = 2^5 \cdot 3 \cdot 5^2$

2. Select a random monic irreducible polynomial $f(x)$ of degree h over \mathbb{Z}_p .

Example: Select the irreducible polynomial $f(x) = x^4 + 3x^3 + 5x^2 + 6x + 2$ of degree 4 over \mathbb{Z}_7 .

It turns out that if we consider a finite field that consists of polynomials in $\mathbb{Z}_p[x]$ of degree less than h , then its size will be $q = p^h$. Denote

this finite field \mathbb{F}_q . Consider the multiplication of these polynomials performed modulo $f(x)$, then the discrete logarithm problem is feasible if p and h were chosen as described in step 1 [Pohlig-Hellman algorithm].

Example: The elements of the finite field F_{2401} are represented as polynomials in $\mathbb{Z}_7[x]$ of degree less than 4 with multiplication performed modulo $f(x)$.

3. Select a random primitive element (i.e. a generator of the multiplicative group) $g(x)$ of the field F_q .

Example: $g(x) = 3x^3 + 3x^2 + 6$. We can do discrete logarithm with base $g(x)$ because it's a primitive element.

4. For each element $i \in \mathbb{Z}_p$, find the discrete logarithm

$$a_i = \log_{g(x)}(x + i)$$

of the field element $(x + i)$ to the base $g(x)$.

Example:

$$\begin{aligned} a_0 &= \log_{g(x)}(x) &= 1028 \\ a_1 &= \log_{g(x)}(x + 1) &= 1935 \\ a_2 &= \log_{g(x)}(x + 2) &= 2054 \\ a_3 &= \log_{g(x)}(x + 3) &= 1008 \\ a_4 &= \log_{g(x)}(x + 4) &= 379 \\ a_5 &= \log_{g(x)}(x + 5) &= 1780 \\ a_6 &= \log_{g(x)}(x + 6) &= 223 \end{aligned}$$

5. Select a random permutation π on the set of integers $\{0, 1, \dots, p - 1\}$.

Example: Consider the permutation $(6, 4, 0, 2, 1, 5, 3)$, i.e. $\pi(0) = 6$, $\pi(1) = 4$, etc.

6. Select a random integer d , such that $0 \leq d \leq p^h - 2$.

Example: $d = 1702$.

7. For each $i \in \{0, 1, \dots, p - 1\}$, compute

$$c_i = (a_{\pi(i)} + d) \mod (q - 1)$$

Example: $q - 1 = 2400$ and $d = 1702$, thus

$$c_0 = (a_6 + d) \mod 2400 = 1925$$

$$c_1 = (a_4 + d) \mod 2400 = 2081$$

$$c_2 = (a_0 + d) \mod 2400 = 330$$

$$c_3 = (a_2 + d) \mod 2400 = 1356$$

$$c_4 = (a_1 + d) \mod 2400 = 1237$$

$$c_5 = (a_5 + d) \mod 2400 = 1082$$

$$c_6 = (a_6 + d) \mod 2400 = 310$$

8. The public key and secret keys are

$$PK = ((c_0, c_1, \dots, c_{p-1}), p, h)$$

$$SK = (f(x), g(x), \pi, d)$$

Encryption

Given the public key $PK = ((c_0, c_1, \dots, c_{p-1}), p, h)$ and a binary message m of length $\lfloor \lg \binom{p}{h} \rfloor$, do the following:

1. Consider m as an integer in the binary form. Transform m into a binary vector $M = (M_0, M_1, \dots, M_{p-1})$ of length p having exactly h 1's as follows:

(i) Set $l \leftarrow h$

(ii) For i from 1 to p do the following:

If $m \geq \binom{p-i}{l}$, then set $M_{i-1} \leftarrow 1$, $m \leftarrow m - \binom{p-i}{l}$, $l \leftarrow l - 1$.

Otherwise, set $M_{i-1} \leftarrow 0$.

Example: $\lfloor \lg \binom{p}{h} \rfloor = \lfloor \lg \binom{7}{4} \rfloor = 5$. Consider a binary message 10110, then $m = 22$ and the loop values are:

$$i = 1, l = 4, m = 22, M_0 = 1$$

$$i = 2, l = 3, m = 7, M_1 = 0$$

$$i = 3, l = 3, m = 7, M_2 = 1$$

$$i = 4, l = 2, m = 3, M_3 = 1$$

$$i = 5, l = 1, m = 0, M_4 = 0$$

$$i = 6, l = 1, m = 0, M_5 = 0$$

$$i = 7, l = 1, m = 0, M_6 = 1$$

Note that $\binom{n}{0} = 1$ for $n \geq 0$, and $\binom{0}{l} = 0$ for $l \geq 1$. Thus, we get $M = (1, 0, 1, 1, 0, 0, 1)$.

2. Compute ciphertext c as

$$c = \sum_{i=0}^{p-1} M_i c_i \mod (p^h - 1)$$

Example:

$$c = (c_0 + c_2 + c_3 + c_6) \mod 2400 = 1521$$

Decryption

To recover the message m from c , do the following:

1. Compute

$$r = (c - hd) \mod (p^h - 1)$$

Example:

$$r = (1521 - 4 \cdot 1702) \mod 2400 = 1913$$

2. Compute

$$u(x) = g(x)^r \mod f(x)$$

Example:

$$u(x) = (3x^3 + 3x^2 + 6)^{1913} \mod (x^4 + 3x^3 + 5x^2 + 6x + 2) = x^3 + 3x^2 + 2x + 5$$

3. Compute $s(x) = u(x) + f(x)$, a monic polynomial of degree h over \mathbb{Z}_p

Example:

$$s(x) = u(x) + f(x) = x^4 + 4x^3 + x^2 + x$$

4. Factor $s(x)$ into linear factors over \mathbb{Z}_p :

$$s(x) = \prod_{j=1}^h (x + t_j)$$

where $t_j \in \mathbb{Z}_p$.

Example: Try all \mathbb{Z}_p to find the roots of $s(x)$.

$$s(x) = x(x+2)(x+3)(x+6)$$

$$t_1 = 0, t_2 = 2, t_3 = 3, t_4 = 6$$

5. Computer a binary vector $M = (M_0, M_1, \dots, M_{p-1})$ as follows. The components of M that are 1 have indices $\pi^{-1}(t_j)$, $1 \leq j \leq h$. The remaining components are 0.

Example:

$$\pi^{-1}(0) = 2$$

$$\pi^{-1}(2) = 3$$

$$\pi^{-1}(3) = 6$$

$$\pi^{-1}(6) = 0$$

Thus, we get that

$$M = (1, 0, 1, 1, 0, 0, 1)$$

6. The message m from M is recovered as follows:

(i) Set $m \leftarrow 0$, $l \leftarrow h$

(ii) For i from 1 to p do the following:

If $M_{i-1} = 1$, then set $m \leftarrow m + \binom{p-i}{l}$ and $l \leftarrow l - 1$.

The final value of m is the integer message m . Example: The values in the loop:

$$i = 1, m = 0, l = 4, M_0 = 1$$

$$i = 2, m = 15, l = 3, M_1 = 0$$

$$i = 3, m = 15, l = 3, M_2 = 1$$

$$i = 4, m = 19, l = 2, M_3 = 1$$

$$\begin{aligned}
i = 5, m = 22, l = 1, M_4 &= 0 \\
i = 6, m = 22, l = 1, M_5 &= 0 \\
i = 7, m = 22, l = 1, M_6 &= 1 \\
m = 22, l = 0
\end{aligned}$$

We get that $m = 22$ or 10110 in the binary form.

Notes

The correctness of Chor-Rivest encryption system can be proved using some algebra. In practice, the recommended size of the parameters are $p \approx 200$ and $h \approx 25$ (e.g. $p = 197$, $h = 24$).

The density of the knapsack set c_0, c_1, \dots, c_{p-1} is $p/\lg(\max(c_i))$, which is large enough to thwart the low-density attacks on the general subset sum problem.

The major drawbacks of the Chor-Rivest scheme is that the public key is fairly large, namely, about $(ph \cdot \lg p)$ bits. (For the suggested parameters, this is about 36000 bits).

References

For more details about the Chor-Rivest encryption scheme, check out the *Handbook of Applied Cryptography* [section 8.6.2, p.302] at <http://cacr.uwaterloo.ca/hac/about/chap8.pdf>

6.046
L22 Sublinear
Time Algorithms

12/4

- Models
 - Classical Approximation
 - diameter of point set
 - Property Testing
 - "reachability" of list
 - connectedness of a graph
-

Big Data

impossible to access all of it
(missed)

Could we know if you are connected to ~~any~~ everyone?

Could ask people who they know

But that doesn't scale

Plus people born/died

So can we do it?

②

Linear was the gold std in this class

But can we do something w/o viewing most of the data?

We couldn't exactly how many inde are left handed?
(pretend its binary)

Can't are all 'in connected

We could approx how many inde left handed
or what %

~~can't~~ SAMPLING!
6.042

call 'is there a large group of connected inde

↳ Yes/no

- not sampling, some new technique we'll learn
- 99% of nodes are connected

(3)

Will push results further + use more approximation approach

"Classical" approximation

- output is ~~it~~ that is close to value of optimal sol for given input
- not enough time to construct sol

property testing

Output is correct ans for given input or
at least other inputs that are close

Classical Approximation

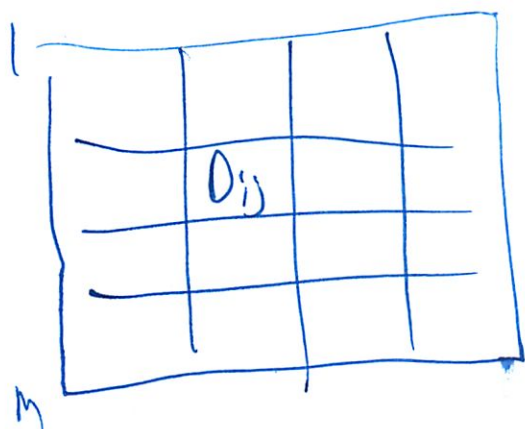
- deterministic time alg
- approximate ans
- sub-linear time

diameter of point set

m points \mathcal{P}
described by distance matrix D

④

D



input size = m^2

Satisfies triangle inequality

$$\hookrightarrow D_{ij} \leq D_{ik} + D_{kj}$$

Symmetric

$$D_{ij} = D_{ji} \quad \forall i, j$$

D_{ij} is distance from i to j

Diameter is distance b/w max points



Could scan whole table and look for max D_{ij}
 $\hookrightarrow m^2$

But because triangle inequality + symmetry
we can do better

So can at a 2-approximation

5

Goal: 2-approximation

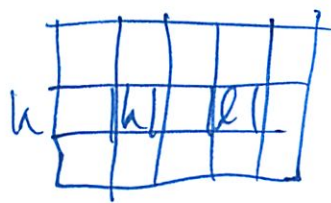
$$\text{output } k, l \text{ s.t. } D_{kl} \geq \frac{\max_{i,j} D_{ij}}{2}$$

1. pick k arbitrarily

2. look at row that k is in

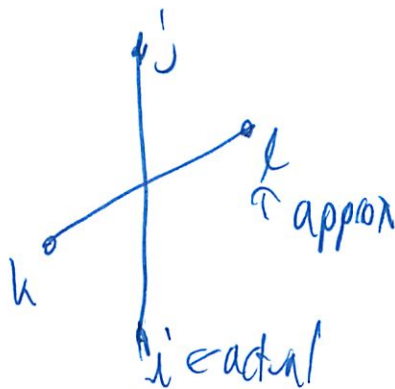
3. Pick the max entry $D_{kl} \rightarrow l$

4. Output D_{kl}



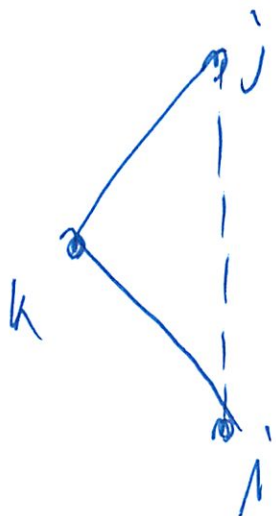
So why does that work?

Must show $D_{ij} \leq D_{ik} + D_{kl}$ by triangle inequality



6

So



known $D_{ij} \leq D_{ik} + D_{kj}$

$$\leq D_{kl} + D_{kl} \text{ symmetry}$$

Since we picked it

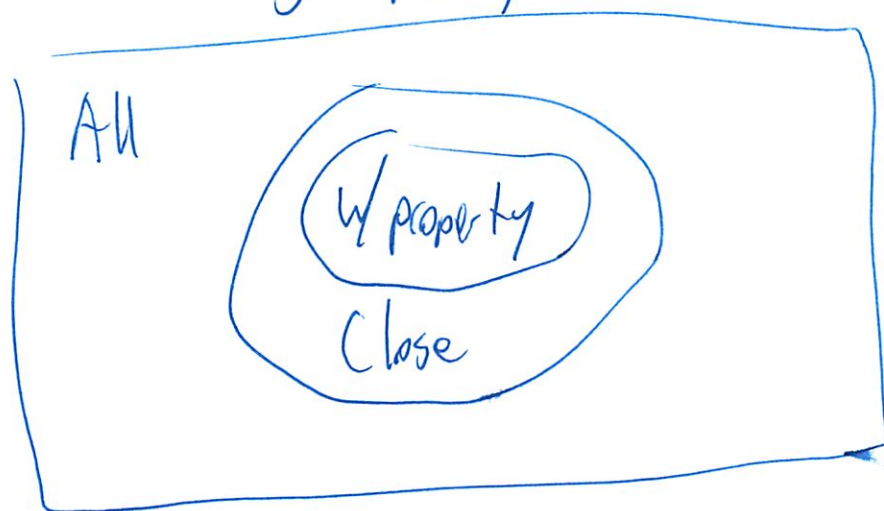
$$\leq 2D_{kl}$$

$$O(m) = O(n/2)$$

Ah Since
 $D_{kl} \geq D_{ik}$
Since we scanned row k is in
I'm pretty sure

Property Testing

Quickly distinguish inputs that have a specific property from those that are far from having property



Normally 'not w/ property' = valid ans

Here w/ property or close = valid ans

Separate those w/ property from those w/ weaker ans
So fast sanity check!
In the ball park!

⑧

- Use to check ans
- Rule at every bad answer

Or do a quick ~~a~~ list check
Then if it is, ~~only~~ then do the slower
real function

Properties

Fns
Graphs
Strings
Matrices
Codewords

Must specify representation of object
and allowable queries

Sortness of seq

Is the list sorted?
Must look at every y_i
↳ n items $O(n)$

9

~~The~~ Can't skip any - may be ∞

Can we quickly test that we are close to sorted?

Quick Our goal: running time complexity $O(n \lg n)$

What do we mean by close to sorted?

A list is ϵ -close to sorted if we can delete at most ϵn values to make it sorted.

Otherwise ϵ -far.

(ϵ is given as an input, eg. $\epsilon = \frac{1}{10}$)

~~Q: Na~~

$\hookrightarrow 90\%$ sorted

Can set/choose ϵ

$$\frac{n}{100} = 1\%$$

(10)

Requirements

Input k, y_1, \dots, y_n

Output If $y_1 \leq y_2 \leq y_3 \dots \leq y_n$

Output pass

If y_1, \dots, y_n is not epsilon close to sorted
then output fail.

↑ 1 sided algorithm must always ~~pro~~ be correct on pass
but can be wrong on fail
but correct w/ prob $\geq \frac{3}{4}$
Arbitrary constant

Example on slide

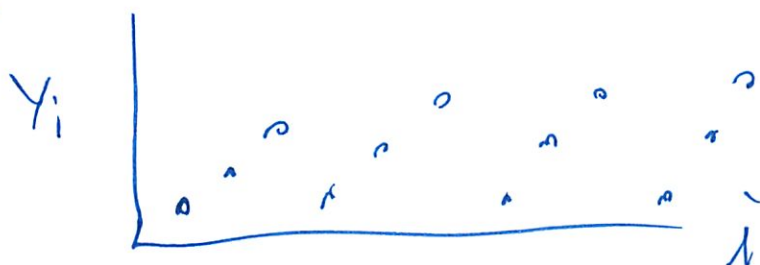
$\frac{1}{4}$ close to sorted

how much of list to delete

far - is must delete at least half el to make sorted

(12)

But



How many elements do we need to delete to
make monotone? $3/4$ ← very far from ~~per~~ correct

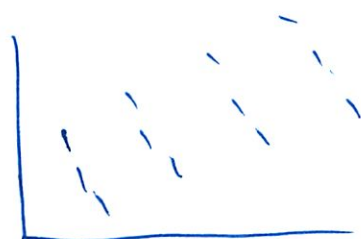
Since most

But passes test unless we pick the
breakpoints!
at least one of

Second attempt

Pick $i < j$ and test $y_i \leq y_j$

But



if compare elements within group = problem
if " " b/w group = passes

13

So we must delete at most $\frac{3}{4}$ elements
↳ pretty far

But prob of passing are good!

First lets add a simplification

↳ w/o generality

Assume list is distinct (ie. $x_i \neq x_j$)

but doesn't make problem easier

Since we can always append item's position to item

$x_1, x_2, \dots, x_n \rightarrow (x_1, 1), (x_2, 2), \dots$

$1, 1, 2, 6, 6 \rightarrow (1, 1), (1, 2), (2, 3), (6, 4), (6, 5)$

Since sort on higher digit that doesn't change
does make sure order remains when
breaking ties

(14)

A test

Do $O(\frac{1}{\epsilon})$ times

1. Pick a random $i \in [1 \dots n]$
2. Look at value of y_i 's value
3. Using that value do a binary search
On the list $y_1 \dots y_n$ for this
Value

While doing binary search must make sure
no inconsistency

Must also check we ended at location i .

4. If we don't end up at location i
or find inconsistency along search
path, output FAIL + halt

5. Output Pass

(15)

So running time

- binary search $O(\lg n)$

- $\frac{1}{\epsilon}$ times

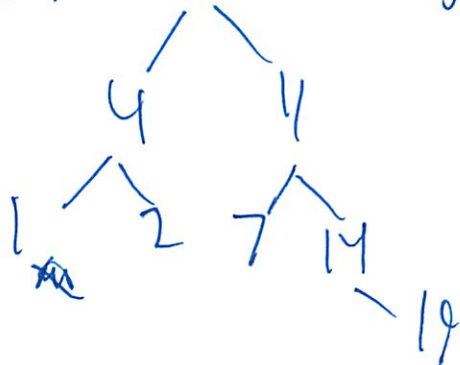
$$O\left(\frac{1}{\epsilon} \lg n\right)$$

So why does this work?

defn index i is "good" if binary search for y_i passes

list 1 4 2 5 7 11 14 19

b-search 5 \leftarrow go to middle



\leftarrow Since checking if binary search puts you at right spot

i 1 2 3 4 5 6 7 8

Y
Good

Look for 7 \rightarrow right

11 \rightarrow left

7 \rightarrow 5

where started, so good \odot

16)

If picked any value, we'll end up where started w/o inconsistency

~~1~~ 1 is good

2 ~~1~~ \rightarrow see value 2

< 5

< 4

$\neq 1$
value

(X) Not good

) more

3 \rightarrow Also not good (X)

(need to review more slowly)

If list sorted, all els are good

Pick $O(k)$ is and make sure they are all good

Does it work?

1. Always Pass if sorted works

2. If far, then Fail w/ high prob

(17)

Instead show an equivalent

$$\text{ie } p \rightarrow q$$

Equivalent

$$\neg q \rightarrow \neg p$$

So show if we output PASS w/ prob $\geq \frac{1}{4}$
Then we are ϵ close to sorted

If we ever find a bad i , we ~~pass~~ fail

If list likely to pass the test ^{w/ prob $\geq \frac{1}{4}$} , then at least
 $\geq (1-\epsilon)n$ i 's are good

Proof If $\geq \epsilon n$ bad i 's then $\frac{c}{\epsilon}$ loops

Choose o w/ prob prob

$$\text{prob} \geq 1 - \underbrace{(1-\epsilon)^{c/\epsilon}}_{e^{-c}}$$

If pick constant big enough, this will be
 $\geq \frac{3}{4}$

(18)

If ever pick a bad i , list will fail

Proof for $i < j$ both good need to show $y_i < y_j$

Let h = least common ancestor of i, j

Search for i went left of h and search
for j went right of $h \rightarrow$

$$y_i < y_h < y_j$$

(Pretty much lost)

Thus list is ϵ -close to monotone

delete $< \epsilon n$ bad elements

So is in monotone order

Usually the other case is harder

Usually contrapositive

" must understand structure of bad inputs

(19)

Bottom level is the neighbors

↳ like a neighbor test

Top levels are like our pick random test

Must still fill in the other $\log(n)$ levels

Testing Connectedness of a Graph

Given Graph G

n vertices

Max degree d

(missed, on slide)

Graph is ϵ -close to connected if can add $< \epsilon d n$ edges to make it connected

Today: ok to violate max d requirement

(20)

~~Set~~ G is

Input ϵ, G

Output G connected \rightarrow PASS

G ~~is~~ ϵ -far-connected \rightarrow FAIL
w/ prob $\geq \frac{3}{4}$

Idea: If G is far connected, lots of nodes must be in small components

So if G is far from connected

then we must have many connected components

So many components must be small

And there must be many nodes in small components

21

→ Do $O\left(\frac{1}{\epsilon d}\right)$ times

→ Pick random node s , and run BFS from s until

- or \mathcal{C}_s
- a) $\# \geq \frac{2}{\epsilon d}$ distinct nodes seen
 - b) $\#$ OR see that \mathcal{C}_s is component of size $< \frac{2}{\epsilon d}$ nodes, in which case output FAIL + halt

→ If reach this point → output PASS

Will always pass if close

Will fail if far usually

Running time

$$\frac{1}{\epsilon d} \cdot d \cdot \underbrace{\frac{2}{\epsilon d}}_{\substack{\text{run BFS} \\ \# \text{ steps}}} = O\left(\frac{1}{\epsilon^2 d}\right)$$

(22)

Must have at least ϵn components

(see slides...)
↳ Behavior

Each small component has at least 1 node
disjoint

Likely to hit nodes w/ high probability

These cause tester to fail

(see slide)

Lecture 22: Sublinear Time Algorithms

- Models
- Classical Approximation
 - diameter of a point set (see slides)
- Property Testing: approximation for decision problems
 - "sortedness" of a list
 - connectedness of a graph

Testing "sortedness" of a list

def. list of size n is ϵ -close to sorted if can delete $\leq \epsilon n$ elements to get a sorted list

Requirements of property testing algorithm:

Input ϵ , list $y_1 \dots y_n$

Output

- if $y_1 \leq y_2 \leq \dots \leq y_n$ output "PASS"
- if $y_1 \dots y_n$ not ϵ -close to sorted output "FAIL" with prob $\geq 3/4$

note: $p \Rightarrow q$

is equivalent to $\neg q \Rightarrow \neg p$

so this is the same as

"If likely to output PASS (with prob $> 3/4$) on this list, then list must be ϵ -close to sorted."

