CS170

25 Jan 1996
Thursday

HW#0
1. Time slots for sections
2. Time slots for midterm
3. Photograph

Θ NOTATION

Defn: $f, r \quad \mathbb{N} \to \mathbb{R}^+$

$$f(n) = \Theta(r(n)) \iff \exists \text{ integer } n_0, \exists \text{ real } c > 0$$
$$\text{s.t. } \forall n \geq n_0$$
$$f(n) \leq c\, r(n)$$

"f grows no faster than r"

Ex $f(n) = 3n^2 - 5n + 7$
- claim $f(n) = \Theta(n^4)$
  proof by induction

Ex $f(n) = 2n + 25\log_2 n$
claim $f(n) = \Theta(n)$
$n_0 = 1, \quad c = 27$
$n \geq 1; \quad 2n + 25\log_2 n \stackrel{?}{\leq} 27n$
$n \geq 1; \quad \log_2 n \leq n$
$2^n \geq n$

Why Θ notation?
- Succintness
- ignores constants
- easy to manipulate

M/c : 1000 instr/sec (1000000)

algo	running time	problem size in 1 second	prob size in 1 minute	prob size in 1 hour
A_1	n	1000 / 10^6	$60{,}000 = 6\times10^7$	$3{,}600{,}000$
A_2	$n\log_2 n$	140 / 62746	4,893	200,000
A_3	n^2	31 / 1000	244	~1897
A_4	n^3	10 / 100	39	153
A_5	2^n	9 / 19	15	21

- Rules for Θ notation:

$f, g, r, s : \mathbb{N} \to \mathbb{R}^+$

$f(n) = \Theta(r(n))$; $g(n) = \Theta(s(n))$ $\quad K>0$: real

✓ (i) $Kf(n) = \Theta(r(n))$

(ii) $f(n) + g(n) = \Theta(r(n) + g(n))$

(iii) $f(n)\,g(n) = \Theta(r(n)\,g(n))$

✓ (iv) $f(n) + g(n) = \Theta(\max\{r(n), s(n)\})$; pointwise

Pf (i) Need to show n_0', c'

s.t. $Kf(n) \le c' r(n) \quad \forall n \ge n_0'$

soln $c' = Kc$; $n_0' = n_0$ ∎

Pf (iv) $\exists n_f, c_f : \forall n \ge n_f \Rightarrow f(n) \le c_f r(n)$

$\exists n_g, c_g : \forall n \ge n_g \Rightarrow g(n) \le c_g s(n)$

$f(n) + g(n) \le c_f r(n) + c_g s(n)$

define $c' = \max\{c_g, c_f\}$

$\therefore c_f \le c'$; $c_g \le c'$

$\le c'(r(n) + s(n)) \quad \forall n \ge \max(n_f, n_g) = \bar{n}$

at each n consider $\max(r(n), s(n))$

$\therefore r(n) \le \max(r(n), s(n))$

$s(n) \le \max(r(n), s(n))$

$\therefore (\cdot) \le 2c' \max\{r(n), s(n)\}$

$\therefore \forall n \ge \bar{n} \quad f(n) + g(n) \le \bar{c} \max\{r(n), s(n)\}$; $\bar{c} = 2c'$

QED

Reminder $a_1, a_2, \ldots$ sequence

$\lim_{n\to\infty} a_n = A$ means $\forall \epsilon > 0 \; \exists \; n \in \mathbb{N}$

s.t. $\forall n \geq n_\epsilon : |a_n - A| < \epsilon$

Thm $f, g \quad \mathbb{N} \longrightarrow \mathbb{R}^+$

If $\lim_{n\to\infty} [f(n)/g(n)]$ exists (is finite > 0) then

$$f(n) = \Theta[g(n)]$$

pf $\forall \epsilon > 0 \; \exists \; n_\epsilon : \forall n \geq n_\epsilon \quad |f(n)/g(n) - A| < \epsilon$

lim exists $\Rightarrow \exists A \in \mathbb{R}^+$

$$\frac{f(n)}{g(n)} - A \leq \left| f(n)/g(n) - A \right| < \epsilon$$

$$\Rightarrow f(n) - A\, g(n) \leq \epsilon\, g(n)$$

$$\Rightarrow f(n) \leq (\epsilon + A) g(n) \quad ; \text{can choose any } \epsilon > 0 \text{ say } \epsilon = 1$$

$\therefore c = 1 + A \qquad f(n) \leq c\, g(n) \quad \forall n \geq n_c$

Ex $[3n^3 + 5n^{2\cdot 5} + (-6)]/(5n^2 - 16n) \stackrel{?}{=} \Theta(n)$

pf $\lim_{n\to\infty} (3n^3 + 5n^{2\cdot 5} - 6)/(5n^3 - 16n^2) = 3/5$

Defn: $f, g \quad \mathbb{N} \to \mathbb{R}^+$

$\lim_{n\to\infty} f(n)/g(n) = 0$ then we say $f(n) = o(g(n))$

Ex $\log_c(n) \stackrel{?}{=} o(n)$

$\lim_{n\to\infty} \log_e n / n$: L'Hôpital's rule

$$= \lim_{n\to\infty} 1/n \,/\, 1$$

$$= 0$$

Assume $f(n)$ monotonically increasing

$c \geq 0$ "arb. real constant

$a > 1$

claim $(f(n))^c = O(a^{f(n)})$; exp fct grows faster than any polynomial

CS 170

30 Jan '90
Tuesday

Defn $\lim_{n\to\infty} f(n)/r(n) = 0$ then $f(n) = o(r(n))$

Lemma $f: \mathbb{N} \to \mathbb{R}^+$; f monotonically increasing
i.e. $\forall n: f(n) \le f(n+1)$
$c \in \mathbb{R}$ constant
$a > 1$ constant

$[f(n)]^c = \Theta(a^{f(n)})$

Pf for special case, where f (extended to reals) is differentiable

$$\lim_{n\to\infty} f(n)^c / a^{f(n)} = \lim_{n\to\infty} c f'(n) \cdot f(n)^{c-1} / [e^{(\ln a) f(n)} \cdot \ln(a) f'(n)]$$

$$= c/\ln a \left[\lim_{n\to\infty} f(n)^{c-1} / a^{f(n)} \right]$$

$\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \le x\}$
$\lceil x \rceil = \min\{n \in \mathbb{Z} \text{ s.t. } x \le n\}$

induction on $\lceil c \rceil$ (ceiling of c):
Base case $c \le 0$

$$\lim_{n\to\infty} \frac{f(n)^c}{a^{f(n)}} = \frac{\lim_{n\to\infty} f(n)^c}{\lim_{n\to\infty} e^{f(n)}} = \frac{0}{\infty} = 0$$

Inductive Step: Assume $\lim_{n\to\infty} \frac{f(n)^{c'}}{a^{f(n)}}$ exists for all $c' < k$

Let c be s.t. $\lceil c \rceil = k+1$

can replace Θ w/ o (but fun $\uparrow \infty$ reqd)

Lemma: $n^c = o(a^n)$; $a > 1$

Multiplying two large integers A & B

<n digit>

Usual algorithm: P = 0

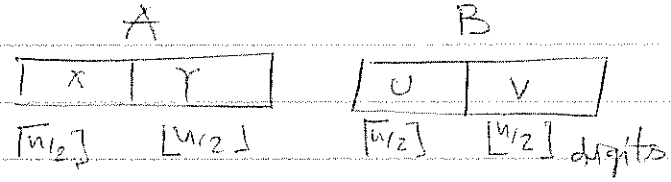
For each digit X of B

$\Theta(n)$ { let S be the product of A & X ← $\Theta(n)$
let s' be S times appropriate power of 10 ← $\Theta(n)$
add s' to P: $\Theta(n)$ ← will have 2n digits
return(P)

loop executed $\Theta(n)$ $\Rightarrow \Theta(n^2)$

Strange algorithm

~~2 8 * 5 6~~ 2 8 * 5 6



$A = (X * 10^{\lfloor n/2 \rfloor} + Y)$ $B = (U * 10^{\lfloor n/2 \rfloor} + V)$

$$XU\,10^{2\lfloor n/2 \rfloor} + (XV + YU) * 10^{\lfloor n/2 \rfloor} + YV$$

multiply (A, B: big numbers, n digits)

$\Theta(n)$ {
if n = 1 then return(A * B)
$Y \leftarrow X = A$ "mod" $10^{\lfloor n/2 \rfloor}$ --- first $\lceil n/2 \rceil$ digits of A
$V \leftarrow U = B$ "mod" $10^{\lfloor n/2 \rfloor}$ --- " ---
$X \leftarrow Y = A$ "div" $10^{\lfloor n/2 \rfloor}$ --- last $\lfloor n/2 \rfloor$ digits of A
$U \leftarrow V = B$ "div" $10^{\lfloor n/2 \rfloor}$ --- " ---

P_1 = multiply (X, U, $\lceil n/2 \rceil$)
P_2 = multiply (X, V, $\lceil n/2 \rceil$)

P_3 = multiply $(Y, U, \lceil n/2 \rceil)$
P_4 = multiply $(Y, V, \lfloor n/2 \rfloor)$

| mod multiply times ⟹ do explicitly

$\Theta(n)$ {
$P_1' = P_1$ times $16^{2\lfloor n/2 \rfloor}$
$P_{23}' = (P_2$ plus $P_3)$ times $10^{\lfloor n/2 \rfloor}$
return $(P_1'$ plus P_{23}' plus $P_4)$

Analyse worst case

Let $T(n)$ be the worst case running time of algorithm "multiply" on input numbers of size n.

$T(1) = \alpha$; constant

Assume $n = 2^k$

$T(n) \leq \beta n + 4T(n/2)$
$\hookrightarrow + (\beta n/2 + 4T(n/4))$
$\beta n + 2\beta n + 16T(n/4)$
$\hookrightarrow 16(\beta n/4 + 4T(n/8))$
$\beta n + 2\beta n + 4\beta n + 64T(n/8)$

pattern: $[\beta n + \beta 2n + \cdots 2^{l-1}\beta n] + 4^l T(n/2^l)$; check by induction
$\leq \beta n 2^l$
$n/2^l = 1$ @ some pt.

$2^l = n \Rightarrow l = \log_2 n$

entire expression $\leq \beta n 2^l + 2^l \cdot 2^l T(1)$
$= \beta n^2 + n^2 \alpha = (\alpha + \beta) n^2$
$= \Theta(n^2)$

Problem is in the recurrance eqn.

General Recurrance Relation

$$T(n) \leq \begin{cases} \alpha & ; n = 1 \\ aT(n/b) + \beta n^k & ; n > 1 \end{cases}$$

assume $n = b^i$ for some $i \in \mathbb{N}$

$$T(n) \leq \beta n^k + aT(n/b)$$
$$\hookrightarrow + a\left[\beta (n/b)^k + aT(n/b^2)\right]$$
$$\leq \beta n^k + a/b^k \beta n^k + a^2 T(n/b^2)$$
$$\hookrightarrow a^2 \left\{ \beta (n/b^2)^k + aT(n/b^3) \right\}$$
$$\leq \beta n^k + a/b^k \beta n^k + (a/b^k)^2 \beta n^k + a^3 T(n/b^3)$$
$$\cdots\cdots$$

Eventually ($n = b^l$, i.e. $l = \log_b n$)

$$\beta n^k \left(1 + a/b^k + (a/b^k)^2 + \cdots (a/b^k)^{l-1}\right) + a^l T(n/b^l)$$

Examine $(1 + c + c^2 + \cdots c^{l-1}) = (c^l - 1)/(c-1)$

$c < 1$	$< 1/1-c$
$c = 1$	$= l$
$c > 1$	$< c^l/c-1 = \Theta(c^l)$ (?)

$a/b^k < 1 \quad T(n) \leq \Theta(n^k) + \alpha a^l$

$\Rightarrow a < b^k$

$\log_b a < k$

$\alpha a^{\log_b n}$

$\alpha (b^{\log_b a})^{\log_b n}$

$\alpha (n^{\log_b a})$

$\alpha (n^r)$

$\therefore T(n) = \Theta(n^k)$

$\therefore \quad a/b^w < 1; \; T(n) = \Theta(n^k)$

when $a/b^k = 1$; $T(n) \leq \beta n^k l \neq \beta + a^l \alpha$

$\beta n^k \log_b n + a^{\log_b n}$ $\Theta(n^k)$

$\therefore (a/b^k) = 1 \quad T(n) = \Theta(n^k \log n)$

$\frac{a}{b^k}) > 1$

$T(n) = \beta n^k (\frac{a}{b^k})^k + \alpha n^{\log_b a}$

$T(n) \leq \beta n^k u (a/b^k)^l + a^l T(1)$

$T(n) = \beta n^k (\frac{a}{b^k})^{\log_b n} + \alpha n^{\log_b a}$

$= \frac{\beta n^k a^{\log_b n}}{b^{k \log_b n}} + \alpha n^{\log_b a}$

$= \frac{\beta n^k a^{\log_b n}}{n^k} + \alpha n^{\log_b a}$

$= \beta a^{\log_b n} + \alpha n^{\log_b a}$

$T(n) = \Theta(n^{\log_b a})$

since $a/b^k > 1$

i.e. $a > b^k$

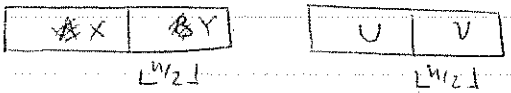
i.e. $(\log_b a > k)$

CS170 Algorithms

Feb 1, 1990
Thursday

$[f(n)]^c = \Theta(a^{f(n)}) \quad ; a > 1$

Multiplication : $\Theta(n^2)$



$A * B = (X * U) * 10^{2\lfloor n/2 \rfloor} + (X * V + Y * U) * 10^{\lfloor n/2 \rfloor} + (Y * V)$

$$T(n) = \begin{cases} \alpha & ; n = 1 \\ \Theta(n) + 4\,T(n/2) & ; n > 1 \end{cases}$$

$T = \Theta(n^2)$!

Examined $f(n) = \begin{cases} a & n = 1 \\ \Theta(n^k) + a\,f(n/b) & n > 1 \end{cases}$

cond: $f(n) = \Theta(n^k) \quad a < b^k$

$\Theta(n^k \log n) \quad a = b^k$

$\Theta(n^{\log_b a}) \quad a > b^k$

$k = 1 \quad a = 4 \quad b = 2$

$\therefore$ here $\Theta(n^{\log_2 4}) = \Theta(n^2)$ Reasoning

looking at recurrance rels we say the way to improve is by changing b or a (k = 1 true) (mean)

$XU + (XV + YU) + YV$ computes as $(X+Y)(U+V)$

form $XU * YV$ and subtract

$\therefore$ in 3 !

(better) compute $(X-Y)(V-U) = (XV + YU) - XU - YV$
smaller size
size will grow otherwise!

XU
YV
add up.

fast multiply (A, B big nos, n # of digits)
If $n = 1$ return $(A * B)$
Break A into X & Y
Break B into U & V
$S_1 = X$ minus Y
$S_2 = V$ minus U
P_1 = fast multiply $(X, U, \lceil n/2 \rceil)$
P_2 = fast multiply $(Y, V, \lfloor n/2 \rfloor)$
P_3 = fast multiply $(S_1, S_2, \lceil n/2 \rceil)$
$P_3' = (P_3$ plus P_1 plus $P_2)$
return $(P_1$ times $10^{2\lfloor n/2 \rfloor}$ plus P_3' times $10^{\lfloor n/2 \rfloor}$ plus $P_4)$

Running time:

$$T(n) \leq \begin{cases} \alpha & ; n = 1 \\ \beta n + 3T(n/2) & ; n > 1 \end{cases}$$

$k = 1 \quad a = 3 \quad b = 2$
$\therefore \Theta(n^{\log_3 2}) = \Theta(n^{1.59})$

Brossard, Bratley:
digits $0 \leq d < 2^{20}$
100 digit # normal = 400 fastest 300msec
$\hookrightarrow$ 602 decimal digits
1000 digits # normal = 40sec fastest 15sec
$\hookrightarrow$ 6000 decimal digits

Record: Schönhage & Strassen $n \log n [\log \log (n)]$? Asymptotically

$O(n \log^k n)$ is implementable usually

Book: Aho, Hopcroft Ch 3 ; 4.1, 4.2, 4.4

Dictionaries

Abstract Data Type

Structure ..... set (finite generally) ← Universe

Operations ..... Insert, Delete, Member, Access

create(S)

create(S) : creates empty set

insert(x,S)

delete(x,S)

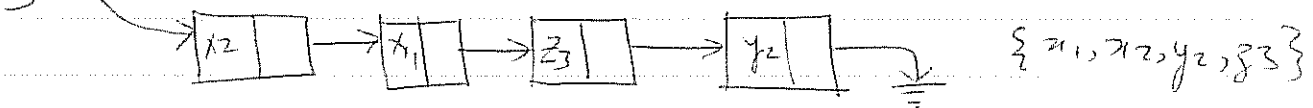
Member?(x,S)

Access(x,S) ← retrieves assoc. info eg tel #

$x, y \in U$

test whether $x = y$

Possible implementation of dictionaries is by linked list.



insert(x,S)

1st see if already in there!

If not tag on to end

$\therefore \Theta(|S|)$

deletion(x,S)

same thing $\Theta(|S|)$ avg case $\sim \frac{|S|}{2}$

member access (x) $\Theta(|S|)$

creation $\Theta(1)$

Assume Universe is "small" $U = \{0, \ldots, M-1\}$

$|U| = M$

Poss. implementation: Array; S is represented by an array

array of {0,1} info

$x \in S \Leftrightarrow A[x] \neq 0$

insert, delete, number access, create : $O(1)$
create $O(M)$ (initialize to zero need)

hashing fn : larger Universe to smaller set.

Universe $|U| = M$ large
set sizes $<< M$
table size t
idea : use a hash fn $h: U \to \{0, \ldots t-1\}$
$A[0..t-1]$ array $x \in S$ iff it is stored at $A[h(x)]$

1. choice of hash fn
2. resolution of collisions

Eg $x \bmod t$ prop of hash fn : ① easy to compute
② distribute evenly $\forall$ all $i \in t$ $|h^{-1}(i)| \approx |U|/t$ (??)
③ "randomness"

CS170 ALGORITHMS 6 Feb 1990

MIDTERM: 2 hrs out of 5-9 pm We Mrch 14

$O(n^{\log_2 3}) = O(n^{1.59})$; multiplication

Dictionary Problem
- insert
- delete
- member
- (Access)

implementation: (1) linked list all ops $\Theta(n)$; create unif
(2) array (size of universe) ; create $\Theta(M)$
others $\Theta(1)$

Hashing table T of size t $\{0, \ldots t-1\}$
universe $U = \{0, \ldots M-1\}$
Hash fn $H: U \to \{0, \ldots t-1\}$
store $x \in U$ at. $T[h(x)]$

Properties: (1) H should be easy to compute
(2) "distributed evenly" i.e. $\forall i$, $0 \le i < t$
$|h^{-1}(i)| \approx |U|/t$
(3) h deterministic

Ex $h(x) = x \bmod t$; HW op. fast
if $t = 2^s$ h picks last s binary digits of x
could have lots of collisions though.
$h(x) = (a \bmod p) \bmod t$ p is prime * $t < p < |U|$

Collisions what if $h(x) = h(y)$?

- Closed Hashing (linear probing)
if location $h(x)$ is occupied, try $(h(y)+1) \bmod t$?

problems ē linear probing
- deletions are next to impossible
- secondary collisions
(cookie monster problem)

Ex. $h(x) = x \bmod 7$

insert 33, 15, 40, 12, 48

The bigger the chain is the more likely it is to grow.

Open Hashing (Seperate Chaining)

Every location in the table points to the beginning of a linked list (will need a little more memory space)

Analysis: Worst case is $O(n)$ (everything hashed to the same location)

"Average Case" (1) h distributes universe U evenly over the table $|h^{-1}(i)| \approx |U|/|T|$ $0 \le i < t$

(Assumption about user) → (2) Any operation INSERT(x) MEMBER(x) DELETE(x) is equally likely any member of U.

set currently stored

$S \rightarrow$ n insertions have occured already

by 2, S is a random subset of size what is the expected time to insert a new element x?

$h(x) = i$

$\Rightarrow$ Expected # of elements of S stored at i is $\frac{|S|}{t}$

∴ insertion time, deletion, member time $\propto |S|/t$

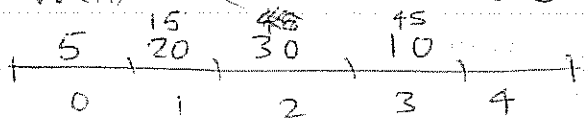
Maybe to ensure not already there!

Expected time for insert, delete, member is $O(n/t)$ (cont

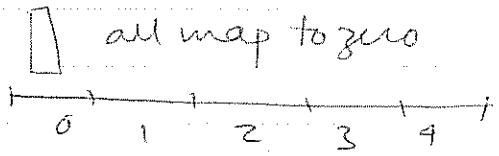
Removing Assumption 2

$h(x) = x \bmod 5$; user decides to insert 5, 10, 20, 30, 15, 4
all map to same location

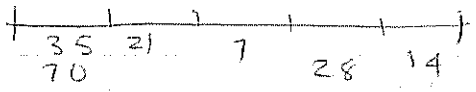
$h'(x) = (x \bmod 7) \bmod 5$



New data: 7, 28, 14, 35, 21, 70



But if we used old HF.



Knowing HF: can break
Knowing set: can get HF
Maybe we can circumvent assumptions about user by using one of a set of HFs by choosing randomly.

IDEA: Choose a hash fn at random.

Ex: $U = \{0, \ldots, M-1\}$
t = table size
$p \ldots t < p \leq M$ prime ※
Choose at random $a, b \in \{0, \ldots, p-1\}$
Use hash fn

$$h_{a,b}(x) = ((ax+b) \bmod p) \bmod t$$

Def $c \in \mathbb{R}$ $\quad t, M \in \mathbb{N}$

$\mathcal{H} \subseteq \{h : \{0, \ldots, M-1\} \to \{0, \ldots t-1\}\}$

$\mathcal{H}$ is c-universal if $\forall x, y \in U \quad x \neq y$

$$\left| \{h \in \mathcal{H} \mid h(x) = h(y)\} \right| \leq c \frac{|\mathcal{H}|}{t} \quad ; \; c = 2/4 \text{ small } \#$$

Fix $x, y : x \neq y$ fraction of functions in $\mathcal{H}$ that produce collision for x & y is $\leq c/t$

c is Universal constant indep. of x, y

No. of your x & y that you are getting

Claim 1 Let $U = \{0, \ldots, M-1\}$, M is prime $\ast$
↳ not prime ⇒ extend

$\mathcal{H} = \{h_{ab} : U \to \{0, \ldots, t-1\} ; h_{ab}(x) = ((ax+b) \bmod M) \bmod t\}$

$0 \leq a, b < M\}$

is c-universal where $c = \left(\lceil M/t \rceil / M/t\right)^2$.

c is small!

Claim 2: Let $S \subset U$ fixed, $|S| = n$

$y \in U$ fixed

$\mathcal{H}$ c-universal class of hash fn

assuming that a random hash fn $h \in \mathcal{H}$ was used to store S, then the expected number of elements of S stored at the same location as y is atmost cn/t.

Any set S not necc random!

Carter & Wegman '79.

CS170

Proof of claim 2:

$$\delta_h(x,y) = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

Fix h. The number of elements of S stored in the same location as z is

$$\sum_{x \in S} \delta_h(x,z)$$

average over all $h \in \mathcal{H}$:

$$= \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} \sum_{x \in S} \delta_h(x,z)$$

$$= \frac{1}{|\mathcal{H}|} \sum_{x \in S} \sum_{h \in \mathcal{H}} \delta_h(x,z) \qquad < c|\mathcal{H}|/t \quad \text{(by defn of c-universal)}$$

$$\leq c/t \sum_{x \in S} 1 = cn/t$$

∎

CS170 Algorithms 8 Feb '90

office hours W 2-3 Th 11-12:30 Yanis
Tu 11-12 W 1-2 Dana

Reading 4.3.3-4.3.4

Last Time: open hashing (if any set of size n equally likely to be stored n/t) t: table size
bad: need to make assumption about i/p.
So introduce randomization ourselves! Leads to
Universal Hashing: $\mathcal{H}$ is a class of hash fns
$h : \{0, \ldots, m-1\} \to \{0, \ldots, t-1\}$
$\longleftarrow u \longrightarrow$

$\mathcal{H}$ is c-universal $\forall x, y \in u \quad |\{h \in \mathcal{H} \mid h(x) = h(y)\}| \leq c\frac{|\mathcal{H}|}{t}$

claim 2: $|S| = n$; randomly chosen h from $\mathcal{H}$
$\forall z \in U$ Expected length of list at location $h(z)$ is $\leq c \cdot n/t$

claim 1: $U = \{0, \dots M-1\}$ M prime ※ $t < M$

$\mathcal{H} = \{h_{ab} ; h_{ab}(x) = ((ax+b) \bmod M) \bmod t \mid 0 \leq a, b < M\}$
$U \to \{0, .. t-1\}$
is a c-universal fn for $c = \left(\lceil M/t \rceil / M/t\right)^2$

Proof: $|\mathcal{H}| = M^2$
for $x \neq y$
when is $h_{ab}(x) = h_{ab}(y) = l$; naturally $0 \leq l < t$

$((ax+b) \bmod M) \bmod t = l$
$((ax+b) \bmod M) \bmod t = l$

By dfn $A \bmod t = l \iff \exists s, 0 \leq s \dots \leq \lfloor \frac{A}{t} \rfloor$ $A = l + st$

$\therefore \exists s, \exists r$: $(ax+b) \bmod M = l + st$
$(ay+b) \bmod M = l + rt$
etc.

$A < M \leq (?) \lfloor M/t \rfloor = \lceil M/t \rceil - 1$ ($\because M$ prime)

for every l, s, r
$xa + 1b = (l + st) \bmod M$
$ya + 1b = (l + rt) \bmod M$
‖ for how many choices of a & b can these eqns have soln?

$\det(\cdot) = x \cdot 1 - y \cdot 1 \neq 0$
$\therefore$ exactly one solution

$\bmod M$: always gives same results as ordinary arithmetic
↑ prime $\therefore$ ※ inverse (mul & add) exist (field)

How about seeing how good a random inputs?

set of n keys is inserted; each order of keys is equally likely to occur as input.