↑
once we can do this, we can repeat to get better probabilities

Comment we didn't specify what should happen

if y_i 's not sorted, but not far.

either output is reasonable, but this leeway allows major speedup!

- Two tests that don't work well:

- 1) Pick random i + test whether $y_i < y_{i+1}$
- 2) Pick random i, j + test whether $y_i < y_j$

Why not?

- Wlog, assume list distinct (i.e., ideally we have $x_1 < x_2 < x_3 \dots$)
(if not, append index to each element
 $x_1 \dots x_n \Rightarrow (x_1, 1), (x_2, 2) \dots (x_n, n)$
breaks ties without changing order)

- The test:

Do $O(1/\epsilon)$ times:

Pick random i
look at y_i 's value +
do binary search on list of y_j 's for
this value

If don't end up at location i
or find an inconsistency along
search path, output "FAIL" + halt

(If no problems found) Output "PASS"

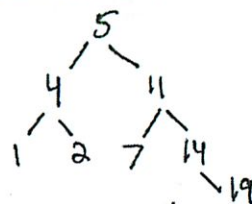
- Runtime $O(\frac{1}{\epsilon} \log n)$

Behavior

def ^{index} i "good" if binary search for y_i successful.

e.g. 1 4 2 5 7 11 14 19

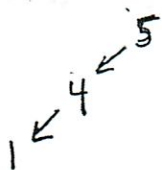
search tree



bin search for 7-19 won't find any inconsistency
 \Rightarrow 5-8 are good

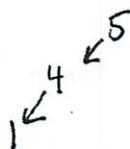
bin search for index 1: (value 1)

ok \Rightarrow ^{index} 1 is good



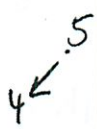
bin search for ^{index} 3: (value 2)

2 not found \Rightarrow index 3 is not good



bin search for index 2: (value 4)

4 found \Rightarrow 2 is good



bin search for index 4: (value 5)

5 found \Rightarrow index 4 good

Behavior (cont.)

- if list sorted,
all i 's good \Rightarrow list always passes
 \uparrow
 uses distinctness

- if list not ϵ -close,
to show: test fails with prob $\geq 3/4$

Equivalently will show:

If list passes test with prob $> 1/4$ then
 must be ϵ -close.

why?

If list passes test with prob $> 1/4$

then $\geq (1-\epsilon)n$ i 's are good

(if $> \epsilon n$ i 's are bad,

then in $\frac{c}{\epsilon}$ loops will

choose one with prob $\geq 1 - (1-\epsilon)^{c/\epsilon}$

$$\geq 1 - e^{-c}$$

$$\geq 3/4$$

\nexists then test fails
 which contradicts that test passes with prob $> 1/4$

Testing "connectedness" of a graph

Given: graph G

n vertices, m edges

max degree d

adjacency list representation



def

G is ϵ -close to connected

if adding $< \epsilon n$ edges transforms it to connected
(today it is ok if transformation violates max degree d requirement)

Property tester requirements:

Input ϵ, G

Output

- if G connected, output "PASS"
- if G not ϵ -close, output "FAIL" with prob $\geq 3/4$

Idea for tester

- If G far from connected,
- \Rightarrow must have many ($\geq \epsilon n$) connected components
- \Rightarrow many small connected components
- \Rightarrow many nodes in small connected components

Algorithm

Do $O(\frac{1}{\epsilon d})$ times:

Pick random node s , run BFS from s until

(a) $\geq \frac{2}{\epsilon d}$ distinct nodes seen \leftarrow good case, continue

or
(b) see that s is in component of size $< \frac{2}{\epsilon d}$ nodes

\uparrow bad case, output "FAIL" and halt

(If never entered (b)) Output "PASS"

Runtime

$O(\frac{1}{\epsilon d})$ loops

$O(\frac{1}{\epsilon d})$ steps of BFS $\times O(d)$ time per step

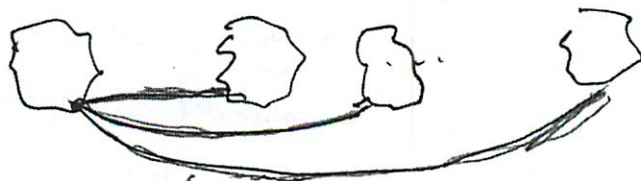
$\Rightarrow O(\frac{1}{\epsilon^2 d})$ time

Behavior

Lemma! If G is ϵ -far from connected
then has $\geq \epsilon d n$ connected components

PF

assume $< \epsilon d n$ components



\uparrow add $< \epsilon d n$ edges connects up graph
 \leftarrow not ϵ -far! $\Rightarrow \square$

Lemma 2 if $\geq \epsilon d n$ conn. components
then $\geq \frac{\epsilon d n}{2}$ components of size $< \frac{2}{\epsilon d}$

Pf $L \leftarrow \# \text{ conn comp}$
 $l' \leftarrow \# \text{ conn comp of size } < \frac{2}{\epsilon d}$
 $l \leftarrow \text{ " " " " " } \geq 2/\epsilon d$

$L = l' + l$
note $l \cdot \frac{2}{\epsilon d} \leq n$ (else too many nodes!)

$$\text{so } l \leq \frac{\epsilon d n}{2}$$

$$\text{so } l' = L - l \geq \epsilon d n - \frac{\epsilon d n}{2} = \frac{\epsilon d n}{2}$$

Observation if $\geq \frac{\epsilon d n}{2}$ components of size $< 2/\epsilon d$
 then $\geq \left(\frac{\epsilon d}{2}\right) \cdot n$ nodes in components of size $< 2/\epsilon d$
 (since each small component has ≥ 1 node)

Putting it together (Lemma 1, 2 + observation):

If G ϵ -far from connected,

$\geq \frac{\epsilon d}{2}$ fraction of nodes in small components
 these reach (b)
 + FAIL!

so $\text{Prob}[\text{fail}] \geq 1 - \left(1 - \frac{\epsilon d}{2}\right)^{c/\epsilon d} \geq 1 - e^{-c/2} \geq 3/4$ for c big enough.

Lecture 22: Sub-linear Time Algorithms

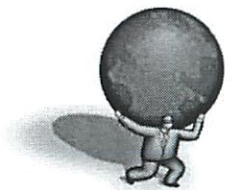
How can we understand?



Today's goal

- Motivation and models
- Classical approximation problems
 - Diameter of a point set
- Property testing: approximation for decision problems
 - “Sortedness” of a list
 - Connectedness of a graph

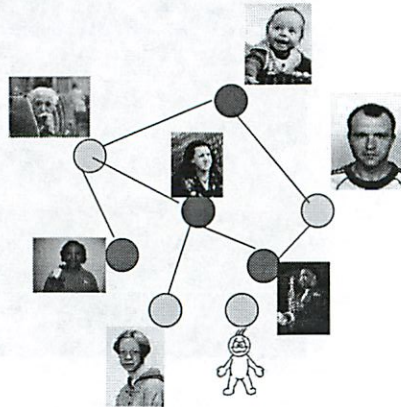
Vast data



- Impossible to access all of it
- Potentially accessible data is too enormous to be viewed by a single individual
- Once accessed, data can change

Connected world phenomenon

- each “node” is a person
- “edge” between people that know each other
- Is the underlying graph connected?



Does earth have the connected world property?

- How can we know?
 - data collection problem is immense
 - unknown groups of people found on earth
 - births/deaths

The Gold Standard in 6.046

- Linear time algorithms!
- Are they adequate?



What can we hope to do *without* viewing most of the data?

- Can't answer “for all” or “exactly” type statements:
 - Exactly how many individuals on earth are left-handed?
 - Are all individuals connected?
- Maybe can answer?
 - approximately how many individuals on earth are left-handed?
 - is there a large group of connected individuals?

What can we hope to do without viewing most of the data?

- Change our goals?
 - for most interesting problems: algorithm must give approximate answer
- we know we can answer *some* questions...
 - e.g., sampling to approximate average, median values

What types of approximation?

- “Classical” approximation for optimization problems:
 - output is number that is close to value of the optimal solution for given input.
 - (not enough time to construct a solution)
- Property testing for decision problems:
 - output is correct answer for given input, or at least for some other input “close” to it.

First:

I. Classical Approximation Problems

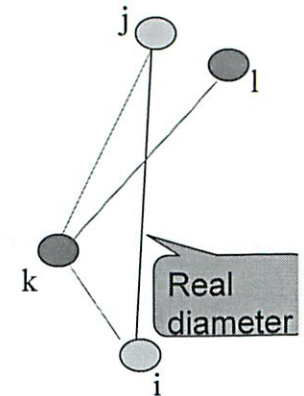
- A very simple example –
 - Deterministic
 - Approximate answer
 - And (of course).... Sub-linear time!

Approximate the diameter of a point set

- Given: m points, described by a distance matrix D , s.t.
 - D_{ij} is the distance from i to j .
 - D satisfies triangle inequality and symmetry.(note: input size $n = m^2$)
- Let i, j be indices that maximize D_{ij} then D_{ij} is the *diameter*.
- Output: k, l such that $D_{kl} \geq D_{ij}/2$

Algorithm

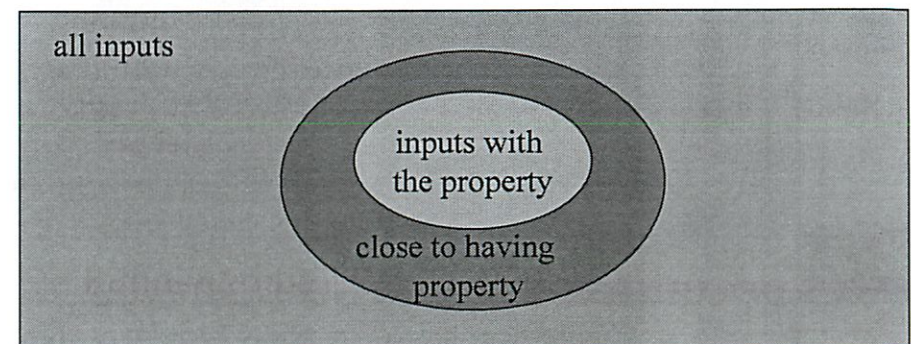
- Algorithm:
 - Pick k arbitrarily
 - Pick l to maximize D_{kl}
 - Output D_{kl}
- Why does it work?
 - $D_{ij} \leq D_{ik} + D_{kj}$ (triangle inequality)
 - $\leq D_{kl} + D_{kl}$ (choice of l + symmetry of D)
 - $\leq 2D_{kl}$
- Running time? $O(m) = O(n^{1/2})$



Main Goal:

II. Property testing

- Quickly distinguish inputs that have specific property from those that are far from having the property



Property Testing

- Properties of any object, e.g.,
 - Functions
 - Graphs
 - Strings
 - Matrices
 - Codewords
- Model must specify
 - representation of object and allowable queries
 - notion of close/far, e.g.,
 - number of bits/words that need to be changed
 - edit distance

Sortedness of a sequence

- Given: list $y_1 y_2 \dots y_n$
- Question: is the list sorted?
- Clearly requires n steps – must look at each y_i

A simple property tester

Sortedness of a sequence

- Given: list $y_1 y_2 \dots y_n$
- Question: can we quickly test if the list close to sorted?

What do we mean by “quick”?

- query complexity measured in terms of list size n
- Our goal (if possible):
 - Very small compared to n , will go for $\log n$

Requirements for algorithm:

- Pass sorted lists
 - Fail lists that are ϵ -far.
 - Equivalently: if list likely to pass test, can change at most ϵ fraction of list to make it sorted
- Probability of success $> \frac{3}{4}$
(can boost it arbitrarily high by repeating several times and outputting “fail” if ever see a “fail”, “pass” otherwise)
- Can test in $O(1/\epsilon \log n)$ time
(and can’t do any better!)

What if list not sorted,
but not far?

What do we mean by “close”?

Definition: a list of size n is ϵ -close to sorted if can delete at most ϵn values to make it sorted.

Otherwise, ϵ -far.

(ϵ is given as input, e.g., $\epsilon=1/10$)

Sorted: 1 2 4 5 7 11 14 19 20 21 23 38 39 45
Close: 1 4 2 5 7 11 14 19 20 39 23 21 38 45
1 4 5 7 11 14 19 20 23 38 45
Far: 45 39 23 1 38 4 5 21 20 19 2 7 11 14
1 4 5 7 11 14

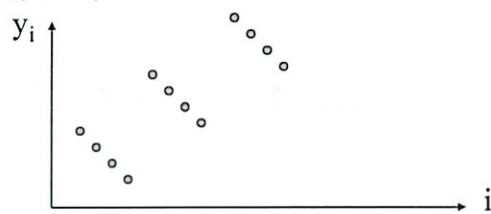
An attempt:

- Proposed algorithm:
 - Pick random i and test that $y_i \leq y_{i+1}$
- Bad input type:
 - 1,2,3,4,5,... $n/4$, 1,2,... $n/4$, 1,2,... $n/4$, 1,2,..., $n/4$
 - Difficult for this algorithm to find “breakpoint”
 - But other tests work well...



A second attempt:

- Proposed algorithm:
 - Pick random $i < j$ and test that $y_i \leq y_j$
- Bad input type:
 - $n/4$ groups of 4 decreasing elements
 $4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9, \dots, 4k, 4k-1, 4k-2, 4k-3, \dots$
 - Largest monotone sequence is $n/4$
 - must pick i, j in same group to see problem
 - need $\Omega(n^{1/2})$ samples



A test that works

- The test:
 - Test $O(1/\epsilon)$ times:
 - Pick random i
 - Look at value of y_i
 - Do binary search for y_i
 - Does the binary search find any inconsistencies? If yes, FAIL
 - Do we end up at location i ? If not FAIL
 - Pass if never failed
- Running time: $O(\epsilon^{-1} \log n)$ time
- Why does this work?

A minor simplification:

- Assume list is distinct (i.e. $x_i \neq x_j$)
- Claim: this is not really easier
 - Why?
 - Can "virtually" append i to each x_i
 $x_1, x_2, \dots, x_n \mapsto (x_1, 1), (x_2, 2), \dots, (x_n, n)$
e.g., $1, 1, 2, 6, 6 \mapsto (1, 1), (1, 2), (2, 3), (6, 4), (6, 5)$
 - Breaks ties without changing order

Behavior of the test:

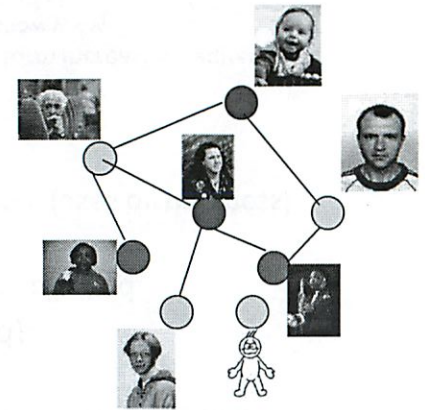
- Define index i to be good if binary search for y_i successful
- $O(1/\epsilon \log n)$ time test (restated):
 - pick $O(1/\epsilon)$ i 's and pass if they are all good
- Correctness:
 - If list is sorted, then all i 's are good (uses distinctness)
 - So test always passes
 - If list likely to pass test,
 - Then at least $(1-\epsilon)n$ i 's are good.
 - Main observation: good elements form increasing sequence
 - Proof: for $i < j$ both good need to show $y_i < y_j$
 - let k = least common ancestor of i, j
 - Search for i went left of k and search for j went right of $k \rightarrow y_i < y_k < y_j$
- Thus list is ϵ -close to monotone (delete $< \epsilon n$ bad elements)

Testing connectedness of a graph

- Given graph G
 - n vertices
 - Max degree d
 - Adjacency list representation
- Is G connected?

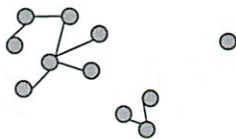
Connected world phenomenon

- Is the underlying graph close to connected?



Close to connected

- Def: G is ϵ -close to connected if can add $< \epsilon dn$ edges and transform it to connected
 - Today: ok to violate max deg d requirement



Property tester:

- Input: ϵ and G
- Output:
 - If G connected, output “PASS”
 - If G not ϵ -close to connected, output “FAIL” with probability $\geq 3/4$
 - (note: if G not connected, but is close, then ok to output either “PASS” or “FAIL”)

Idea:

- If G far from connected, lots of nodes must be in small components!
- More specifically...
 - Will show that if G far from connected
 - Then must have many connected components
 - So many components must be small
 - And there must be many nodes in small components

Behavior

- Lemma 1: If G ϵ -far from connected, then has $\geq \epsilon dn$ components
- Lemma 2: If $\geq \epsilon dn$ components then $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$
- Observation: If $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$ then $\geq \epsilon dn/2$ nodes in components of size $< \frac{2}{\epsilon d}$

These cause
tester to FAIL!

Algorithm:

- Do $O(\frac{1}{\epsilon d})$ times:
 - Pick random node s , and run BFS from s until:
 - $\geq \frac{2}{\epsilon d}$ distinct nodes seen
 - OR see that s is component of size $< \frac{2}{\epsilon d}$ nodes, in which case output "FAIL" and halt
- If reach this point, output "PASS"

Runtime: $O(\frac{1}{\epsilon d})$ loops, each does $O(\frac{1}{\epsilon d})$ steps of BFS, using $O(d)$ time per step – total is $O(\frac{1}{\epsilon^2 d})$

Behavior

- Putting it together: If G ϵ -far from connected, then $\geq \epsilon d/2$ fraction of nodes cause algorithm to fail!
 - So $\text{Prob}[\text{tester fails in one of } \frac{c}{\epsilon d} \text{ loops}]$ is
$$\geq 1 - \left(1 - \frac{\epsilon d}{2}\right)^{\left\lceil \frac{2c}{\epsilon d} \right\rceil} \geq 1 - e^{-c/2} \geq 3/4 \quad (\text{for big enough } c)$$

Lemma 1

If G is ϵ -far from connected, then it has $\geq \epsilon dn$ components

Proof: if $< \epsilon dn$ components, can add $< \epsilon dn$ edges to connect

Lemma 2

If $\geq \epsilon dn$ components then $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$

(see notes for proof)

Observation:

If $\geq \epsilon dn/2$ components of size $< \frac{2}{\epsilon d}$ then
 $\geq \epsilon dn/2$ nodes in components of size $< \frac{2}{\epsilon d}$

Why? Each small component has at least one node.

Problem Set 6

This problem set is due at **11:59pm on Thursday, December 06, 2012.**

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Each problem must be turned in separately to Stellar.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. A proof (or indication) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convolved and obtuse descriptions will receive low marks.

Exercise 5-1. Do Exercise 31.7-1 in CLRS on page 964.

Exercise 5-2. Do Exercise 31.7-3 in CLRS on page 965.

Exercise 5-3. Do Exercise 27.2-3 in CLRS on page 796.

Problem 5-1. One-way functions

Circle the functions that are likely to be one-way. Explain why or why not. State your assumptions clearly.

1. $f(x; y) = x \oplus y$.

2. $f(p; q) = pq$ where p and q are prime.

3. $f(x) = \log_g(x) \bmod p$, where p is a fixed prime and g is a fixed generator of Z_p^* . g is a generator of Z_p^* if the order of $g \bmod p$ is $p-1$. The least positive x such that $a^x \equiv 1 \pmod{p}$ is called the order of $a, \bmod p$.

Problem 5-2. RSA: finding d

For the RSA cryptosystem, suppose the modulus is $n = 55$. If the encryption exponent $e = 7$, what is the decryption exponent d ? Show your work.

Problem 5-3. RSA: walkthrough

For this problem you will determine public and private keys for the RSA cryptosystem and encrypt and decrypt a message according to RSA. You can use Wolfram Alpha web resource for computational help (<http://www.wolframalpha.com/>). You should write down all the computational steps required in each of the parts below.

Entity A chooses the primes $p = 2357$, $q = 2551$, and chooses $e = 3674911$. Answer the following questions.

1. What is A 's public key?
2. What is A 's private key?
3. What is the ciphertext corresponding to $m = 5234673$?
4. Show the steps in decrypting the ciphertext generated in Part 3 above to obtain m .

Problem 5-4. Parallel merging

Design a parallel algorithm that uses $2n$ processors to merge two sorted lists, each of length n , in $O(\log n)$ parallel steps. Assume there is a common shared memory such that any number of processors can access any location in memory on a read at the same time. Assume also that all the elements in the lists are all distinct.

Note: a parallel algorithm that uses $2n$ processors to merge two sorted lists, each of length n , in $O(\log^2 n)$ parallel steps is given in Chapter 27.3 of CLRS. The model of CLRS is more restrictive since more than one processor may not access the same memory location in the same parallel step.

12/3

9:30P

#6 #1 | Onekey functions

1. $f(x; y) = x \oplus y$

Well no.

By definition XOR is reversible

a	b	\oplus
0	0	0
0	1	1
1	0	1
1	1	0

Its the key. right

Value 1111
key 1010

XOR 0101
XOR key 1111 ← back

②

2. $f(p, q) = pq$ where p, q are prime

But what does 1 way mean?

~~We can~~

$$\text{WP} \left[P[f(A(f(x))) = f(x)] < \frac{1}{p(n)} \right]$$

\uparrow $\{0, 1\}$ \uparrow every polynomial

"hard to invert"

not just lossy (not one-to-one)

ie ~~are~~ output n 0s for a input of length n

I don't get $< \frac{1}{p(n)}$ - every prob

hard means prohibitively expensive

does mention multiplying primes as carrying

(3)

Since $p \cdot q$ is $O(n^2)$ where $n = \# \text{ of digits}$

Finding factors $O(2^{(\log N)^{1/3}} (\log \log N)^{2/3})$

pseudopolynomial in $\log N$

Pseudo-polynomial is poly in numeric value of input

weakly NP-complete

ex: testing if n is prime

check $\{2, 3, \dots, \sqrt{n}\}$

takes $\sqrt{n} - 1$ divisions

sublinear in value of n

but exponential in size of $n (\log n)$

What matters is length of encoded input

ex: adding 2 8-digit #s
takes 8 steps

so yes likely I may

(9)

$$3. f(x) = \log_g(x) \pmod{p}$$

p = fixed prime

g = fixed generator of \mathbb{Z}_p^*

g is a generator of \mathbb{Z}_p^* if order of $g \pmod{p}$ is $p-1$

Least positive x s.t.,

$a^x \equiv 1 \pmod{p}$ is called the
order of $a \pmod{p}$

didn't get that...

Generator Polynomial

polynomials that are divisibly
by some fixed polynomial

ie $g(x) = x^2 + x + 1$ $n=5$ $m=2$

5

is $0 \cdot g(x)$ $1 \cdot g(x)$ ~~$x \cdot g(x)$~~

$(x+1)g(x)$ $x^2 g(x)$ $(x^2+1)g(x)$

$(x^2+x)g(x)$ $(x^2+x+1)g(x)$

aka

0 x^2+x+1 x^3+x^2+x

x^3+1 $x^4+x^3+x^2$ x^4+x^3+x+1

x^4+x x^4+x^2+1

which is

00000 00111 01110

so which
ones present

Note any linear combo of code word are also code word,

Note this is like XOR here (GF(2))

00111 \oplus 01001

01110 \oplus

6

Could not find in 6.042 notes...

But remember it...

Generator polynomials

note $m \leq n$

m = degree of generator

n = max degree of polynomial is $n-1$

$\mathbb{F}(q)$ is a finite field

These polynomials divisible by $g(x)$

ie ~~$\mathbb{Z}_2[x]$~~

$$\begin{array}{r} x \\ x^3 + x + 1 \overline{) x^3 + x^2 + x} \end{array}$$

← was made $x \cdot g(x)$

Think I get this now
but how does it help?

log generator

↑ What makes it a generator or not?

(Or is this wrong generator?)

really need to ask here...

order of \rightarrow multiplicative order

~~the~~ multiplicative order of $a \bmod n$ is
Smallest positive int k w/

$$a^k \equiv 1 \pmod{n}$$

ie the order of $4 \bmod 7$

$$4^2 = 16 \equiv 2 \pmod{7} \quad (\times)$$

$$4^3 = 64 \equiv 1 \pmod{7} \quad (\checkmark)$$

$$\text{So } O_7(4) = 3$$

Or they defined that for us

(8)

(c)

know g, p

Have func

$\log_g x \bmod p$

↑ if that ^{when hashed} = a

~~$g^a \bmod p = x$~~

If hash then

$$a = \log_g x \bmod p$$
$$\hookrightarrow g^a \bmod p = x$$

normal ~~log~~ log

mod is $\{0, 1, \dots, p-1\}$
of course

So qv. w/ 1 way

if given a , can ya find a x ?

Diff
12/4
7p

Q9

One way Given a , can find x s.t. $f(x)=a$

~~x~~ can be any one not sure we
can find a
No! not one way

WP defn

Saying bound doesn't matter

becomes ∞ in the limit

Prob is very low

Given a , can find any x

↑
random fn here

(10)

17/5
7:30P

Actually its not that reversible
well kind of

More that can go bit by bit + recover it

Maintenance: Let c' and d' denote the values of c and d at the end of an iteration of the **for** loop, and thus the values prior to the next iteration. Each iteration updates $c' = 2c$ (if $b_i = 0$) or $c' = 2c + 1$ (if $b_i = 1$), so that c will be correct prior to the next iteration. If $b_i = 0$, then $d' = d^2 \bmod n = (a^c)^2 \bmod n = a^{2c} \bmod n = a^{c'} \bmod n$. If $b_i = 1$, then $d' = d^2 a \bmod n = (a^c)^2 a \bmod n = a^{2c+1} \bmod n = a^{c'} \bmod n$. In either case, $d = a^c \bmod n$ prior to the next iteration.

Termination: At termination, $i = -1$. Thus, $c = b$, since c has the value of the prefix $\langle b_k, b_{k-1}, \dots, b_0 \rangle$ of b 's binary representation. Hence $d = a^c \bmod n = a^b \bmod n$.

If the inputs a , b , and n are β -bit numbers, then the total number of arithmetic operations required is $O(\beta)$ and the total number of bit operations required is $O(\beta^3)$.

Exercises

31.6-1

Draw a table showing the order of every element in \mathbb{Z}_{11}^* . Pick the smallest primitive root g and compute a table giving $\text{ind}_{11,g}(x)$ for all $x \in \mathbb{Z}_{11}^*$.

31.6-2

Give a modular exponentiation algorithm that examines the bits of b from right to left instead of left to right.

31.6-3

Assuming that you know $\phi(n)$, explain how to compute $a^{-1} \bmod n$ for any $a \in \mathbb{Z}_n^*$ using the procedure MODULAR-EXPONENTIATION.

31.7 The RSA public-key cryptosystem

With a public-key cryptosystem, we can encrypt messages sent between two communicating parties so that an eavesdropper who overhears the encrypted messages will not be able to decode them. A public-key cryptosystem also enables a party to append an unforgeable "digital signature" to the end of an electronic message. Such a signature is the electronic version of a handwritten signature on a paper document. It can be easily checked by anyone, forged by no one, yet loses its validity if any bit of the message is altered. It therefore provides authentication of both the identity of the signer and the contents of the signed message. It is the perfect tool

for electronically signed business contracts, electronic checks, electronic purchase orders, and other electronic communications that parties wish to authenticate.

The RSA public-key cryptosystem relies on the dramatic difference between the ease of finding large prime numbers and the difficulty of factoring the product of two large prime numbers. Section 31.8 describes an efficient procedure for finding large prime numbers, and Section 31.9 discusses the problem of factoring large integers.

Public-key cryptosystems

In a public-key cryptosystem, each participant has both a public key and a secret key. Each key is a piece of information. For example, in the RSA cryptosystem, each key consists of a pair of integers. The participants “Alice” and “Bob” are traditionally used in cryptography examples; we denote their public and secret keys as P_A, S_A for Alice and P_B, S_B for Bob.

Each participant creates his or her own public and secret keys. Secret keys are kept secret, but public keys can be revealed to anyone or even published. In fact, it is often convenient to assume that everyone’s public key is available in a public directory, so that any participant can easily obtain the public key of any other participant.

The public and secret keys specify functions that can be applied to any message. Let \mathcal{D} denote the set of permissible messages. For example, \mathcal{D} might be the set of all finite-length bit sequences. In the simplest, and original, formulation of public-key cryptography, we require that the public and secret keys specify one-to-one functions from \mathcal{D} to itself. We denote the function corresponding to Alice’s public key P_A by $P_A()$ and the function corresponding to her secret key S_A by $S_A()$. The functions $P_A()$ and $S_A()$ are thus permutations of \mathcal{D} . We assume that the functions $P_A()$ and $S_A()$ are efficiently computable given the corresponding key P_A or S_A .

The public and secret keys for any participant are a “matched pair” in that they specify functions that are inverses of each other. That is,

$$M = S_A(P_A(M)), \quad (31.35)$$

$$M = P_A(S_A(M)) \quad \text{both work} \quad (31.36)$$

for any message $M \in \mathcal{D}$. Transforming M with the two keys P_A and S_A successively, in either order, yields the message M back.

In a public-key cryptosystem, we require that no one but Alice be able to compute the function $S_A()$ in any practical amount of time. This assumption is crucial to keeping encrypted mail sent to Alice private and to knowing that Alice’s digital signatures are authentic. Alice must keep S_A secret; if she does not, she loses her uniqueness and the cryptosystem cannot provide her with unique capabilities. The assumption that only Alice can compute $S_A()$ must hold even though everyone

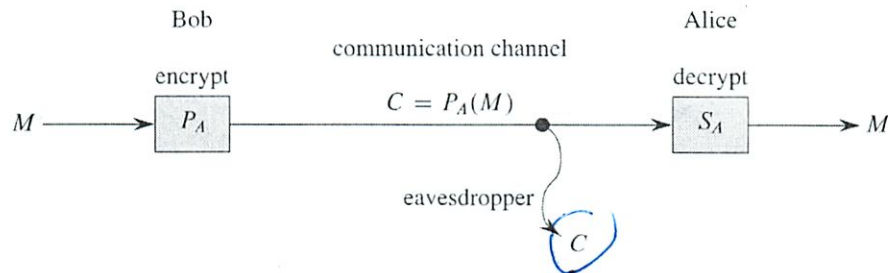


Figure 31.5 Encryption in a public key system. Bob encrypts the message M using Alice's public key P_A and transmits the resulting ciphertext $C = P_A(M)$ over a communication channel to Alice. An eavesdropper who captures the transmitted ciphertext gains no information about M . Alice receives C and decrypts it using her secret key to obtain the original message $M = S_A(C)$.

knows P_A and can compute $P_A()$, the inverse function to $S_A()$, efficiently. In order to design a workable public-key cryptosystem, we must figure out how to create a system in which we can reveal a transformation $P_A()$ without thereby revealing how to compute the corresponding inverse transformation $S_A()$. This task appears formidable, but we shall see how to accomplish it.

In a public-key cryptosystem, encryption works as shown in Figure 31.5. Suppose Bob wishes to send Alice a message M encrypted so that it will look like unintelligible gibberish to an eavesdropper. The scenario for sending the message goes as follows.

- Bob obtains Alice's public key P_A (from a public directory or directly from Alice).
- Bob computes the *ciphertext* $C = P_A(M)$ corresponding to the message M and sends C to Alice.
- When Alice receives the ciphertext C , she applies her secret key S_A to retrieve the original message: $S_A(C) = S_A(P_A(M)) = M$.

Because $S_A()$ and $P_A()$ are inverse functions, Alice can compute M from C . Because only Alice is able to compute $S_A()$, Alice is the only one who can compute M from C . Because Bob encrypts M using $P_A()$, only Alice can understand the transmitted message.

We can just as easily implement digital signatures within our formulation of a public-key cryptosystem. (There are other ways of approaching the problem of constructing digital signatures, but we shall not go into them here.) Suppose now that Alice wishes to send Bob a digitally signed response M' . Figure 31.6 shows how the digital-signature scenario proceeds.

- Alice computes her *digital signature* σ for the message M' using her secret key S_A and the equation $\sigma = S_A(M')$.

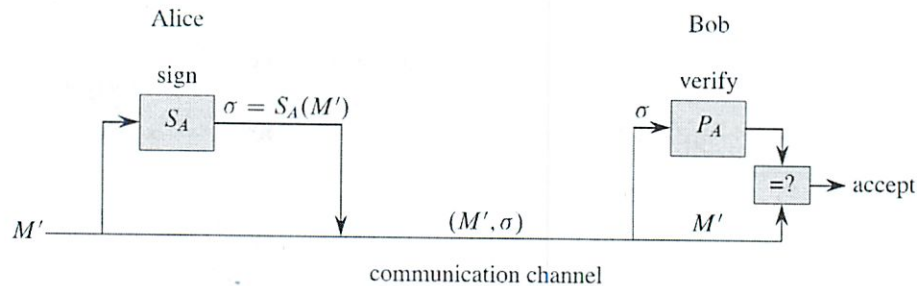


Figure 31.6 Digital signatures in a public-key system. Alice signs the message M' by appending her digital signature $\sigma = S_A(M')$ to it. She transmits the message/signature pair (M', σ) to Bob, who verifies it by checking the equation $M' = P_A(\sigma)$. If the equation holds, he accepts (M', σ) as a message that Alice has signed.

- Alice sends the message/signature pair (M', σ) to Bob.
- When Bob receives (M', σ) , he can verify that it originated from Alice by using Alice's public key to verify the equation $M' = P_A(\sigma)$. (Presumably, M' contains Alice's name, so Bob knows whose public key to use.) If the equation holds, then Bob concludes that the message M' was actually signed by Alice. If the equation fails to hold, Bob concludes either that the message M' or the digital signature σ was corrupted by transmission errors or that the pair (M', σ) is an attempted forgery.

Because a digital signature provides both authentication of the signer's identity and authentication of the contents of the signed message, it is analogous to a handwritten signature at the end of a written document.

A digital signature must be verifiable by anyone who has access to the signer's public key. A signed message can be verified by one party and then passed on to other parties who can also verify the signature. For example, the message might be an electronic check from Alice to Bob. After Bob verifies Alice's signature on the check, he can give the check to his bank, who can then also verify the signature and effect the appropriate funds transfer.

A signed message is not necessarily encrypted; the message can be "in the clear" and not protected from disclosure. By composing the above protocols for encryption and for signatures, we can create messages that are both signed and encrypted. The signer first appends his or her digital signature to the message and then encrypts the resulting message/signature pair with the public key of the intended recipient. The recipient decrypts the received message with his or her secret key to obtain both the original message and its digital signature. The recipient can then verify the signature using the public key of the signer. The corresponding combined process using paper-based systems would be to sign the paper document and

then seal the document inside a paper envelope that is opened only by the intended recipient.

The RSA cryptosystem

In the *RSA public-key cryptosystem*, a participant creates his or her public and secret keys with the following procedure:

1. Select at random two large prime numbers p and q such that $p \neq q$. The primes p and q might be, say, 1024 bits each.
2. Compute $n = pq$.
3. Select a small odd integer e that is relatively prime to $\phi(n)$, which, by equation (31.20), equals $(p-1)(q-1)$.
4. Compute d as the multiplicative inverse of e , modulo $\phi(n)$. (Corollary 31.26 guarantees that d exists and is uniquely defined. We can use the technique of Section 31.4 to compute d , given e and $\phi(n)$.)
5. Publish the pair $P = (e, n)$ as the participant's *RSA public key*.
6. Keep secret the pair $S = (d, n)$ as the participant's *RSA secret key*.

For this scheme, the domain \mathcal{D} is the set \mathbb{Z}_n . To transform a message M associated with a public key $P = (e, n)$, compute

$$P(M) = M^e \bmod n. \quad (31.37)$$

To transform a ciphertext C associated with a secret key $S = (d, n)$, compute

$$S(C) = C^d \bmod n. \quad (31.38)$$

These equations apply to both encryption and signatures. To create a signature, the signer applies his or her secret key to the message to be signed, rather than to a ciphertext. To verify a signature, the public key of the signer is applied to it, rather than to a message to be encrypted.

We can implement the public-key and secret-key operations using the procedure MODULAR-EXPONENTIATION described in Section 31.6. To analyze the running time of these operations, assume that the public key (e, n) and secret key (d, n) satisfy $\lg e = O(1)$, $\lg d \leq \beta$, and $\lg n \leq \beta$. Then, applying a public key requires $O(1)$ modular multiplications and uses $O(\beta^2)$ bit operations. Applying a secret key requires $O(\beta)$ modular multiplications, using $O(\beta^3)$ bit operations.

Theorem 31.36 (Correctness of RSA)

The RSA equations (31.37) and (31.38) define inverse transformations of \mathbb{Z}_n satisfying equations (31.35) and (31.36).

Proof From equations (31.37) and (31.38), we have that for any $M \in \mathbb{Z}_n$,

$$P(S(M)) = S(P(M)) = M^{ed} \pmod{n}.$$

Since e and d are multiplicative inverses modulo $\phi(n) = (p-1)(q-1)$,

$$ed = 1 + k(p-1)(q-1)$$

for some integer k . But then, if $M \not\equiv 0 \pmod{p}$, we have

$$\begin{aligned} M^{ed} &\equiv M(M^{p-1})^{k(q-1)} \pmod{p} \\ &\equiv M((M \bmod p)^{p-1})^{k(q-1)} \pmod{p} \\ &\equiv M(1)^{k(q-1)} \pmod{p} \quad (\text{by Theorem 31.31}) \\ &\equiv M \pmod{p}. \end{aligned}$$

Also, $M^{ed} \equiv M \pmod{p}$ if $M \equiv 0 \pmod{p}$. Thus,

$$M^{ed} \equiv M \pmod{p}$$

for all M . Similarly,

$$M^{ed} \equiv M \pmod{q}$$

for all M . Thus, by Corollary 31.29 to the Chinese remainder theorem,

$$M^{ed} \equiv M \pmod{n}$$

for all M . ■

The security of the RSA cryptosystem rests in large part on the difficulty of factoring large integers. If an adversary can factor the modulus n in a public key, then the adversary can derive the secret key from the public key, using the knowledge of the factors p and q in the same way that the creator of the public key used them. Therefore, if factoring large integers is easy, then breaking the RSA cryptosystem is easy. The converse statement, that if factoring large integers is hard, then breaking RSA is hard, is unproven. After two decades of research, however, no easier method has been found to break the RSA public-key cryptosystem than to factor the modulus n . And as we shall see in Section 31.9, factoring large integers is surprisingly difficult. By randomly selecting and multiplying together two 1024-bit primes, we can create a public key that cannot be “broken” in any feasible amount of time with current technology. In the absence of a fundamental breakthrough in the design of number-theoretic algorithms, and when implemented with care following recommended standards, the RSA cryptosystem is capable of providing a high degree of security in applications.

In order to achieve security with the RSA cryptosystem, however, we should use integers that are quite long—hundreds or even more than one thousand bits

long—to resist possible advances in the art of factoring. At the time of this writing (2009), RSA moduli were commonly in the range of 768 to 2048 bits. To create moduli of such sizes, we must be able to find large primes efficiently. Section 31.8 addresses this problem.

For efficiency, RSA is often used in a “hybrid” or “key-management” mode with fast non-public-key cryptosystems. With such a system, the encryption and decryption keys are identical. If Alice wishes to send a long message M to Bob privately, she selects a random key K for the fast non-public-key cryptosystem and encrypts M using K , obtaining ciphertext C . Here, C is as long as M , but K is quite short. Then, she encrypts K using Bob’s public RSA key. Since K is short, computing $P_B(K)$ is fast (much faster than computing $P_B(M)$). She then transmits $(C, P_B(K))$ to Bob, who decrypts $P_B(K)$ to obtain K and then uses K to decrypt C , obtaining M .

We can use a similar hybrid approach to make digital signatures efficiently. This approach combines RSA with a public *collision-resistant hash function* h —a function that is easy to compute but for which it is computationally infeasible to find two messages M and M' such that $h(M) = h(M')$. The value $h(M)$ is a short (say, 256-bit) “fingerprint” of the message M . If Alice wishes to sign a message M , she first applies h to M to obtain the fingerprint $h(M)$, which she then encrypts with her secret key. She sends $(M, S_A(h(M)))$ to Bob as her signed version of M . Bob can verify the signature by computing $h(M)$ and verifying that P_A applied to $S_A(h(M))$ as received equals $h(M)$. Because no one can create two messages with the same fingerprint, it is computationally infeasible to alter a signed message and preserve the validity of the signature.

Finally, we note that the use of *certificates* makes distributing public keys much easier. For example, assume there is a “trusted authority” T whose public key is known by everyone. Alice can obtain from T a signed message (her certificate) stating that “Alice’s public key is P_A .” This certificate is “self-authenticating” since everyone knows P_T . Alice can include her certificate with her signed messages, so that the recipient has Alice’s public key immediately available in order to verify her signature. Because her key was signed by T , the recipient knows that Alice’s key is really Alice’s.

Exercises

31.7-1

Consider an RSA key set with $p = 11$, $q = 29$, $n = 319$, and $e = 3$. What value of d should be used in the secret key? What is the encryption of the message $M = 100$?

31.7-2

Prove that if Alice's public exponent e is 3 and an adversary obtains Alice's secret exponent d , where $0 < d < \phi(n)$, then the adversary can factor Alice's modulus n in time polynomial in the number of bits in n . (Although you are not asked to prove it, you may be interested to know that this result remains true even if the condition $e = 3$ is removed. See Miller [255].)

31.7-3 ★

Prove that RSA is multiplicative in the sense that

$$P_A(M_1)P_A(M_2) \equiv P_A(M_1M_2) \pmod{n}.$$

Use this fact to prove that if an adversary had a procedure that could efficiently decrypt 1 percent of messages from \mathbb{Z}_n encrypted with P_A , then he could employ a probabilistic algorithm to decrypt every message encrypted with P_A with high probability.

★ 31.8 Primality testing

In this section, we consider the problem of finding large primes. We begin with a discussion of the density of primes, proceed to examine a plausible, but incomplete, approach to primality testing, and then present an effective randomized primality test due to Miller and Rabin.

The density of prime numbers

For many applications, such as cryptography, we need to find large “random” primes. Fortunately, large primes are not too rare, so that it is feasible to test random integers of the appropriate size until we find a prime. The *prime distribution function* $\pi(n)$ specifies the number of primes that are less than or equal to n . For example, $\pi(10) = 4$, since there are 4 prime numbers less than or equal to 10, namely, 2, 3, 5, and 7. The prime number theorem gives a useful approximation to $\pi(n)$.