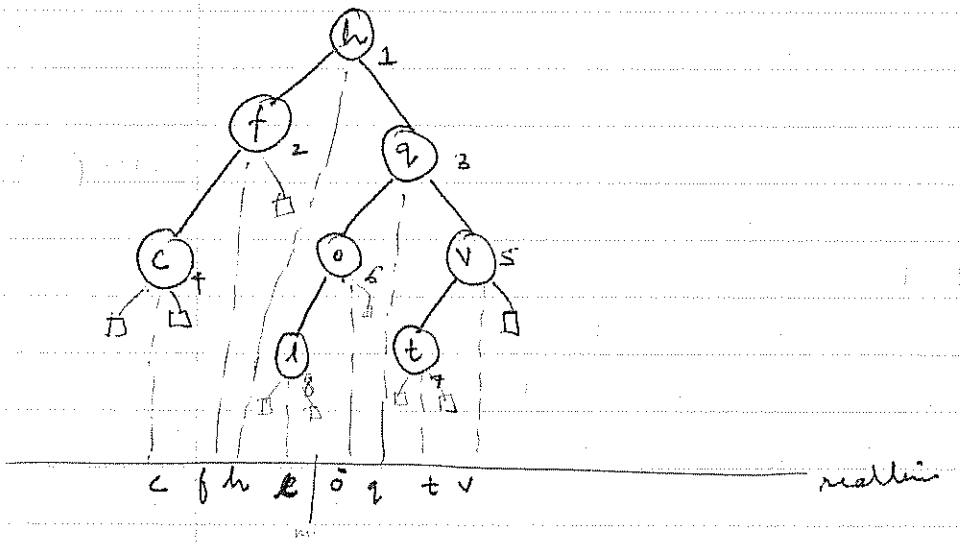
{h, c, f, l, o, q, t, v}

plan1 h f q c v o t l : 1/8!

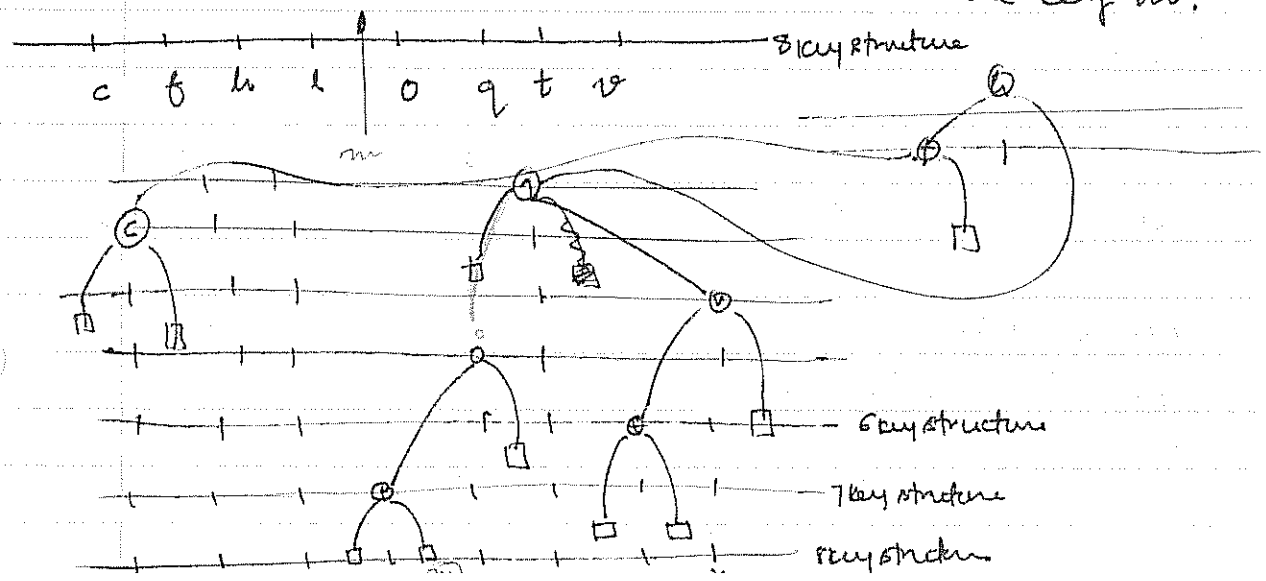
plan2 q v c t h o f q : 1/8!

What do trees look like in "Expected Case"?

↳ Bad Assumptions. But lets see.



come around looking for the key m.



To find a key in the 8 key structure
(e.g. 11)
(1) recursively find key in 7 key structure
(2) maybe perform one more comparison
↳ Prob $\leq 2/8 = 2/8$ (Prob of additional comparison)
↳ ∵ of ——•—— sort of ckt

E[# of comparisons necessary to go from (i-1) key structure to the i key structure] is at most $2/i$

Conclusion: Expected # of comparisons necessary to find a key in an n key structure is $\leq (2/n + 2/n-1 + \cdots - 2/3 - 2/2 + 2)$
$= 2H_n$
$\approx 2\ln(n)$
$\approx 1.38 \log_2 n$

CS170 Algorithms (Th) 15 Feb

Last Time: AVL trees
HW: Join " " in log. times
Random insertions in B.S.T. (confusing)

Thm: A n keys from ordered universe U, ($x \in U$ fixed) For each π, permutation of A, consider T_π the binary search tree that results from inserting the elements of A into an initially empty tree according to the order dictated by π.

$S_\pi(x)$ = # of comparisons necessary to search for x in T_π

Claim: Assuming that each π is as likely as the others, the Expected # of comparisons necessary to search for x

$$S(x) = \frac{1}{n!} \sum_{\text{all } \pi} S_\pi(x) \leq 2H_{n+1} < 2\ln n \approx 1.38 \log_2 n$$

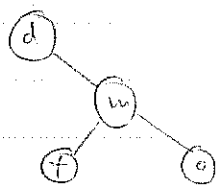
↑ n^{th} Harmonic #

CS170

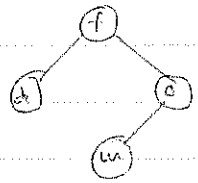
Ex: $\{d, f, m, o\}$ $n = 4$

π_1 d m o f $\quad T_{\pi_1}$

π_2 f d o m



T_{π_2}



$x = 'e'$ $S_{\pi_1}('e') = 3$

$S_{\pi_2}('e') = 2$

<u>cor</u> Expected search time for any element in such a random tree is $O(\log n)$

Expected time req'd to build such a tree is $O(n \log n)$

↳ ∵ each subtree is random

$\therefore = \sum_{i=1}^{n} \log i \doteq n \log n$

<u>Proof</u> It suffices to consider $x \notin A$ (Worst case)

$p_i(x)$ = prob that there is a comparison between x & the element of A that was ~~the~~ ith ~~one~~ inserted as the i^{th} one.

$$p_n(x) = \begin{cases} 1/n & \text{if } (x > a \ \forall a \in A) \text{ or } (x < a \ \forall a \in A) \\ 2/n & \text{otherwise} \end{cases}$$

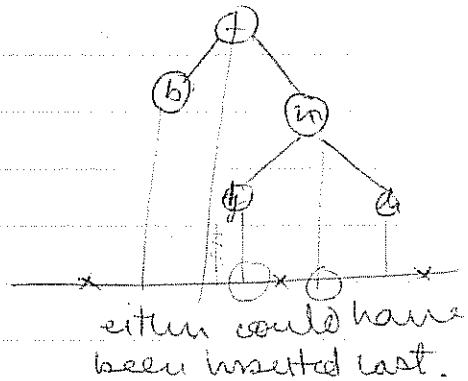
$p_n(x) \leq 2/n$

$p_i(x) \leq 2/i$

$\exists i$ random variable

↳ that counts the # of comparisons between x & the i^{th} element of A added to the tree

indicator $\begin{cases} \Pr(z_i = 1) = p_i \\ \Pr(z_i = 0) = 1 - p_i \end{cases}$



$\therefore E[z_i] = p_i \leq 2/i$

$z = z_1 + z_2 + \cdots z_n$

$E(z) = E(z_1) + E(z_2) + \cdots E(z_n) \leq 2 \sum_{1 \leq i \leq n} 1/i$

BOT! There is a big problem ∈ deletions

Problems ∈ BSTs:
① Making assumptions about i/p dist
② No deletions
↳ If you repeatedly insert & delete, the tree loses balance, becomes $O(\sqrt{n})$

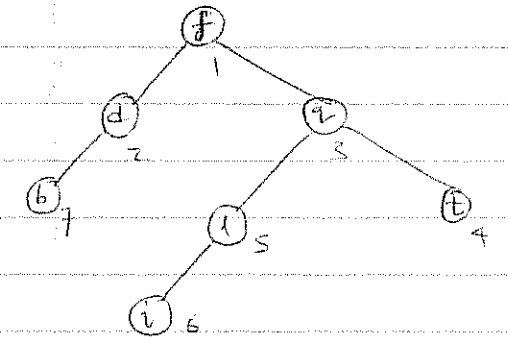
∴ Want similar result ∈ no assumptions on i/p & no deletions.

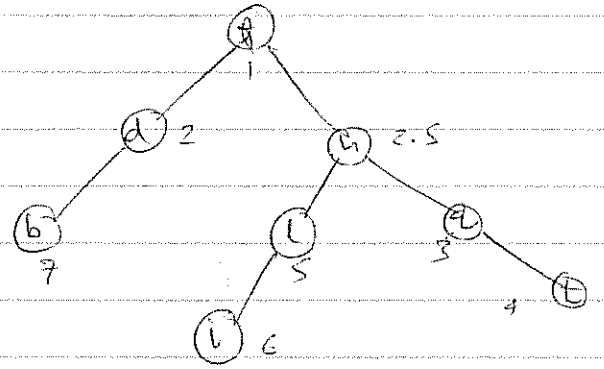
∴ Do some kind of randomness in datastructure
"push randomness into algo."
if A never changes:
- generate a random permutation of A
- insert according to permutation.

Problem: need to know A in advance and no changes A.
But if know A → can do ordinary sort.
∴ useless

{f, d, q, t, l, i, b}

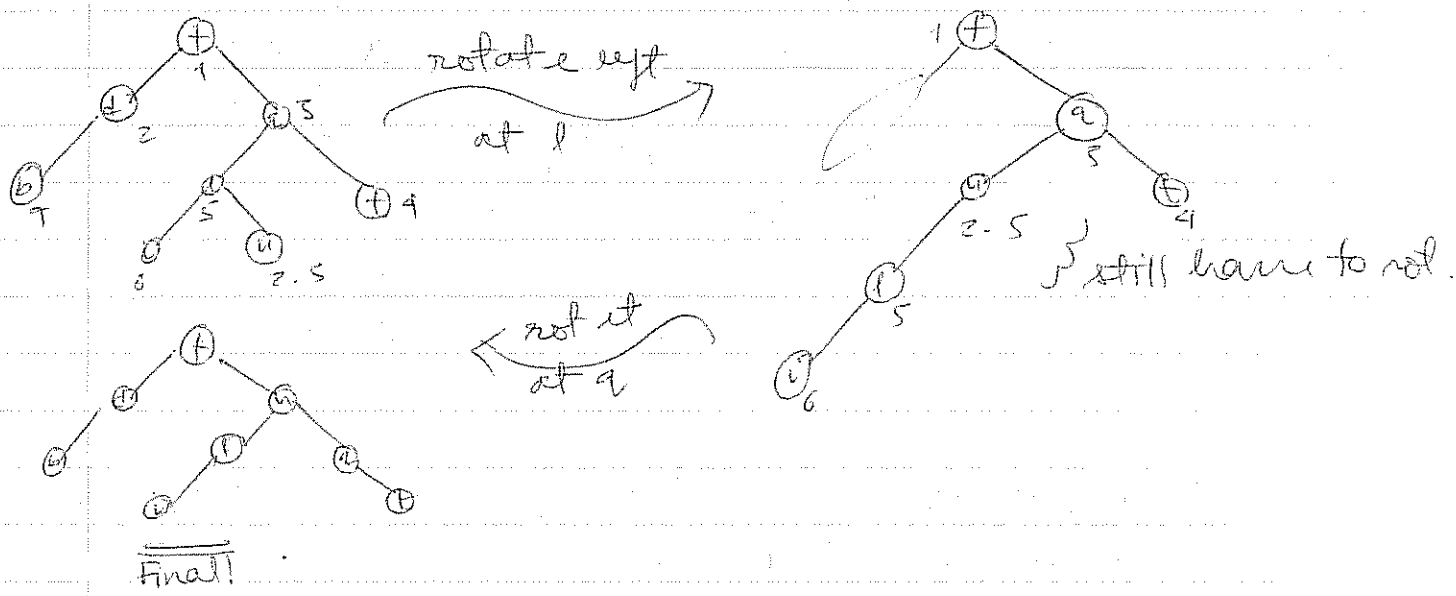
Want to insert 'n' as if it appeared in a certain place of the permutation, eg *





any child's priority is greater than that of child →

∴ must restructure original tree! By a rotate left at 'l'



S = set of "items"

item (key, priority)

from ordered universe ∈ ℝ

not a heap = ~~heap~~ TREAP for S

binary tree for S
in in-order ∈ respect to keys
in "heap-order" ∈ respect to priorities

∀ node z : priority of parent of z ≤ priority of z

▷ # of treaps is one only!
Recursively prove: root has to be...

{if all keys distinct
& all priorities distinct}

UPDATE ROUTINES

insert (K, prio, T)
if T = NULL then set T to new node (K, prio)
else if K = T→key then return
else if K < T→key then insert (K, prio, T→left child)
if T→lc→prio < T→prio then Rotate Right
else insert (K, prio, T→rc)
if T→rc→prio < T→prio then Rot left (T)

Your NULL must be some universal node whose priority = ∞.

delete (K, T) {no matter what its priority is}
if T = NULL then return
if T→key = K then
if T→lc→prio < T→rc→prio then Rotate Right
else Rotate Left (T)
if T→lc = T→rc = NULL then set T to NULL
if K < T→key then delete (K, T→lc)
else delete (K, T→rc)

time for insert & delete?
depends on depth of node

set A of keys, for each $a \in A$ you get to choose the priority $p(a)$ to perform (a, p(a))
↳ just choose a random number as p(a) !!!
↳ uniform permutations!
↳ must be from a very large range

Randomized search tree for A is a TREAP with items $(a, p(a))$, $a \in A$

$p(a)$ i.i.d. rnv

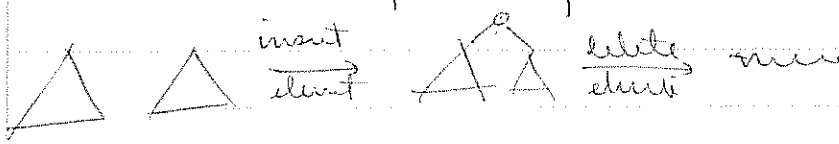
random numbers

RSTs behave the same way as BSTs with "random input" assumption

Thm In RST for n elements

Searches, insertions, deletions take $\Theta(\log n)$ expected time {expectation over random choices made by algorithm themselves}

How about joining/splitting TREAP? (Exercise)



CS170 Algorithms 20 Feb Tuesday

Last Time
① analyzed random b.s.t.
② randomized b.s.t. (we are generating randomness ourselves - no assump about Real world)

SORTING

Input: S (multi) set of items.
each item having a key
from an ordered universe

OUTPUT ① Sequence $a_1 \ldots a_n$ with the property $a_i.key \leq a_{i+1}.key \quad \forall\ 1 \leq i < n$
② $\{a_1, a_2, \ldots, a_n\} = S$ (kind of a joke)

Case: keys are from "bounded universe"
$U = \{0, \ldots, B-1\}$

Bucket Sort:

O(B) 1. Initialize each entry of an array $T[0..B-1]$ to empty set
O(N) 2. For each element $a \in S$ insert a into $T[a.key]$
O(B) 3. W = empty
For $i = 0$ to $B-1$
append $T[i]$ onto W

→ Total running time is $O(N+B)$

Space = $O(B)$

RadixSort:

assume keys are K-tuples of K digits $(d_{K-1}, d_{K-2}, \ldots d_0)$, where $d_i \in \{0, \ldots B-1\}$

lexicographic order $<_L$

$(d_0) <_L (d_0') \iff d_0 < d_0'$

$K > 1$

$$(d_{K-1}, d_{K-2}, \ldots, d_0) <_L (d'_{K-1}, d'_{K-2}, \ldots d'_0)$$

$$\iff \begin{cases} d_{K-1} < d'_{K-1} \\ \text{or } d_{K-1} = d'_{K-1} \text{ AND } (d_{K-2}, \ldots d_0) <_L (d'_{K-2}, \ldots d'_0) \end{cases}$$

correspond exactly

Can apply bucket sort recursively.

LexicoSort(S, K, i, B) S: set of n K-tuples with digits from $[0, \ldots B-1]$, $0 \le i < K$

(K-1 first time)

O(B) { 1. Initialize array T[0..B-1] to empty

O(n) { 2. For each $a = (d_{K-1}, d_{K-2}, \ldots d_i \ldots d_0)$ in S do insert a into T[d_i]

? { 3. If $i > 0$ then for $j = 0 \ldots B-1$ } do LexicoSort(T[j], K, i-1, B) }

O(B) { 4. W = empty; for j = 0 to B-1 do append T[j] onto W

5. return(W)
S = W

? /* Basically * sort according * to the i rt * most digits, * forget the * carriones * to the left */

Running time = ?

$O(K(n+B))$
$O(Kn+KB)$ (?)
↳ this will be $\underline{O(B^K)}$!

3. fixed $a \in S$
"touched" by algorithm $O(K)$ times
$O(1)$ times for each $0 \le i < K$

Look at the entire analysis again
Anyway its a bad way of doing it.

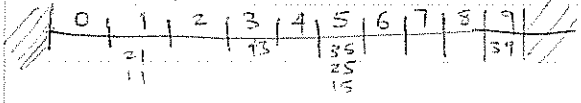
```
True RadixSort( S, K, B)
  W = S
  for i = 0 to K-1 do
    for j = 0 to B-1 do T[j] = empty
    for each a = (d_{K-1}, ..., d_0) in W in turn do
      insert a into T[d_i]
    W = ∅
    for j = 0 to B-1 do
      append T[j] onto W
  S = W
```

do any K times

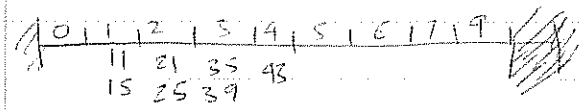
from before $O(n+B)$

same as bucketsort using i^{th} digit of the # as a key

Ex. $S = \{21, 35, 39, 25, 43, 15, 11\}$
$B = 10$; $K = 2$



$W = 21, 11, 43, 35, 25, 15, 39$



$= 11, 15, 21, 25, 35, 39, 43$

Running Time $O(K(n+B))$ Xtra space $O(B)$

a new non trivial exc.
- try to find an integer sorting algo that works in $O(n \log K)$

for radix sort { if $B = n$ sort integers in the range $0 \ldots n^K - 1$ in time $O(Kn)$

See if you can figure out the $O(n \log K)$ algo.

Assumption:

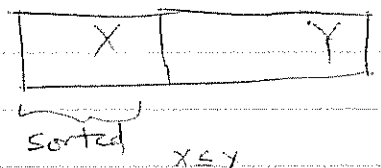
"insert a into $T[d_i]$"
↳ not const time possibly
must go along w/ this assumption
i.e. can index into an array in $O(1)$ time

Comparison-Based sorting
no restrictions on U
assume $x, y \in U$, can tell whether $x < y$ / $x = y$ / $x > y$ in $O(1)$ time

Selection Sort
Assume i/p is an array $A[1 \ldots n]$

$O(n^2)$ { For $i = 1$ to $n-1$
$O(n)$ { determine j $1 \le j \le n$ s.t. $\rightarrow i \le j \le n$
$A[j] \le A[k]$ $\forall k$ $1 \le k \le n \rightarrow i \le r \le n$
swap($A[i], A[j]$)

INVARIANT

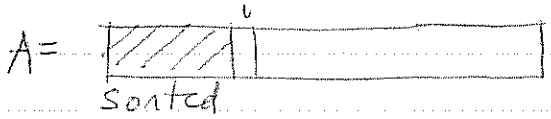


Ex. $\Theta(n^3)$ algo which is not purely wasteful
some found $\Theta(n^{\log n})$
used the opposite of ÷ & conquer
↳ multiply & subtract

Insertion Sort

For $i = 2$ to n
insert $A[i]$ into sorted
subarray $A[1], \ldots A[i-1]$

INVARIANT-



# of comparison $O(n^2)$
or $O(n \log n)$
if you use binary search

BUT the # of data moments is still $O(n^2)$

good for "almost sorted" arrays
if out by say 10/11 positions
$\Rightarrow O(kn)$

i.e. every element is out of its correct place by $\leq k$ positions

To make $O(n \log n)$ use AVL trees

CS170 Algorithms 22 Feb '90 Thursday

Manber p. 127-141

Sorting
BUCKET
RADIX

Comparison based
SELECTION
INSERTION
$O(n^2)$

MERGE sort $O(n \log n)$ [Can merge two arrays in/o extra sp. but probably non trivial]

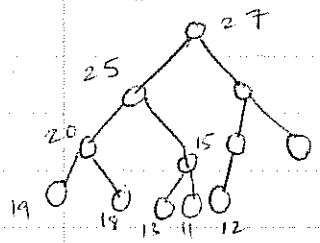
"IN SITU" c/o using extra memory (& O(c)/O(log n))

HEAPSORT:

max(Heap) n keys

build perfect binary tree on n nodes
- depths of leaves can differ by ≤ 1 (h, h-1)
- at most one non-leaf node has only one child
- depth of leaf is h-1 ⇒ all leaves to the right have depth h-1.

perf. Bin. Tree defn WRONG



defn HEAP

key(node) ≤ key(parent(node))

DeleteMax(H)

delete say 27 ⇒ move 12 up. swap 12 & larger child recurs

r = root(H)
x = value(r)
c = rightmost leaf(H) (@ bottom most level)
value(r) = value(c)
delete leaf c
REHEAP(r)

REHEAP(r)

if leaf(r) then return
if no_rightchild(r) then (⇒ ∘ - ∘)
if value(lchild(r))

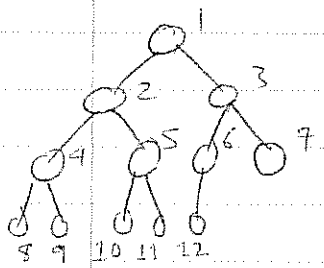
> value(r) then
swap(value(r), value(rchild(r)))
return
else (2 children)
if value(lchild(r)) >
value(rchild(r)) then
swap(value(r), value(lchild(r)))
REHEAP(lchild(r))
else
symmetric case

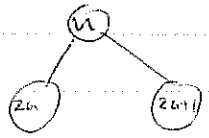
if value(lchild(r)) > value(r) then

There will be at most $2\lceil \log(n+1) \rceil$ comparisons!

How can you do this via no pointers?

IMPLICIT REP^N of PERFECT BIN TREE.





∴ can store in an array (∵ know indices of children)

"level numbering"

array T[1..]

root T[1]

parent T[i] = T[⌊i/2⌋]

lchild(T[i]) = T[2i]

rchild(T[i]) = T[2i+1]

leaf?(T[i]) → ∴ (2i > n)

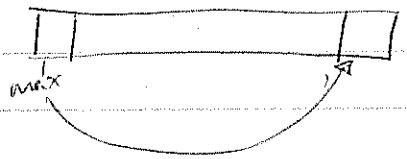
heap with n nodes

Rightmost leaf T[n] something is a leaf if it doesn't have a left child!

HeapSort T[1..n]

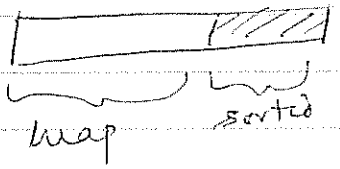
1 create maxheap in T } O(n)

2 for k = n downto 2 do
  swap(T[1], T[k]) ; O(1)
  reheap (T, 1, k-1) ; O(log n)



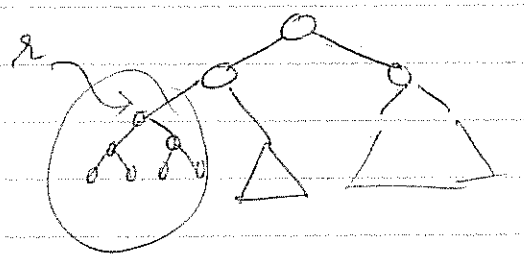
n times

↳ O(n log n)

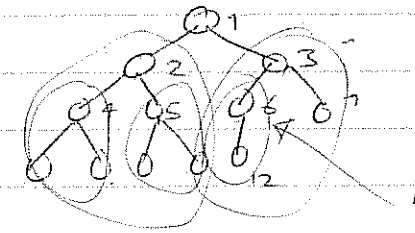


$\sum$ comparisons $\leq 2n\lceil \log_2(n+1) \rceil$

reheap (T, r, m)
r position in T
r < m



Create MaxHeap ; How?