Theorem 31.37 (Prime number theorem)

$$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln n} = 1.$$

■

The approximation $n / \ln n$ gives reasonably accurate estimates of $\pi(n)$ even for small n . For example, it is off by less than 6% at $n = 10^9$, where $\pi(n) =$

#2 | RSA finding d

Suppose $\text{mod} = n = 55$

if $e = 7 = \text{encryption exponent}$

What is decryption exponent d ?

Reading chap
Should read rest at some point
Theory Refresher...

$n = pq$ \checkmark primes

e is rel prime to $\phi(n) = (p-1)(q-1)$

Rel prime = coprime
 $\gcd(a, b) = 1$
always a pair!

Compute d as multiplicative inverse
of $e \text{ mod } \phi(n)$

$$\frac{1}{x} = x^{-1} \quad \frac{a}{b} \rightarrow \frac{b}{a}$$

②

do this w/ the Chinese-Remainder Theorem

↳ G.042 called this the Pheizer

$$de = 1 \pmod{(p-1)(q-1)}$$

Pheizer

The water jug problem

$$3 = s \cdot 21 + t \cdot 26$$

$$\text{Since } \gcd(21, 26) = 1$$

An integer is a linear combo of a, b
iff it is a multiple of $\gcd(a, b)$

Repeat: over till 3 gallons

1. Fill 21 gal jug

2. Pour all into 26

2b. If 26 ever full, empty it and

2c. Continue filling 21 into 26

(3)

So always a s, t so that

$$\gcd(a, b) = sa + tb$$

actually find w/ Reverse

aka extended Euclid's algorithm

$$\gcd(a, b) = \gcd(b, \text{rem}(a, b))$$

$$\begin{aligned} \text{ie } \gcd(\overset{259}{20}, \overset{70}{49}) &= \gcd(20, 49) \\ &= \gcd(49, 21) \\ &= \gcd(21, 7) \\ &= \gcd(7, 0) \\ &= 7 \end{aligned}$$

Same steps, just w/ extra bookkeeping

3.4 is Solving modular linear eqn

(4)

So actually.

Factor 55 to p, q

No efficient alg

~~Don't~~ number field size is best

Or just write all $0 \rightarrow \sqrt{n}$

Guess that is bigger than $\mathbb{P} e^{1,2}$

factor 55 is 5×11

So ϕ is $(4) \cdot 10 = 40$

Pick e ~~7/40~~

$e \rightarrow 7$ (given)

Check $\gcd(7, 40) = 1$ ✓

So d is $\frac{1}{7} \bmod 40$ ✓

(5)

That is just $\frac{1}{7} \dots$

$$7^{-1} \pmod{40} = 23$$

weird $7^{-1} = \frac{1}{7}$ ~~but~~

$$7^{-1} \pmod{40} = 23$$

$$\text{but } \frac{1}{7} \pmod{40} = \frac{1}{7}$$

Or something special about
modular multiplicative inverse

$$a^{-1} \equiv x \pmod{m}$$

is

$$ax \equiv aa^{-1} \equiv 1 \pmod{m}$$

a, m must be coprime

ie $3^{-1} \equiv x \pmod{11}$

is like $3x \equiv 1 \pmod{11}$

Try Since $3 \cdot 4 \pmod{11} = 12 \pmod{11} = 1$

(6)

Find w/ extended euclidean alg
(write up)

12/4
7:30p
0+1

TA's 2+3 no pulverizer needed to be shown
don't need to show every math step

#3 RSA Walkthrough

$$p = 2357$$

$$q = 2551$$

$$e = 3674911$$

$$n = 6012707$$

$$d = 2356 \cdot 2550 = 6007800$$

$$d = 422191$$

a.) $P = (3674911, 6012707)$

b.) $S = (422191, 6012707)$

c.) Encrypt $m = 5234673$

$$5234673^{3674911} \pmod{6012707}$$

$$3650502$$

d.) Decrypt

$$3650502^{422191} \pmod{6012707}$$

$$= 5234673 \quad \text{✓}$$

12/3
11:20P

#4 | Parallel merging

blast from the past...

Parallel algorithm

2n processors

Merge 2 sorted lists

each length = n

$O(\lg n)$ parallel steps

Common shared memory

↳ opposite of previous problem

parallel access to memory

all objects distinct

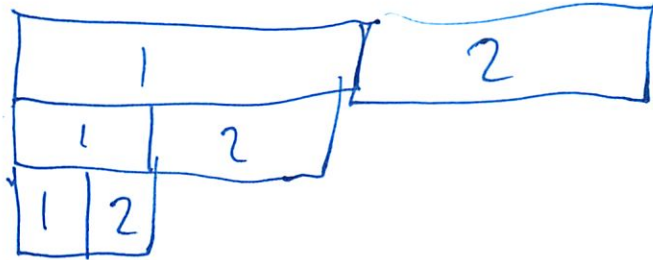
27.3 Multi-threaded Merge sort

$O(\lg^2 n)$ steps

but only 1 processor at same
memory in same parallel step

2.3.1 Normal Merge Sort

Divide, Conquer + Combine



etc

then combine pieces

$O(n)$ time

2 piles of cards

Sort each

~~then take 2 cards from top of both~~

~~put~~ place face up

put ^{top} card onto output pile

Smaller

if 1 pile empty, place

Code uses sentinel card

(3)

$$T = 2T\left(\frac{n}{2}\right) + \theta(n) \\ = \theta(n \lg n)$$

Multithreaded

already divide + conquer

make 1st recursive call n/ spawn

Sync links them back up

Merge-Sort(A, p, r)

if

$p < r$

$$q = \lfloor (p+r)/2 \rfloor$$

Spawn Merge-Sort(A, p, q)

Merge Sort ~~(A, p, q)~~
(A, q+1, r)

Sync

Merge(A, p, q, r)

$\theta(n)$

So $\theta(n \lg n)$

in series I think

(4)

In parallel

$$= MS(n/2) + \Theta(n)$$

$$= \Theta(n)$$

So $\frac{MS(n)}{MS_{\infty}(n)} = \Theta(\lg n)$

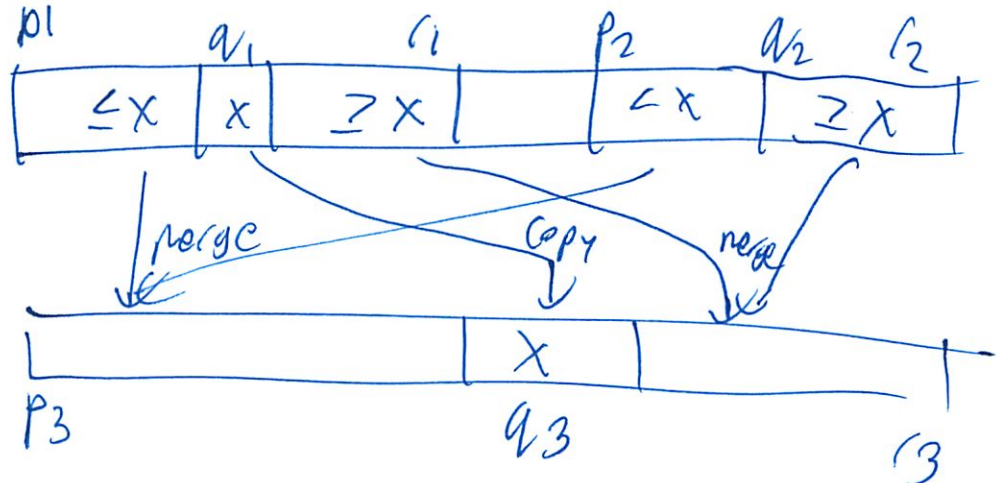
\swarrow serial \nwarrow parallel

not really parallelism.

Since Merge is serial

but can fix w/ nested parallelism

Merge



Find middle item of ~~the~~ part 1
use that as median

Find that in part 2 using binary search
Lac

5

Then merge

Copy x in

(recursively merge $[p_1 \dots q_1 - 1]$ $[q_2 \dots q_2 - 1]$)
" " $[q_1 + 1 \dots r_1]$ $[q_2 + 1 \dots r_2]$

So can we do this at same time?
Yeah that is the point

Both sides in parallel

Max # of elements is at least $\frac{3n}{4}$

$$n_2 \leq n_1$$

$$\text{So } n_2 = \frac{2n_2}{2} \leq \frac{(n_1 + n_2)}{2} = \frac{n}{2}$$

$$\begin{aligned} \lfloor n_1/2 \rfloor + n_2 &\leq n_1/2 + n_2/2 + n_2/2 \\ &= (n_1 + n_2)/2 + n_2/2 \\ &\leq n/2 + n/4 \\ &= 3n/4 \end{aligned}$$

Binary Search $O(\lg n)$

why?
Oh I see how x
chosen

6

$$PM(n) = PM_{\infty}(3n/4) + \Theta(\lg n)$$

So $\Theta(\lg^2 n)$ 'chow
master theorem

$$a=1$$

$$b=4$$

$n^{\lg 4} = 1$ vs $\lg n$

n vs $\lg n$

forget

↳ forgot ...

don't want to look up
Since I like the simplified
version....

Oh it saves no cases of Master Theorem

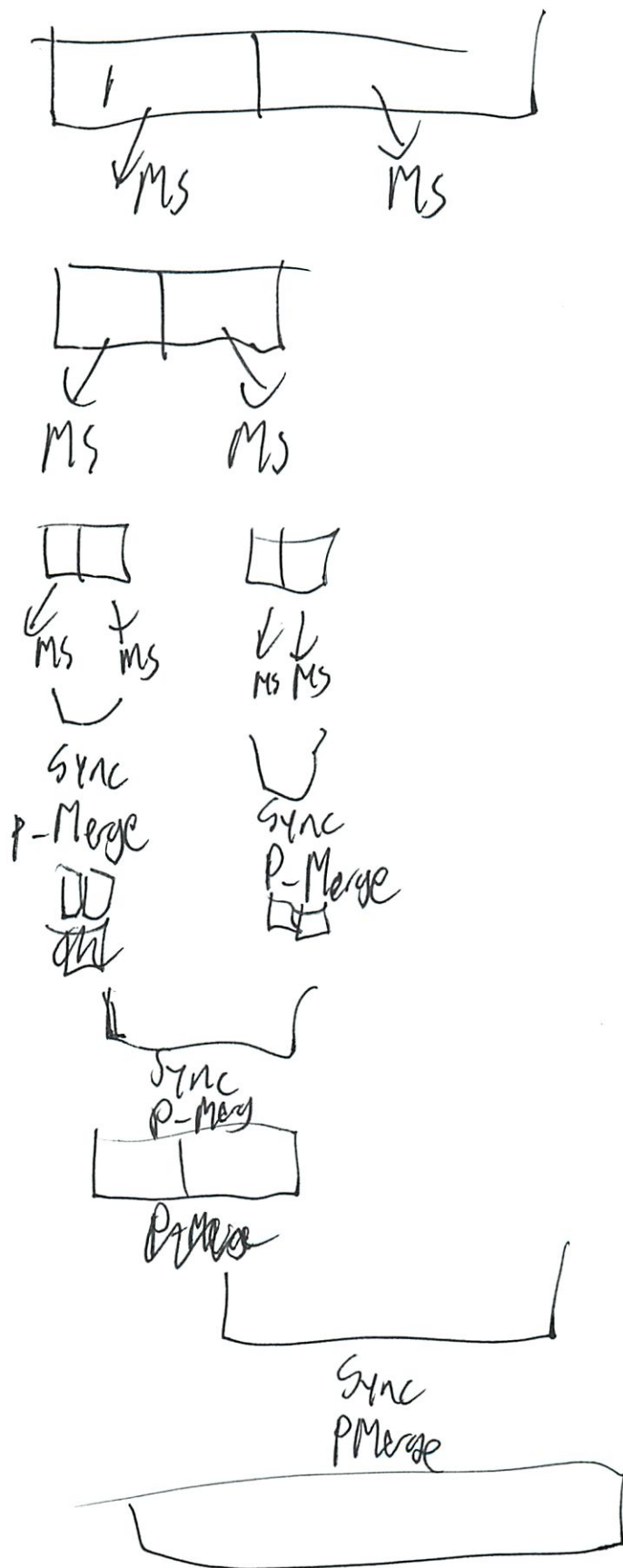
So put together to multi-threaded merge sort

Seems pretty simple

Well 2 things parallel now

So how much is parallel...

A hand-drawn diagram of a cell. It consists of an outer oval boundary. Inside this boundary is a smaller, roughly rectangular structure representing the nucleus. A single line connects the nucleus to the outer boundary, representing a nuclear pore.



8

So we have sync

$$\begin{aligned} \text{PMS}_1(n) &= 2 \text{PMS}_1(n/2) + \text{PM}_1(n) \\ &= 2 \text{PMS}_1(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

$$\begin{aligned} \text{PMS}_\infty(n) &= \text{PMS}_\infty(n/2) + \Theta(\lg^2 n) \\ &= \Theta(\lg^3 n) \end{aligned}$$

↑ that was merge

↑ also have the split

Now back to problem

We just have merge not

So just that clever part

Like where picking pivot

But what is $\lg n$ is $\lg^2 n$ ↑
I don't get how they found that!...

①

They will show $f(n) = \Theta(n^{\lg a} \lg^k n)$
has sol $\Theta(n^{\lg a} \lg^{k+1} n)$

Master Theorem notes

$n^{\lg a}$ vs $f(n)$

For each $n^c \cdot (\lg n)^d$

1. Compare c s
bigger c

2. Compare d s
bigger or $\cdot \lg n$
ie. $n^{\lg a} \cdot \lg n$

$n^{\lg 4}$ vs $\lg n$

~~$n^1 (\lg n)^0$ vs $n^0 (\lg n)^1$~~

~~So bigger c~~

~~So just n^1~~

1 vs $\lg n$

(18)

So

$$n^0 (\lg n)^0 \text{ vs } n^0 (\lg n)^1$$

So c's same

So d = $\lg n$

$$\lg n \lg n$$

$$\lg^2 n \quad \textcircled{\vee} \text{ verified}$$

But how do we change that?
The other side must be n^1

Need recurrence w/ i: hmmm

n^1 would be $O(n)$ (X) too high

$\lg n$ would be tie and $\lg^2 n$ (X) same

Those are the only choices....

Unless the other side was different...

$$\textcircled{\bullet} 1 \text{ vs } 1 \\ \text{would be } \Theta(\lg n)$$

⑪

Does that help?

I feel it should....

Well then $O(n^{\log_b a} \lg^k n)$

being $f(n)$

leads to $T(n)$

but isn't that related to other side

Oh well....

Oh they're put in on RHS
So related

Anyway a $T(3n/4) + \Theta(1)$ solution?
 $\Theta(\lg n)$ which comes from Binary Search

But I may be totally barking up the wrong tree
here w/ this analysis....

(12)

Ways to do w/ out binary search

Take the median...

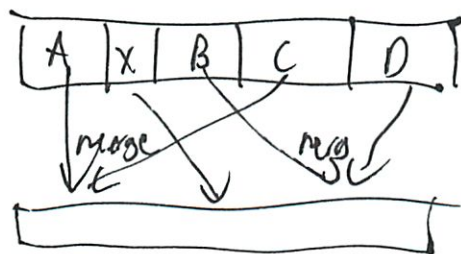
But I feel its something related to sharing memory...

* Accessing same memory location in same parallel steps
Its probably something class

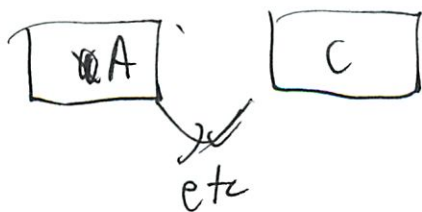
Ah they split again to merge



They do



So then



? So double recursive here
any way to fix?

(3)

What if we just pick median

Then could never reduce

Since could be



I think I'm overthinking this

Common

(how does one get better at these?)

Just start and pull from memory and merge



Then cross it off

but must merge these

Don't confuse machine level vs actual actions

Not sure fully what I mean by that

Don't understand where $\log_2 n$ comes from
for merge - see math - but doesn't feel...
Since binary search + merge?

(14)

Still confused on how to ~~sh~~ use shared memory

Any # of processes can access same place in memory

So before we split merge future

here we do something to do both everything

X Y
pull from each?

Normal pulls for card from each

well looks at both piles, picks lowest

i We do half of deck

But that is what we had!

Where do we cut the deck??

Think card metaphor might break down here...

How does multiple people looking at card help?

This is the card metaphor

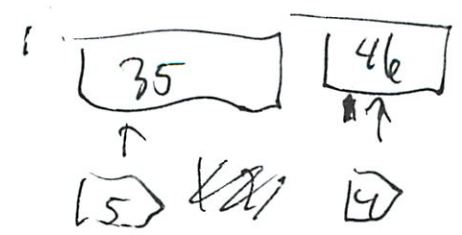
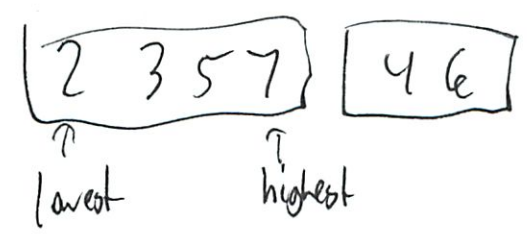
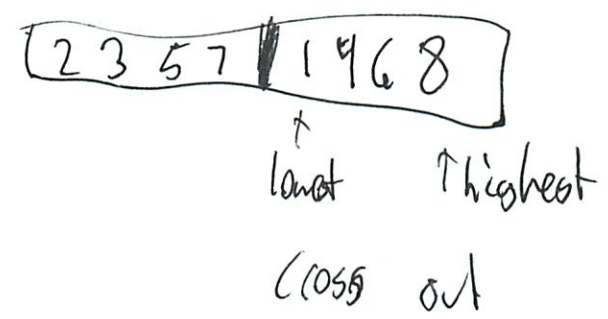
Before we split cards off

Now we are not ⁱⁿ half then finding partner

i from both ends

↳ low and high

look at top + bottom of card pile



16
I like this

but how is this is accessing same memory at once?

I kinda see this is processing both

Where as before we give each their
own set of responsibility

But also when only 1 left

5

↑↑
highest
lowest

for odd #

That we can't have

Then $T(n) = PM\left(\frac{\lg n}{2}\right) + \Theta(1)$

$$n^{\lg 2} \text{ vs } 1$$

$$n^0 \text{ vs } 1$$

$$1 \text{ vs } 1$$

$$1 \lg(n) \text{ @}$$

not totally sure

(17)

But how find min, max?

Well sorted

and we store offset in

That way don't have to adjust (cross out)

(18)

12/4
2:00P
OK

4) both sides at once (my sol)

Still half n

Winda ~~can~~ both sides

TA but can't be linear

try to relate #s in list to others
not just all

? TA similar to the book example

Their model: only 1 can read mem

Here all n processors can read mem

~~Edge case~~

Book: Tear paper in half

Only 1 person can hold a half at once

Here: Each person has photocopy/Google Doc
? real time

19

How many processors in book?

Why $\log_2 n$?

Since gives chunk to each processor

Each spans 2

↳ don't care distribution that reuses

So $\log n$ tree
height

each processor $\log n$

Since binary search

and $\log n$ of them

$\log^2(n)$

Which $\log n$ to get rid of?

binary search - allows to keep splitting in half
split off

(13) (20)

Reason we split in half is b/c mem can't be shared

? So that must be hint

So assign processor for each value
Binary search puts them all together

Know where is in your list
is binary sorted

Take list 1 have median

Do Binary search in list 2

So know where put in final list

Put it together

Already sorted so median b/w
do for each #

Q (21)

All n binary search
going ~~where~~ in parallel $\log(n)$

Doing index

Not search for the ones in the gap

Why is ok to divide up work

Not waiting

how how many elements be smaller in 2nd list

so know index of final

so 7th item

Do for every item

↳ in each list

how many under me in this list
in other list \rightarrow binary search

15/22

$\lg(n)$ parallelized

$n \lg(n)$ non parallelized

work $2n \lg n =$ non-parallelized

span $\lg n =$ parallelized

(write up)

12/5
8p

off by 1?

0	1	2	3	4	0	1	2	3	4
0	3	5	7	9	1	2	4	6	8

0	1	2	3	4	5
0	1	2	3	4	

? ? ? ? ? ?
0+0 1+0 1+1 1+2 2+2 2+3
1+0

? there
would be
if inserted etc

Michael Plasmier
6.046 R07

P-set 6 #1
OH TA

One-Way Functions

Given an output can you easily find an input that produces that output?

a) $f(x, y) = x \oplus y$

~~Yes~~ (No) This is the quintessential reversible function.

$$\underbrace{(\text{key} \oplus \text{Value})}_{\text{output}} \oplus \text{key} = \text{Value}$$

That's not quite why
It's easy to go bit by bit and
see what input you need for
a given output.

②

b) $f(p, q) = pq$ where p, q are prime

(Yes.) Factoring primes is difficult to do.

The best known algorithm is in
 $O(2^{(\log N)^{1/3}} (\log \log N)^{2/3})$ time

Which is on pseudo-polynomial in $\log N$
The number of bits required to represent N .

③

$$c) f(x) = \log_g(x) \bmod p$$

p = fixed prime

g = fixed generator of \mathbb{Z}_p^*

So ~~the~~ $f(x)$ is our output

Can apply log rule

$$g^{f(x)} \bmod p = x$$

Here we can easily find x , given $f(x)$

(No)

Michael Plasmier
6.046 R07

P-set 6 #2

Find d , w/ $n=55$ $e=7$

1. Factor $n=55$

Try all values $2 \dots \sqrt{55}$

Get 5, 11 as p, q

2. Calculate ϕ as $(5-1)(11-1) = 40$

3. Verify $\gcd(7, 40) = 1$ ✓

4. Calculate $7^{-1} \bmod 40$ as 23,

5. $d = 23$,

Michael Plasmore
6.046 R07

P-set 6 #3

RSA Walk through

$$p = 2357$$

$$q = 2551$$

$$e = 3674911$$

~~XXXXXXXXXX~~

1. Compute $n = pq$

$$n = 6012707$$

2. Compute $\phi = (p-1)(q-1)$

$$\phi = 2356 \cdot 2550 = 6007800$$

3. Verify $\gcd(3674911, 6007800) = 1$ ✓

4. Compute

$$d = e^{-1} \bmod \phi$$

$$= 3674911^{-1} \bmod 6007800$$

$$= 422191$$

②

a) $P = (e, n)$

$$= (3674911, 6012707)$$

b) $S = (d, n)$

$$= (422191, 6012707)$$

c) Encrypt $m = 5234673$

$$C = M^e \mod n$$

$$= 5234673^{3674911} \mod 6012707$$

$$= 3650502$$

d) Decrypt

$$M = C^d \mod n$$

$$= 3650502^{422191} \mod 6012707$$

$$= 5234673 \quad \checkmark$$

Michael Plasmeyer
6.046 R09

P-Set 6 #4
OH TA

Parallel Merging

On each processor

For each element, the

look up current list position

just read list offset

do a binary search in other list

read list offset for where it would be
if it was in that list. ie 3

i 0 1 2 3 4
value

1	2	4	6	8
---	---	---	---	---

3 would be at position 2

add these 2 values together

place in master sorted list at that offset

②

This is $\Theta(\log n)$ to do the binary search
but all at once \rightarrow so $\Theta(\log n)$ parallelized

would be $\Theta(n \log n)$ unparallelized

Problem Set 6 Solutions

This problem set is due at **11:59pm** on **Thursday, December 06, 2012**.

Exercise 5-1. Do Exercise 31.7-1 in CLRS on page 964.

Exercise 5-2. Do Exercise 31.7-3 in CLRS on page 965.

Exercise 5-3. Do Exercise 27.2-3 in CLRS on page 796.

Problem 5-1. One-way functions

Circle the functions that are likely to be one-way. Explain why or why not. State your assumptions clearly.

1. $f(x; y) = x \oplus y$.

2. $f(p; q) = pq$ where p and q are prime.

3. $f(x) = \log_g(x) \bmod p$, where p is a fixed prime and g is a fixed generator of Z_p^* . g is a generator of Z_p^* if the order of $g \bmod p$ is $p-1$. The least positive x such that $a^x \equiv 1 \pmod{p}$ is called the order of $a, \bmod p$.

Solution:

The first function is not one-way because, given $z = f(x; y)$ it is easy to generate a y_0 for any x_0 such that $f(x_0; y_0) = z$. The second function is hard to invert under the assumption that integer factorization is hard, as long as it is only defined over prime numbers. The third function is not one-way because it is easy to exponentiate modulo p . Note: the inverse of exponentiation, computing discrete logarithms, is hard.

Problem 5-2. RSA: finding d

For the RSA cryptosystem, suppose the modulus is $n = 55$. If the encryption exponent $e = 7$, what is the decryption exponent d ? Show your work.

Solution: $n = 5 \cdot 11$, so $\phi(n) = 4 \cdot 10 = 40$. Then $d = e^{-1} = 23 \bmod \phi(n)$, because $7 \cdot 23 = 161 = 1 \bmod 40$.

Problem 5-3. RSA: walkthrough

For this problem you will determine public and private keys for the RSA cryptosystem and encrypt and decrypt a message according to RSA. You can use Wolfram Alpha web resource for computational help (<http://www.wolframalpha.com/>). You should write down all the computational steps required in each of the parts below.

Entity A chooses the primes $p = 2357$, $q = 2551$, and chooses $e = 3674911$. Answer the following questions.

1. What is A 's public key?
2. What is A 's private key?
3. What is the ciphertext corresponding to $m = 5234673$?
4. Show the steps in decrypting the ciphertext generated in Part 3 above to obtain m .

Solution:

A computes $n = pq = 6012707$ and $\phi = (p - 1)(q - 1) = 6007800$. A chooses $e = 3674911$ and, using the Extended Euclidean algorithm, finds $d = 422191$ such that $ed \equiv 1 \pmod{\phi}$. A 's public key is the pair $(n = 6012707, e = 3674911)$, while A 's private key is $d = 422191$.

Encryption: To encrypt message $m = 5234673$, B uses an algorithm for modular exponentiation to compute

$$c = m^e \bmod n = 5234673^{3674911} \bmod 6012707 = 3650502$$

and sends this to A .

Decryption: To decrypt c , A computes

$$c^d \bmod n = 3650502^{422191} \bmod 6012707 = 5234673$$

Problem 5-4. Parallel merging

Design a parallel algorithm that uses $2n$ processors to merge two sorted lists, each of length n , in $O(\log n)$ parallel steps. Assume there is a common shared memory such that any number of processors can access any location in memory on a read at the same time. Assume also that all the elements in the lists are all distinct.

Note: a parallel algorithm that uses $2n$ processors to merge two sorted lists, each of length n , in $O(\log^2 n)$ parallel steps is given in Chapter 27.3 of CLRS. The model of CLRS is more restrictive since more than one processor may not access the same memory location in the same parallel step.

Solution: Let's call the two input lists list A and list B , and output list C .

Have a processor consider one element in list A . A processor that gets assigned $A[i]$ will do a binary search on list B to find index j such that $B[j] < A[i] < B[j+1]$. $A[i]$ has $i-1$ numbers in list A

and j numbers in list B that are smaller than itself. The rank of $A[i]$ in the merged list is $i+j$ and all numbers are distinct, so $A[i]$ can be placed in $C[i+j]$. This process has $O(\log n)$ complexity. Now assign $2n$ processors to each work on one of $2n$ elements in A and B . Since multiple processors can read from shared memory simultaneously, the $2n$ processors can all work in parallel and finish the merging process in $O(\log n)$ time.

L23 Compression

12/6

- Intro

- Lossless Compression

- some benchmarks + high level ideas
- RLE, Huffman, Lempel Ziv, graphs

- Lossy Compression

- images / movies / music
- maintain a set
Bloom Filters

Pretty high level introduction

Lossless

- can get data back perfectly

Lossy

- can't get all info back
- so eye/ear can't tell the difference

②

How do we store big data?

So that it is easy to manipulate + process!

But also that it's small

In general \rightarrow we can't

Can't compress every m -bit string to something smaller than $(\leq m-1)$ bit strings

$(m \text{ bit}) \rightarrow (m-1 \text{ bit})$ (\times) No

Must have a 1:1 mapping

Since 2^m strings and $\underbrace{2^{m-1} + 2^{m-2} + \dots}_{< 2^m}$

No 1:1 mapping possible!

But may be able to compress certain strings

Actually less than half the strings are compressible

Deciding if compressible is hard \rightarrow ~~known~~ P.805

(3)

But we know some English strings uncommon

So we use larger strings to encode them

~~Data represents enough info for task, but not everything~~

Lossless

Run-Length Encoding

0000001110001100000000000

So 6x0s, 3x1s, 3x0s, 2x1s, 8x0s

So store 6, 3, 3, 2, 1

Fax machines do since often long runs of 0s

Also used in JPEG - substrate

But w/ very short runs may actually make file bigger

Huffman Coding

a →

b →

c →

d →

e →

build tree

← most common, so short

(4)

(missed)

Lempel-Ziv

Use pointers to previous places where
can substring

ie $(10, 20)$ = copy next 20 strings from
location 10

Goal when lots of similar repetitions in file
details are not on final

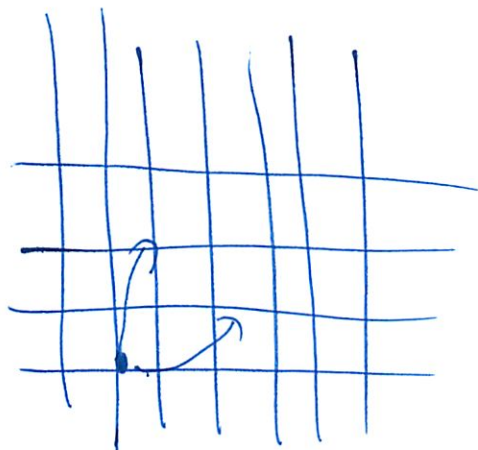
Compressing web graphs

Often sites are similar
Just diff links

So save canonical website
and just change the diffs

Could use small ints to represent offsets

5



represent starting pt ^{could be as big as n}
and offsets

Some small constants

Images/Movies etc

Drop high order Fourier coefficients

Drop undetectable info

Lossy compression & storing sets

losing info slows ya down
doesn't kill ya

6

Bloom Filters

- support membership queries
- saves space (+)
- but false positives (-)

Maintaining a Set

Given $S = \{x_1, \dots, x_n\}$ $x_i \in D$

We have static case, where S is
given in init

- insertions not a problem - not now
- deletion not supported

Goal: membership queries $\{x\}$
is $y \in S$

Desire

- small space
- fast query time

So We're Ans

bitmap hash Fn

Array A of length $|D|$
 A_i if $i \in S$ ↳ space

(7)

Lookup: $O(1)$ by array index
Space $O(D)$ really sucks

• Another solution

Write x_i 's in sorted order

$$x_{i1} \leq x_{i2} \leq \dots$$

So do binary search

~~Space~~ Lookup: $O(\lg n)$

Space: $O(n \log d)$

↳ didn't hear why (review)
~~space~~ $d = \text{dictionary size}$

Can't do better than this w/o losing info
So not that bad

⑧

Can you do better?

If S can be any subset of D

then $2^{|D|}$ subset needs sep encoding

So then need $|D|$ bits

$\uparrow \log$ for 2^0 possibilities

if S ~~to subsets of size at most N~~

Can be any subset of size N

$\binom{|D|}{n}$ subsets of size N

6.042 work

$$\left(\frac{|D|}{n}\right)^n \leq \binom{|D|}{n} \text{ lower bound}$$

Need at least \log of that ~~many~~ many bits

$$\text{at least } \Omega(n \log |D|) \text{ bits}$$

So that is optimal space

(Clever has proved - need to think to do this)

9

What if we allow a small false positive rate?

Then not every subset needs a sep encoding
We are no longer lil

ie if y is in set, must always answer YES

if y is not in set, usually answer No

but might mess up (false positive) + answer YES
for at most 10% of the y 's

Why is this reasonable?

(could have program that tells ya allowable passwords)

S = non allowed passwords (ie dictionary)

Checks that your password is not in this set S

Will always say dictionary words not ok

But sometimes another password might be denied
Even though it's ok

But no blacklisted pw get through

(10)

So how do they work?

What is std way of using algorithm?
(missed)

try 1 hash

To ↓ false positives, we hash h times

You tell it what false pos rate you want

Bloom Filters

Use k hash fns h_1, \dots, h_k mapping

D to $\{1 \dots m\}$

- array of m bits, initialized to 0

Insert Insert x ; for each $1 \leq i \leq k$
Set $h_i(x)$ to 1

Query $x \in S$? Output YES if $\forall 1 \leq i \leq k, h_i(x) = 1$
else output No

10

Never setting $1 \rightarrow 0$

Only $0 \rightarrow 1$

(this is pretty clever - I hadn't thought of!)

Behavior If $x \in S$ then always output Yes

If $x \notin S$ then might be Yes

↳ but that is false positive
keep rate low!

Example

00000000000000000000

insert x_1

get $h_1(x_1)$ $h_2(x_1)$ $h_3(x_1)$

Set those to 1

0100000100001010

Note does not
need to be
in order

insert x_2

$h_1(x_1)$ $h_1(x_2)$ $h_2(x_1)$ and $h_2(x_2)$ $h_3(x_1)$ $h_3(x_2)$
0100010010000101

(12)

Do a query y_1

See $h_1(y_1)$ $h_2(y_1)$ $h_3(y_1)$

all of these must be 1 to
be in ~~the~~ dictionary

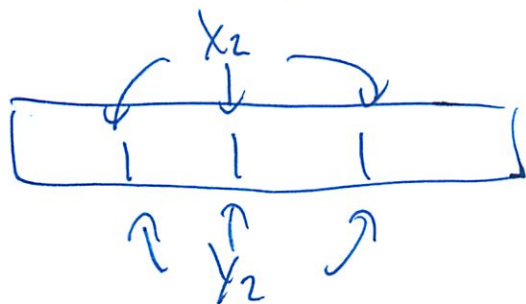
~~No~~ ☒ Correct

y_2

does map ~~the~~ to 1 in each

↳ ☒ Correct

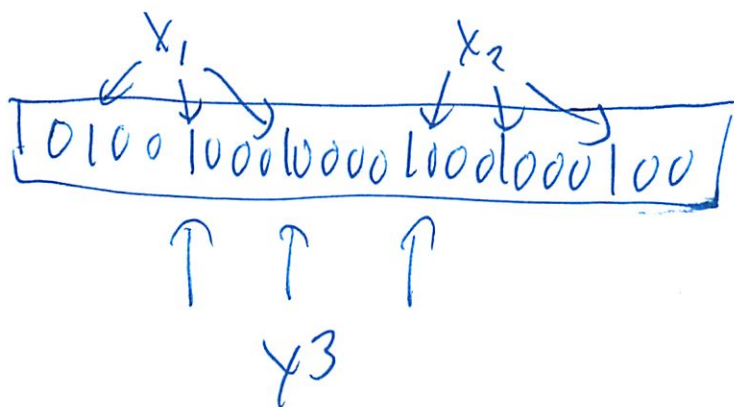
⊛ We can see that maps to same place x_2
But normally don't know that



(13)

But chance for false pos

x_3



Outputs yes but is No
(X) Wrong

What is the false positive rate?

1. Assume $k \ll n$

(goal upper bound on # of 0

bits turned to 1

(need to think more this way!)

2. Assume
Hash functions perfectly random
Not that reasonable of an assumption

(14)

Some calculations

1. After all items get hashed in, what is prob that it will survive + maintain 0

$p' \equiv$ Prob that i th bit of bloom filter is still 0

(note same for all i since random assumption)

$$= \left(\underbrace{\left(1 - \frac{1}{m}\right)}_{\text{survive for fixed } x} \right)^{\underbrace{k}_{\text{for all } k \text{ hash fns}} n}$$

survive for fixed x
for all k hash fns

survive after all n
elements inserted

$$\approx e^{-kn/m} = p$$

good approx ↗ else

15

What is probability we remain 0?

What is the fraction of 0s we have left
After it is done?

Simple Linearity of Expectation w/ Indicator
Variable calculation

2) fraction of 0s after S inserted $= p$

$$p = \frac{1}{n} \sum I_i \quad \text{where } I_i = \begin{cases} 1 & \text{if } i\text{th bit remains 0} \\ 0 & \text{otherwise} \end{cases}$$

(flipped - weird!)

$$E[p] = \frac{1}{n} \sum E[I_i]$$

$$= E[I_i]$$

$$= \text{prob}[I_i = 1]$$

$$= p'$$

$$= e^{-kn/m}$$

(16)

3) What is prob of a false positive?

$$\begin{aligned} & \text{Prob}[y \notin S \text{ if false positive}] \\ &= \text{Pr}[\text{all } h_i(y) \text{'s land on a "1"}] \\ &= (1-p)^k \\ &\approx (1-p')^k \\ &\approx (1-p)^k \\ &\approx (1 - e^{-kn/m})^k \end{aligned}$$

It is known that p is very close to $E[p]$
w/ high prob

I don't need to know why

So what does this give us?

(17)

Minimized false positives w/ calculus

What they got (shipping what they did)

How to pick k ?

large k

⊕ helps find 0 if $y \neq s$

⊖ decreases # of 0 bits in array

So more likely to get a false positive

So there is this tradeoff

No clear ans

Must minimize false positives via derivatives

Find k to minimize prob of false positives
via derivatives

$$\text{gives } k = (\ln 2) \left(\frac{m}{n} \right) \approx 1.7 \left(\frac{m}{n} \right)$$

$$\text{gives vs } p = \frac{1}{2}$$

↳ half 1s half 0s

(18)

Then prob of false positive is $\frac{1}{2^k}$

$$\text{Prob}[\text{False pos}] = \frac{1}{2^k} = (1.6185)^{m/n}$$

So if know what we want (ie $= \frac{1}{10}$)

~~the then picking m~~

Then can solve for m in terms of n
know how much more (missed)

So constant factor more than n

Lossless ^{was} $m \log d$

$$m = O(n)$$

$$k = O(1)$$

Runtime $O(1)$ k is constant

Space $O(|S|)$

(size of dictionary)

(19)

Could we do better?

No - its known
Simple proof

Nice properties

Easy to Union 2 Dictionaries
Just union of bits (AND)

But Deletion + Counting non supported
Some Counting Bloom filters

Can use to spell check

Or store all passwords

Can speed up joins in db

Find which cache has pg

70

Compression

Lots of ways to compress

Takes advantage of special properties of data

Very connected to learnability + randomness

Predicting compressibility

Lecture 23: Compression

- Intro
- Lossless Compression
 - some "buzzwords" → high level ideas
RLE, Huffman, LZ
 - graphs
- Lossy Compression
 - Images, movies, music ...
 - maintaining a set
Bloom Filters

see
slides

← these notes

Maintaining a set

Given $S = \{x_1, \dots, x_n\}$ $x_i \in D$

Goal a way of storing S'
so that

membership queries?

is $y \in S$?

are supported.

Desire

- small space
- fast query time

A solution:

- array A of length $|D|$
 $A_i = 1$ if $i \in S$
0 o.w.

Space = $|D|$

query time = 1

← can be huge!

← best possible

Another Solution:

- write X_i 's in sorted order

$$X_{i_1} \leq X_{i_2} \leq X_{i_3} \leq \dots$$

$$\text{space} = n \log |D| \leftarrow \text{best possible (see below)}$$

$$\text{query time} = O(\log n) \quad (\text{binary search})$$

\nwarrow not terrible, but not great.

Can you do better?

- if S can be any subset of D :

$2^{|D|}$ subsets of D

each needs separate encoding

$$\Rightarrow \log(2^{|D|}) = |D| \text{ bits required to get } 2^{|D|} \text{ possibilities}$$

- if S can be any subset of size n :

$\binom{|D|}{n}$ subsets of size n

$$\left(\frac{|D|}{n}\right)^n \leq \binom{|D|}{n}$$

$$\Rightarrow \Theta(n \log |D|) \text{ bits required to get } \binom{|D|}{n} \text{ possibilities}$$

allowing errors means not every set
has to have a separate encoding.
Can this help?

But:

What if small false positive rate is ok?

i.e. if $y \in S$, always answer "yes"
if $y \notin S$, answer "no" for most y
but might answer "yes"
(say for 10% of y 's)

Why would this be reasonable?

Example: Dictionaries (Bloom)

• hyphenation of words

easy words: 90%

hard words: 10%

store $S = \text{hard words}$ in compressed way

simple rules \leftarrow (relatively fast
not always correct)
table lookup \leftarrow (slow
always works)

Given word w :

is $w \in S$?

if yes
no

do table lookup
apply simple rules

\leftarrow might do table lookup
even if w
is easy.
costly, but not
a crisis.

Tradeoff: cost of extra table lookups
vs. savings on storing S .

Bloom Filters

Goal: maintain a set $S = \{x_1, \dots, x_n\}$ under inserts + queries

Bloom Filter: uses k hash fctns h_1, \dots, h_k mapping to $\{1..m\}$

- array of m bits
initially all 0

- Algorithm:

Insert x :

For each $1 \leq i \leq k$,
Set $h_i(x)$ to 1

Query $x \in S?$:

Output "yes" if for all $1 \leq i \leq k$
 $h_i(x) = 1$

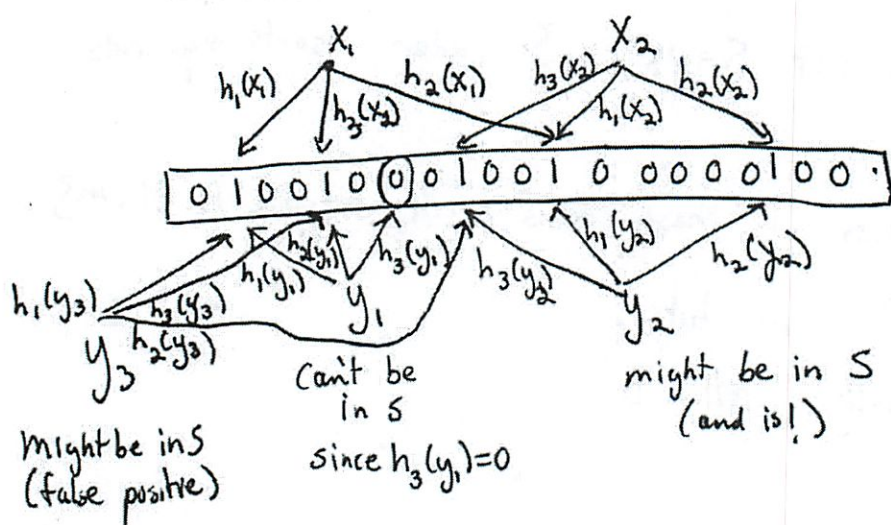
else output "no"

Behavior:

if $x \in S$, then always outputs "yes"

if $x \notin S$, then might output "yes"

"False positive" \leftarrow might be ok
if not too many

Example

$n = \# \text{ elements}$

$m = \text{range of hash fctn.}$

$\equiv \text{size of Bloom Filter}$

$k = \# \text{ of hash fctns}$

What is false positive rate?