$\lfloor n/2 \rfloor$ ← largest non leaf node

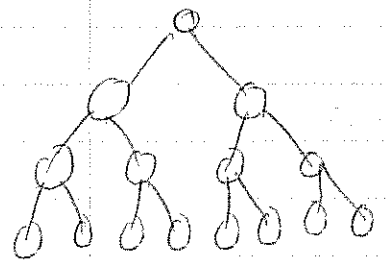
all childre have heap prop
⇒ reheap at parent

for $i = \lfloor n/2 \rfloor$ downto 1
    reheap(T, i, n)

→ $O(n \log n)$ ⟸ takes $< 2 \log n$ comparisons

actually will be even faster!
∵ low down tree depth a lot less, where most of the comparisons take place

assume $n = 2^h + 1$: totally perfect



level l	How many?	# of comp.
3	4	$2 \cdot 1$
2	2	$2 \cdot 2$
1	1	$2 \cdot 3$ (two comp @ each step)
l	2^{l-1}	$2(h-1)$

general

Total # of comp

$$\sum_{1 \le l \le h} 2^{l-1} \, 2(h-l+1)$$

$$= \sum 2^l (h-l+1)$$

$$n+1 = 2^{h+1}$$

$$2^l = \frac{(n+1)}{2^{h+1-l}}$$

$$= \sum_{1 \le i \le h} \frac{n+1}{2^{h+1-l}} (h+1-l)$$

$$= (n+1) \sum_{1 \le i \le h} i/2^i \quad \to \text{converge (ratio test)}$$

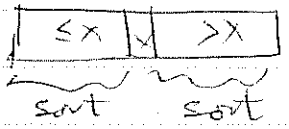
$$< C$$

$O(n)$

Exploiting Randomness : QUICKSORT

D-C T[1..n]

1. Get "middle" element x
   at most half of keys $\le x$
   " " " " " $> x$



2. partition T into pieces $\le x, > x$ } O(n) plausible

3. rec. sort each piece

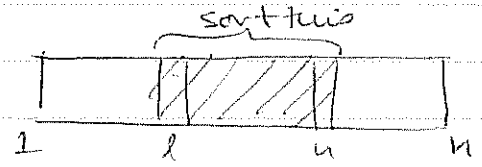
$$R(n) = O(n) + 2R(n/2) \quad ; \text{ ignoring } 1$$

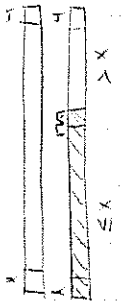
$\hookrightarrow O(n \log n)$

▷ Tough so randomly pick one!
∵ Random ⇒ ½ prob of being in middle 50%

QUICKSORT (T, l, u) T[1..n] array
$1 \le l \le u \le n$

1. If $l \ge u$ then return
2. Pick a random i,
   $l \le i \le u$
   let $x = T[i]$
3. "pivot" on x & return m largest index s.t. $T[i] \le x \ \forall\ i \le m$
   swap(T[i], T[m])





4. Quicksort (T, l, m-1)
   Quicksort (T, m+1, u)

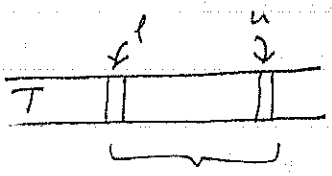
CS170 — Algorithms — 27/2/90 Tuesday

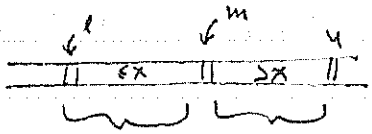
Manber pp 127-145

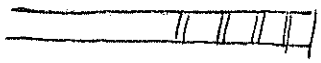
Last Time: heapsort in situ $O(n \log n)$

QUICKSORT (T, l, u)

1. if $l \geq u$ then return
2. pick random i, $l \leq i \leq u$
   at $x = T[i]$
3. swap (T[i], T[l])
   pivot on x & return m:
   swap (T[m], T[l])
4. Quicksort (T, l, m-1)
5. Quicksort (T, m+1, u)

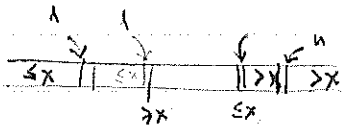




How do you prevent 

* Do the smaller subproblem first! C: order of no consequence
  ⇒ use only logarithmic stack space

```
PIVOT (T, L, U, X) : integer
    l = L; u = U
    while l < u do
        while l ≤ u and T[l] ≤ X do l = l+1
        while l ≤ u and T[u] > X do u = u-1
        if l < u then swap(T[l], T[u])
                      l = l+1
                      u = u-1
    return (l-1)
```



this loop moves l upto apt till > x

when collide ⇒ stop

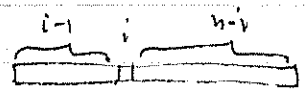
* # of comparisons in PIVOT = $u - l + 1$ = "n"

"n"+1

$R(n)$ = expected no. of comparisons when applying quicksort to n elements.

(assuming all elements distinct)

$R(1) = 0$ (comparisons among the elements themselves)



(from pivot)

$$R(n) = (n+1) + \sum_{1 \le i \le n} \frac{1}{n} \left(R(i-1) + R(n-i) \right)$$

$R(i-1)$: left subproblem; $R(n-i)$: right subproblem, $j = n-i+1$

$$= (n+1) + \frac{1}{n} \sum_{1 \le i \le n} R(i-1) + \frac{1}{n} \sum_{1 \le j \le n} R(j-1)$$

$$= (n+1) + \frac{2}{n} \sum_{1 \le i \le n} R(i-1)$$

$$n R(n) = n(n+1) + 2 \sum_{1 \le i \le n} R(i-1)$$

consider the difference $nR(n) - (n-1)R(n-1)$

$$= n(n+1) + 2 \sum_{1 \le i \le n} R(i-1) - \left((n-1)n + 2 \sum_{1 \le i < n} R(i-1) \right)$$

$$= 2n + 2R(n-1)$$

$$\therefore \quad n R(n) = 2n + (n+1) R(n-1)$$

$$= 2 + \left(\frac{n+1}{n}\right) R(n-1)$$

$$R(n) = 2 + \left(\frac{n+1}{n}\right) \left[2 + \left(\frac{n-1}{n-1}\right) R(n-2) \right]$$

$$= 2 + \frac{2(n+1)}{n} + \left(\frac{n+1}{n-1}\right) R(n-2)$$

$$= 2 + \frac{2(n+1)}{n} + \frac{n+1}{n-1} \left[2 + \frac{(n-1)}{n-2} R(n-3) \right]$$

$$= 2 + 2\left(\frac{n+1}{n}\right) + 2\left(\frac{n+1}{n-1}\right) + \frac{n+1}{n-1} \frac{n-1}{n-2} \left[2 + \frac{n-2}{n-3} R(n-3) \right]$$

Discern a pattern

Pattern

$$2 + 2(n+1)\,{}^{1}/_{n+1} + 2(n+1)\,{}^{1}/_{n-1} + 2(n+1)/_{n-2} + \cdots + 2(n+1)\,{}^{1}/_{k} + \cdots + (n+1)\,{}^{1}/_{1}\,R(1-2)$$

Recall $R(0\,1) = 0$

$\therefore$ the last term is $+1 = 3$

$\therefore$ stops at $2(n+1)\,{}^{1}/_{3}$.

$= 2(n+1)\left(H_{n+1} - 1 - {}^{1}/_{2}\right)$

$= 2(n+1)\,H_{n+1} - 3(n+1)$; an exact expression

$\therefore$ Expected running time is $O(n \log n)$

/* End Sort */

SELECTION

Set S $x_1, x_2, \ldots x_n$ integer k $1 \le k \le n$

WANT: k^{th} smallest element of S

(i) Sort & choose k^{th} element $\Rightarrow O(n \log n)$

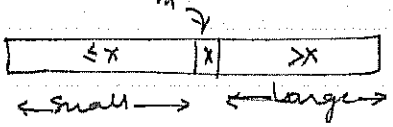
How about $\Theta(n)$?

when $k = 1$ linear

$k = 2$ linear

$k = {}^{n}/_{2}$?

Any Qsort: Pivot (don't need all of Qsort)



$k < m$ look at left part only

$k > m$ look at right part only

Select (S, k) (assume $1 \le k \le |S|$)

0. If $|S| = 1$ then return only el't in S
1. Pick random $a \in S$
2. partition S into $SMALL = \{x \in S \mid x \le a\}$

$LARGE = \{x \in S \mid x > 1\}$

(kind of wasteful!! since using an itre elem some place

3. if $K \le |SMALL|$ then return (SELECT(SMALL, K))
else return (SELECT(LARGE, K-|SMALL|))

$R(n,k)$ expected no. of comparisons made by SELECT(S,K) $|S|=n$ (assuming all elts distinct)

$R(1,1) = 0$

$$R(n,k) = n+1 + \sum_{k \le i \le n} \tfrac{1}{n} R(i,k) + \sum_{1 \le i \le k} \tfrac{1}{n} R(n-i, k-i)$$

Claim $R(n,k) \le cn$ for some constant c.
i.e. $R(n,k) = O(n)$

By induction

$$R(n,k) = n+1 + \frac{1}{n} \sum_{k \le i \le n} c \cdot i + \frac{1}{n} \sum_{1 \le i < k} c \cdot (n-i)$$

$$= (n+1) + \frac{c}{n} \sum_{1 \le i \le n} i + \frac{c}{n} \sum_{1 \le i < k} (n-i)$$

$$= (n+1) + \frac{c}{n}\left[\left[\frac{n(n+1)}{2}\right] - \frac{k(k-1)}{2}\right] + (k-1)n - \frac{k(k-1)}{2}$$

$$= (n+1) + \frac{c}{2}(n+1) + \frac{c}{n}\left((n-k)\cdot(k-1)\right)$$

max — find bd on this

$$f(k) = (n-k)(k-1) = -k^2 + (n+1)k - n$$

$$f'(k) = -2k + (n+1)$$

$$\max k = \frac{n+1}{2}$$

$$\therefore R(n,k) \le (n+1) + \frac{c}{2}(n+1) + \frac{c}{n}\left(\frac{n-1}{2}\right)\left(\frac{n-1}{2}\right)$$

$$\le (n+1)\left(1 + \frac{c}{2}\right) + \frac{c}{n}\left(\frac{n-1}{2}\right)^2$$

$$= n\left(\tfrac{c}{2}+1\right) + \left(\tfrac{c}{2}+1\right) + c/n \quad \frac{(n^2-2n+1)}{4}$$

$$\left(+\frac{cn}{4} - \frac{c}{2} + \frac{c}{4n}\right)$$

$$= n\left(\tfrac{3c}{4}+1\right) + \left(1+\tfrac{c}{4n}\right) \overset{?}{\leq} cn$$

If you choose c sufficiently large eq 8 works!

Expected running time of SELECT is $O(n)$

QUICKSORT Exp # of comps $2(n+1)H_{n+1} - 3(n+1)$
SELECTION " " " " $O(n)$
Want $O(n)$ comparisons in worst case

SELECT (S, k)
0. if $|S| = 1$ then return only el't in S
1. pick random $a \in S$
2. partition S into $SMALL = \{x \in S \mid x \le a\}$
$LARGE = \{x \in S \mid x > a\}$
3. if $k \in |SMALL|$ then return (SELECT(SMALL, k))
else return (SELECT(LARGE, $k - |SMALL|$))

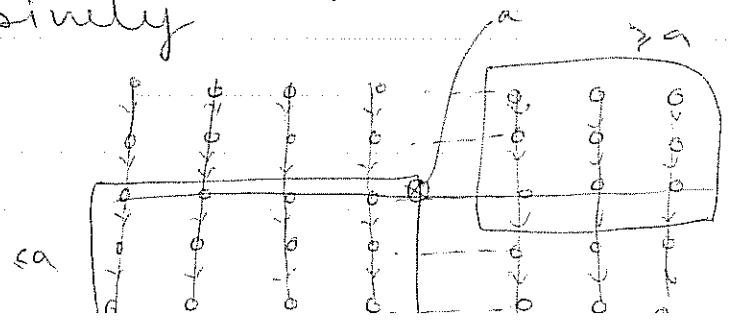
$O(n)$

IDEA: pick a good $a \in S$
Assume magically a chosen s.t. $|SMALL| \le \alpha |S|$
$|LARGE| \le \alpha |S|$
$\alpha < 1$
$\alpha = 99/100$

∴ Recurrance Relu is
$R(n) \le cn + R(\alpha n)$
claim $R(n) \le [c/1-\alpha]\, n = O(n)$

Method for finding a "good" a ((assume $|S| \ge 50$) (can ignore the small cases))

$O(n)$ (i) break S into $\lceil n/5 \rceil$ groups of at most 5 elements each
$O(n)$ (ii) sort each group (const time to sort 5 elements)
$R(n/5)$ (iii) let M be the set of medians from the groups.
find "a", the median of M.
↪ recursively



This guarantees
$|SMALL| \le 3/4\,|S|$
$|LARGE| \le 3/4\,|S|$

$R(n) \leq O(n)$ if $n \leq 50$

Looking at (3). $n > 50$ $R(n) \leq cn + R(n/5) + R(3/4\, n)$

claim $R(n) \leq 20cn$ works for $n \leq 50$ (work at log 64)

$R(n) \leq cn + 20cn/5 + 20c(3/4\, n)$

$\leq cn + 4cn + 15cn = 20cn$

$1/5 + 3/4 = 19/20$

20 come from $1/1-a = 1/1-19/20 = 20$

$1/3 + 3/4 \Rightarrow$ sum larger than 1 $\therefore$ won't work (choosing something larger than 5 will work too.)

(Blum, Floyd, Pratt, Rivest) 1970

LOWER BOUNDS

PROBLEM || ex. S set of n keys. Determine the largest key in S.

Q. How many comparisons between keys in S must *any* algo perform?

```
S ⊂ {1, ..., m}
For i = 1 to m do T[i] = 0;
For each x ∈ S do T[x] = 1;
max = 0
For i = 1 to m do
    if T[i] = 1 then max = i
return(max);
```

Not a single key comparison!

∴ Any is slightly shaky

MODEL of computation
only operations on keys are
store (copy) key
compare two keys
Previous algo. was from a diff model of computation. [used key as an integer]

Claim: Any "comparison based" algo to find the maximum must perform at least $n-1$ comparisons.

(1) Every key, except for max must have "lost" a comparison.
(2) In every comparison only one loses.

Problem: S Set of n keys. Determine the largest & smallest key

Q: How many key-key comparisons must any comparison based algo. make?

Adversary argument:
game: algo asks questions
$x_5 \geq x_6$ adversary answers; the answers must be consistent
adversary attempts to force the algo to ask many questions.
"strategy for the adversary"

strategy: key in S can be in one of four states
N: key was never involved in any comparison
W: key won at least one comparison

L : key lost @ least one comparison, it never ~~lost~~ won

WL : key won some & comp & lost some comp.

comp x : y	Response	New States	Info Gain
N N	x > y	W L	2
W N	x > y	W L	1
WL N	x > y	WL L	1
L N	x < y	L W	1
W W	x > y	W WL	1
L L	x > y	WL L	1
W L	x > y	W L	0
WL L	x ? y	WL L	0
W WL	x > y	W WL	
WL WL	answer consistently	WL WL	

Max, Min $N-1$ DISTINCT keys must have lost some comparison, $N-1$ DISTINCT keys must have won some comparison.

$2 \times (N-1)$ pieces of info.

If N men can make only $N/2$ comp. of the sort where both x & y are untouched

Remaining $N/2$ pieces of info must be gained by 1 comparison each.

$\Rightarrow$ at least $N/2$ comp + $(N-2)$ comp each

$\therefore 3N/2 - 2$ comp must be done

CS170 ALGORITHMS March 6

O(n) deterministic SELECTION - (Pratt)

Lower Bounds for Problems

B lower bound for problem P
⇒ for each algo. A that solves P ∃ an input X
s.t. A (when) would run on i/p X takes at least B steps.

✓ ∀ each algo ∃ i/p

✗ NOT ∃ i/p X ∀ algo A s.t. algo A run on X takes ≥ B steps

✗ NOT ∀ algo ∀ i/p.

class of algos
model of computation
primitive computational steps — COMPARISONS

eg. max of $\{x_1, x_2, x_3 \ldots x_7\}$

If general then B(n), x_A of size n
also for infinitely many n (not all n, nor all $n \geq N$)

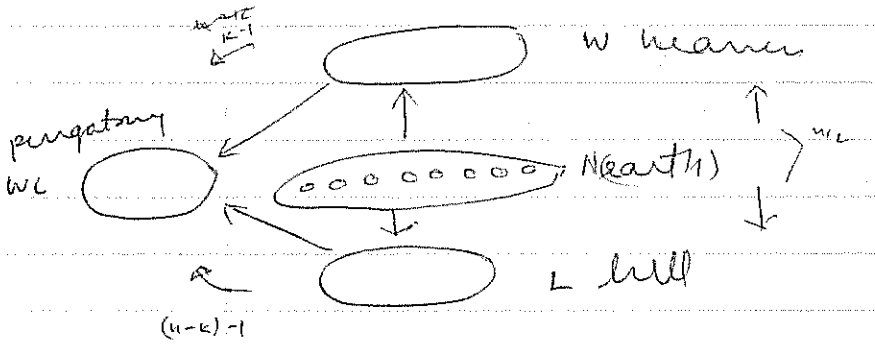
Adversary argument: "run algo A & construct "bad i/p" x_A on the fly"

Game: Algo asks questions of the form
"is $x_i > x_j$?"
the adversary answers consistently
Goal of the adversary: force algo to ask many questions

New: Adversary strategy: rules for how to answer Qs
Accounting: proof that adversary will require the algo to ask "many" Qs.

Find the smallest & largest of n keys

Last time

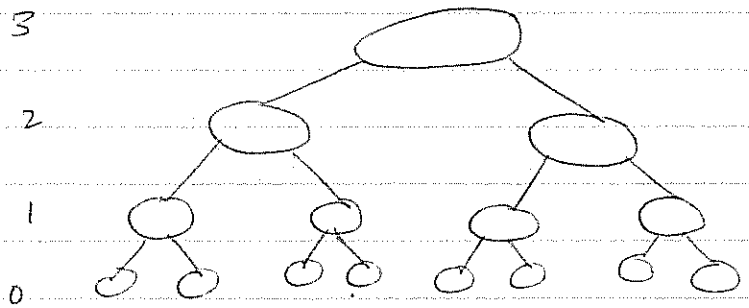


@the end one in heaven
one in hell

So keep as many from this

Problem: Sorting 2^{14} keys

level algorithms - deterministic & comp.-based



Each x_i is stored at some node of T.

~~BINARY TREE~~

x_i - - $N(x_i)$ node where x_i is stored

&

INVARIANT: $N(x_i)$ left relative of $N(x_j)$ then $x_i < x_j$

$N(x_i)$ is an ancestor of $N(x_j)$

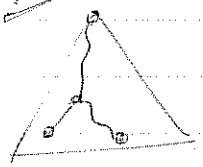
$\Rightarrow$ it is unknown whether $x_i \gtrless x_j$

(node is its own ancestor. Also more than one key in a node is allowed)

A node N at level i can hold at most 2^i keys in the subtree rooted at N

initially all keys in root node

relations



Algo must move all keys to the leaves.

Rules for answering questions $x_i \lessgtr x_j$?

(Rule 1) $N(x_i) = N(x_j) = N$ assume $x_i < x_j$
move x_i to the l.c. of N, x_j to r.c. of N

(Rule 2) $N(x_i)$ is left relative of $N(x_j)$
$\Rightarrow x_i < x_j$ assumed
↳ Dumb Algo because not giving away anything

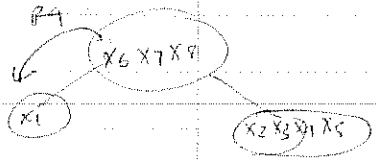
R3 $N(x_i)$ is STRICT (i.e. not same) ancestor of $N(x_j)$

& $N(x_j)$ is in left subtree of $N(x_i)$
Then Assume $x_j < x_i$ & move x_i to r.c. of $N(x_j)$

[& symmetric]

Also examine ~~&~~ invariant cond: a node N at level i can't hold at most 2^i keys in the subtree rooted at N.

∴ R4 as soon as one node fills up, throw down all in ancestor on the other side



↳ In his words

R4 " if subtree of node N at level i contains 2^i keys, move all remaining (& future) keys stored in parent of N to sibling of N"

Algo has to force adversary to move all keys to leaves.

Claim: Algorithm needs at least $n/2$ comparisons to move all the keys one level down in tree

Lvl (3) k levels ∴ sorting $8 \times 8 \Rightarrow 3 \times 8/2 = 12$

in general $n = 2^k$

$\Rightarrow$ need $\frac{n}{2} \log n$ or $k \frac{n}{2}$

↑ # of levels

Heap sort $2n \log_2 n$

asymptotically can't do better

HW-4 : TUESDAY
Prob 1 can be handed in ≈ HW-5

Quicksort ≈ median finding → O(n)
is thus O(n log n)

Last time: lower bounds

∀ algo A ∃ i/p x_A : A takes ≥ B steps on x.

↓

comparison based (when arithmetic ⇒ new complexity)

adversary argument

game: algo asks Qs $x_i < x_j$?
adversary has to answer consistently

Sorting $n = 2^k$ keys
Showed any comparison based "sorting" algo must make at least $n/2 \log_2 n$ comps.

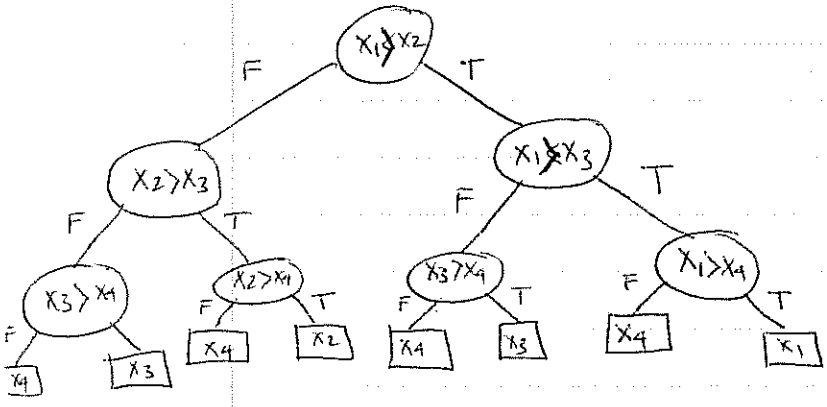
INFORMATION THEORETIC LOWER BOUNDS

Model of comp: Decision Tree
same as comparison based deterministic
fixed input size

Ex: find the maximum of four elements

```
imax = 1
for i = 2 to 4 do
    if X[i] > X[imax] then imax = i
return (imax)
```

What if variables used to store the a_i, not array?



Computation in a decision tree starts at root proceeds to a leaf.
Path taken depends on outcome of comp's

"running time on i/p $x_1 \ldots x_n$" = length of root leaf path being followed.

"worst case running time" ≡ height of tree.

Want: large height

lemma: Binary Tree ≤ height k has at most 2^k leaves

corollary: If a Binary Tree has N leaves then it must have height at least $\geq \lceil \log_2 N \rceil$

Decision Tree: leaf ←yields (corresponds to)→ possible result.

Problem has $\geq N$ many possible results
⇒ decision tree must have many leaves

CS 170

$\Rightarrow$ decision tree has large height $\lceil \log_2 N \rceil$

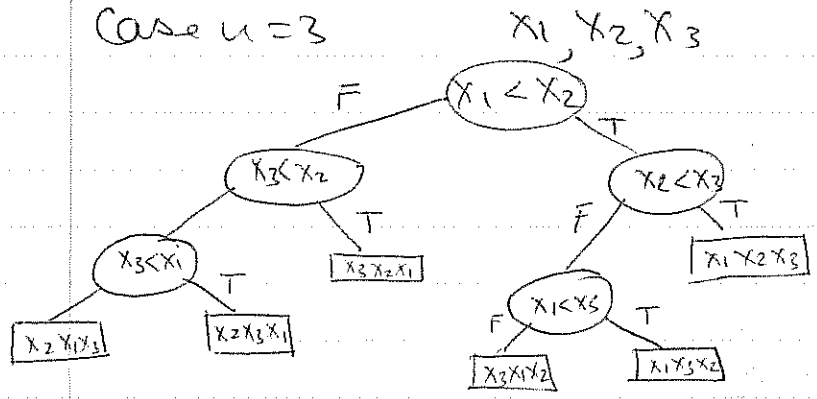
Problem: Find the max of 1000 keys

1000 possible outcomes
$\therefore$ decision tree has @least 1000 leaves
$\Rightarrow$ decision tree has height @least $\geq \lceil \log_2 1000 \rceil$
$\Rightarrow$ comp based algo must make at least 10 comp. $\lceil 10 \rceil = 10$

not very exciting

Prob Sorting 1024 keys
How many diff results

Case $n=3$



Outcomes $\Rightarrow$ Permutations

$\therefore$ 1024! outcomes

$\therefore$ ht of any decision tree for sorting is at least
$\lceil \log_2 1024! \rceil$
in general $\lceil \log n! \rceil$

Any decision tree for sorting n keys has height at least $\lceil \log_2 n! \rceil$

$\Rightarrow$ For any comparison based algo. for sorting 'n' keys $\exists$ an i/p which forces the algo to make at least $\lceil \log n! \rceil$ comps

$n! \cong \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$\log_2(n!) = \frac{1}{2}\log_2 2\pi + \frac{1}{2}\log n + n\log n - n\log e$

$\Rightarrow$ @least $n\log_2 n - 1.44\ldots n + \frac{1}{2}\log_2 n + 1.3287$

Still unknown whether this bd can be achieved
(has been done ~~better~~ at this upto 17)

EOF (Midterm)

Algorithmic Paradigm

- brute force search (trial & error)
- incrementally (induction)
- divide & conquer
- Dynamic Programming

Dynamic Programming

$\int \frac{dcabin}{cabin} = \log cabin + C$

log length l
$n-1$ specified location

log to be cut in n pieces

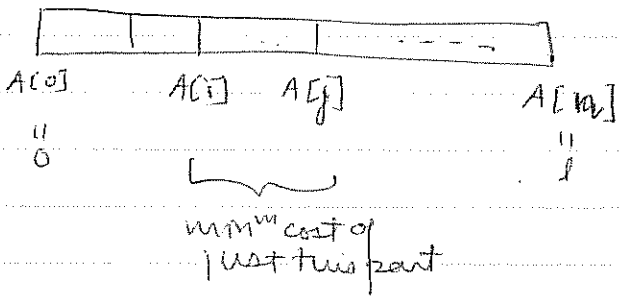
cost of cut = length of piece being cut

min^m cost cutting seq.

we want cost (min^m cost cutting sequence)

Glass seq

General




```
COST(i,j)
    if i+1=j then return(0)
M = ∞
    for k = i+1 to j-1
        M = min(cost(i,k) + cost(k,j))
    return(M + A[j] - A[i])
                 length
```

assume $j-i=m$ (# of primitive pieces)

$$T(m) = m + \sum_{1\le h<m} [T(h)+T(m-h)] \quad ; \quad T(1)=1$$

($(m+1)/(m-1)$)

$$= m + 2\sum_{1\le h<m} T(h)$$

recall Qsort (then $\frac{1}{2}n^2 \le$)

$$T(m) - T(m-1) = m + 2\sum_{1\le h<m} T(h) - (m-1) - 2\sum_{1\le h<m-1} T(h)$$

$$= 1 + 2T(m-1)$$

$$T(m) = 1 + 3T(m-1)$$

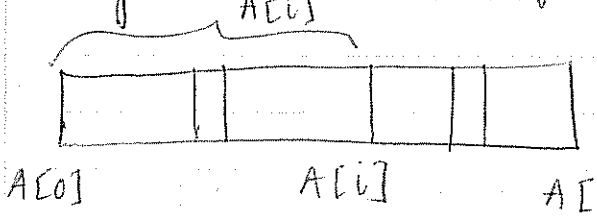
$$\ge 3T(m-1) \Rightarrow O(3^m)$$

n pieces
n^2 cuts
don't repeat all these computations
remember results

INFORMATION THEORETIC LOWER BOUND-

$\lceil \log_2 N \rceil$ $\lceil \log_2 n! \rceil$

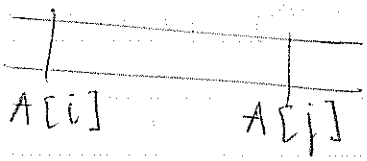
Dynamic Programming



cost of cut = length of piece being cut

Cost(i,j) cost of min cost sequence to cut piece between A[i] & A[j] into "atomic" parts

- if $i+1 = j$ return [0]
- $M = \infty$



- for $k = i+1$ to $j-1$
  $M = \min\{M, \text{cost}(i,k) + \text{cost}(k,j) + (A[j] - A[i])\}$
- return(M)

$$T[l] = l + 2 \sum_{1 \le i < l} T(i)$$

$> 3^{l-1}$ exponential

Cost(i,j) can be called for @ most $O(n^2)$ diff pairs (i,j)
For every (i,j) actual computations should be performed at most once, result should be remembered & never recomputed.