Assume 1) $kn < m$

good upper bound on total # of bits set
to 1 for n elements with K
hash fctns

2) hash fctns are perfectly random
(+ independent)

Some calculations:

1) After all S hashed into B.F.:

$p' \equiv \text{prob [ith bit of BF is still 0]}$

$$= \left[\underbrace{\left(1 - \frac{1}{m}\right)^K}_{\substack{\text{prob } x \in S \\ \text{doesn't hash} \\ \text{to } i \text{ under } h_j}} \right]^n \quad \left. \vphantom{\left[\left(1 - \frac{1}{m}\right)^K \right]^n} \right\} \begin{array}{l} \text{prob no } x \in S \\ \text{hashes to } i \text{ under} \\ \text{any } h_j \end{array}$$

$$\approx e^{-kn/m} \equiv p$$

① note: p is a good approx to p'

2) What is $\rho \equiv$ fraction of 0 bits in BF after S inserted?

$$\rho = \frac{1}{n} \sum_i I_i \quad \text{where} \quad I_i = \begin{cases} 1 & \text{if } i\text{th bit of BF still 0} \\ 0 & \text{o.w.} \end{cases}$$

$$\begin{aligned} E[\rho] &= \frac{1}{n} \cdot \sum_i E[I_i] \\ &= E[I_i] \\ &= \Pr[I_i = 1] = p' \\ &\approx \rho \quad \text{from ①} \end{aligned}$$

3) Prob[$y \notin S$ is false positive]

$$= \Pr[\text{all } h_i(y)\text{'s land on a "1"}]$$

$$= (1-p)^k \quad \searrow \text{②}$$

$$\approx (1-p')^k \quad \searrow \text{①}$$

$$\approx (1-p)^k$$

$$= (1 - e^{-kn/m})^k$$

• How should we pick k ?

- more hash fctns gives a tradeoff

⊕ helps find a 0 for $y \notin S$

⊖ decreases number of 0 bits in array
 \Rightarrow more likely to get false positive

- find k to minimize prob of false positive
 via derivatives

gives $k = (\ln 2) \left(\frac{m}{n}\right) \approx (0.7) \left(\frac{m}{n}\right)$

$$p = 1/2$$

$$\text{prob [false positive]} = \left(\frac{1}{2}\right)^k \approx (0.6185)^{m/n}$$

note: if need false positive rate $<$ some
 constant, then $m = O(n)$ is sufficient!
 $k = O(1)$

$$\Rightarrow \text{Space} = O(|S|)$$

$$\text{lookup time} = O(1)$$

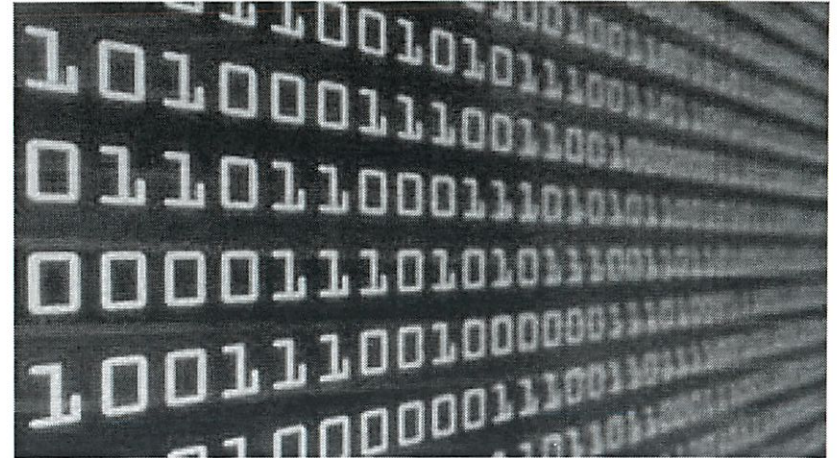
• Can we beat Bloom Filters?

Known fact: any scheme with $\leq \epsilon$ false positive rate needs
 $m \geq n \log 1/\epsilon$

B.F. uses $m \geq (1.44) n \log 1/\epsilon$

Lecture 23: Compression

How do we store big data?



How do we represent data?

- So far:
 - So that it is easy to manipulate/process!
- Another consideration:
 - short

Today's
goal!

How can we compress data?

- Bad news: In general, we can't ... not even by one bit:
 - To represent *every* m -bit string with at most $(m-1)$ bits: each m -bit string should be represented by a different $(\leq m-1)$ -bit string
 - There are 2^m m -bit strings,
 - But there are only $2^m - 1$ different $(\leq m-1)$ -bit strings!

What if we know *something more* about our data?

- Some examples:
 - Frequency of characters, certain sequences differ
 - E.g. Some English letters are uncommon, so use larger strings to encode them
 - Data represents certain type of object – take advantage of characteristics of the object
 - Picture, movie, music
 - Set
 - Graph

Lossless compression:
a few words

Data compression: two paradigms

- Lossless:
 - Can get back the original sequence perfectly
 - Run-length encoding
 - Huffman-coding
 - Lempel-Ziv
- Lossy:
 - Keep enough information for the task, but not everything
 - Pictures, video, music,: .jpg, .mpg, .mp3
 - Wavelets
 - Bloom filters

Lossless compression:
a few buzzwords
and very high level ideas

Run length encoding

- Given bits:
000000111000110000000000
- Encode by lengths of successive runs:
6 0's, 3 1's, 3 0's, 2 1's, 9 0's
(Actually, need only store vector 6,3,3,2,9)

Great for fax machines!

Also used in jpeg

Lempel-Ziv

- Lempel Ziv
 - Use pointers to previous places where saw the substring
 - i.e., “copy the string starting at location 10 for the next 20 letters” --- works well if cheaper to represent (10,20) than the values of 20 letters.
 - Many variants (LZ77, LZ78, LZW...)

Huffman coding

- Variable length encoding –
 - Use shortest strings to encode most frequent letters
 - E.g. Use 3 bits to encode “a” and “e”, 4 bits for “f”, “s”, “m” and 5 bits to encode “z”, “x”
 - Need prefix-free property
 - Bad if $a \rightarrow 00$, $b \rightarrow 111$ and $z \rightarrow 00111$
Is “00111” decoded to “ab” or “z”?

See CLRS Chapter 16.3 for more details!

Compressing graphs

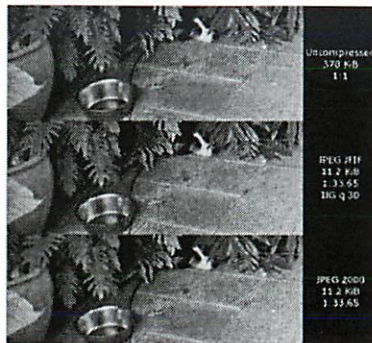
- Adjacency matrix: n^2 bits
- Adjacency list: $m \log n$ bits
- In general, can't do much better
 - There are $2^{\frac{n^2}{2}}$ many graphs, each needing a distinct representation, so need $\frac{n^2}{2}$ bits
 - Can also give counting argument for graphs with m edges

Compressing Web graphs

- Many nodes have similar links
 - Can express u's node as "copy v's links and make the following modifications"
- Destinations of links exhibit locality
 - Can use small integers to express destinations relative to source of link
 - e.g. grid graph + short edges:
 - nodes are pairs (x,y)
 - (x,y) connected to $(x \pm 1, y \pm 1)$
 - (x,y) also connected to $(x+d, y+e)$ for constant d,e – can represent edge by $\log d + \log e$ bits instead of $2 \log n$ bits.

Images/Movies/...

- Store via Fourier representation
 - Do we really care about the "high order Fourier coefficients"?
 - Drop undetectable information....



Lossy compression

Lossy compression: storing sets

Bloom filters

- Data structure for representing a set to support membership queries
 - (+) Save lots of space
 - (-) false positives

A nice property:

- union of two sets represented by Bloom filters of size m (same hash functions)
 - Take “or” of the bits

Bloom filter

- Goal: Maintain set $S = \{x_1, x_2, \dots, x_n\}$ of n elements
- Bloom filter:
 - array of m bits
 - Given k independent hash functions h_1, \dots, h_k mapping x_i 's to $\{1, \dots, m\}$
 - Storage Algorithm:
 - initially all m bits are 0
 - For each $x_i \in S$, for each $1 \leq j \leq k$, set bit $h_j(x_i)$ to 1
 - On query y :
 - For each $1 \leq j \leq k$, check if bit $h_j(x_i)$ is 1
 - If yes for all j , output “in S ”, else output “not in S ”

Other features?

- Deletions?
- Counting?

Applications of Bloom Filters: Dictionaries

- Unix spell-checker:
 - Store dictionary in BF
 - While spell-checking, look up each word in BF
 - False positive causes you to ignore misspelled word
- Unsuitable passwords:
 - Store dictionary + words of edit distance 1 in BF
 - Don't allow passwords that seem to be in BF
 - Might not allow you to use a perfectly good password

Other applications:

- Databases: speed up semi-join
- Distributed caching: find which (if any) of cooperating caches holds a web page
- P2P networks: locate objects
- Network routing
- And many many many more...

Compression

- Lot's more!
 - Ways to compress
 - Applications
 - Better storage
 - Cheaper communication
 -
 - Maybe surprisingly:
 - Connected to learnability, randomness

Recitation 1

12/7

(Missed due to travel)

Recitation 11: Graph Diameter Testing and Bloom Filters

1 Graph Diameter Testing

1.1 Definitions

Diameter: The diameter D of an undirected, unweighted graph $G = (V, E)$ is the max shortest distance between 2 vertices.

ϵ -close to diameter D : A graph $G = (V, E)$ is ϵ -close to having diameter D if adding or removing up to ϵn^2 edges can transform it into a graph G' that has diameter D .

C -neighborhood: The C -neighborhood of a vertex v is the set of all vertices u that can be reached from v in at most C hops. We call $\{v\} \cup v$'s neighborhood a **ball** and call v the center. A vertex u' is in the **C -boundary** of a ball if it is at a distance of at most C from any vertex in the ball. This also means the distance of u' to the center v is at most $2C$.

1.2 Theorem

We wish to check if a graph $G = (V, E)$ has diameter at most D with boundary $\beta(D) = 4D + 2$. That is, we wish to accept *all* graphs G that have diameter D and wish to reject graphs that are ϵ -far from having diameter $\beta(D)$ with probability at least $2/3$. Graphs are ϵ -near to having diameter $\beta(D)$ but with diameter greater than D may be accepted or rejected.

1.3 Algorithm

1. Select S vertices
2. For each vertex, perform BFS, terminating the search when:
 - (a) k vertices have been reached. If so, go on to next vertex.
 - (b) all vertices in the D -neighborhood have been reached. If so, reject.
3. Accept if all vertices have a D -neighborhood of at least k .

Where $S = \frac{4n}{\epsilon m}$ and $k = \frac{2n}{\epsilon m}$. We can assume that $m \geq n - 1$ since otherwise the graph is disconnected and has an undefined diameter and may be rejected without testing.

1.4 Proof

To help prove this algorithm upholds the theorem statement, we first prove two lemmas.

1.4.1 Lemma 1

If the C -neighborhood of each vertex in G contains at least k vertices, then the graph G can be transformed into a graph G' with diameter at most $4C + 2$ by adding at most $\frac{n}{k}$ edges.

1.4.2 Proof of Lemma 1

Cover the graph G in disjoint balls of size C . This can be done by:

1. Pick a vertex v and find its C -neighborhood.
2. Repeat by only picking v that are not in the C -boundary of any existing ball.
3. Stop when you can no longer pick any vertex.

There are at most $\frac{n}{k}$ balls as each ball has at least k vertices. Connect the center of all balls to the first ball created. This adds at most $\frac{n}{k} - 1$ edges.

Each vertex u is in the C -boundary of some vertex v . This means that it is at most $2C$ from a center v . Let u_1 and u_2 be the two vertices that have the max shortest path. The longest this path can be is $2C$ from two centers v_1 and v_2 . The two centers are at most 2 hops away in the modified graph because all centers are connected to the first center. Therefore, this path is at most $4C + 2$.

1.4.3 Lemma 2

If the C -neighborhood of at least $(1 - \frac{1}{k})n$ vertices contain at least k vertices, then the graph can be transformed into a graph with diameter $4C + 2$ by adding at most $\frac{2n}{k}$ edges.

1.4.4 Proof of Lemma 2

This follows closely from Lemma 1, but don't take any centers if there are fewer than k vertices in the C -neighborhood. There are at most $\frac{n}{k}$ balls. There are at most $\frac{n}{k}$ vertices that are not in any ball or C -boundary of any center (the ones that don't have a C -neighborhood of size at least k).

Connect the center of all balls to the first ball's center as in Lemma 1. Connect all vertices not in any ball or C -boundary to the first ball's center as well. This adds at most $\frac{2n}{k} - 1$ edges to the graph. Similarly to above, the transformed graph's diameter is $4C + 2$.

1.4.5 Final Proof

If the graph's diameter is at most D , the algorithm will accept. What remains is that we have to reject graphs that are ϵ -far from diameter $\beta(D) = 4D + 2$ with probability at least $2/3$.

We call the vertices that have D -neighborhoods of size less than k bad. If there are at most $\frac{n}{k}$ vertices, we can transform it into a graph with diameter $4D + 2$ by Lemma 2. Graphs that are ϵ -far from having diameter $\beta(D)$ or less have to have more than $\frac{n}{k}$ bad vertices.

The chance that we only select good vertices when dealing with a graph that is ϵ -far from having diameter $\beta(D)$ is

$$(1 - \frac{1}{k})^S = (1 - \frac{1}{k})^{2k} < e^{-2} < \frac{1}{3}.$$

Therefore the chance of selecting at least one bad vertex is greater than $2/3$, which means the algorithm will reject with probability at least $2/3$.

1.4.6 Analysis of Runtime

This algorithm picks S vertices and runs a BFS for k steps for a total time of $O(Sk) = O(\frac{8n^2}{\epsilon^2 m^2})$. Since $m \geq n - 1$, the runtime is equivalent to $O(\frac{1}{\epsilon^2})$.

2 Bloom Filters

2.1 Distributed Cache

There are two servers 1 and 2, each of which has a cache that can fit 100 files. There is a third, larger server (server 3) that has all possible files. It costs 1 to send a packet between server 1 and 2. It costs 2 to send a packet between servers 1 and 3 or between 2 and 3.

- It takes 1 packet to send a request for a file
- It takes 1 packet to receive a file.
- It takes 1 packet to send a request for a Bloom filter.
- It takes 1 packet to receive a Bloom filter.

Suppose every second, server 1 wants to request 10 files and that server 2 has 4 of them.

2.1.1 Scheme 1: Just query the server 3

This would take 2 packets per request and response for a total of 2 packets \cdot 2 cost / packet \cdot 10 = 40.

2.1.2 Scheme 2: Query server 2, then server 3

It takes 10 packets to query whether each file is on server 2 and gets 4 packets back in response for a total of 14 between server 1 and 2. It then queries server 3 for the remaining 6 files for a cost of 24. This has a total cost of 38.

2.1.3 Scheme 3: Use Bloom filters

Server 1 asks 2 for a Bloom filter and only queries server 2 for the necessary files. It takes 1 packet to request the Bloom filter and 1 packet to receive it for a total cost of 2. It then requests and gets the 4 files from server 2 for a cost of 8. It then queries server 3 for 6 files, which still costs 24 for a total cost of 34 cost.

2.1.4 Scheme 4: Continuous update

Rather than having to query for what files are on each server, server 1 and server 2 continuously send each other updates as to what files they have. If each server was to send the entire list of files it had, it would cost 100 each time a server wished to update the other if it sent file names. If the server instead sent Bloom filters, it would only cost 1.

2.2 Bloom Filter as dictionary

We wish to augment the Bloom filter to provide the dictionary function. You have a set S of n key-value pairs (x_i, v_i) where $v_i \in \{0, 1, 2\}$, and support insertions and queries. Each key x_i appears at most once.

The query we wish to answer is given x , if x is a key of some pair in S , return its value with probability $1 - 2\epsilon$. Otherwise, report anything.

We can do this using 3 Bloom filters, one for each of the v_i values. That is, let B_0, B_1, B_2 be Bloom filters. B_j will store the keys x_i if the pair (x_i, j) is in the set S .

If the value is only in one Bloom filter, we will return the right value. However, if x appears in multiple Bloom filters, we randomly return one of the values.

If each Bloom filter is $m = n \log(1/\epsilon)$, then each Bloom filter has a chance ϵ of returning a false positive. If x is in fact a key in S , there is a 2ϵ chance of it returning as a false positive in a Bloom filter it is not actually inserted into since each of the other Bloom filters has a ϵ chance of giving a false positive for x .

Thus, we have an augmented Bloom filter that takes $O(n \log(1/\epsilon))$ bits and reports the right value given that x is a key with probability $1 - 2\epsilon$.

2.2.1 Modification

What happens if instead of using 3 Bloom filters, we only use two Bloom filters B_1 and B_2 ? If a key is not found in any Bloom filter, the data structure returns the value 0 instead.

The data structure will require less space. It will also return the correct value for keys mapping to 1 or 2 with probability $1 - \epsilon$ instead of $1 - 2\epsilon$, however, for keys mapping to 0, the probability it returns the correct answer is still $1 - 2\epsilon$.

Interactive

Proof + Zero

knowledgeProofs

- Graph non-isomorphism
 - defs of interactive proofs, zero-knowledge
 - Graph isomorphism
- Do case evals!
-

Complexity theoryEfficiently solvable P" verifiable NP

↳ if we are handed a proof

BPP - randomized P

or

But is there a randomized efficiently verifiable
(randomized NP)

②

Questions

$$P = NP?$$

$$P = BPP?$$

Is randomness necessary for efficient solvability

Can we do something deterministic

that is the same as randomly

has not been solved

Previous example fell

Randomness affects how we efficiently
verify proofs

(see slides)

(3)

Verifiable: don't need to know time to create proof

Alice is not at all time bounded

Bob can only work in poly time

Bob checks to accept/reject

Graph isomorphism

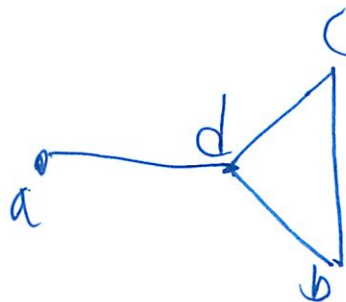
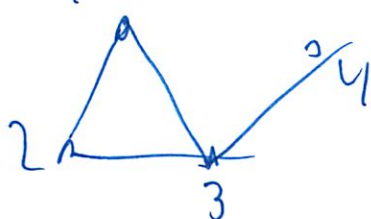
$G \cong G'$ if bijection $V \rightarrow V'$

such that (u, v) in G iff $(\phi(u), \phi(v))$ in G'

rename
nodes

w/ our renaming fn ϕ

ex



ϕ : $\begin{matrix} 1 \rightarrow c \\ 2 \rightarrow b \\ 3 \rightarrow d \\ 4 \rightarrow a \end{matrix}$

could have switched

Q4

Needs to preserve edges + non edges

Just relabel vertices

$(1, 4)$ in G and $(1', 4')$ in G'

$(1, 3)$ not in G and $(1', 3')$ not in G'

Given 2 graphs are they isomorphic?

Not known to be ~~NP~~ P

Lots of heuristics

But no good way to solve

known to be in NP

Not known in Co-NP

(missed why)

How would you prove there is no isomorphism

Not known to be NP-complete

But we don't think it is

5

Don't know how to prove they are not isomorphic:

How could Alice convince us they are not isomorphic
w/o trying all possible φ

The 70s

no Internet

no cable TV

everyone watched the same 3 TV channels

and the same ads

The Pepsi Challenge

Here i modified:

Given 1 cup (Coke or Pepsi)

Is it Pepsi?

If you get it k times in a row
then we'll believe you

⑥
If you could tell diff
get it right every time

If you can't get each right $\frac{1}{2}$ time
So prob get it right k times in a row $(\frac{1}{2})^k$
~~1/2~~

You either know or you don't!
↳ deterministic

So if get k times in a row you either
get it right, or really, really, really lucky

So how can we use this to have Alice prove
 G, G' are isomorphic or not

Assume we can shuffle the graph (random permutations)
↳ picks random ordering of $1 \dots n$: y_1, \dots, y_n

(7)

Alice we relable the vertices of G
w/ the new names

node i in $G \rightarrow$ node y_i in G'

node (i, j) in $G \rightarrow$ edge (y_i, y_j) in G'

Interactive Proof

For $G_0 \neq G_1$

Alice can tell them apart since
she can do ∞ calculations at once

Then I give her another random ~~one~~ shuffle
If she says they are the same \rightarrow then they are

P-time Bob

Pick random bit $\in \{0, 1\}$ \leftarrow coin or penny

Pick random shuffle γ

Sends $\gamma \circ G_F$ \rightarrow (Alice)

⑧

Alice should know which one we sent

Should be isomorphic to G_1 , not G_0

↳ Alice will know which one it is isomorphic
ie which one it came from

(not sure if I heard this right)

Bob

← $b = \text{guess for } c$

Alice

Bob checks if $b = c$

if not fail + halt

if so, repeat whole thing k times

Why does this work?

If $G_0 \not\cong G_1$: $\forall c, G_c \cong G_0$ or G_1

but not both so Alice knows c
(even as reshuffled)

⑨

But if $G_0 \equiv G_1$ then Alice can guess c w/ prob
at most $\frac{1}{2}$

That means if get it correct ~~after~~ after k times
the prob Alice guesses right each time is
w/ prob $(\frac{1}{2})^k$

Interactive Proofs

Don't need to write down each possibility

Since randomness

and the verifier interacts w/ the proof

not something done in ~~advance~~ advance

only tries stuff in response to
one of our queries

Invented in 1985

10

* Note we are not proving graph isomorphism
but non-graph isomorphism *

~~IP~~

Completeness if proposition is true, prover convincing
Verifier to always accept

Soundness if proposition is false true no matter
what (missed, see slides)

Efficient interactive proofs give you all P-space
so more powerful than efficient classical problems

$IP = PSPACE$

Can randomness change how we efficiently verify a
problem?
If can't toss coins $IP = NP$
If prover cons in prob time (see slides)

11

Zero knowledge

Prove theorem to you w/o verifier doesn't learn anything except the statement of theorem

Believe something is true w/o you knowing why

Convince you G, H not isomorphic, but will convince you they are not

Can Bob convince anyone that graphs are non-isomorphic?

Next verifier would prob pick diff random shuffles---

(12)

Each time we send her q to Alice

Since we send $f \circ G_c$

So get little info after that

V doesn't know anything except for truth of statement

(Take ^{Prot} further classes on this)

After interaction V knows

($V = \text{Bob}$)

- history of interactions
- if statement is true

Also given truth of statement V could generate interactions on his own w/ same distribution

Can generate whole distribution of that on its own

Reason Bob could not do at start is he doesn't know if it is actually true or not

(13)

Since earlier we knew the actual correspondence
But could not generate that on your own ...

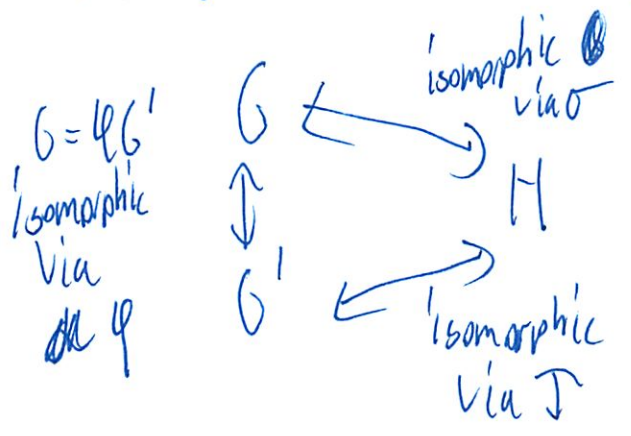
Zero-Knowledge

Interactive Proof for Graph Isomorphism $G \cong G'$

Don't want to see isomorphism

So add 3rd graph H , which is isomorphic to other graph

~~$G \cong G'$ isomorphic via ϕ~~



$$H = \sigma \circ G = \sigma \circ \phi \circ G' = \tau \circ G'$$

So $\tau = \sigma \circ \phi$

(4)

Idea

- Use 3d graph H s.t. $H \cong$ to both
- Convince V that $G + G'$ are both \cong to H
So $G \cong G'$

IP for $G \cong G'$

Alice: picks random σ

$$H \leftarrow \sigma \circ G$$

Sends H to Bob

Bob: picks $b \in \{1, 2\}$ randomly

Alice: If $b=1$, sends σ to Bob, else sends τ to Alice

Bob: Checks the isomorphism \cong if $b=1$ checks that $H = \sigma \circ G$
else checks that $H = \tau \circ G'$

Since if G, G' are isomorphic, is some isomorphism for Alice to send (σ and τ)

(15)

If not isomorphic

Then at most one of them is isomorphic to H

So prob that Bob picks the not isomorphic one
is $\leq \frac{1}{2}$

So we repeat these k times

If $G \cong G'$, both \cong to it so Alice can
always find isomorphism σ or τ

If $G \not\cong G'$, at most one \cong Alice can't
Send isomorphism prob $\geq \frac{1}{2}$

~~Bob picks $\{1, 2\}$ first~~

~~Alice picks correspondence~~

16

Why Zero Knowledge?

Cryptography

Guy tried to teach it to kids
Didn't help

Why 0 knowledge?

Bob can't figure
(missed, see slides)

If one way fns exist ~~there~~ exists a 0 knowledge
IP for any IP problem

Philosophical: Can efficiently prove some things that
are not efficiently provable w/ classical proofs

Practical Reason: Passwords + identification
Secure protocols

(Hmm how much is 6.858 project like this?)

17

6.046 Conclusion

Take more theory classes
Upper level classes
Grad classes

Good luck on exam!

Lecture 24:

Interactive Proofs + Zero Knowledge

- Graph non isomorphism
- "defs" of IP, zero-knowledge
- Graph Isomorphism

(most of this lecture is on slides)

Announcements

Course Evaluation time!

<https://web.mit.edu/subjectevaluation/>

def.

Random permutation of graph G :

- pick random ordering of $1..n$

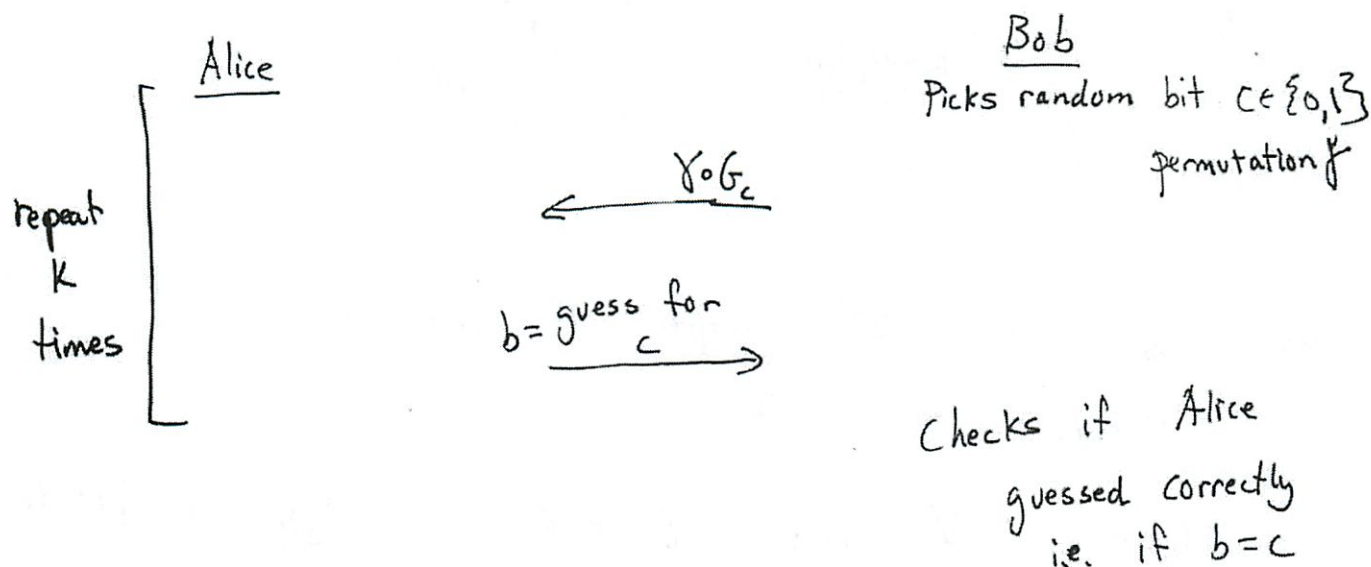
y_1, \dots, y_n

- relabel vertices of G by new names

node i in $G \rightarrow$ node y_i in G'

edge (i, j) in $G \rightarrow$ edge (y_i, y_j) in G'

Interactively Proving $G_0 \neq G_1$:



Why is this good?

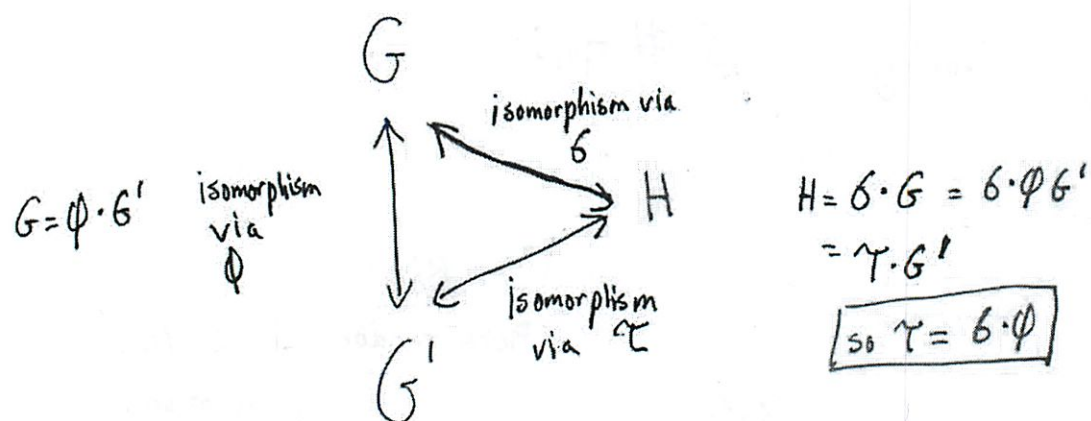
if $G_0 \neq G_1$: ($\gamma \circ G_c$ can only be isomorphic to one of G_0 or G_1)
 $\gamma \circ G_c \cong G_0$ or $\gamma \circ G_c \cong G_1$ but not both

So Alice knows c

if $G_0 \cong G_1$: ($\gamma \circ G_c$ is isomorphic to both of G_0 and G_1)
 Alice can guess c with prob $\leq \frac{1}{2}$

\therefore after K times, Alice correct on each
 with prob $\leq \frac{1}{2^K}$

Interactive Proof Graph Isomorphism of (G, G')



idea . use a 3rd graph H which is isomorphic to both

• Convince V that G & G' are both isomorphic to H

which implies $G \cong G'$

But Careful! can't give both δ & γ away, otherwise, verifier learns $\phi = \delta^{-1} \cdot \gamma$
 ↓ not Zero Knowledge

What to do?

flip a coin + give away either δ or γ
 but not both

Interactive Proof of $G \cong G'$

15

Alice: (prove)

picks random permutation σ

$$H \leftarrow \sigma \circ G$$

sends H to Bob

Bob: (verify)

- picks $b \in \{1, 2, 3\}$ randomly
- Sends b to Alice

Alice:
if $b=1$,
sends G to B

do not
send both!

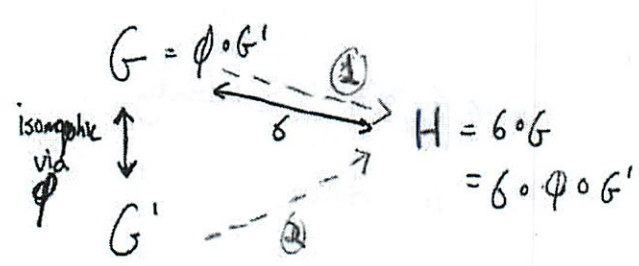
else ($b=2$)
sends $\underbrace{(\sigma \circ G)}_{= H}$ to B

Bob:

if $b=1$
checks $H = G \circ G'$

if $b=2$
checks $H = H \cdot G'$

Interactively Proving Graph Isomorphism of (G, G')



- Present
- (1) δ — ask for isomorphism from H to G
 - or
 - (2) $\delta \circ \phi$ — ask for isomorphism from H to G'

• if $G \cong G'$ then both are isomorphic to H
 i.e. $H = \delta \circ G = \delta \circ \phi \circ G'$

• if $G \not\cong G'$ then at most one isomorphic to H
 A fails with prob $\frac{1}{2}$

Complexity theory

Lecture 24: Interactive Proofs and Zero Knowledge

Efficiently solvable:	P	BPP (randomized P)
Efficiently verifiable:	NP	???

Can randomness change what can (and how to) efficiently verify?

Central complexity theory questions:

- $P = NP$? (whatever can be efficiently verified is also efficiently solvable?)
- $P = BPP$? (is randomness necessary for efficient solvability?)

Today: Randomness affects how we can efficiently verify proofs!

- Interactive proofs
- Zero-knowledge Interactive proofs

Example: n is a product of 2 primes

Alice: Prover

(unbounded computation time)



Bob: Verifier

(poly time)



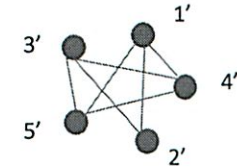
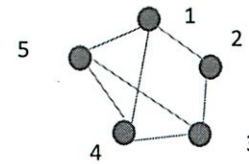
p, q

If p, q prime and $n=pq$ accept
else reject

Graph Isomorphism: How difficult is it?

- Not known to be in P
- In NP (proof = correspondence)
- Not known to be in Co-NP
 - How would you prove there is NO isomorphism?
- Not known to be NP-complete
 - But we don't think it is...

Example: G and G' are isomorphic

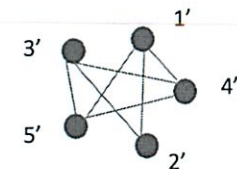
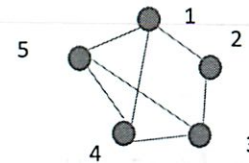


G and G' isomorphic (denoted $G \cong G'$) if there is a correspondence (bijection) $\varphi: V \rightarrow V'$ such that any edge (u, v) in G iff edge $(\varphi(u), \varphi(v))$ in G' .

e.g., $(1,4)$ in G and $(1',4')$ in G'

$(1,3)$ not in G and $(1',3')$ not in G'

Proving that G and G' are isomorphic

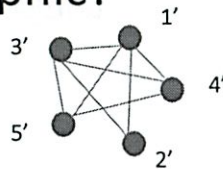
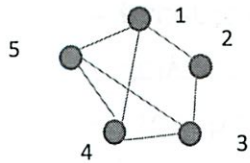


Correspondence φ



If correspondence good,
ACCEPT, else REJECT

How do you prove that G and G' are not isomorphic?



?



Can't try all φ !

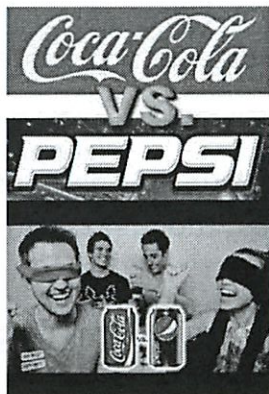
Shortest known classical proof is exponential in n

A quick detour...

- An important prehistorical note....

The Pepsi challenge: (1975)

- Can you tell the difference?



The Pepsi challenge: (1975)

- Can you tell the difference?
- A way to prove it:
 - We give you random samples of Coke and Pepsi
 - If you get it right k times in a row, then we'll believe you
- Why?
 - If you can tell the difference, you get it right every time
 - If you can't tell the difference, you get it right with probability $\frac{1}{2}$ each time, so probability you get it right k times in a row is $1/2^k$
 - i.e., if you get it right k times, you either know or you are really lucky!

Proving that G_0 and G_1 are NOT isomorphic

- Bob (verifier):
 - Flips coin $c \in \{0,1\}$ picks random “shuffle” γ
 - Sends randomly shuffled version of G_c i.e., $H = \gamma \circ G_c$ to Alice (prover)
- Alice (prover):

(Note, if G_0 and G_1 are NOT isomorphic, then H is isomorphic to only one... so Alice can figure out which one was sent)

 - If H isomorphic to G_0 then output $b=0$
 - Else output $b=1$
- Bob (verifier):
 - If Prover gets it right each time “ACCEPT” else “REJECT”

permutation

Interactive Proofs

- As in NP
 - The verifier is polynomial time
 - the prover is “all powerful” (unbounded computation time)
- Two new ingredients:
 - Randomness: verifier tosses coins, can err with small probability
 - Interaction: rather than “reading” proof, verifier interacts with prover

Interactive Proofs

[Goldwasser Micali Rackoff 1985]

- Prover:
 - Knows the proof
 - No run time bounds
- Verifier:
 - Doesn't know anything
 - Probabilistic: can toss coins
 - Polynomial time algorithm
 - Accepts or rejects the proposition



Interactive Proofs

(P,V) is *interactive proof system* for set of theorems L if

1. Completeness:

- If proposition x is true (i.e., $x \in L$), prover can behave in a way that convinces verifier to always accept (with probability 1)

2. Soundness:

- If proposition is false (i.e., $x \notin L$), then, no matter what the prover does, verifier rejects with high probability
- i.e., probability of accept is $\leq \frac{1}{2}$
 - If repeat k times, probability of accept $\leq \frac{1}{2^k}$

Efficient interactive proofs are
MORE POWERFUL
than efficient classical proofs!

The complexity class IP

- Decision problems L such that L has an interactive proof system

IP=PSPACE

- After Graph non-isomorphism, Non-SAT, *number* of satisfying assignments,...
- Thm: IP=PSPACE

Complexity theory

Efficiently solvable:	P	BPP (randomized P)
Efficiently verifiable:	NP	$IP = PSPACE$

Can randomness change what can (and how to) efficiently verify? WE THINK SO!

Remarks

- If verifier can't toss coins, then $IP=NP$
- If prover must run in poly time, then $IP=\text{probabilistic poly time}$

Interactive Proofs

- A third new ingredient
 - Zero Knowledge: verifier doesn't learn anything except for the statement of the theorem



I will not tell you why G and H are not isomorphic, but I will **CONVINCE** you that they are not!

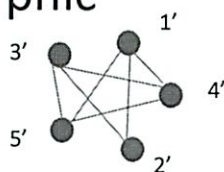
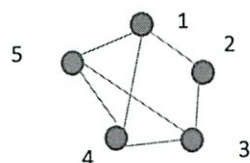
Zero Knowledge of Graph non-isomorphism

- Could Bob convince anyone else that the graphs are non-isomorphic?
 - The next verifier would probably pick different random shuffles...

Zero Knowledge Interactive Proofs

- After interaction, V “knows”
 - Statement of theorem is true
 - History of interaction
- Zero-knowledge: V didn't learn anything except for truth of statement
 - i.e., given truth of statement, V could generate interactions on his own with same distribution
 - A fascinating definition... take more crypto courses!

Back to previous example: G and G' are isomorphic



Correspondence φ



If correspondence good,
Accept, else reject

Verifier learns $G \cong G'$ and *correspondence*

Very high level idea for
proving $G \cong G'$:

- Alice: (knows a correspondence φ s. t. $G = \varphi \circ G'$)
 - Produces a THIRD graph G' which is isomorphic to both G and H ! (randomly permutes G via σ)
 1. This means she can give correspondence from G to H (i.e. $H = \sigma \circ G$)
 2. And a correspondence from G' to H (i.e. $H = \sigma \circ \varphi \circ G'$)
- Bob:
 - randomly decides if Alice should demonstrate 1 or 2
 - Since he only sees one of them, he doesn't actually see the correspondence between G and G'

Interactive Proofs

- Zero Knowledge: verifier doesn't learn anything except for the statement of the theorem



I will not give you the
correspondence, but I
will prove that I could
have if I had wanted
to!

Back to proving $G = \varphi \circ G'$



- Randomly permute nodes of G to get $H = \sigma \circ G (= \sigma \circ \varphi \circ G')$
- Send H to Bob
- Toss coin b and send to Alice
- If $b=0$, send σ (map from G to H)
- If $b=1$, send $\sigma \circ \varphi$ (map from G' to H)
- Check σ or $(\sigma \circ \varphi)$

Why does this work?

- If $G \cong G'$
 - Alice knows φ demonstrating $G \cong G'$
 - Since Alice chose σ , it is no problem to compute $H = \sigma \circ \varphi \circ G'$ and to output σ or $\sigma \circ \varphi$
- If $G \not\cong G'$
 - $H \cong G$ or $H \cong G'$ (or neither, but not both)
 - With probability $\frac{1}{2}$, Alice cannot demonstrate an isomorphism since there is none!

Note: For proving graph isomorphism

- Verifier poly time
- Prover all powerful?
 - Here, Prover only needs to know the correspondence!!!

Why zero knowledge?

- Bob can't figure out φ from σ or $\sigma \circ \varphi$
- But could figure out φ from σ and $\sigma \circ \varphi$
 - So can't repeat k times?
 - Must pick new σ each time!

Which theorems have interactive zero knowledge proofs?

- If one-way functions exist then there exists a zero knowledge interactive proof for any IP problem

Why interactive proofs, why zero-knowledge????

- A philosophical reason:
 - Can efficiently prove statements that are not efficiently provable with classical proofs
- A practical reason:
 - Passwords and identification
 - prove that “I am Ronitt Rubinfeld” so that no eavesdropper can mimic me later
 - Secure protocols
 - prove that I am behaving honestly

True zero-knowledge:

- Quote from a colleague in 1988:
“I explained it [zero-knowledge] to my kids, and they understood!”
 - they know that they didn’t learn anything”

6.046: some final words

- Let’s hope it wasn’t zero-knowledge!
 - nor zero-fun!
- Take more theory classes!
 - Lots of good choices – complexity, crypto, all kinds of algorithms...
- **GOOD LUCK ON THE EXAM!**

Zero-knowledge proof

From Wikipedia, the free encyclopedia

Read 12/13 apt

In cryptography, a **zero-knowledge proof** or **zero-knowledge protocol** is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.

Contents

- 1 Abstract example
- 2 Definition
- 3 Practical example
- 4 Variants of zero-knowledge
- 5 Applications
- 6 History and results
- 7 See also
- 8 Notes
- 9 External links

Abstract example

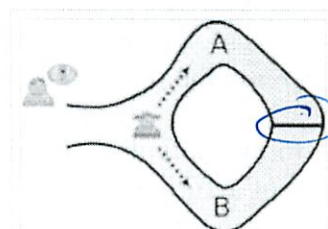
Oh remember this from GRC!

There is a well-known story presenting some of the ideas of zero-knowledge proofs, first published by Jean-Jacques Quisquater and others in their paper "How to Explain Zero-Knowledge Protocols to Your Children".^[1] It is common practice to label the two parties in a zero-knowledge proof as Peggy (the prover of the statement) and Victor (the verifier of the statement).

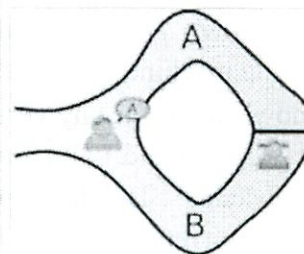
In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a circle, with the entrance on one side and the magic door blocking the opposite side. Victor says he'll pay her for the secret, but not until he's sure that she really knows it. Peggy says she'll tell him the secret, but not until she receives the money. They devise a scheme by which Peggy can prove that she knows the word without telling it to Victor.