Array C[0..n, 0..n] initialized to undefined

new algo:

if C[i,j] ≠ 'undef' then return(C[i,j])

& replace every return(·) by {C[i,j] = (·); return C[i,j]}

how often do we go thru loop?

* { for k = i+1 to j-1
m = min{m, cost(i,k) + cost(k,j) + (A[j] - A[i])}

Invariant!
∵ change to itself.

(1) loop(*) is executed for each i, j at most once.
each time there are atmost O(n) iterations
↳ corresp to the min value

∴ excluding recursive calls time spent in (*) is $O(n^3)$

(2) The ops take O(1) time per invocation

How many invocations are there overall.

1 C(0, n)
2 invocations in body of loop(*)
This body is executed $O(n^3)$ times (sec 1)

⇒ There are $O(n^3)$ invocations
each invocation takes const time
⇒ $O(n^3)$

How to obtain the actual opt. cut sequence

$B[0..n, 0..n]$

$B[i,j] = k$ if the opt. cut sequence for "obliterating" piece $A[i]..A[j]$ has the first cut at position $A[k]$

* { for $k = i+1$ to $j-1$

$M = \min(M, \ldots$

$\Rightarrow$ [whenever M changes then $B[i,j] = k$

```
Bestseq(i,j)
    if i+1 = j then return
    k = B[i,j]
    output k
    Bestseq(i,k)
    Bestseq(k,j)
```

BOTTOM UP APPROACH:

Compute the optimal cost of cutting "short" pieces first, then compute the optimal cost for cutting larger & longer pieces

O(n) for $i = 0$ to $n-1$ do $C[i,i+1] = 0$

for $l = 2$ to n do

  for $i = 0$ to $n-l$ do

    $M = \infty$

    for $k = i+1$ to $i+l-1$ do

      $M = \min\{M, C[i,k] + C[k,i+l] + (A[i+l] - A[i])\}$

    $C[i,i+l] = M$

return $(C[0,n])$

$l = j-i$ .. "length"

i .. left end pt

$i+l$ right end

EDIT DISTANCE BETWEEN STRINGS

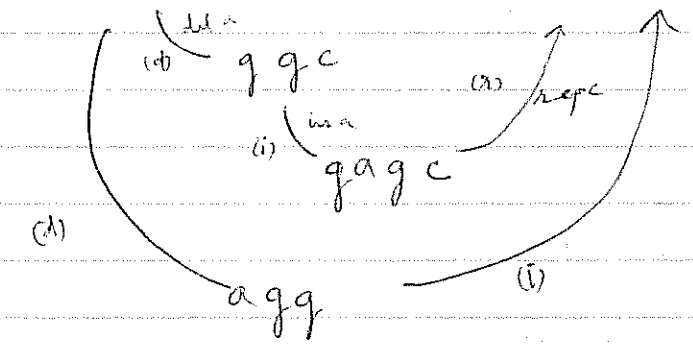
$A = a_1 a_2 \ldots a_n \qquad B = b_1 \ldots b_m \qquad a_i, b_j \in \Sigma$ (finite alphabet)

"How far" are A & B apart?

How many primitive operations are needed to transform A into B.

(i) inserting a char
(ii) delete a char
(iii) replace a char by a diff one

eg $A = agqc \qquad B = qaqg$



acgt
DNA seq.

$A(i) = a_1 a_2 \ldots a_i$

$B(j) = b_1 \ldots b_j$

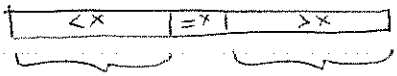
$C(i,j)$ .. $\overset{\min}{*}$ of operations necessary to transform $A(i)$ into $B(j)$

What happens to a_i in opt transformation.

case 1: a_i is deleted ⇒ can put off till end
~~case 2~~: $A(i-1)$ transforms to $B[j]$ then delete a_i

case 2: a_i is inserted to match b_j
$A(i-1)$ transforms to $B[j-1]$, then replace by char to b_j

5. Quicksort (& median as pivot)



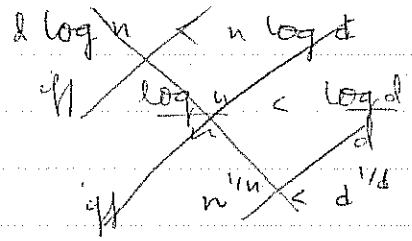
$$T(n,d) = \begin{cases} O(1) & n \in C \\ O(n) + \max\{T(n_1,d_1) + T(n_2,d_2)\} \end{cases}$$

↖ must assume worst case

$n_1 \le n/2 \quad n_2 \le n/2$

$d_1 + d_2 = d - 1$

worst when $d_1 = d_2$ (concavity arg)



$x^{1/x}$ is an increasing function for $x > e$

EDIT DISTANCE:

$A = a_1 \dots a_n$
$B = b_1 \dots b_m$

delete
insert
replace

$A(i) = a_1 \dots a_i$
$B(j) = b_1 \dots b_j$

want: min # of operations to transform $A \to B$

Claim: In optimal edit sequence for transforming A into B the individual operations commute.

(i.e. wouldn't delete something you had inserted)

equiv. class of all opt. transf. from A to B

same class true between: inserts, deletes, replaces } as long as they involve diff chars

But in opt seq no char changed twice

⇒ There is an optimal transformation from A to B with all changes occuring in left to right order, in particular the last change is rightmost.

What change could happen last?

Case 1: last change to left of a_n is it must be that $a_n = b_m$

Case 2: last change involves a_n

Case 3: last change to the right of a_n

Case 1 ⇒ ~~Last~~ must ~~remove a_n~~ ! transform $A(n-1)$ to $B(m-1)$

Case 3: (must be of form insert b_m)
⇒ transform $A(n)$ to $B(m-1)$

Case 2-1 a_n is deleted

Case 2-2 a_n is replaced
⇒ must be replaced (by b_m)

Case 3 : must be of form b_m

$C(i,j)$ the min^n # of ops to transform $A(i)$ to $B(j)$

can know if i or $j = 0$ then return (j)
if $j = 0$ then return (i)

$$M = \min \begin{cases} C(i,j-1)+1 & ; \text{case 3} \\ C(i-1,j)+1 & ; \text{case 2-1} \\ C(i-1,j-1) + \text{unequal}(i,j) & \text{case 1, case 2.2} \end{cases}$$

exp time
many times
too many calls

$$\text{unequal}(i,j) = \begin{cases} 1 & a_i \neq b_j \\ 0 & a_i = b_j \end{cases}$$

return (M)

Dyn. Prog. Paradigm

array $T[0..n, 0..m]$ initialized to 'undef'

if $T[i,j] \neq$ 'undef' then ret $T[i,j]$

if $i = 0$ then $T[i,j] = j$ ret (j)

if $j = 0$ then $T[i,j] = i$ ret (i)

$M = \min \{$ $\}$

$T[i,j] = M$ ret (M)

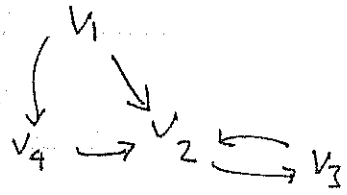
$O(mn)$

GRAPH THEORY [Manber ch7]

$G = (V, E)$ V: set of vertices, E: set of edges

$E \subseteq V \times V$

for $v_i, v_j \in V$ $(v_i, v_j) \in E$



Representation:

Adjacency matrix - Size = $|V| \times |V|$

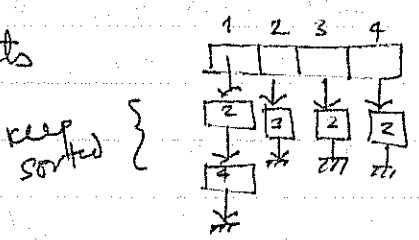
	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	1	0	0
4	0	1	0	0

$A[i,j] = 1 \Leftrightarrow (v_i, v_j) \in E$

$= 0$ otherwise

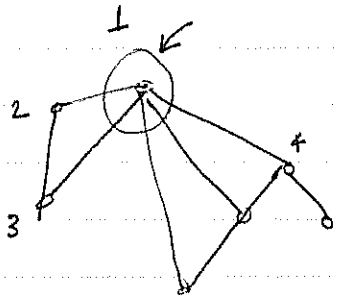
Drawback: Sparse matrix ⇒ too much space

Adjacency list -
Build an array of lists



Space = $|V| + |E|$;

Undirected graph ⇒ $|V| + 2|E|$; large $|E|$ ($\sim |V|^2$) then not an adv.



valid undirected graph

- Depth First Search

DFS (G, v) ← prevents cycles

preWork → mark (v) ; try to avoid getting in circles

for every edge (v, w) in E

if w not marked then { DFS (G, w)

may have to work here → return

} postwork

Search: visiting all nodes

order in which we traverse nodes depends on the initial vertex.

initial one is root.

↳ Recall ~~searching~~ traversing tree.

Connected Graph:

G is connected if $\forall u, v \in V$

there exists a path between u, v.

Path p in G is a sequence of vertices $\langle v_1, v_2 \ldots v_k \rangle = p(v_1, v_k)$

s.t. no vertex is repeated ["simple path"]

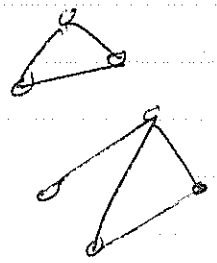
DFS visits all vertices if G is connected

one recursive call for each vertex.

$\therefore O(|E| + |V|)$ = O of E plus V when connected

Connected components

A graph



→ two connected components

DFS on (1) then start from a new

```
connected_components(G)
    compNum = 1
    while there exists an unmarked v ∈ V
        DFS(G, v) with preWork: v.comp = compNum
        compNum = compNum + 1
    return;
```

component in which v lies.

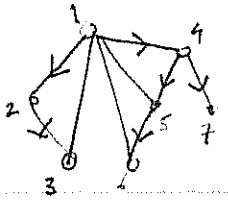
$T_{DFS} = O(|V| + |E|)$

will stick to connected now ∵ we can generalize to this from unconnected.

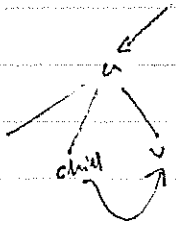
```
DFS_Numbering(G, V)
    DFS_Num = 1
    DFS(G, V) with preWork: { v.dfn = DFS_Num
                            { DFS_Num = DFS_Num + 1
```

```
DFS_Tree(G, v)
    DFS(G, v) with postWork: put (v, w) in TreeEdgeSet
```

Tree : connected acyclic graph



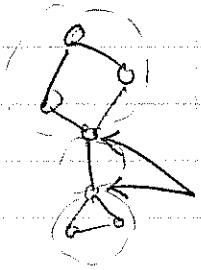
Good to get a tree out of the graph.
every edge in the graph either a tree or a back edge!
DFS_Tree partitions E into tree edges(T) & backward edges(B)
(u,v) Assume u is visited before v



Imposing an order on vertices!

BICONNECTED COMPONENTS:
G is biconnected if $\forall u,v \in V$ $\exists$ paths $P_1(u,v)$, $P_2(u,v)$ s.t P_1, P_2 have no common vertices except uv.

Biconnected component: Maximal subset of E s.t. the induced subgraph is biconnected.



Can have common vertices after splitting graph into biconnected components.

Articulation point: Vertex whose removal disconnects the graph

Edges that are in a cycle belong to the same biconnected component

Cycle: Path whose first & last vertices are the same
Eg $\langle v_1, v_2, \ldots v_k, v_1 \rangle$

for any two edges $e \neq e' \in E$ can prove e & e' belong to the same biconnected component
$\Leftrightarrow$ they belong to the same cycle
To find bicon. comp need articulation points
DFS to identify artic pts as well as traverse

Graph Algo: cont Manber Ch7 BB Ch 6

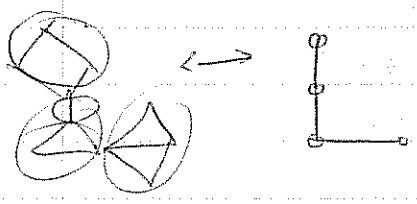
Defn: A biconnected graph is a graph if for any $u, v \in V$ ∃ two disjoint paths between u & v

Defn: A biconnected component is a maximal set of edges s.t. the induced subgraph is biconnected

Defn: Articulation point is a vertex whose removal disconnects the graph

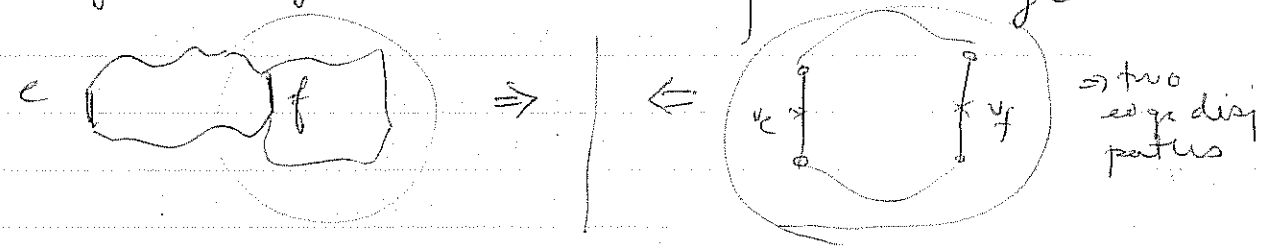
Each graph has a corresp biconnected tree

each node ↔ biconnected comp
each edge ↔ art pt



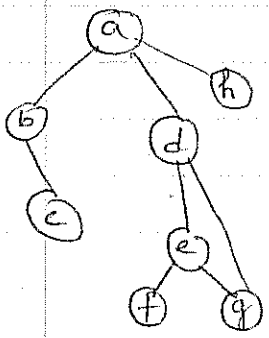
Claim: Two edges are in the same biconn comp ⇔ there is an edge containing them

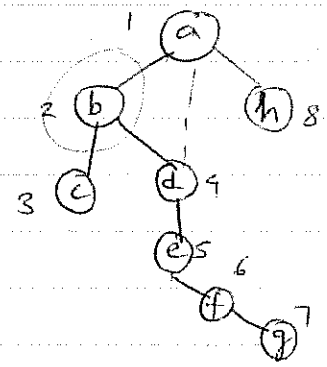
Pf: If ∃ a cycle containing two edges



Claim: Each edge is contained in exactly one biconnected component

Tree has upto $|V|$ edges
Need $O(|V| - |E|)$ ~~edge~~ ops.





b, c

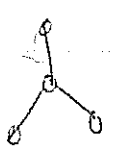
v.Number: the * we get from DFS

v.Lowest: least DFS # that a desc. connects too in one step.

If w is a child of v & w.Lowest $\geq$ v.number, then v is an articulation point

Idea: search for articulation points.
use a stack to keep vertices
Doing 3 DFS essentially. (could do in 1? C?)
DFS(G) {this numbers vertices}
BC(v)
insert v into stack
v.Lowest = v.number
for all (v,w) st w ≠ parent
if w.number > v.number {w is a child}
BC(w)
if w.Lowest ≥ v.number then
remove all vertices from stack until v is reached ⇒ bi comp
put v back into stack
v.Lowest = min(v.Lowest, w.Lowest)
else v.Lowest = min(v.Lowest, w.Number)

look at not! still works



A node can be an artic pt more than once
Thats why we put back on stack

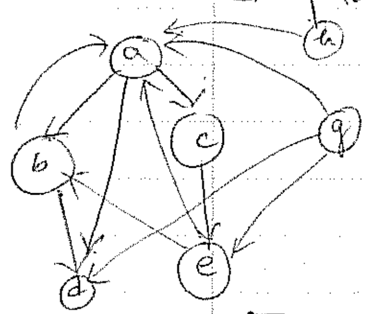
a b c d e f g h

1 2 3 4 5 6 7 8 ← v. number

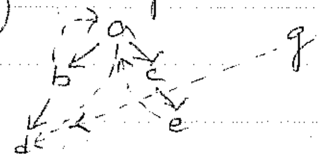
tam		
a	a	1
ab	b	1 2
abc	c	1 2 3
abcb [bc]	ab	1 2 3
	d	1 2 3 4 → 1

← Build on this

Depth First Search in Directed Graphs



(u,v) is a directed edge means that v is adjacent to u.



← Get Forest even though graph was "connected"

DIRECTED D.F.S.

Tree Edges (a,b) (b,d)

Forward Edge (a,d) 'an edge to a desc. which isn't a child'.

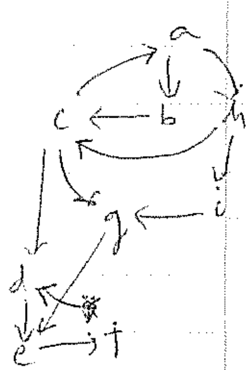
Back Edge (b,a), (e,a)

Edges from a node to an ancestor

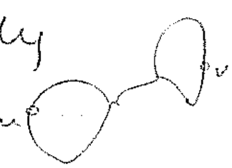
iff from undirec → Cross Edges (g,d) (e,v (?)) Edges between unrelated nodes

Defn: A graph is a strongly connected if for any vertices v,u ∈ V ∃ a path from v to u & from u to v

Defn: A strongly connected component is a maximal subset of vertices st the induced subgraph is strongly connected.



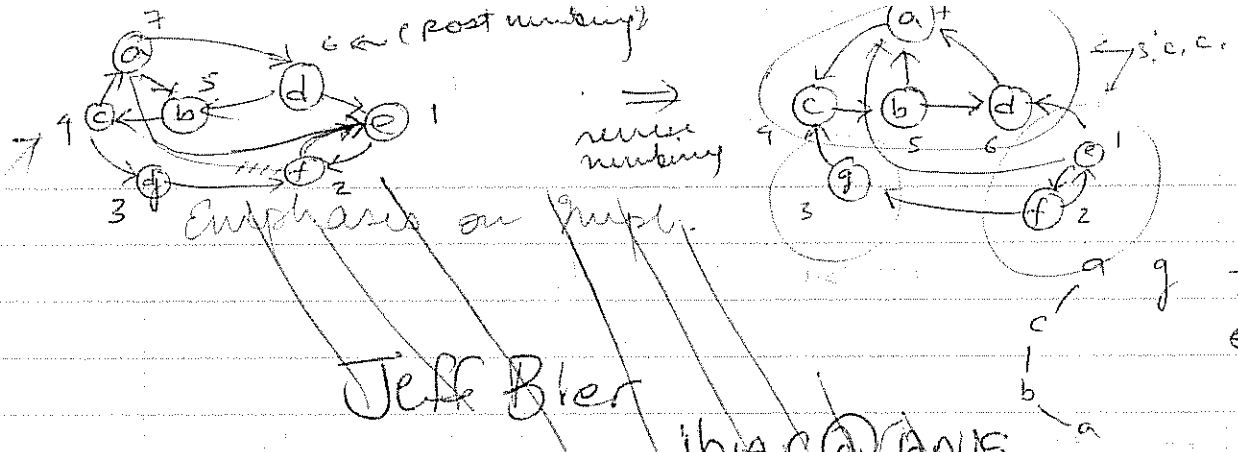
Claim: If u, v are in the same strongly connected component ⇔ there is a connected circuit containing them



Claim: Each vertex is contained in exactly one strongly connected component.

Nodes of the strongly connected comp. graph ⇔ comp. edges ⇔ edges between comp

Strongly Con Comp Graph is Acyclic



Jeff Blier
jblier@janus

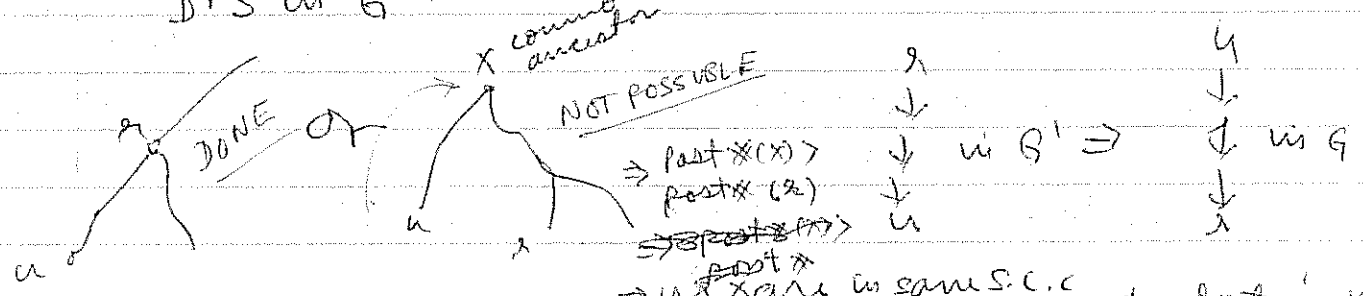
1. Use directed depth first search to get post number vertices (i.e. number v the last time you see it)
2. Reverse all the edges of G and get G'
3. Directed DFS on G', always starting with the vertex with the highest post number
4. Trees are exactly the strongly connected comp!

Pf of Algos correctness

⇒ u, v end up in the same tree after DFS of G' the root is r.

u ≠ r ∴ post number (r) > post number (u)

DFS in G:



∴ r is an ancestor or r is to the right of u.

∴ assumption: G is "connected" i DFS of G gives a tree which spans G.

in which case, postnum (x) > postnum (y) ⇒ x is ancestor of y in G i.e. x reaches G.

now if u, r end up in same tree after DFS of G' the root must be r (w.o.l.g) in G.

DFS undirected graphs — bicon comp
directed graphs — strongly con comp

Topological Sorting:

DAG - directed acyclic graph
= no directed cycles

eg ancestry graph
task precedence task A can't start unless B complete

Topological sorting Prob: DAG (V,E) $|V|=n$
number vertices in V from 1 to n so that for all "arcs" $(v,w) \in E$ label(v) < label(w)

(only for pictures)

Claim: In a DAG there always exists a source (vertex) of indegree 0.
charac of a DAG (the said straightforward)

Idea of Algo: Find a source, label it, remove it from graph; iterate

$n = |V|$ $e = |E|$

(2n many ways label a DAG; hard)

1. Input: Graph (V,E)
   determine indegree of every vertex — $O(n+e)$
   for each $v \in V$ do indegree(v) = 0
   for $v \in V$ do
   for each arc (v,w) in out(v) do
   increment indegree(w)

2. Find all "sources" — $O(n)$
   SOURCES = $\emptyset$
   for all $v \in V$ do if indegree(v) = 0 then
   insert v into SOURCES

```
BEGIN
  initialize S to be empty;
  visit, mark, and stack v;
  WHILE S is nonempty DO
    WHILE there is an unmarked vertex w adjacent to Top(S) DO
      visit, mark, and stack w
    END {while there is an unmarked vertex};
    pop S
  END {while S = nonempty};
END {DFS}
```

3. Iterati

at most once for each vertex $\therefore O(n) + \sum_{v \in V} O(\text{outdegree}(v)) = O(n+e)$

$O(1)$ { $O(\text{indegree}(v))$ { $O(1)$ }

```
Global = 0
while sources ≠ 0 do
    choose and remove some vtx v from SOURCE
    increment Global
    label(v) = Global
    for each arc (v,w) in out(v) do
        decrement indegree(w)
        if indegree(w) = 0 then insert w into SOURCES
if Global < |V| then graph was not acyclic
```

/*Also DFS then: back edges ⇒ cycle*/

Overall $O(n+e)$ is Linear Time.

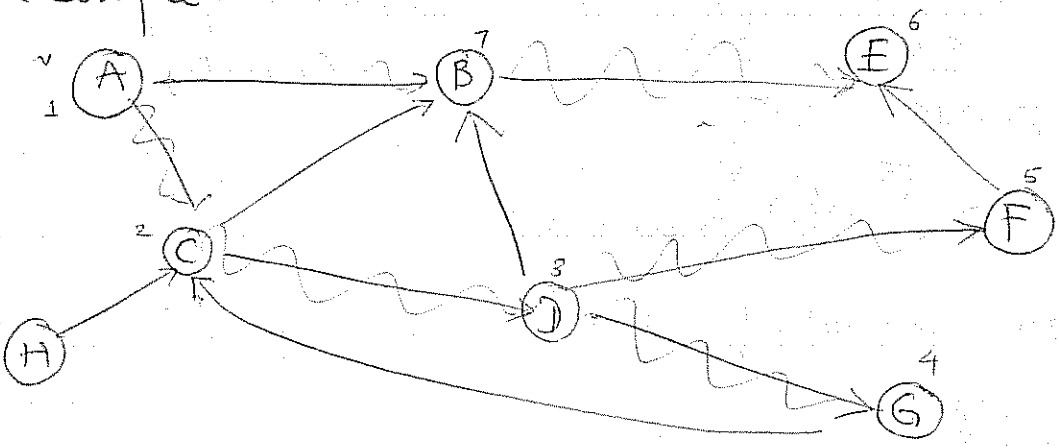
DFS revisited: graph (V, E) vertex $v \in V$
want: all vertices that are reachable from v.

```
for all x ∈ V do label(x) = ∅ - 1
Global = 1
  label(v) = 1
  STACK = ∅
    for each arc(v,w) in out(v) do put w on STACK; label(w) = 0
    while STACK ≠ 0 do
        choose and delete w from top of STACK
        increment Global
        label(w) to Global
        for each arc (w,u) in out(w) do
            if label(u) ≤ 0 then
                put u on STACK
                label(u) = 0
```

BFS

```
BEGIN
  initialize Q to be empty;
  visit and mark v; insert v into Q
  WHILE Q nonempty DO
    x := RemoveFromQ(Q);
    FOR each unmarked
        vertex w adjacent to x DO
        visit and mark w;
        insert w in Q
    END{for};
  END {while Q nonempty};
END {BFS}
```

Example:



E
G
F
C
B

STACK

What if we used a Queue instead of a stack?

A labeled 1 B labeled 2 C lab 3 E lab 4 D lab 5 F lab 6 G lab

Queue B C

B C E D F G Queue empty!

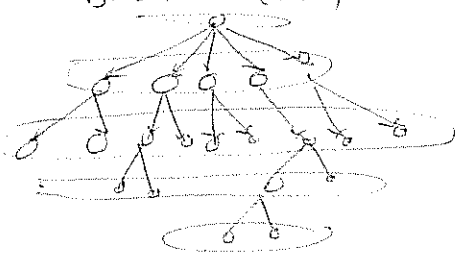
C beginning of Queue

Different tree results (∵ BFS) when Queue

BFS tree: (The ~~one~~)

in the directed case tree looks like

BFS tree (Dir)



can lends exist

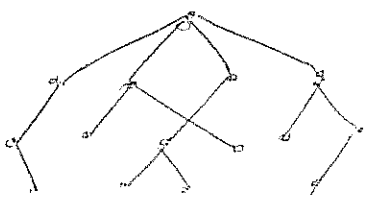
"downward edges"

"go over only one level"

BFS tree (Undir)

a nonex edge over two levs (adj level / same level)

If you ✱ levels



(assuming unit distances)

Using BFS one can solve following problem in time $\Theta(|V| + |E|)$

graph (V, E) $v \in V$

for each $w \in V$ determine the shortest (directed) path from v to w

Question: what if edges have different "lengths"?

directed graph (V, E)

with edge weights $weight(e)$ ... non neg reals

"length" of a directed path = sum of edge wts along the path.

"distance" from v to w shortest length of any directed path from v to w.

SINGLE SOURCE SHORTEST PATH PROBLEM

given $v \in V$ for all $w \in V$ we want the distance from v to w.

Iterative DFS

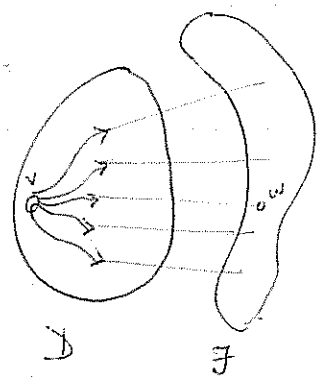
$\forall w \in V$ label$(w) = 0$
Global = 1; label(v); stack = $\emptyset$
$\forall (v,w) \in$ out(v) push w on STACK
while STACK $\neq \emptyset$ do
  pop w from top of STACK
  if label$(w) = 0$ then
    Global++; label(w) = Global
    $\forall (w,u) \in$ out(w) push u on STACK

single source shortest paths [Manber p. 204, BB p 87]
(d, G, v)

$wt(u,v) \in \mathbb{R}_+$, length $P = \sum_{e \in P} wt(e)$, dist$(v,u) = \min_{P(v,u)} \{\text{length } P\}$

DONE: vertices w for which we've calculated $w$dist
FRINGE: vertices w : $w \notin$ DONE & $\exists u \in$ DONE : $(u,w) \in E$

induction: Have found shortest paths to w; $\forall w \in$ DONE
find shortest path to some $u \in$ FRINGE



step: Pick $w \in F$: u.dist + $wt(u,w)$ = min$_{x \in D, y \in F}$ {x.dist + $wt(x,y)$}

Assume $\exists P(v,w) = \langle v \ldots d f \ldots w \rangle$ $d \in D$, $f \in F$
length$(P) <$ u.dist + $wt(u,w)$
$\Rightarrow$ d.dist + $wt(d,f) <$ u.dist + $wt(u,w)$
CONTRAD.

$O(|V|)$ Algorithm: $\forall u \in V$ do $\{u.done = false; u.dist = \infty\}$

$v.dist = 0$

$O(|V| + \log|V|)$ create Minheap, $\forall u$ u.loc point to location in HEAP

while $\exists$ undone vertex

pick w st w.dist is min among the undone vertices ← update HEAP $O(|V| \cdot \log|V|)$

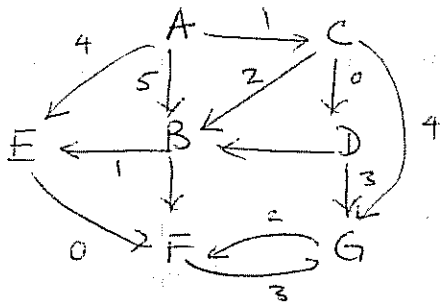
w.done = true

$\forall (w,z) \in out(w)$ do

if z.done = false

$z.dist = \min\{z.dist, w.dist + wt(w,z)\}$ update → $O(|E| \log|V|)$

$T = O(|V| + |E|\log|V|)$



A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
✓	5	1		4		
	3					5
	2					4

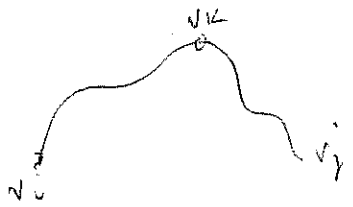
(constructing paths is trivial: just store prev.)

All pairs shortest paths:

dynamic programming $T = O(|V|^3)$

$P^k(v_i, v_j)$ = path from v_i to v_j w/ intermediate vertices in $\{v_1, \ldots v_k\}$

$P^k(v_i, v_j)$ { uses v_k: $P^{k-1}(v_i, v_k) + P^{k-1}(v_k, v_j)$
doesn't use v_k: $P^{k-1}(v_i, v_j)$



distance: $D^k(i,j) = \min\{D^{k-1}(i,j), D^{k-1}(i,k) + D^{k-1}(k,j)\}$

Maintain a table D

$[|V| \times |V|]$ prev. ex

$$\begin{bmatrix} 0 & 5 & 1 & \infty & 4 & \infty & \infty \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix} \begin{matrix} D^0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ D^{|V|} \end{matrix}$$

update table for each k.

Alg For k=1 to |V| do
For i=1 to |V| do
For j=1 to |V|
$D[i,j] = \min\{D[i,j], D[i,k] + D[k,j]\}$

Last Time: Shortest Paths
Single Source
all pairs

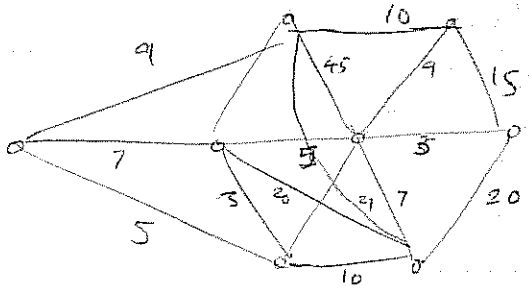
HWK6: Next Tue

MINIMUM SPANNING TREES
$G = (V, E)$ undirected
T Spanning Tree of G
minimal connected subgraph of G that contains all of V

edge weights (= costs = lengths)

Min Spanning Tree of G: a spanning tree of G s.t. the sum of lengths of its edges is minimal.

Application:



Lemma: $G = (V, E)$ undirected, weighted
$\emptyset \neq W \subset V$
$T \subseteq W$ subset of edges of graph induced by W
W promising $\Leftrightarrow$ $\exists$ MST of G that contains T,

$\rightarrow$ $G = (V, E)$ undirected, weighted
$W \subset V$; T subset of edges with both endpts in W
T promising

$$E_W = \{(u,v) \in E \mid u \in W, v \notin W\}$$

57/60
70/75
39/60
70/75

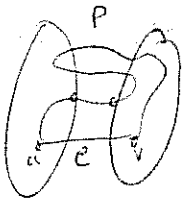
If e is an edge in E_W of minimal cost then $T \cup \{e\}$ is promising

Pf T promising
$\Rightarrow \exists$ min span tree $\bar{T}$ of G that contains T.
pick edge e of minimal cost in E_W.

If e is in $\bar{T}$ then $T \cup \{e\}$ is promising
e not contained in $\bar{T}$.

(u, v)
e W $\notin W$

$\exists$ a unique path p in $\bar{T}$ connecting u and v.
p doesn't contain e!

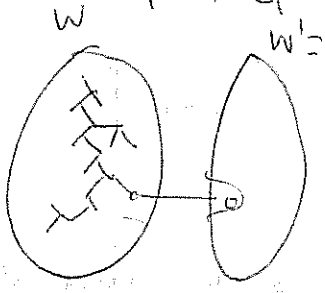


$\Rightarrow$ somewhere there is an edge $e' = \{u', v'\}$ with $u' \in W, v' \notin W$
$cost(e) \leq cost(e')$
consider $T^* = \bar{T} \setminus \{e'\} \cup \{e\}$

T^* is min span tree!
as $cost(T^*) \leq cost(T)$
$T \cup \{e\} \not\subset T^*$
$\Rightarrow T \cup \{e\}$ is promising QED

Idea for algorithm

increase the sets W & T from single vertex & $\emptyset$ to set of all vertices & spanning tree.

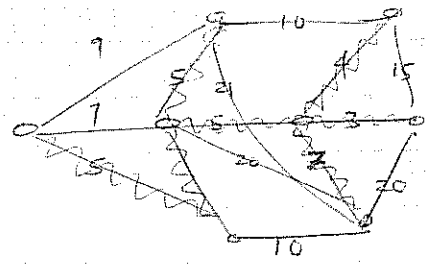


MST(V, E)
choose some vtx v from V
let W = {v} ; T = ∅
while W ≠ V do
among edges with exactly one end point in W
choose one of minimal weight
e = {w, w'} | w ∈ W
insert e into T
insert w' into W

(n-1) iterations

as many trees in a graph

Apply to example -



Loosely formulate

Assume graph is complete (can take ∞ using nulls)

MAINTAIN: vtx u in W' = V \ W
closest[u] vtx w in W
s.t. cost({u, w}) is minimal

(* find shortest edge *)
START at some arbitrary vtx by choosing an arbitrary vertex from V
then choose u ∈ W'
besttvtx = u
for each w' ∈ W' \ {u} do
if cost({w', closest[w']}) < cost({besttvtx, closest[besttvtx]})
then besttvtx = w'
insert {besttvtx, closest[besttvtx]} into T
insert besttvtx into W

O(n)

(* update closest[ ] *)
for each neighbor x of best vtx do
if x ∉ W and cost({x, besttvtx}) < cost({x, closest[x]}

O(n)

then closest[x] = best vtx.

$O(n^2)$ Prims algorithm

What if not $O(n^2)$ edges

Priority queue

INSERT (object, key)

DELETE (object) → every edge pts into datastructure!

DELETEMIN

(objects, keys)

HEAP or BALANCED SEARCH TREE

edges — lengths

Idea: Store all edges with exactly one endpoint in W in a priority queue PQ.

(* initialization *)

for each neighbor x of v do

insert ($\{v, x\}$, cost($\{v, x\}$)) into PQ

worst case $O(n \log n)$

$O(\deg(v) \log(\deg(v)))$

(* finding shortest edge *)

$e = \{u, v\}$ = DELETEMIN

add e to T

if $u \in W$ then insert v into W

else insert u into W.

$O(\log n)$ (worst case)

$\Rightarrow O(\log n)$ total $= O(n \log n)$

$e = v^2$

$\Rightarrow \log e = O(\log n)$ (!)

(* update PQ *)

For each neighbor x of p do

if $x \in W$ then delete DELETE($\{p, x\}$)

else INSERT($\{p, x\}$, cost($\{p, x\}$))

$\Rightarrow$ Total is $\sum_{p \in V} \deg(p) O(\log n) = O(e \log n)$ $\therefore$ entire algorithm $O(n + e \log n)$

ECS225b
CS170

DSP
Algorithms

April 12
~~April 12~~

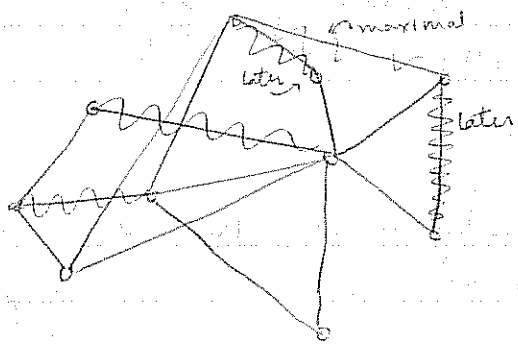
10 Moffitt
M 8-9 (Soda) 3-4
W 4-5 Soda
F 12-1

Tu between class
Th " "
W 2-3

Min Span Trees
$O(n^2)$
$O(\frac{e}{n} \log n)$

Matchings in Graphs

$G = (V, E)$ undirected graph
matching $M \subset E$
every vtx in V incident to at most one edge in M



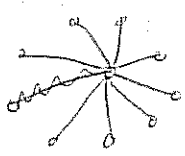
matching: maximal
maximum
perfect

maximal - cannot be extended i.e.
$\nexists$ matching M' in G st $M \subset M'$ $|M| < |M'|$
maximum - matching of largest possible size

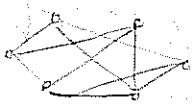
perfect - every vtx incident to exactly one edge of M.

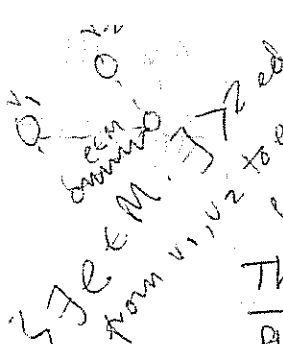
Does every connected graph have a perfect matching?

How does one find a maximum matching?



$G = (V, E)$ is very dense $\Leftrightarrow$ (i) $|V| = 2n$ (even # of vtcs)
(ii) every vtx in V has deg $\geq n$





Thm: Every graph has a perfect matching

Pf by construction

Assume M is a maximal matching for G and $|M| < n$

$\Rightarrow \exists$ at least two unmatched vtcs v_1, v_2

no neighbor of v_1 (or v_2) can be unmatched

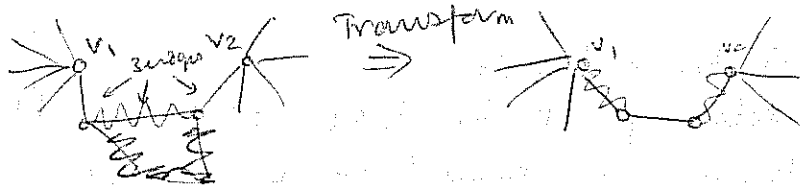
[Else could extend matching]

(because of maximality of M)

$\Rightarrow \exists \geq 2n$ edges emanating from v_1 & v_2 that end at matching edge in M

$|M| < n$

By pigeonhole $\Rightarrow \exists$ 3 edges emanating from v_1 or v_2 that end at the same matching edge

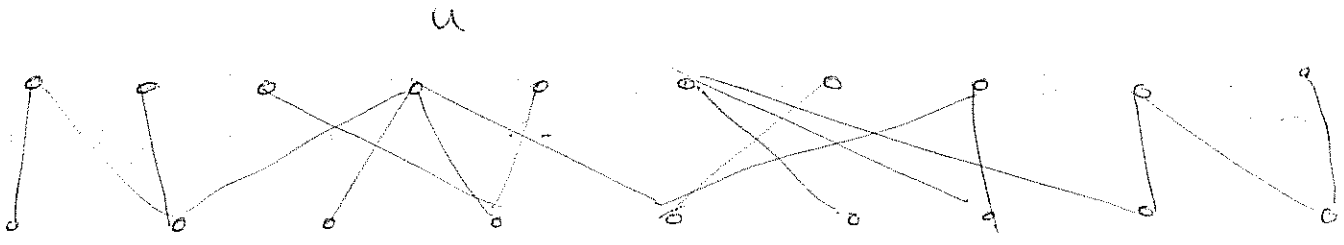


How can you do this efficiently? (Data Struc etc)

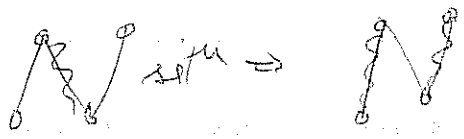
BIPARTITE GRAPHS $G = (V, E, U)$

vtx sd $V \cup U$ $\quad V \cap U = \emptyset$

all edges are of form $\{u, v\}$ $u \in U$, $v \in V$



Max # of Boy-Girl Pairs (7-7 relns)
Repairs — Repairman
edges ⇒ which job he can do
use ⇒ find maximum # of jobs

Look for  ⇒

once all v'tx's in one set are matched ⇒ can't do any better

examine this carefully

alternating path for M:
path from $u \in U$ to $v \in V$ both unmatched
along the path edges alternate between M and $E \setminus M$.

Claim: If an alternating path exists for M, then ∃ a matching M', $|M'| = |M| + 1$
("flip" all edges along alternating path)

a same as in G true for (?) graphs but much harder to prove

Thm: A matching M is maximum ⇔ there is no alternating path for M. || For Bipart

(A matching M is not maximum ⇔ there is an alternating path for M)

∃ alt path ⇒ M isn't maximum TRIVIAL (above)
(perform edge "flip" on alt. path)

Need to show: M is not maximum
$\Rightarrow \exists$ an alt. path

Let P be some maximum matching of G

$|P| > |M|$

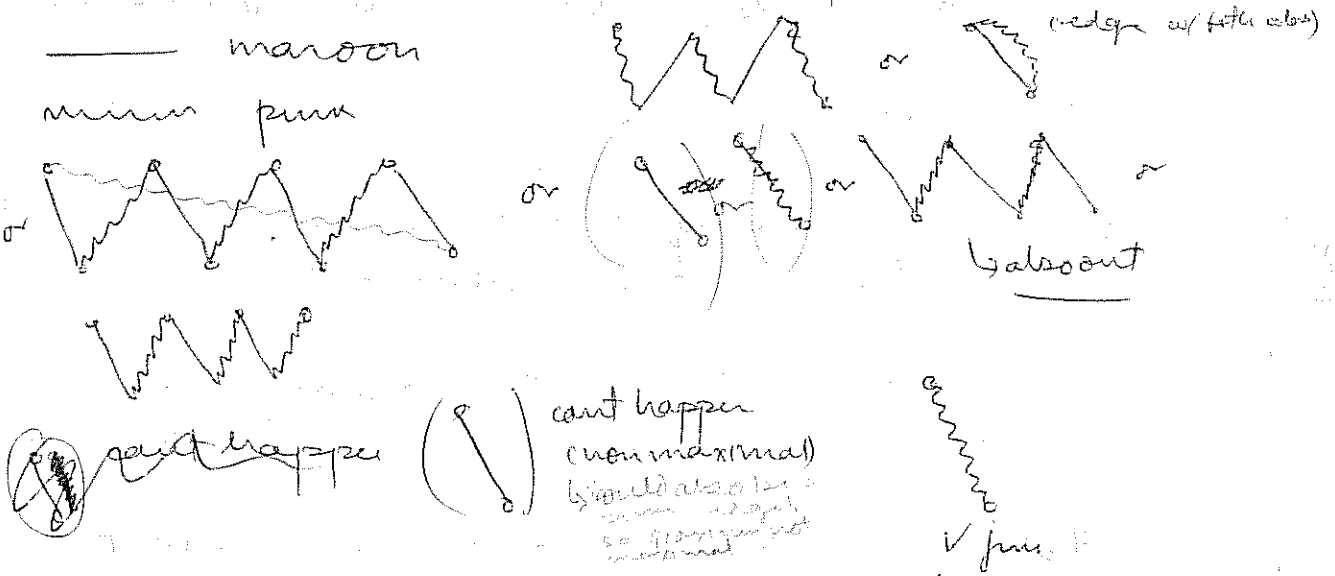
Consider graph induced by $P \cup M$

pink edges — maroon

not necc disjoint

in G' every vtx is incident to: at most ~~2~~ one pink edge
at most one maroon edge
$\Rightarrow$ every vtx in G' $\Rightarrow$ incident to at most two edges

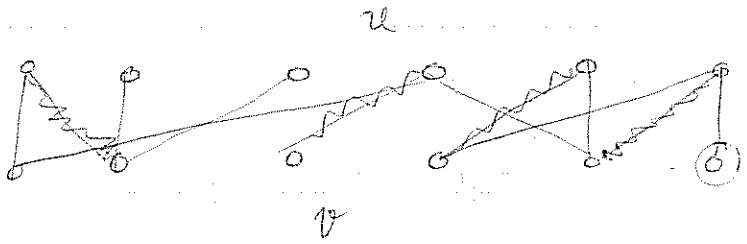
look at connected components of G'
$\Rightarrow$ cycles or paths only are possible



we must have a case like "V" there are more pink than maroon and in the rest of the configs there are equal #'s.

MATCHINGS

- very dense graphs
- bipartite graphs



largest possible matching ⇒ look at alternating paths

↳ Gives systematic way of enlarging the matching (& knowing when can't if no alt. path then done)

How to find alternating path

orient edges not in the matching to go from v to u & vice versa

(i) direct edges
(ii) do DFS starting at unmatched vtx in V.

$$\Theta(\underbrace{|V|}_{n}+|U|+\underbrace{|E|}_{e})$$

do these at most n times! (since increase matching by at least one each time!)

$$\Rightarrow O(n(n+e))$$

Can do better!

Do BFS level by level stop when you hit a layer of unmatched nodes. From the BFS tree can find alt. paths. Do flipping & repeat algo.

↳ $O(\sqrt{n}(n+e))$ using BFS
Hopcraft & Karp

- Best known

General Graphs: long time, is it in P?

- $O(n^5)$ Edmonds
- $O(\sqrt{n}(n+e))$ S. Micali V. Vazirani Karpes course cornell

REDUCTIONS

Want to solve problem A
We know how to solve (maybe similar) problem B.

could tweak algo for problem B.
How about Transforming A to problem B
(Reduce)

Ex. Cyclic Shift Problem
Input instance: two strings $\alpha = a_0 a_1 \ldots a_{n-1}$
$\beta = b_0 b_1 \ldots b_{n-1}$

Question: Is β a cyclic shift of α?
i.e. is there a k such that $b_i = a_{(k+i) \bmod n}$

cedar
dance

Can reduce this to Substring Problem (SP)

Input instance two strings $\gamma = c_0 \ldots c_{m-1}$
$\delta = d_0 \ldots d_{m-1}$

Question: Is δ a substring of γ?
Is there some λ s.t. $c_{\lambda+i} = \delta_i \ \forall \ 0 \le i < k$

Reduce cyclic shift problem (CSP) to SP
$\gamma = \alpha\alpha$
$\delta = \beta$

If there is an algo for solving SP in time $T(m,n)$
then there is an algo for solving CSP in time $T(2n, n)$
$+ O(n)$ ← transformation (reduction) time

CS170

Ex: $S_1, \ldots, S_k$ collection of sets

System of distinct representatives (SDR)

$r_1, \ldots, r_k$ $r_i \neq r_j$ if $i \neq j$

$r_i \in S_i$ for $i = 1 \ldots k$

$S_1 = \{1,2\}$ (SDR) $r_1 = 1$

$S_2 = \{2,3\}$ $r_2 = 2$

$S_3 = \{1,3\}$ $r_3 = 3$

$S_1 = \{1,2\}$ $S_4 = \{2,4\}$

$S_2 = \{2,3\}$ $S_5 = \{1,4\}$

$S_3 = \{1,2,3\}$

SDR!

Finding an SDR: Input instance $S_1, S_2, \ldots S_k$

Question: $\exists$ an SDR, and if yes, produce such a system.

Halls Theorem: An SDR exists for $S_1, S_2, \ldots S_k$

$\Leftrightarrow$ for every $\{i_1, i_2, \ldots i_m\} \subseteq \{1, \ldots, k\}$

$|S_{i_1} \cup S_{i_2} \cup \ldots S_{i_m}| \geq m$

Prob (i) 2^k subsets

(ii) not constructive

Finding a Bipartite Matching:

Input instance: Bipartite graph $G = (V, E, U)$

Question: produce a maximum matching.

Reduction of finding an SDR to finding a maximum bipartite matching.

form graph: $V = \{S_1, S_2, \ldots, S_k\}$

$U = S_1 \cup S_2 \cup S_3 \cup \ldots \cup S_k$ ← can do in linear time C's integers, indexed if R neg

$(v_i, u_j) \in E$ if $u_j \in S_i$

there is an edge $\{i, S_k\}$ $\Leftrightarrow$ $i \in S_k$

$M = |S_1| + |S_2| + \ldots |S_k|$

If there is an algo for finding a maximum matching in a bipartite graph in time $T(n,e)$, then there is an algorithm for "Finding an SDR" in time $T(k+|V|+M, M) + O(k)$

(figure out which matches) ↑ makes sense of solution

↑ time ↑ vertices ↑ edges

Ex. 3-clique Input instance $G=(V,E)$ undirected graph.

Question: Does G have a 3-clique
Are there 3 vertices in G so that any one is adjacent to two other ones? (is there a Δ?)

Alg (i) Check all $\binom{n}{3}$ subsets of 3 vtcs $O(n^3)$

Alg (ii) Form the adjacency matrix A

$A[i,j]=1$ if $(i,j)\in E$
$=0$ o.w.