First, Victor waits outside the cave as Peggy goes in. They label the left and right paths from the entrance A and B. Peggy randomly takes either path A or B. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Providing she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path. Note that Victor does not know which path she has gone down.

However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path that

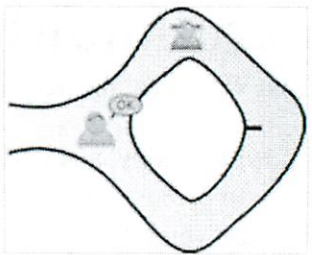


Peggy randomly takes either path A or B, while Victor waits outside



Victor chooses an exit path

she had entered by. Since Victor would choose A or B at random, she would have a 50% chance of guessing correctly. If they were to repeat this trick many times, say 20 times in a row, her chance of successfully anticipating all of Victor's requests would become vanishingly small (about one in 1.05 million).



Peggy reliably appears at the exit Victor names

Definition

Al's Baba's Cave from SW

Aug - remember from summer in LA...

A zero-knowledge proof must satisfy three properties:

- 1. **Completeness**: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
- 2. **Soundness**: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
- 3. **Zero-knowledge**: if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some simulator that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

The first two of these are properties of more general interactive proof systems. The third is what makes the proof zero-knowledge.

What? prover + verifier interact. Prover knows power

Zero-knowledge proofs are not proofs in the mathematical sense of the term because there is some small probability, the soundness error, that a cheating prover will be able to convince the verifier of a false statement. In other words, they are probabilistic rather than deterministic. However, there are techniques to decrease the soundness error to negligibly small values.

A formal definition of zero-knowledge has to use some computational model, the most common one being that of a Turing machine. Let P, V , and S be Turing machines. An interactive proof system with (P, V) for a language L is zero-knowledge if for any probabilistic polynomial time (PPT) verifier \hat{V} there exists an expected PPT simulator S such that

$$\forall x \in L, z \in \{0, 1\}^*, \text{View}_{\hat{V}}[P(x) \leftrightarrow \hat{V}(x, z)] = S(x, z)$$

The prover P is modeled as having unlimited computation power (in practice, P usually is a Probabilistic Turing machine). Intuitively, the definition states that an interactive proof system (P, V) is zero-knowledge if for any verifier \hat{V} there exists an efficient simulator S that can reproduce the conversation between P and \hat{V} on any given input. The auxiliary string z in the definition plays the role of "prior knowledge". The definition implies that \hat{V} cannot use any prior knowledge string z to mine information out of its conversation with P because we demand that if S is also given this prior knowledge then it can reproduce the conversation between \hat{V} and P just as before.

The definition given is that of perfect zero-knowledge. Computational zero-knowledge is obtained by requiring that the views of the verifier \hat{V} and the simulator are only computationally indistinguishable, given the auxiliary string.

Practical example

We can extend these ideas to a more realistic cryptography application. In this scenario, Peggy knows a Hamiltonian cycle for a large graph, G . Victor knows G but not the cycle (e.g., Peggy has generated G and revealed it to him.) Peggy will prove that she knows the cycle without revealing it. A Hamiltonian cycle in a graph is just one way to implement a zero knowledge proof; in fact any NP-complete problem can be used, as well as some other difficult problems such as factoring.^[2] However, Peggy does not want to simply reveal the Hamiltonian cycle or any other information to Victor; she wishes to keep the cycle secret (perhaps Victor is interested in buying it but wants verification first, or maybe Peggy is the only one who knows this information and is proving her identity to Victor).

To show that Peggy knows this Hamiltonian cycle, she and Victor play several rounds of a game.

- At the beginning of each round, Peggy creates H , an isomorphic graph to G (i.e. H is just like G except that all the vertices have different names). Since it is trivial to translate a Hamiltonian cycle between isomorphic graphs with known isomorphism, if Peggy knows a Hamiltonian cycle for G she also must know one for H .
- Peggy commits to H . She could do so by using a cryptographic commitment scheme. Alternatively, she could number the vertices of H , then for each edge of H write a small piece of paper containing the two vertices of the edge and then put these pieces of paper upside down on a table. The purpose of this commitment is that Peggy is not able to change H while at the same time Victor has no information about H .
- Victor then randomly chooses one of two questions to ask Peggy. He can either ask her to show the isomorphism between H and G (see graph isomorphism problem), or he can ask her to show a Hamiltonian cycle in H .
- If Peggy is asked to show that the two graphs are isomorphic, she first uncovers all of H (e.g. by turning all pieces of papers that she put on the table) and then provides the vertex translations that map G to H . Victor can verify that they are indeed isomorphic.
- If Peggy is asked to prove that she knows a Hamiltonian cycle in H , she translates her Hamiltonian cycle in G onto H and only uncovers the edges on the Hamiltonian cycle. This is enough for Victor to check that H does indeed contain a Hamiltonian cycle.

Completeness

If Peggy is honest, she can easily satisfy Victor's demand for either a graph isomorphism (which she has) or a Hamiltonian cycle (which she can construct by applying the isomorphism to the cycle in G).

Zero-Knowledge

Peggy's answers do not reveal the original Hamiltonian cycle in G . Each round, Victor will learn only H 's isomorphism to G or a Hamiltonian cycle in H . He would need both answers for a single H to discover the cycle in G , so the information remains unknown as long as Peggy can generate a distinct H every round. If Peggy does not know of a Hamiltonian Cycle in G , but somehow knew in advance what Victor would ask to see each round then she could cheat. For example, if Peggy knew ahead of time that Victor would ask to see the Hamiltonian Cycle in H then she could generate a Hamiltonian cycle for an unrelated graph. Similarly, if Peggy knew in advance that Victor would ask to see the isomorphism then she could simply generate an isomorphic graph H (in which she also does not know a Hamiltonian Cycle). Victor could simulate the protocol by himself (without Peggy) because he knows what he will ask to see. Therefore, Victor gains no information about the Hamiltonian cycle in G from the information revealed in each round.

Soundness

If Peggy does not know the information, she can guess which question Victor will ask and generate either a graph isomorphic to G or a Hamiltonian cycle for an unrelated graph, but since she does not know a Hamiltonian cycle for G she cannot do both. With this guesswork, her chance of fooling Victor is 2^{-n} , where n is the number of rounds. For all realistic purposes, it is infeasibly difficult to defeat a zero knowledge proof with a reasonable number of rounds in this way.

Variants of zero-knowledge

Different variants of zero-knowledge can be defined by formalizing the intuitive concept of what is meant by the output of the simulator "looking like" the execution of the real proof protocol in the following ways:

- We speak of *perfect zero-knowledge* if the distributions produced by the simulator and the proof protocol are distributed exactly the same. This is for instance the case in the first example above.
- *Statistical zero-knowledge* means that the distributions are not necessarily exactly the same, but they are statistically close, meaning that their statistical difference is a negligible function.
- We speak of *computational zero-knowledge* if no efficient algorithm can distinguish the two distributions.

Applications

Research in zero-knowledge proofs has been motivated by authentication systems where one party wants to prove its identity to a second party via some secret information (such as a password) but doesn't want the second party to learn anything about this secret. This is called a "zero-knowledge proof of knowledge". However, a password is typically too small or insufficiently random to be used in many schemes for zero-knowledge proofs of knowledge. A zero-knowledge password proof is a special kind of zero-knowledge proof of knowledge that addresses the limited size of passwords.

One of the most fascinating uses of zero-knowledge proofs within cryptographic protocols is to enforce honest behavior while maintaining privacy. Roughly, the idea is to force a user to prove, using a zero-knowledge proof, that its behavior is correct according to the protocol. Because of soundness, we know that the user must really act honestly in order to be able to provide a valid proof. Because of zero knowledge, we know that the user does not compromise the privacy of its secrets in the process of providing the proof. This application of zero-knowledge proofs was first used in the ground-breaking paper of Oded Goldreich, Silvio Micali, and Avi Wigderson on secure multiparty computation.

History and results

Zero-knowledge proofs were first conceived in 1985 by Shafi Goldwasser, Silvio Micali, and Charles Rackoff in a draft of "The Knowledge Complexity of Interactive Proof-Systems".^[3] While this landmark paper did not invent interactive proof systems, it did invent the **IP** hierarchy of interactive proof systems (see *interactive proof system*) and conceived the concept of *knowledge complexity*, a measurement of the amount of knowledge about the proof transferred from the prover to the verifier. They also gave the first zero-knowledge proof for a concrete problem, that of deciding quadratic nonresidues mod m . In their own

words:

Of particular interest is the case where this additional knowledge is essentially 0 and we show that [it] is possible to interactively prove that a number is quadratic non residue mod m releasing 0 additional knowledge. This is surprising as no efficient algorithm for deciding quadratic residuosity mod m is known when m 's factorization is not given. Moreover, all known *NP* proofs for this problem exhibit the prime factorization of m . This indicates that adding interaction to the proving process, may decrease the amount of knowledge that must be communicated in order to prove a theorem.

The quadratic nonresidue problem has both an **NP** and a **co-NP** algorithm, and so lies in the intersection of **NP** and **co-NP**. This was also true of several other problems for which zero-knowledge proofs were subsequently discovered, such as an unpublished proof system by Oded Goldreich verifying that a two-prime modulus is not a Blum integer.^[4]

Oded Goldreich, et al., took this one step further, showing that, assuming the existence of unbreakable encryption, one can create a zero-knowledge proof system for the *NP*-complete graph coloring problem with three colors. Since every problem in **NP** can be efficiently reduced to this problem, this means that, under this assumption, all problems in **NP** have zero-knowledge proofs.^[5] The reason for the assumption is that, as in the above example, their protocols require encryption. A commonly cited sufficient condition for the existence of unbreakable encryption is the existence of one-way functions, but it is conceivable that some physical means might also achieve it.

On top of this, they also showed that the graph nonisomorphism problem, the complement of the graph isomorphism problem, has a zero-knowledge proof. This problem is in **co-NP**, but is not currently known to be in either **NP** or any practical class. More generally, Goldreich, Goldwasser et al. would go on to show that, also assuming unbreakable encryption, there are zero-knowledge proofs for *all* problems in **IP=PSPACE**, or in other words, anything that can be proved by an interactive proof system can be proved with zero knowledge.^[6]

Not liking to make unnecessary assumptions, many theorists sought a way to eliminate the necessity of one way functions. One way this was done was with *multi-prover interactive proof systems* (see interactive proof system), which have multiple independent provers instead of only one, allowing the verifier to "cross-examine" the provers in isolation to avoid being misled. It can be shown that, without any intractability assumptions, all languages in **NP** have zero-knowledge proofs in such a system.^[7]

It turns out that in an Internet-like setting, where multiple protocols may be executed concurrently, building zero-knowledge proofs is more challenging. The line of research investigating concurrent zero-knowledge proofs was initiated by the work of Dwork, Naor, and Sahai.^[8] One particular development along these lines has been the development of witness-indistinguishable proof protocols. The property of witness-indistinguishability is related to that of zero-knowledge, yet witness-indistinguishable protocols do not suffer from the same problems of concurrent execution.^[9]

Another variant of zero-knowledge proofs are non-interactive zero-knowledge proofs. Blum, Feldman, and Micali showed that a common random string shared between the prover and the verifier is enough to achieve computational zero-knowledge without requiring interaction.^[10]

See also

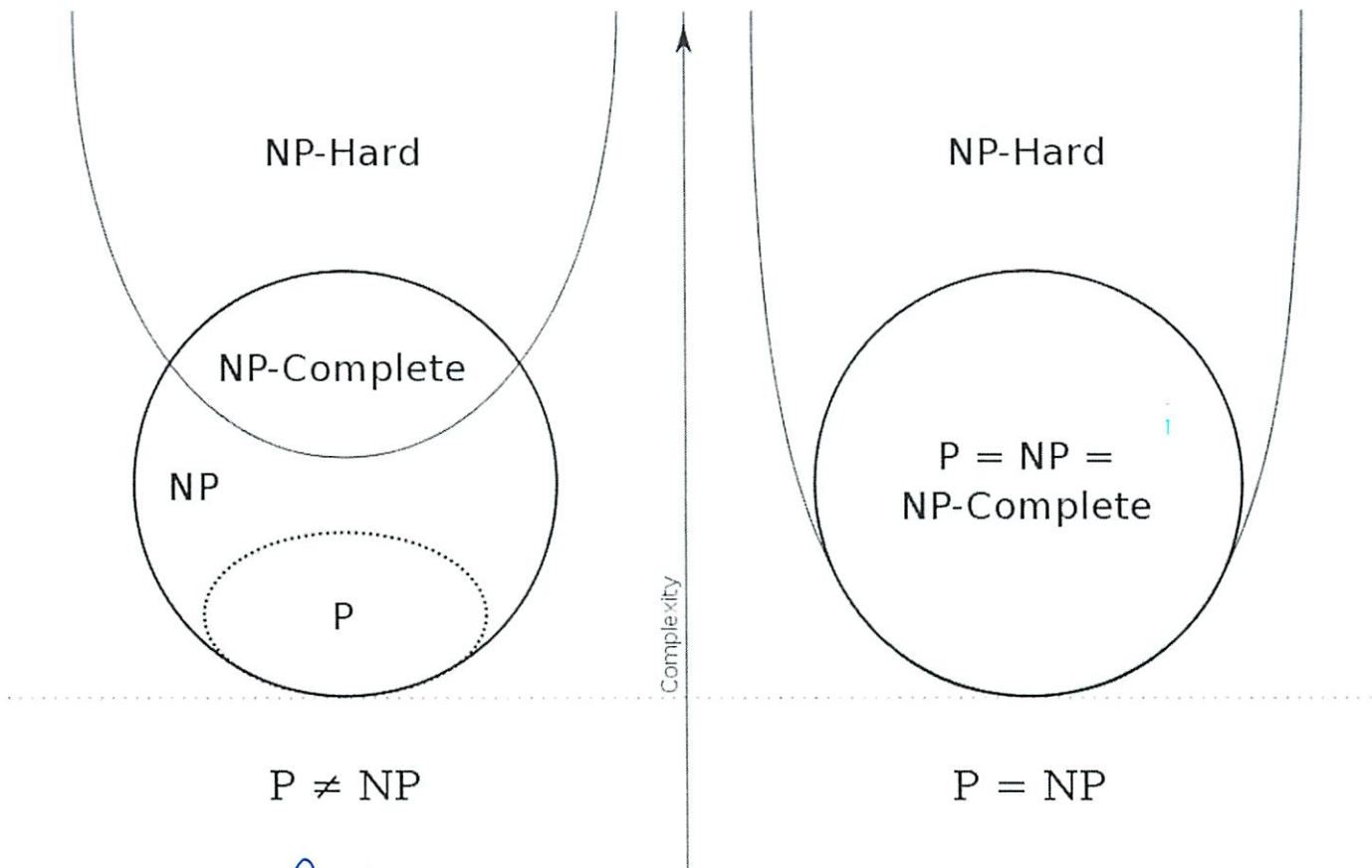
- Arrow information paradox
- Cryptographic protocol
- Feige–Fiat–Shamir identification scheme
- Proof of knowledge
- Topics in cryptography
- Zero-knowledge password proof
- Witness-indistinguishable proof

Notes

1. ^ Quisquater, Jean-Jacques; Guillou, Louis C.; Berson, Thomas A. (1990). "How to Explain Zero-Knowledge Protocols to Your Children" (<http://www.cs.wisc.edu/~mkowalczyk/628.pdf>) . *Advances in Cryptology - CRYPTO '89: Proceedings* **435**: 628–631. <http://www.cs.wisc.edu/~mkowalczyk/628.pdf>.
2. ^ <http://www.bennetyee.org/ucsd-pages/ZKP.html>
3. ^ Goldwasser, S.; Micali, S.; Rackoff, C. (1989), "The knowledge complexity of interactive proof systems" (<http://crypto.cs.mcgill.ca/~crepeau/COMP647/2007/TOPIC02/GMR89.pdf>) , *SIAM Journal on Computing* (Philadelphia: Society for Industrial and Applied Mathematics) **18** (1): 186–208, doi:10.1137/0218012 (<http://dx.doi.org/10.1137/0218012>) , ISSN 1095-7111 (<http://www.worldcat.org/issn/1095-7111>) , <http://crypto.cs.mcgill.ca/~crepeau/COMP647/2007/TOPIC02/GMR89.pdf>
4. ^ Goldreich, Oded (1985). "A zero-knowledge proof that a two-prime moduli is not a Blum integer". *Unpublished manuscript*.
5. ^ Goldreich, Oded; Micali, Silvio; Wigderson, Avi (1991). "Proofs that yield nothing but their validity". *Journal of the ACM* **38** (3): 690–728. doi:10.1145/116825.116852 (<http://dx.doi.org/10.1145/116825.116852>) .
6. ^ Ben-Or, Michael; Goldreich, Oded; Goldwasser, Shafi; Hastad, Johan; Kilian, Joe; Micali, Silvio; Rogaway, Phillip (1990). "Everything provable is provable in zero-knowledge". In Goldwasser, S.. *Advances in Cryptology--CRYPTO '88. Lecture Notes in Computer Science*. **403**. Springer-Verlag. pp. 37–56.
7. ^ Ben-or, M.; Goldwasser, Shafi; Kilian, J.; Wigderson, A. (1988). "Multi prover interactive proofs: How to remove intractability assumptions" (<http://theory.lcs.mit.edu/~cis/pubs/shafi/1988-stoc-bgkw.pdf>) . *Proceedings of the 20th ACM Symposium on Theory of Computing*: 113–121. <http://theory.lcs.mit.edu/~cis/pubs/shafi/1988-stoc-bgkw.pdf>.
8. ^ Dwork, Cynthia; Naor, Moni; Sahai, Amit (2004). "Concurrent Zero Knowledge". *Journal of the ACM* **51** (6): 851–898. doi:10.1145/1039488.1039489 (<http://dx.doi.org/10.1145/1039488.1039489>) .
9. ^ Feige, Uriel; Shamir, Adi (1990). "Witness Indistinguishable and Witness Hiding Protocols". *Proceedings of the twenty-second annual ACM Symposium on Theory of Computing (STOC)*. doi:10.1145/100216.100272 (<http://dx.doi.org/10.1145/100216.100272>) .
10. ^ Blum, Manuel; Feldman, Paul; Micali, Silvio (1988). "Non-Interactive Zero-Knowledge and Its Applications". *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*: 103–112. doi:10.1145/62212.62222 (<http://dx.doi.org/10.1145/62212.62222>) .

External links

- Applied Kid Cryptography (<http://www.wisdom.weizmann.ac.il/~naor/PUZZLES/waldo.html>) – A simple explanation of zero-knowledge proofs using Where's Waldo? as an example
- A gentle introduction to zero-knowledge proofs with applications to cryptography (<http://www.austinmohr.com/work/files/zkp.pdf>)
- How to construct zero-knowledge proof systems for NP (<http://www.wisdom.weizmann.ac.il/~oded>)



↑ What we know now

(need to review NPC vs NP-H)

Zero knowledge

Main article: Zero-knowledge proof

Not only can interactive proof systems solve problems not believed to be in **NP**, but under assumptions about the existence of one-way functions, a prover can convince the verifier of the solution without ever giving the verifier information about the solution. This is important when the verifier cannot be trusted with the full solution. At first it seems impossible that the verifier could be convinced that there is a solution when the verifier has not seen a certificate, but such proofs, known as zero-knowledge proofs are in fact believed to exist for all problems in **NP** and are valuable in cryptology. Zero-knowledge proofs were first mentioned in the original 1985 paper on **IP** by Goldwasser, Micali and Rackoff, but the extent of their power was shown by Oded Goldreich, Silvio Micali and Avi Wigderson.^[5]

What is the certificate
exactly again?



Final

Michael E Plasmeyer
Student



(https://wikis.mit.edu/gradebook-
/confluence/support@mit.edu)

STUDENT DETAIL

6.046J



R07

Michael Plasmeyer
Cumulative Grade: 230

/display
/GBMGuide
/Gradebook+Module+Overview
Comment:

View More

Sort By:

Filter...

	Due Date	Points	Max Points	Weight
Problem 1-1	09-25-2012	3.00	3	1
Problem 1-2	09-25-2012	3.00	3	1
Problem 1-3	09-25-2012	3.00	3	1
Problem 1-4	09-25-2012	2.00	3	1
Problem 2-3	10-05-2012	2.00	3	1
Problem 2-1	10-05-2012	2.00	3	1
Problem 2-2	10-05-2012	2.00	3	1
Problem 2-4	10-05-2012	1.00	3	1
Quiz 1	10-11-2012	32.00	80	80
Problem 3-1	10-25-2012	1.00	3	1
Problem 3-2	10-25-2012	1.00	3	1
Problem 3-3	10-25-2012	3.00	3	1
Problem 3-4	10-25-2012	2.00	3	1
Problem 4-1	11-06-2012	4.00	6	1
Problem 4-2	11-06-2012	5.00	6	1
Quiz 2-1	11-14-2012	10.00	15	0
Quiz 2-2	11-14-2012	10.00	20	0
Quiz 2-3	11-14-2012	2.00	20	0
Quiz 2-4	11-14-2012	7.00	25	0
Quiz 2-5	11-14-2012	15.00	20	0
Problem 5-1	11-27-2012	2.00	3	0

Problem 5-2	11-27-2012	2.00	3	0
Problem 5-3	11-27-2012	1.00	3	0
Problem 6-1	12-06-2012	3.00	3	0
Problem 6-2	12-06-2012	3.00	3	0
Problem 6-3	12-06-2012	3.00	3	0
Problem 6-4	12-06-2012	3.00	3	0
Final Exam	12-17-2012	103.00	150	0

[SHOW FIRST PAGE](#)