} eigenvalues of A are int.

is $A[i,i]=1$ Square A to A^2 $\quad A^2[i,j]=\sum_{1\le k\le n} A[i,k]*A[k,j]$

$A^2[i,j]\neq 0 \Leftrightarrow \exists k$ s.t. $A[i,k]=1$ and $A[k,j]=1$

Determining whether G contains a 3clique amounts to checking whether there is a pair (i,j) s.t $A^2[i,j]\neq 0$ and $A[i,j]\neq 0$

(1) form A $\quad O(n^2)$
(2) form A^2 $\quad T(n)$
(3) check entries $O(n^2)$

If there is an algo. for squaring an $n\times n$ matrix in time $T(n)$ then is an algo for solving 3-clique in $T(n)+O(n^2)$

$O(n^{2.36})$ current record

→ examine converse
no algo for 3-clique in $O(n^2)$ ⇒ matrix Mult can't be in $O(n^2)$

CS170 Algorithms April 19, '90 Thursday

REDUCTIONS

Problem A reduces to problem B if there is an algo. for solving ~~an instance~~ of problem B in time $T(n)$ there is an algo for solving problem A in time

$$R(n) + T(n') + S(n')$$

$R(n)$ ... time for transforming an instance of prob A of size n to an instance of problem B of size n'.

$S(n')$ ... "making sense of this solution" i.e. time to transform soln for B to soln for A.

Last time - 3clique

$O(n^2) + T(n) + O(n^2)$ $\quad \therefore T(n)$: Mult $n \times n$

If there is no algo for solving problem A in time $R(n)+T(n')+S(n')$, then there is no algo for solving problem B in time $T(n)$.

(Trivial) Matrix Squaring reduces to matrix multiplication
Matrix Multiplication reduces to matrix squaring

A, B $n \times n$ matrices : Form $2n \times 2n$ matrices

$$\begin{pmatrix} 0 & B \\ A & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & B \\ A & 0 \end{pmatrix}^2 = \begin{pmatrix} BA & 0 \\ 0 & AB \end{pmatrix}$$

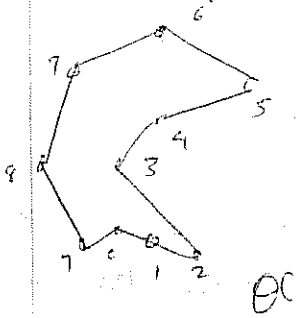
∃ an algo for matrix squaring i running time $T(n)$
$\Rightarrow$ ∃ an algo for multiplying 2 $n \times n$ matrices in time $T(2n) + O(n^2) + O(n^2)$

Ex: Simple Polygon Problem

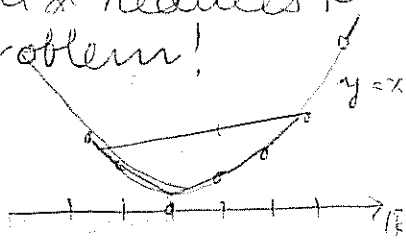
Input instance is a set S of n points in the plane each specified by pair of real #s.

Output an ordering $p_0, p_1, \ldots, p_{n-1}$ of the points that corresponds to a simple polygon w/ corners $p_0, p_1, \ldots p_{n-1}$ in CCW

all pts on line $\Rightarrow$ cost



Claim: Sorting n real # reduces to the Simple Polygon Problem!



T n real #s
$\Theta(n)$ form $S = \{(t, t^2) \mid t \in T\}$

Solve the simple polygon problem for S
$(p_0, p_1, \ldots p_{n-1})$

$O(n)$ determine k st $x(p_k)$ minimal
For $i = 0$ to $n-1$ output $x(p_{(k+i) \bmod n})$

If there is an algo for solving the simple polygon problem in time $T(n)$, then $\exists$ an algo for sorting n real #s in time $T(n) + O(n)$

Model of comp

? From which can't do SPP in better than $O(n \log n)$
$\exists$ no. algo. for sorting n reals in time $O(n \log\log n + cn)$

comp basics

Polynomial Time

$O(n^2)$ { $2n^2+3n+5$, $3n^2$, $2^{100000!} \cdot n^2$ }

If the running time of an algo. is $O(n^c)$ for some fixed const c,

polynomial ⇒ P

polynomial time algo
⇒ "efficient"

Problem tractable if ∃ a polynomial time algo to solve it.

P : class of all tractable problems
class of all problems for which ∃ a polynomial time algo

Ex : Sorting, searching, squaring matrices, bcon comp. matching ... just about everything we did

K-clique

Problem instance: input undirected graph G, integer K

Question: Does G have a clique of size K?

- not known to be in P

Decision Problems :

Problem for which solution is YES / NO

problem : Find out the size of the max. clique in a graph G.
↳ can do c "n" clique problems!

Extreme : is the 29^{th} bit of o/p 0 or 1?

Number SAT?

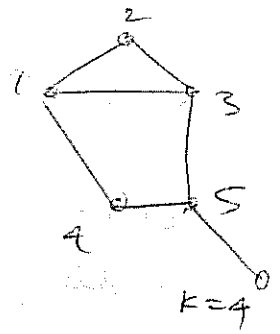
Think in terms of UNIX

Language Recognition Problem

view the i/p for a problem as a string of chars
language corresponding to problem is
set of all strings that are inputs which
produce yes answers.

Format for problem

K
n
1
2 adj. lists
.
.
.
n - - - - -



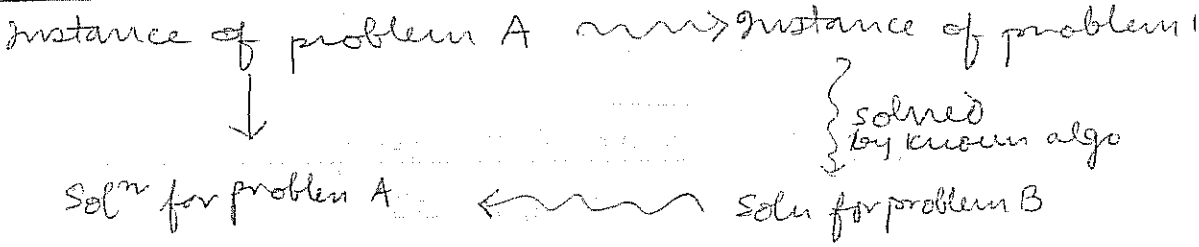
4 \n
6 \n
1: 2 3 4 \n
2: 1 3 \n
4: 1 5 \n
5: 3 4 6 \n
6: 5 \n E
F

no 4 clique ⇒ not in our language!

language ⟷ decision problem

Language L_1 reduces to language L_2 ⇔ ∃ an algo Δ that transforms strings into strings s.t.
$W \in L_1 \Leftrightarrow \Delta(W) \in L_2$

Reductions



DECISION PROBLEMS Yes/No Yes/No

One approach to DecProb: LANGUAGE RECOGNITION

language L_1 reduces to language L_2
$\exists$ an algo Δ that transforms strings w to strings $\Delta(w)$ s.t. $w \in L_1 \iff \Delta(w) \in L_2$

P: problems solvable in polynomial time.
class of languages $\forall$ which $\exists$ a polynomial time algo to decide if a string w is in the language or not

poly time
A is an algo that decides membership in L_2 } $\Rightarrow$
L_1 reduces to L_2

$\exists$ an algo B for deciding membership in L_1
given string w (1) Apply Δ to w to obtain $s = \Delta(w)$
Recall $w \in L_1 \Leftrightarrow s \in L_2$
(2) Apply A to s; if A answers yes, output yes else no.

$\Delta \to$? poly $\Rightarrow$ change are dyes a bit
L_1 polynomially reduces to L_2 if $\exists$ a poly time algo Δ that transforms strings w to strings $\Delta(w)$ s.t. $w \in L_1 \Leftrightarrow \Delta(w) \in L_2$

Lemma: (i) L_1 is poly reduc to L_2 and (ii) L_2 is in P $\Rightarrow L_1 \in P$

Pf: (i) $\Rightarrow \exists$ algo Δ $\exists$ a polynomial p
$w \in L_1 \Leftrightarrow \Delta(w) \in L_2$ w/ i/p N, Δ terminates within $p(|w|)$ steps

(ii) $\Rightarrow \exists$ an algo $\mathcal{A}$ $\exists$ a polynomial q, $\mathcal{A}$ on i/p s answers yes iff $s \in L_2$ - $\mathcal{A}$ takes time $q(|s|)$

Need to show: ~~$\exists$~~ polynomial time algo

i/p w

(1) apply Δ to produce $s = \Delta(w)$ — $p(|w|)$
(2) apply $\mathcal{A}$ to s — because $|s| \leq p(|w|)$, $q(|s|)$
o/p yes if $\mathcal{A}$ says yes
o/p no if $\mathcal{A}$ says no

$\therefore$ running time is $p(|w|) + q(|s|)$
$= q(p(|w|))$
is a poly w/t

(Composition) Lemma: (i) L_1 poly reducible to L_2 and (ii) L_2 poly-reduc to L_3

$\Rightarrow L_1$ poly reduc to L_3.
$w \in L_1 \Leftrightarrow \Delta(w) \in L_3$

$w \in L_1 \Leftrightarrow \Delta(w) \in L_2 \Leftrightarrow \Delta'(s) = \Delta'(\Delta(w)) \in L_3$

NON-DETERMINISM

k-coloring of a graph G: each vertex of G is assigned one of k colors s.t. no two adjacent vertices are assigned the same colour.

$G = (V, E)$

Find me a 3-coloring of G, and explain how you got it!

not the same

easier Convince me that G is 3 colorable

very hard

Prove to me that in every 3 coloring of G each color appears equally often

any given can convince w/ 3 coloring very fast with be one

nondeterministic algorithm: "usual primitives" (all CS170 so far except for randomness) and an nd-choice function.

"goto label1 or goto label2" ← small choice

A nondeterministic algorithm recognizes language L $\Leftrightarrow$ Given an i/p string x, it is possible to connect the nd-choices encountered during the execution of the alg into real choices s.t. the outcome of the algorithm will be to accept the i/p string x if and only if $x \in L$.

If $x \in L$ then there must be an execution sequence for the algorithm that leads to an "accept" outcome

how do you show NOT 3colorable! much harder

termination? how do you show NOT 3colorable!

If $x \notin L$ then there must be NO execution sequence for the algo leading to an "accept" outcome!

→ We aren't saying what's going to happen!

Example:

Problem: Is $G = (V, E)$ 3 colorable

Recognise the language of all strings that represent 3-colorable graphs

(1) For each vtx v of G do
goto label1 or goto label2 or goto label3
label1: color(v) = green ; next v
label2: color(v) = black ; next v
label3: color(v) = blue ; next v

(2) For every edge $\{x, y\}$ do if color(x) = color(y) then loop forever

output accept

nondeterministic algo A

running time of algo A for i/p $x \in L$
= length of shortest execution sequence that leads to an accept-outcome

running time of alg A for i/p of length n:
maximum running time of the algo A for string $x \in L$ with $|x| = n$.

nondeterministic polynomial time algorithm:
NP ---- class of languages that can be recognized by some polynomial time nondet algorithm.
"Guess a soln & verify in Polytime"
NP = P ?

CS170 Algorithms April 26 Thurs

languages – decision problems

HWK #8 due Tue May 8

polynomial time reductions.

L_1 poly-reduces to L_2
$L_2 \in P$
$\Rightarrow L_1 \in P$

Nondeterministic
nd-choice specify a finite list of choices "goto label1 or goto label2"

accepting a language L nondet alg A for L.

$w \in L \Rightarrow$ "there must be an accepting path in A"

$w \notin L \Rightarrow$ "there must be No accepting path in A"
↳ don't say anything about No inputs
maybe it says no, maybe it loops forever.

NP: class of problems for which a nondet algos runs in poly time
$\exists$ poly q
$w \in^q \Rightarrow$ "there must be an accepting path for A

of length at most $q(|w|)$

$w \notin L \Rightarrow$ "there is NO accepting path"

Nondet algo: Unlimited guesses in a way

"guess a soln & verify it to be a valid soln"

Ex k-colorability

Ex Instance Graph $G = (V, E)$ unlimited
integer k
Q. Is G not k-colorable?

Ex k-clique Instance graph $G = (V, E)$
integer k
Q: Does G have a k-clique?

Informal 1) Guess a subset S of vtxs of G of size k
2) For each pair $x, y \in S$, $x \neq y$
check that $\{x, y\}$ is an edge in G

Formal alg: Input $G = (V, E)$, int k
(1) $S = \emptyset$
for each vtx $v \in V$ do
goto label$_1$ or goto label$_2$
label$_1$: include v in S
next v
label$_2$: next v
(2) If $|S| = k$ then
for each pair $x \neq y$ in S do
if $\{x, y\} \notin E$ then loop forever
accept

Q: Is there no K-clique?

if Pr NP hard $\in$ NP
no Pr known s.t. co Pr is in NP

Ex Compositeness - The i/p is a binary string w
Q - does w represent a composite number?

n composite $\exists\ k_1\ k_2 \in \mathbb{N}\ \ 1 < k_1 < n\ :\ k_1 * k_2 = n$

Informal Algo : (1) Guess strings w_1 & w_2
(2) check $w_1 \neq 1$ & $w_2 \neq 1$
(2) compute the product p of the numbers represented by w_1 & w_2
(3) check that $p = x$ rep by w

(1) → guessing a string s:
string = "0"
Loop forever
goto label1 or goto label2 or goto label3
label1: append '1' to s
next
label2: append '0' to s
next
label3: exit loop

→ will generate a string

Ex : Primality input binary string w
Question: Does w represent a number that is not composite? (i.e. a PRIME?)

Lemma: (Number Theory) If n odd $n > 2$,

n prime $\iff$ $\exists\ x \in \mathbb{N}$ $0 < x < n$

(i) $x^{n-1} = 1 \bmod n$

(ii) for each prime factor p of $n-1$ $x^{(n-1)/p} \not\equiv 1 \bmod n$

(1) "guess an x"

(2) check that $0 < x < n$

(3) check that $(x^{n-1} \bmod n) = 1$

↑ simple: never blows up (mod)

(4) Guess numbers $p_1, p_2, p_3, \ldots p_k$

check that $p_1 \cdots p_k = n-1$

for each p_i recursively apply algo to check that p_i is prime

(5) for each p_i check that $(x^{(n-1)/p} \bmod n) \neq 1$

good randomized algos

Compositeness – the problem not hard but finding the factors is very hard.

Lemma (1) lang L_1 poly red to L_2

(2) L_2 is in NP

Then $L_1 \in NP$

Defn: A language L is NP-hard (problem X) iff every language (problem) in NP is polynomially reducible to L (X).

Defn: A problem X is NP-complete iff

(i) $X \in NP$

(ii) X is NP hard

Interesting: if X is NP-complete & $\exists$ a poly time

deterministic algo for X then P=NP.
if then there is a polynomial time deterministic algo for every problem is in NP.

Y in NP

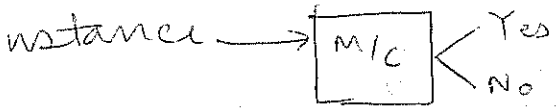
Lemma Problem X is NP-complete if
(i) X belongs to NP
(ii) there is some NP complete problem Y, s.t Y polynomially reduces to X

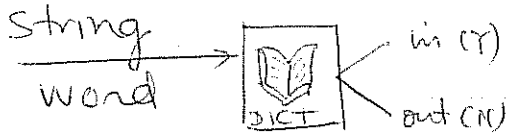
Pf: Need to show that X is NP hard, i.e every problem Z in NP is polynomially reducible to X.

⇒ { (i) Z poly reduces to Y (as Y NP complete)
(ii) Y poly reduces to X
(iii) Z poly reduces to X

QED

Decision Problems ~ languages





Examples: Does G have an HC?

SAT: vars $x_1 \ldots x_n$ clauses $c_1 \ldots c_m$

$c_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ (3 CNF form)

can always put B. exp in 3CNF!

$\exists$? an assgn $\phi = \bigwedge_{i=1}^{m} c_i$ (Garey-Johnson)

s.t. ϕ = TRUE? f: Poly Transform to 3CNF

Decision Problem $\xrightarrow{\text{poly}}$ Search Problem

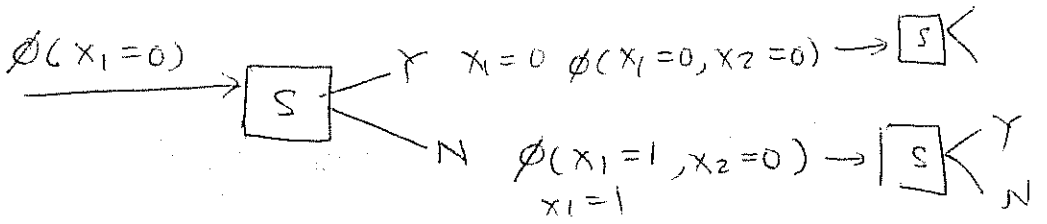
$\exists$? SAT find SAT assgn

But can show $\geq$! ($\xleftarrow{\text{poly}}$)

i.e. both equally hard!

Pf assume there is an algo S for ~~search~~ decision SAT

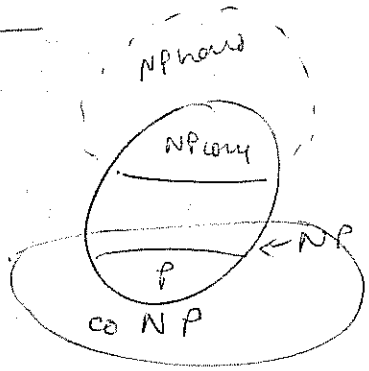
Want to construct an algo for search SAT



$T(\text{Search}) =$
$(\text{Total } \# \text{ of Search Prob}) \times (T(S))$

$= O(n)$ Tim S
$\Rightarrow O(n)$ poly $\rightarrow$ poly
$O(n)$ exp $\rightarrow$ exp

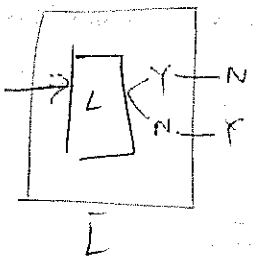
Prob π is $\in$ NP-hard
if $\forall L \in NP \quad L \leq_{poly} \pi$

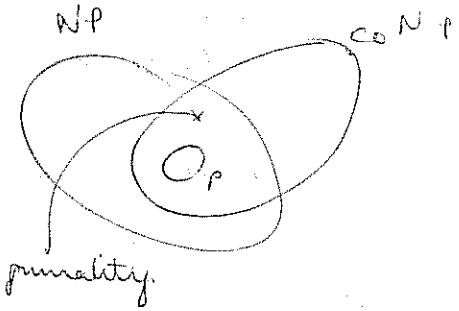


$coNP = \{ L \mid \bar{L} \in NP \}$

$\bar{L} = \{ \text{strings } x \mid x \notin L \}$

P Co P
ψ ψ
$\bar{L}$ L

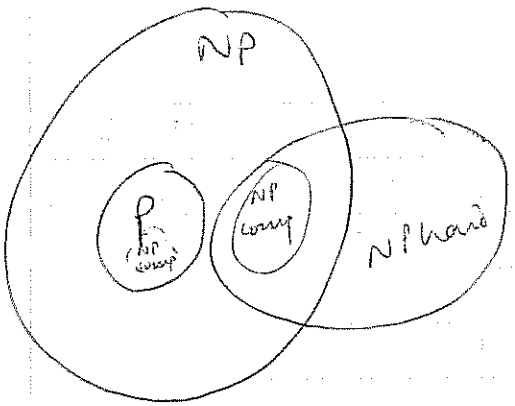




CS 170

Algorithms

27 April Monday



To show problem Prog NP complete must show

1. Prog $\in$ NP
2. Choose problem Q $\in$ NP complete
3. Find reduction (poly time) from Q to prog

$\forall R \in NP, R \propto Q \propto Prog$

4. Verify this takes poly time

① if P = NP ....

② if P ≠ NP stop trying to solve NP complete prob

Acceptability vs Decidability

M/c accepts L
vs
M decides L
} in Polynomial time

after some fixed (maybe unknown) Poly time Yes/No o/p

May take a bit long to reply

If you have M_1 which decides L you can build a m/c M_2 which accepts L.

Converse is not true!

Note that if we have a m/c which accepts $\overline{L}$

then we can run them in P & answer the decidability problem.

U universe of elements
$S_1, \ldots, S_K$ $\quad S_i \subseteq U \ \forall i$

Question: can we partition U into sets A & B

$(A \cup B = U, \ A \cap B = \emptyset)$

s.t. $\forall i \ \exists \ a_i, b_i \in S_i$ s.t. $a_i \in A$ & $b_i \in B$

Show this is NP-complete

① clearly in NP
② Reduce to SAT

ands of ors tough
ors of ands easy

SAT: Given CNF ; How about DNF

Reduce SAT to Set Partition

Take ϕ, some CNF form & "convert" it to an instance of set partition.

What does are want $S_1, \ldots S_K, U$ to be?
SAT: $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots x_n, \bar{x}_n$, $c_i = (x_1 \vee x_3 \vee \bar{x}_5)$ eq/

$U = \{x_1, \ldots x_n\}$ Hint one of the elements in the $\bar{x}_1, \bar{x}_1, \ldots \bar{x}_n$, False
Universe is "false"
For each clause $c_i = (x_1 \vee x_3 \vee \bar{x}_5)$ $S' = (x_1, x_3, \bar{x}_5, \text{false})$
$\forall x_i$ = $S = (x_j, \bar{x}_j)$

NP poly red
NP hardness
... how do we get started

prob X is NP hard $\Leftrightarrow$ every prob in NP poly red to prob X

X is NP complete
$X \in NP$

How to show a problem is NP-complete
(i) show $Z \in NP$
(ii) Show $\exists$ an NP complete problem that poly reduces to Z

Boolean Expressions

variables (truth valued)
negation $\bar{x}$
conjunction $\wedge$ and
disjunction $\vee$ or

$(x_1 \vee x_2 \wedge \bar{x}_3) \vee ((\bar{x}_2 \vee x_4) \wedge (\overline{x_3 \wedge x_4}))$

for each subs of a truth value, for each variable, the expr evaluates to a truth value,

Q: $\exists$? an subst that renders expr "true"?

Expression is CNF (Conj. Normal Form)
"conjunction of clauses"
clause ... "disjunction of literals"
literal ... variable or negated variable

Danger of Blowing Up!

Can always take a Boolean expr to CNF by applying distr, assoc, laws, deMorgan

SAT : input instance is a Boolean formula $\mathcal{F}$ in CNF

Question : Is $\mathcal{F}$ satisfiable?
is there a truth value substitution for the variables that renders $\mathcal{F}$ true?

Thm : SAT is NP-complete (Stephen Cook)

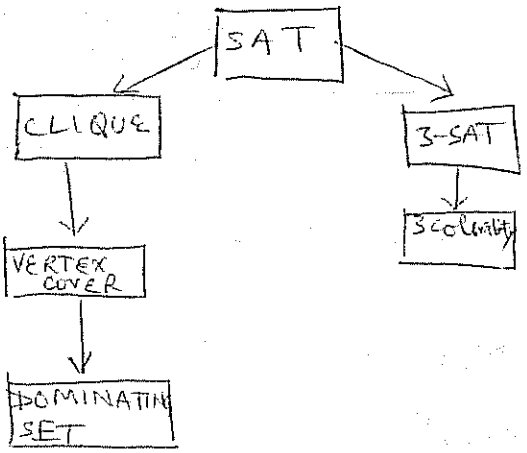
Pf - (i) SAT is in NP (trivial)

(ii) if $Z \in NP$ then $Z \xrightarrow{\text{poly}}$ SAT

Z in NP

$\exists$ non det poly time algo A for solving it.

take A ~~transform it to a Boolean expr and~~ & input x (in CNF), & transform it to a Boolean expr $\mathcal{F}_{A,x}$ s.t. $\mathcal{F}_{A,x}$ is sat $\Leftrightarrow$ A accepts x.



CLIQUE
graph $G = (V, E)$
k-clique $W \subseteq V$, $|W| = k$
$x \neq y \in W \Rightarrow \{x, y\} \in E$

VERTEX COVER
$U \subseteq V$ such that
if $\{x, y\} \in E \Rightarrow x \in U$ or $y \in U$

DOMINATING SET
$D \subseteq V$
$x \in V \Rightarrow x \in D$ or x is adjacent to some $y \in D$

CLIQUE - instance: graph $G=(V,E)$, int K

Q: does G have a K-clique?

VERTEX COVER - instance: graph $G=(V,E)$; int c

Q. does G have a vertex cover of size $\le c$?

DOMINATING SET - instance: $G=(V,E)$; int t

Q. does G have a dominating set of size $\le t$?

3-SAT - instance: Boolean formula in CNF; each clause has 3 literals.

Lemma: CLIQUE is NP-complete

Pf: Need to show (1) CLIQUE is in NP

(2) some NP-complete prob poly reduces to CLIQUE, CLIQUE

(1) ✓ (last time)

(ii) take any instance $\mathcal{F}$ of SAT and produce an instance (G,K) of CLIQUE, s.t. $\mathcal{F}$ is satisfiable $\Leftrightarrow$ G has K cliq

instance of SAT: vars $x_1, x_2, \dots x_M$

clauses: $c_1, \dots, c_m$

$c_i = (x_{i_1} \vee \bar{x}_{i_2} \vee \bar{x}_{i_3} \vee x_{i_4} \cdots)$

instance of CLIQUE; int $K=m$

graph $G=(V,E)$

$V = \{(i,j) \dots$ var x_j appears in clause c_i

$(i,\bar{j}) \dots \bar{x}_j$ " " " c_i

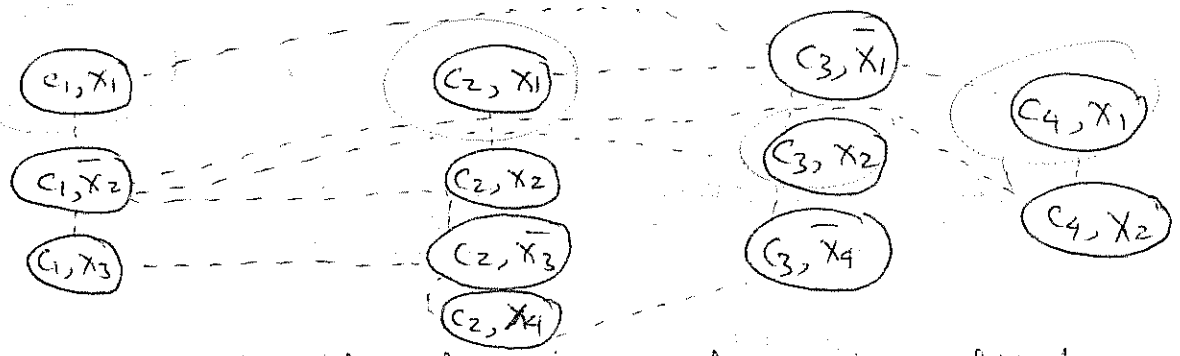
$$\mathcal{E} = \Big\{ \{(i,j),(i',j')\} \; ; i \neq i'$$
$$\{(i,\bar{j}),(i',\bar{j}')\} \; ; i \neq i'$$
$$\{(i,j),(i',\bar{j}')\} \quad i \neq i', \; j \neq j'$$

which ones are NOT in the set?!

Ex $(x_1 \vee x_3 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_3 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (x_1 \vee x_2)$

$x_1 = t$
$x_2 = t$

c_1 c_2 c_3 c_4



⇒ not in set

Can construct ~~edges~~ vertices in poly time
Can check edge in graph in poly time
⇒ G can be constructed from $\mathcal{F}$ in poly time

Need to show $\mathcal{F}$ is satisfiable ⇔ G has m-clique

(i) $\mathcal{F}$ is satisfiable ⇔ ∃ a truth assignment that renders some literal true in every clause

For each clause i

∃ var x_j s.t. x_j is true
or ∃ var $\bar{x}_j$ s.t. $\bar{x}_j$ is true

Can choose (i,j) or $(i,\bar{j})$ for each i.
(i,j) and $(i',\bar{j})$ cannot occur simultaneously

CS170 — Algorithms — 3 May '90

SAT is NP-complete

X is NP complete iff (i) X is in NP & (ii) some NP complete problem Y (chosen) reduces to X.

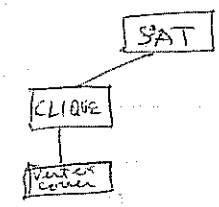
Final: Sec1 Thu 05/17; 8-11 Wheeler 130 — Sec2 Wed 05/16 12^{30}-3^{30} 3105

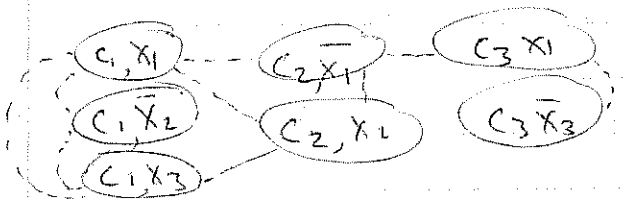
CLIQUE G, K

Q: Does G have a K-clique?

$\leftarrow c_1 \rightarrow \quad \leftarrow c_2 \rightarrow \quad \leftarrow c_3 \rightarrow$

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3)$





connect all unless vertices involved are negations of one another.

⇒ Clique iff ∃ SAT assignment

Lemma: "Vertex Cover" is NP-complete

Instance: Graph $G' = (V', E')$ int c

Q: Is there a $C \subseteq V'$ with $|C| = c$, s.t. every edge in E' is incident to some vtx in C.

(i) Is VTX Cover in NP?

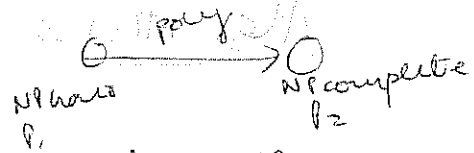
Yes, "guess" C and then verify for each $e \in E'$ that E is incident to some vtx in C.

(ii) Show CLIQUE poly reduces to vertex cover

Recall CLIQUE: Inst graph $G = (V, E)$ int K

Q: Is there a $K \subseteq V$ s.t. $|K| = K$ and every two vertices in K are adjacent?

given G & K must produce G' & c (poly time)



$\Rightarrow p_1 \in NP$

G' : $V' = V$

E' : $\bar{E} = \{(x,y) \mid x \neq y \in V, \{x,y\} \notin E\}$

$c = |V| - k$

Claim: G has a k-clique $\Leftrightarrow$ G' has a vertex cover of size c

Proof ($\Rightarrow$) $K \subseteq V$, $|K| = k$ s.t. $x, y \in K \Rightarrow \{x,y\} \in E$

Consider $C = V \setminus K$; $|C| = |V| - k = c$

Is C a vertex cover of G'?

Need to show: If $u \notin C$ and $v \notin C$ then u & v are not adjacent

($u \notin C$: $u \in K$; $v \notin C$: $v \in K$)

$\Rightarrow (u,v) \in E \Leftrightarrow \{u,v\} \notin E'$

Proof ($\Leftarrow$) C vtx cover for G'

$|C| = c = |V| - k$

Consider $K = V \setminus C$; $|K| = |V| - c = k$

Is K a k clique of G?

Need to show $(x,y) \in K \Rightarrow \{x,y\} \in E$

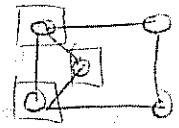
$x \in K \Leftrightarrow x \notin C$; $y \in K \Leftrightarrow y \notin C$

$\Rightarrow (x,y) \notin E'$ (by prop of vtx cover)

$\Leftrightarrow (x,y) \in E$

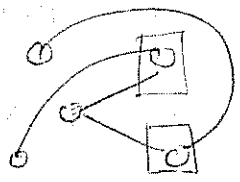
$\Rightarrow K$ is a clique

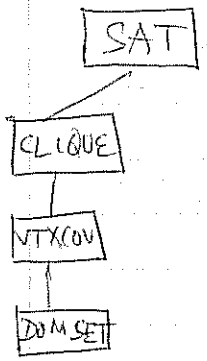
G :



3 clique: □

G'

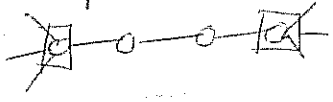




Lemma: dominating set is NP-complete
instance: graph $G' = (V', E')$ int d
Q: $\exists$? $D \subset V$ with $|D| = d$ s.t. $\forall x \in V'$
$x \in D$ or x adjacent to a vtx in D

(i) Dom Set is in NP ("guess" D then verify)
(ii) Reduce (polynomially) Vtx cover to dom set

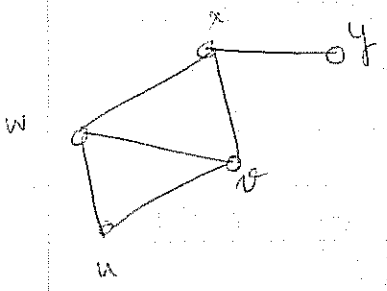
VTX COVER - instance is graph $G = (V, E)$ int c
Q: is there a $C \subset V$ with $|C| = c$ s.t.
$(x,y) \in E \Rightarrow x \in C$ or $y \in C$?
given G & c need to construct G', d (poly time)

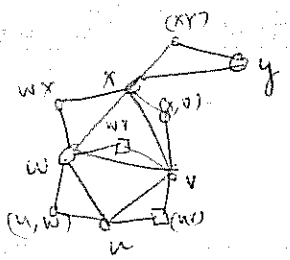
~~Dom Set is always~~ VTX cover is always Dom Set
converse not true ⟶ 

$V' = V \cup \{(x,y) \mid \{x,y\} \in E\}$
$E' = E \cup \{\{x,(xy)\},\{y,(xy)\}\} \mid (x,y) \in E\}$

$d = c$

G:





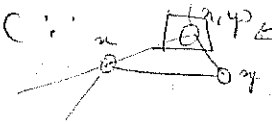
: G'

Need to show: G has a vtx cover of size c $\Leftrightarrow$ G' has a dom set of size d ($= c$)

Pf ($\Rightarrow$) Assume C is vtx cover of size c (for G)
if $(x,y) \in E$ then $x \in C$ or $y \in C$
((claim: $D = C'$ is dom set for G'
$\hookrightarrow$ $x \in C$ or x is adjacent to some vertex in G, namely y

What about isolated nodes? never in vtx cover, always in dom set.

$\Leftrightarrow (xy) \in V' \Rightarrow (x,y) \in E \Rightarrow x \in C$ or $y \in C$
But (xy) adjacent to x and to y in G'

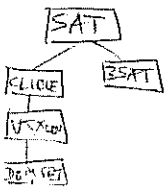
Pf ($\Leftarrow$) D is dom set for G' $|D| = c$
difficult ($\because$  is in dom set

$\therefore$ must transform $\square$ to one of neighbors
if D contains a vertex of form (x,y) then replace by (if necessary "pad new x" to size c)

Claim: new $D \subset V$ is a vertex cover for G

$(x,y) \in E$;
(xy) is adjacent to some vertex in D
But x and y are the only vertices adjacent to (xy) in G'
$\Rightarrow x \in D$ or $y \in D$
$\Rightarrow D$ is a vertex cover for G.

QED



Lemma: 3SAT is NP-complete
3-SAT: instance $\mathcal{F}$ a boolean formula in CNF with exactly 3 literals per clause
Q: is $\mathcal{F}$ satisfiable?

Proof: (i) is in NP ($\because$ special case of SAT)
(ii) SAT poly reduces to 3-SAT
SAT: instance $\mathcal{F}$ boolean formula in CNF
Q: is $\mathcal{F}$ satisfiable?
Given $\mathcal{F} = C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_m$
produce $\mathcal{F}' = C_1' \wedge C_2' \wedge \ldots \wedge C_n'$ each clause having 3 literals
$\mathcal{F}$ satisfiable $\Leftrightarrow \mathcal{F}'$ satisfiable

$B = \{c_1, c_2, \ldots c_m\}$

$B' = \phi$

while $B \neq \phi$ do

pick clause C from B and delete it

case 1: C has exactly 3 literals

include C in B'

case 2: C has < 3 literals

duplicate literals & include C in B'

case 3: C has > 3 literals

introduce new var, form C' & C'' include C' into B include C'' into B

$C = \alpha_1 \vee \alpha_2 \vee \alpha_3 \ldots \vee \alpha_j$

$(\alpha_1 \vee \alpha_2 \vee z) \wedge (\bar{z} \vee \alpha_3 \vee \alpha_4 \vee \ldots \vee \alpha_j)$

C' — new variable — C''

C true $\Leftrightarrow$ at least one α_k is true

say α_4 true

then can procure $C' \wedge C''$ to be true (by choosing z)

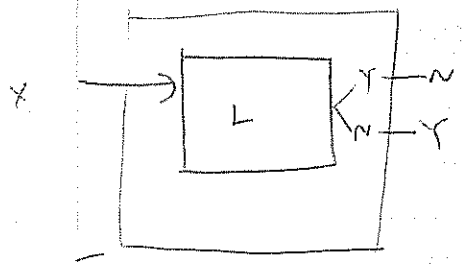
again if C never true $\Leftrightarrow$ no α_k true

$\Rightarrow$ can never be able to pick right z!

CS170 (Section) May 7, 1990
Monday

$L \in P$: $\exists$ poly time M/c M s.t. $x \in L \Rightarrow$ M answers YES & halts
$x \notin L \Rightarrow$ M " NO & halts
} decides L

$\overline{L} = \{x \in \Sigma^* \mid x \notin L\}$
every x



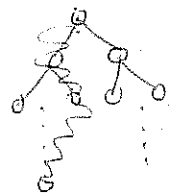
$\overline{L}$ $\therefore P = coP$

recognizes / accepts lang. } $L \in NP$: $\exists M$: $x \in L \Rightarrow$ M runs poly time, answers YES, halts
$x \notin L \Rightarrow$ M never halts or says NO

$\overline{L}$: $x \notin L$ Want $\overline{M}$ run poly time YES halt
But can't be sure if halts! so NP not necc coNP

NP m/c guesses (~~randomly~~ nondeterministically) runs in poly time det.

Computation tree:

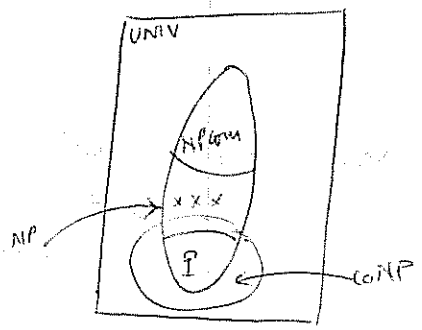


Knows when to go ← guessing point
Find one path that takes it to a YES state ⇒ will guess that

NP m/c : smart guesser (correctness)
: can check (certify) in poly time ‖ characterizations

③ Look for Poly

④



is NP = coNP ?

⟨11.14⟩ Π: und in $G = (V, E)$; $K \in \mathbb{Z}$

$\exists ?\ V' \subseteq V \mid |V'| = K : (V', E')$ acyclic

T.P.T. "NP-compl" $\leq \pi \leq$ "NP"

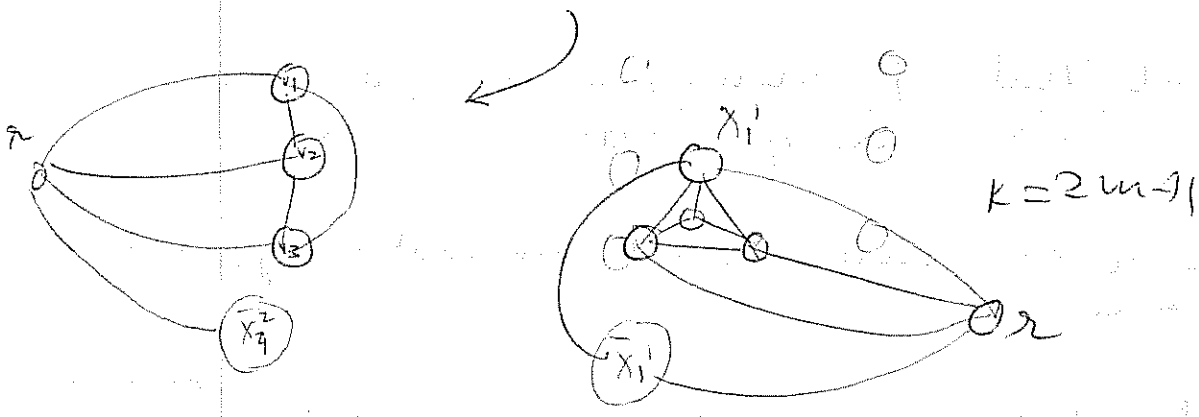
Red^{n} from 3SAT : clauses $c_1, \ldots, c_m$

$c_i = (x_1 \vee x_2 \vee x_3)$

$\exists ?$ truth assgn
true variable $\longrightarrow V'$

$c_1 = (x_1 \vee \bar{x}_2 \vee x_3)$

$K = m$ (+1) ← root



① Hamiltonian
$K = 2$ ⇒ cycle / path

② restrict the problem

CLIQE ⟶ our problem is harder than clique add ind-ep set always
∴ will always find no set
this is a restriction of the original problem

SAT

Instance: Boolean formula $\mathcal{F}$ in CNF

Q: Is $\mathcal{F}$ satisfiable?

3-SAT

Instance: Boolean formula $\mathcal{F}$ in CNF each clause has exactly 3 literals

Q: Is $\mathcal{F}$ satisfiable?

Ex: $\mathcal{F} = (x_1 \vee x_3 \vee x_4 \vee \bar{x}_5 \vee x_6) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_6) \wedge (x_4 \vee x_5)$ --> $(x_1 \vee x_3 \vee z_2)(\bar{z}_2 \vee x_4 \vee \bar{x}_5 \vee x_6)$ (same instance) $(\bar{z}_2 \vee x_4 \vee z_3)(\bar{z}_3 \vee \bar{x}_5 \vee x_6)$

$(x_2 \vee \bar{x}_3 \vee z_1)(\bar{z}_1 \vee x_4 \vee \bar{x}_6)$

$(x_4 \vee x_5 \vee x_5)$

Lemma: 3-colorability is NP complete

Instance: graph $G = (V, E)$

Q: Can G be 3 colored ie $\exists$ fn $c: V \to \{R, B, G\}$ s.t. $\{x,y\} \in E \Rightarrow c(x) \neq c(y)$

(i) 3 colorability is NP ~~complete~~. ∵ can guess colors & verify each edge

(ii) some NP complete problem [3SAT] polynomial reduces to 3COLORABILITY

Instance: Boolean Formula $\mathcal{F}$ in CNF exactly 3 literals/clause

Q: Is $\mathcal{F}$ satisfiable?

Given formula $\mathcal{F}$ one must construct graph $G_{\mathcal{F}}$ s.t. $\mathcal{F}$ is satisfiable $\Leftrightarrow$ G is 3-colorable.

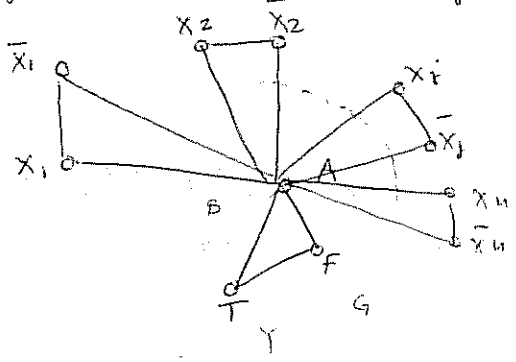
$\mathcal{F} = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$C_i = (\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$

vars: $x_1 \ldots x_n$

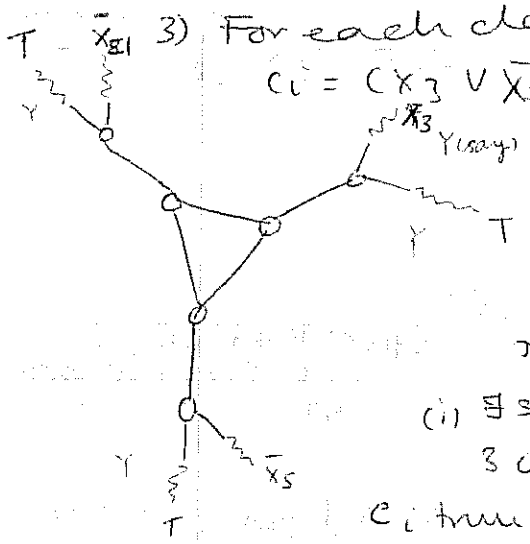
1) construct Δ

2) for each variable x_i, append a Δ to A



3) For each clause c_i introduce a "clause gadget" consists of 6 vtxs

$c_i = (x_3 \vee \bar{x}_5 \vee \bar{x}_{21})$



finally G_f $\Big\|$ $(6m + 2n + 3)$ vtces
$(12m + 3n + 3)$ edges

Now we are to show f sat $\Leftrightarrow$ G_f 3 colorable

(i) $\exists$ SAT $\Rightarrow$ G_f 3 colorable

3 coloring G_f inside c_i is trivial by assgn

c_i true $\Leftrightarrow$ one of the literals is true

asymmetry (a) $\bar{x}_{21}, \bar{x}_5$ true is colored T then can do 3 colorability

(b) $\bar{x}_{21}$ true $\bar{x}_5$ false

(c) $\bar{x}_{21}, \bar{x}_5$ false

↙ instead, to show (ii) f not SAT $\Rightarrow$ G_f not 3 colorable

must show ∀ no matter how started coloring f clause gadget which screws you up.

Planar Graph: Every planar graph is 4-colorable

lemma: It is NP complete to decide whether (↓ can't say no) a planar graph can be 3-colored.

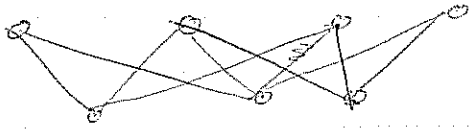
Planar 3-colorability

instance : planar graph G
Q : can G be 3 colored?

3 colorability instance : graph G
Q : Can G be 3 colored

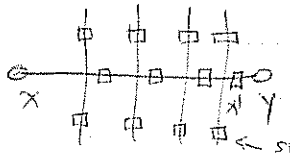
Given G have to produce planar graph G' s.t. G is 3 colorable $\Leftrightarrow G'$ is 3 colorable

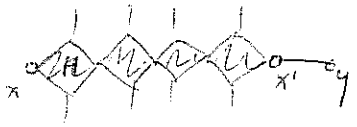
Embed G in the plane



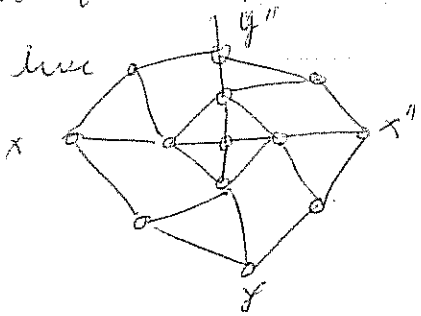
Draw ($O(n^4)$ intersections)

any edge





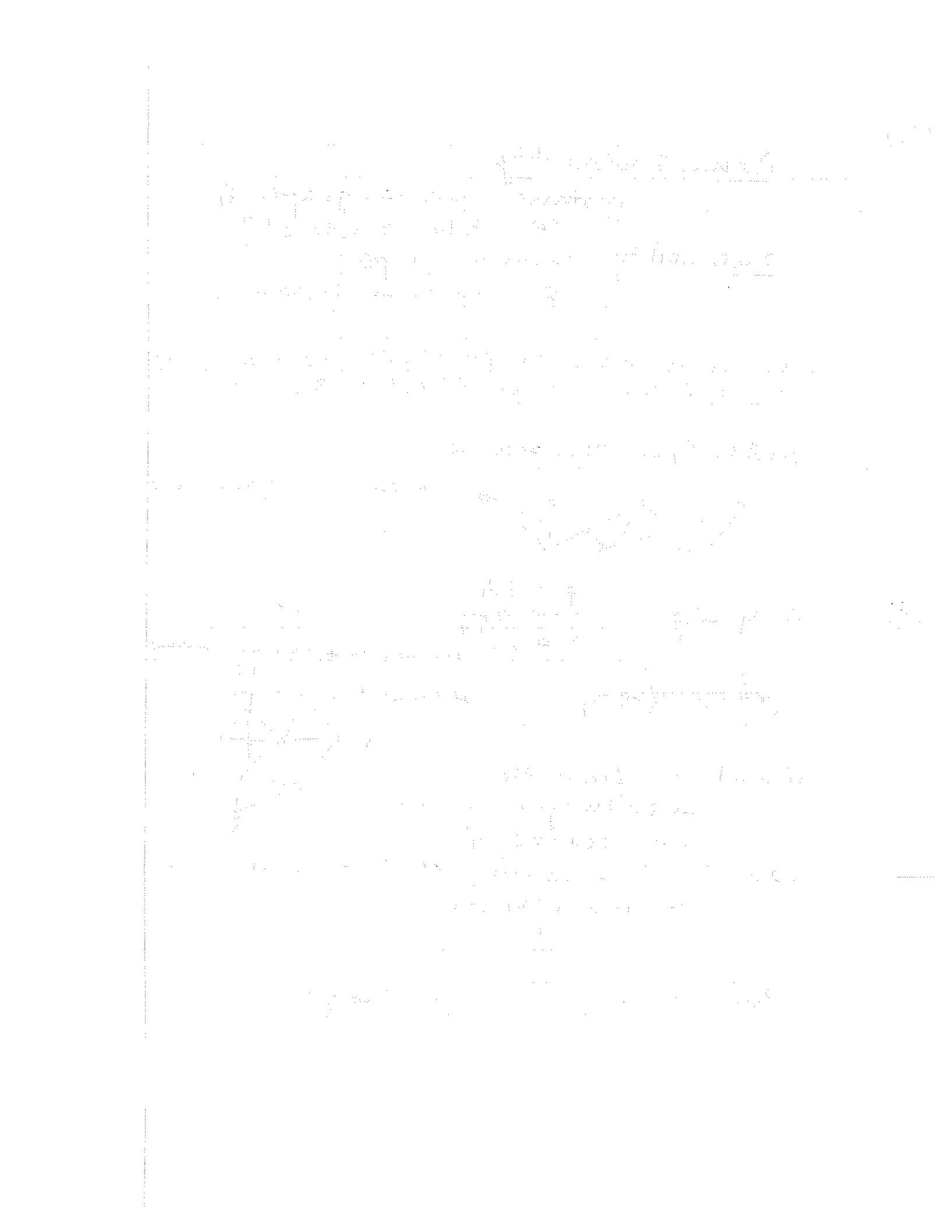
H looks like



Claim 1: H is 3 colorable
and always $c(x) = c(x'')$
and $c(y) = c^{-2}(y'')$

Claim 2: $\exists$ a coloring of H s.t. $c(x) = c(x'') = c(y) = c(y'')$
and also $c(x) \neq c(y)$
$\|$ $\|$
$c(x'')$ $c(y'')$

Play around $\bar{c}$  3 col of $G \Rightarrow$ 5 col of G'



CS170 Algo 9 May '90 Thursday

1. 3-colorability
2. planarity & 3 colorability

Hamiltonian Path
instance: graph G
Q. Is there a simple path that touches every vertex exactly once?

Hamiltonian cct: simple cct

Travelling Salesperson Problem (TSP):
instance: Graph G
positive edge lengths (which are integers)
Lint.
Q: Is there an HC in G of length $\leq L$

Cct Brd: moving the drill more exp. than manufactured

3-DIMENSIONAL MATCHING
instance: X, Y, Z 3 disj sets $|X| = |Y| = |Z| = n$
$T \subset (X \times Y \times Z)$ (triples)
also (hyperedges)
Q: Is there an $M \subset T$
s.t. every $x \in X$ appears in exactly one triple in
and " $y \in Y$ " " " " " "
and " $z \in Z$ " " " " "

Can you construct maximal bipartite graph?

PARTITION:
inst. Sequence X of integers (positive)
Q: Let X be partitioned into A and B
s.t. $\sum_{a \in A} a = \sum_{b \in B} b$

2, 25, 7, 15, 3, 8, 9

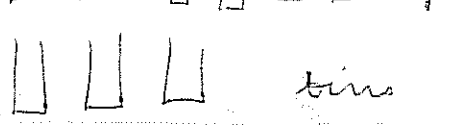
binary — hard
unary — easy (poly in i/p size)

KNAPSACK:

Instance: sequence X
for every $x \in X$ pos int $s(x)$
pos. int $w(x)$
positive int S, W
Q: Is there a $Y \subseteq X$ s.t. $\sum_{y \in Y} s(x) \leq S$ and $\sum_{y \in Y} w(y) \geq W$?

BIN PACKING:

Instance: Sequence X of (+)ve integers
positive int B
positive int K
Q: Can one partition X to subsequences $Y_1, Y_2, \ldots Y_K$ s.t. for each $1 \leq i \leq K$
$\sum_{x \in Y_i} x \leq B$



Analogy w/ files in 5 disks
very good approx algos exist!

CROSSWORD PUZZLE CONSTRUCTION:

construction or solution

Instance: Set D of words, an "array" $A[n,n]$
Can one assign a character to each array position s.t. each column & each row is a word in D.

↳ spec. Dark spots in $A[n,n]$ ⇒ still NP complete

QUADRATIC CONGRUENCE

Instance: 3 pos. integers a, b, c
Q: Is there an x $0 < x < c$ s.t.
$x^2 \bmod b = a$ "Sqrt of a" (?)
Amazing that SAT red to this!
The other is this to SAT ok ∵ formula

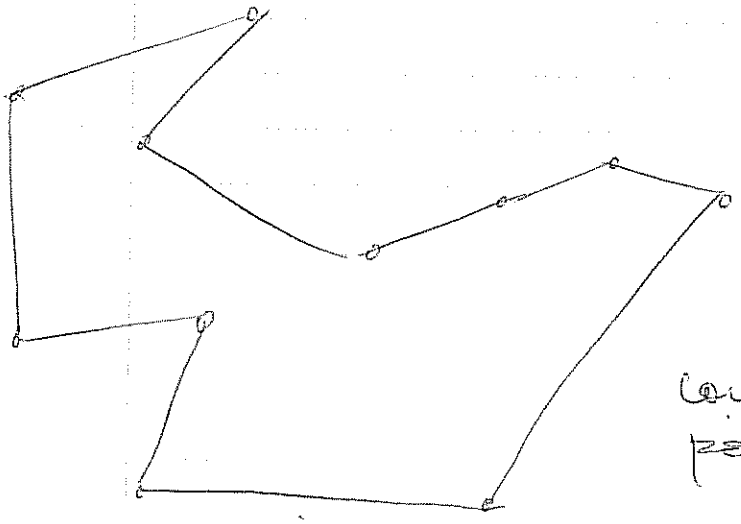
GEOGRAPHY (Generalized)

instance: directed graph $G = (V, E)$
start vtx (V)

two person game.
X of visited vertices
x current vtx
move: choosing outarc from x that ends at a vtx $y \in X$.

$X = X \cup \{y\}$
y - ~~cur~~ current vtx

Q.: Starting at v is there a first move that will force a win?
PSPACE complete Chess - finite game

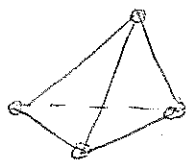


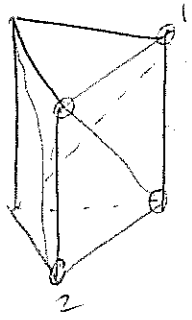
simple polygon
can always cut into Δ.
in such a way that
corners of Δs are original
points

could also allow internal points! → Steiner Pts

Any simple polygon can be Δated w/o using Steiner Points.

3-dimensional polyhedra
analog of Δ is tetrahedron





can do so that line connecting vtcs is out of the solid.

May not be possible to tetrahedralise because 3 base the 4th above always out.

∴ need Steiner pts.

From SAT

Instt: 3 Dim polyhedron P

Q: can P be tetrahedralise a/o Steiner Pts?

Even: know can do in 1 St. Pt.

Can you do in none : NP complete.

TIME IN: 6:33pm NAME: ADNAN AZIZ TIME OUT: 8:30

100/100

CS 170
R. Seidel

Midterm

March 14th, 1990

This is an open book exam. You may use any books or notes that **you** brought along. Remember that clarity of presentation is deemed important. All questions have equal weight.

GOOD LUCK!

20 1. Draw a decision tree for finding the third largest of the four keys x_1, x_2, x_3, x_4.

20 2. Let $U = \{0,1,2,3,4,5\}$ and $T = \{0,1,2\}$. Consider the following five functions from U to T:

$$h_1(x) = (x+1) \bmod 3 \qquad h_2(x) = (x+2) \bmod 3 \qquad h_3(x) = x \bmod 2$$

$$h_4(0) = h_4(5) = 2 \quad h_4(1) = h_4(2) = 1 \quad h_4(3) = h_4(4) = 0$$

$$h_5(0) = h_5(1) = 2 \quad h_5(2) = h_5(3) = 1 \quad h_5(4) = h_5(5) = 0$$

Determine a function h_6 from U to T so that the set $\{h_1, h_2, h_3, h_4, h_5, h_6\}$ is a 1-universal class of hash functions.

20 3. Let f and g be two functions in $\mathbf{N} \to \mathbf{R}^+$ so that $f(n) = O(g(n))$.
Define the function $h : \mathbf{N} \to \mathbf{R}^+$ as $h(n) = f(2n)$.

Is it necessarily true that $h(n) = O(g(n))$? Prove your answer.

20 4. You are given n keys and an integer k with $1 \le k \le n$. Give an efficient algorithm to find *any one* of the k smallest keys. (For example, if $k=3$, the algorithm may provide the first-, the second-, or the third-smallest key. It need not know the exact rank of the key it outputs.) How many key comparisons does your algorithm make? (Hint: Don't look for something complicated. One insight gives a short, simple algorithm.)

Give a lower bound, as a function of n and k, on the number of comparisons needed to solve this problem. (As model of computation use comparison based algorithms.)

20 5. For a sequence $S = (x_1, \ldots, x_n)$ of n real numbers let $d(S)$ denote the number of distinct elements in S.
Show that it is possible to sort S in worst case time $O(n \log d(S))$.
You must not assume that $d(S)$ is known in advance.

CS170 Algorithms : M.T. 1 Adnan Aziz

Q3. Given $f, g : \mathbb{N} \to \mathbb{R}^+$

$f(n) = O(g(n))$ Given

$h(n) \doteq f(2n)$

Is it ~~not~~ ok true that $h(n) = O(g(n))$?

(20)

Let's consider $f(n) = 2^n$ & $g(n) = 3^n$

clearly $f(n) = O(g(n))$

as $2^n < 1 \times 3^n \quad \forall n \geq 1 \quad \{n_0 = 1, c = 1\}$

$h(n) = f(2n) = 2^{2n} = 4^n$

Now $4^n \neq O(3^n)$

for if it were, $\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+$ s.t.

$$4^n \leq c 3^n \quad \forall n \geq n_0$$

$$\Leftrightarrow n \log 4 \leq n \log 3 + \log c \quad \forall n \geq n_0$$ (∵ $\log x$ is monotonic, i.e. $x \leq y \Leftrightarrow \log x \leq \log y$ $(x, y > 0)$)

$$\Leftrightarrow n \log(4/3) \leq \log c \quad \forall n \geq n_0$$ (note $\log 4/3 > 0$)

But for any $c \in \mathbb{R}^+$, we have the inequality false for all $n \geq \log c / \log(4/3)$, forcing a contradiction!

$\Rightarrow$ It is not NECCESARY that $h(n) = O(g(n))$

good ✓

Q2. Given $U = \{0,1,2,3,4,5\}$ $T = \{0,1,2\}$
Consider the following five functions from $U \to T$

$h_1(x) = (x+1) \bmod 3$ $\quad h_2(x) = (x+2) \bmod 3$ $\quad h_3(x) = x \bmod 2$

$h_4(0) = 2$	$h_4(2) = 1$	$h_4(4) = 0$
$h_4(1) = 1$	$h_4(3) = 0$	$h_4(5) = 2$

$h_5(0) = 2$	$h_5(2) = 1$	$h_5(4) = 0$
$h_5(1) = 2$	$h_5(3) = 1$	$h_5(5) = 0$

Determine $h_6 : U \to T$ so that $\{h_1, \ldots h_6\}$ is 1-universal

$|H| = 6$ $\quad |T| = t = 3$

	h_1	h_2	h_3	h_4	h_5	h_6
0	1	2	0	2	2	1
1	2	0	1	1	2	1
2	0	1	0	1	1	1
3	1	2	1	0	1	1
4	2	0	0	0	0	1
5	0	1	1	2	0	1

× ⇒ collision

✓ ⇒ O.K.

	h_3	h_2	h_1	h_4	h_5	# of h_i causing collision
(0,1)	✓	✓	✓	✓	×	1
(0,2)	×	✓	✓	✓	✓	1
(0,3)	✓	×	×	✓	✓	2
(0,4)	×	✓	✓	✓	✓	1
(0,5)	✓	✓	✓	×	✓	1
(1,2)	✓	✓	✓	×	✓	1
(1,3)	×	✓	✓	✓	✓	1
(1,4)	✓	×	×	✓	✓	2
(1,5)	×	✓	✓	✓	✓	1
(2,3)	✓	✓	✓	✓	×	1
(2,4)	×	✓	✓	✓	✓	1
(2,5)	✓	×	×	✓	✓	2
(3,4)	✓	✓	✓	×	✓	1
(3,5)	×	✓	✓	✓	✓	1
(4,5)	✓	✓	✓	✓	×	1

for any (x,y) $x \neq y$ we can have more than 2 collisions (thus no longer 1 universal)

↳ $\{$# of h_i causing collision for any x, y $x \neq y\}$ $\leq \frac{|H| \times c}{t} = \frac{6 \times 1}{3} = 2$

∴ we must choose h_6 so that it doesn't have collisions for (0,3) (1,4) (2,5)

∴ let $h_6(0) = 0$
$h_6(1) = 1$
$h_6(2) = 2$
$h_6(3) = 1$
$h_6(4) = 2$
$h_6(5) = 0$

Clearly this $h_6(x)$ has no collision for (0,3), (2,5), (1,4)

So it is 1-Universal

QED

Q1 We want a decision tree for finding the third largest of four keys $\{x_1, x_2, x_3, x_4\}$

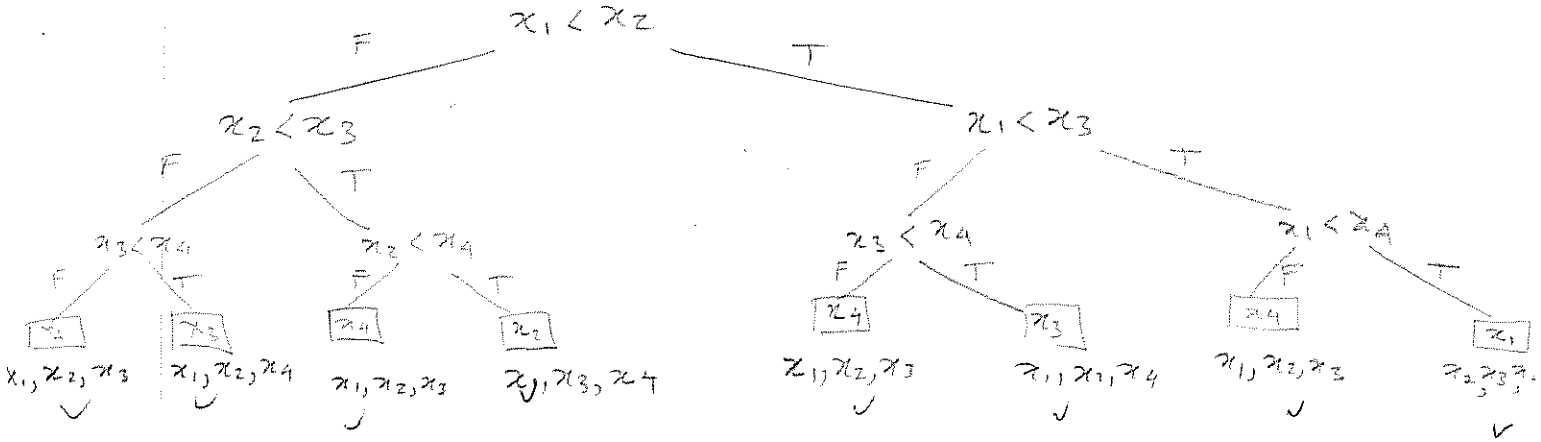
(20)

(NOT necc. optimal!)

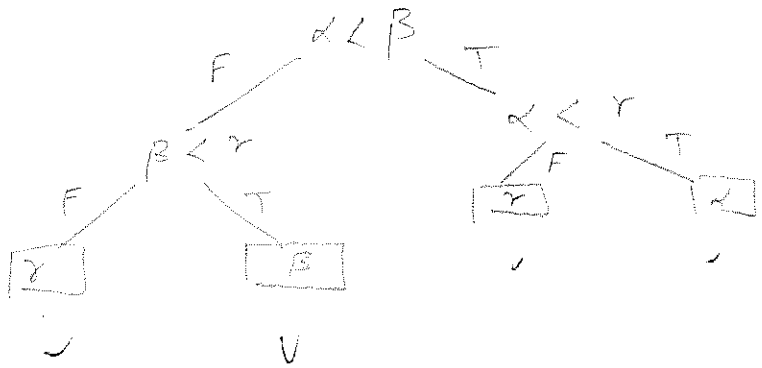
Third largest of 4 elements is same as second smallest
Finding the 3rd largest is the same as finding the second smallest which is a little easier to do so I'm determining the second smallest.

Algorithm: (i) find min
(ii) look at remainders for min^m $\Rightarrow$ this is the 3rd largest

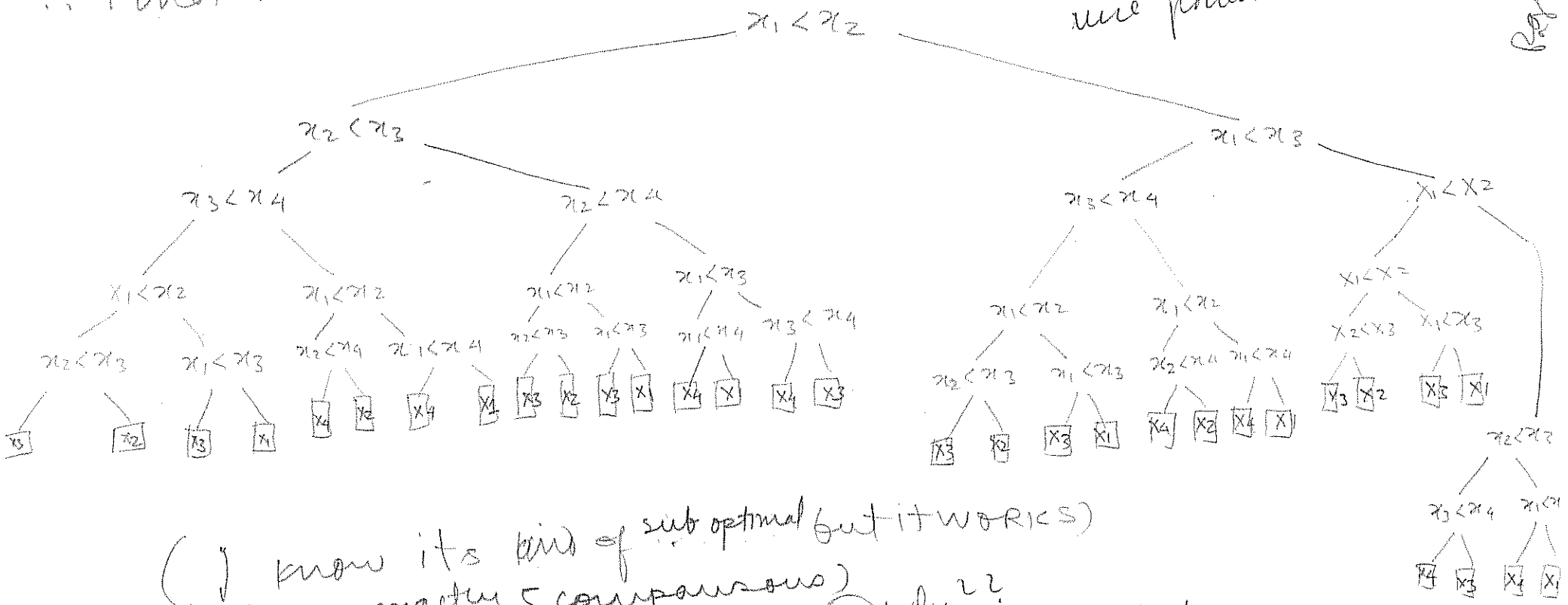
(i) Can find minimum in $N-1 = 4-1 = 3$ comps.



(ii) Find min^m of α, β, γ $\Rightarrow$ 2 comps



∴ Final tree is



nice presentation!

(I know it's kind of sub optimal but it WORKS)

(Turns exactly 5 comparisons)

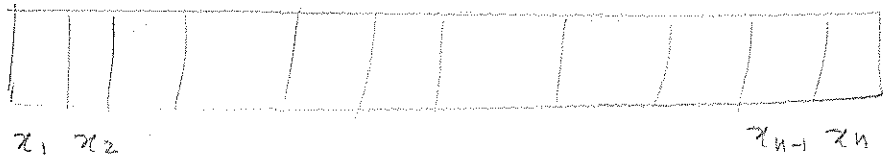
Actually you must make 5 comparisons in the worst case for any algo, so this isn't all that bad.

Some pruning could have been done to reduce average case time.

Why??

It is possible to do with 4 comp's!

Q4.



POINT 1. We must examine more than $(N-K)$ keys for, if we examine less say $(N-K)-c$ where $c \geq 0$ there would be $N-[(N-K)-c] = K+c$ keys which we didn't look at. $c \geq 0 \Rightarrow K+c \geq K$ therefore an adversary could make K of the $K+c$ keys smaller than the smallest that we checked implying that we were wrong in choosing a desired element by examining only $(N-K)-c$ keys.

Now with $(N-K)+1$ keys the adversary has only $(K-1)$ keys. If we choose the min of the $(N-K)+1$ keys we have a guarantee that at most $(K-1)$ keys are less than it $\Rightarrow$ it is one of the K^{th} smallest

So in $[(N-K)+1]-1$ comparisons we can find the min^m. i.e. $(N-K)$ comparisons this is a lower bound to as is clear from the adversary argument (we must subject each of the keys to a com

Thus the adversary can force us to look at least $(N-K)+1$ keys. Looking at $(N-K)+1$ keys requires $[(N-K)+1]-1$ comparisons $= (N-K)$ Hence $(N-K)$ comparisons is a lower bound

It is the best lower bound also, because the following algo (based on the above discussion) finds a suitable element in $(N-K)$ comparisons

Algo: Find the minimum of any subset of $(N-K+1)$ elements $\Rightarrow$ takes $(N-K+1)-1 = N-K$ comp.

[Finding the min^m of X elements can be shown to be optimally done in $X-1$ comparison - well known]

Q5. $S = (x_1, x_2, \ldots x_n)$ a sequence of n real #s
$d(S)$ = # of distinct elements in S

T.P.T. S can be sorted in $O(n \log[d(S)])$
[$d(S)$ is not known in advance.]

(20) Consider the following scheme:

The data structure used is the AVL tree
Each node has two fields: ① the element
② a count of the number of occurances

Each time we fetch a new element from S we look for it in the tree. (looking at nodes in the standard BST search way)
If it is already in the tree, we increment the count field of the corresponding node by 1.
If it is not in the tree, we create a new node with the field ① the element, set count ② to one.
and Balance the tree if necessary.

In this way we finally have an AVL tree. To output the answer, do a scan of the tree and with each node's element out according to the multiplicity established in the corresponding count field.

good

Comments: ① The output is sorted because of the way we construct the AVL tree
② the worst case is $O(n \log d(S))$ ✓

↳ The point to note is that there are exactly $d(S)$ nodes at the end of the day. Therefore the height of the tree is at most $\lfloor \log d(S) \rfloor$ ✓

Regret

For an AVL tree (i) creation takes const time.
(ii) insertion takes time prop to ht.
~~(iii) deletion takes time prop to ht~~
(iii) looking for element take time prop to ht

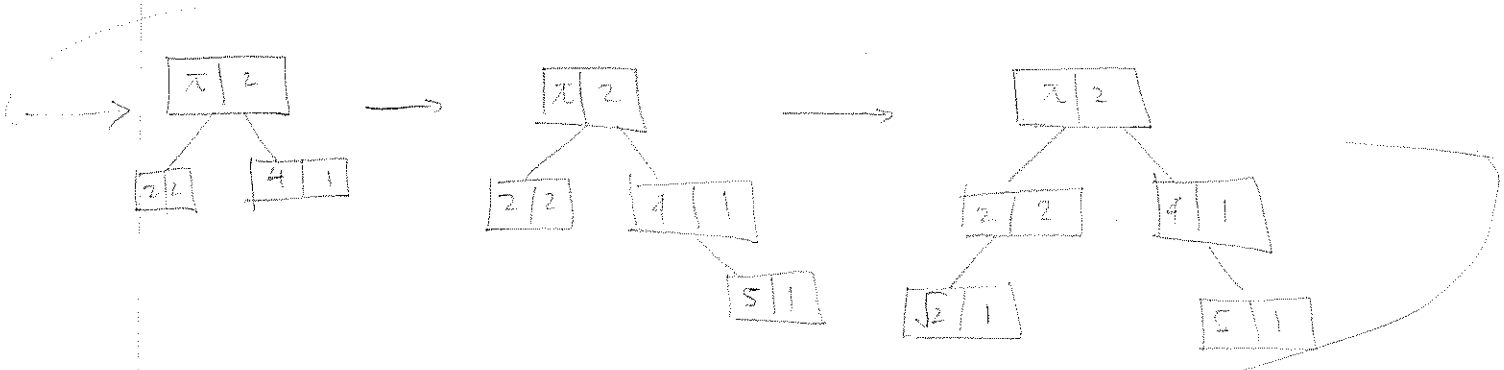
Thus inserting an element takes
no more than $c \times \lfloor \log d(s) \rfloor$ time
as the ht is bounded by $\lfloor \log d(s) \rfloor$ the final ht

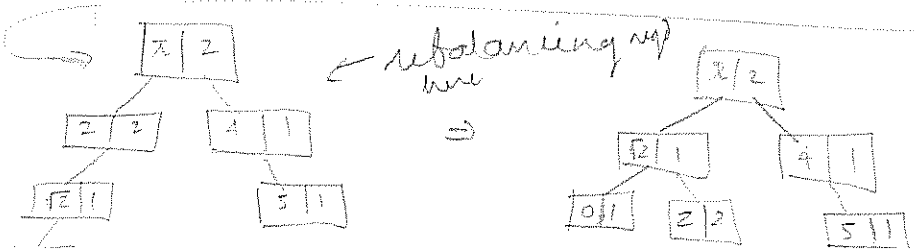
Writing out also takes time proportional only to n the number of elements.

A3: For each element insertion into to the AVL tree take $O(\log d(s))$ time inserting n elements takes $O(n \log(d(s))$ time.

∴ the sort takes only $O(n \log d(s))$ time.
(Never did we assume $d(s)$ before hand)

eg. $\{2, \pi, 2, 4, \pi, 5, \sqrt{2}, 0\}$ node : | * | count |



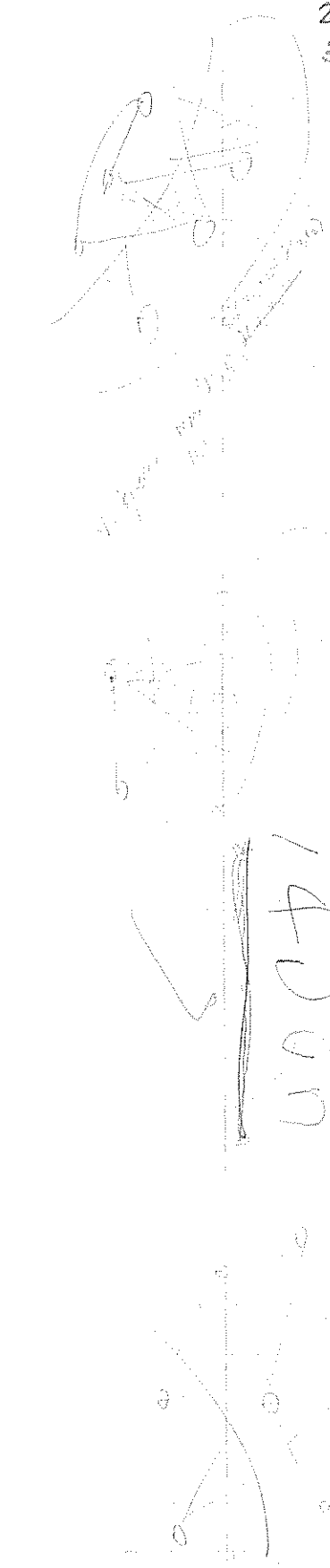


output $\{0, \sqrt{2}, 2, 2, \pi, \pi, 4, 1\}$

1. Lee : ice from Mary
2. Roommate : Donald Knuth Am #49
3. Kerzin's Engagement
4. Hills college ← can get in
5. fasting fina : will be happy when it's one
6. Vikram
7. Hit a hundred

---

1. RA
2. Neuro (Ili)
3. CS170
4. Music
5. Women
6. Exceptional aspects of Amruta way to Vikram Pareekh experim



Rough

$\{x_1\ x_2\ x_3\ x_4\}$ find min in 3 comp
find min in 2 comp

$x_1 < x_2$

$x_3 < x_2$ $(N-K)$

$x_1 < x_3$ → Yes ⇒ try $x_4 < x_1$
→ No ⇒ try $x_4 < x_3$

$x_1\ x_2\ x_3\ x_4$ $x_4 < x_2 < x_1$

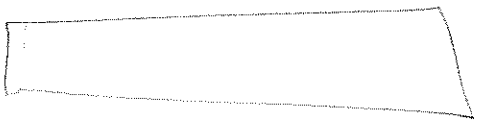
~~$x_1 < x_2$~~

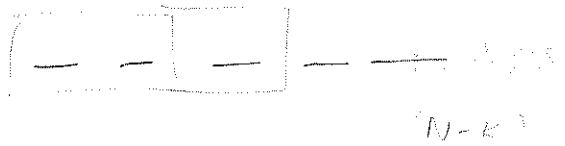
$x_1\ x_3\ x_4$

~~$x_1 < x_3$~~

~~$x_3 < x_1$~~

$x_3\ x_4$





cant do c/o looking at at least N-k elements