6.046 Spring 2024

LECTURE 1 2/6/24 11AM

Interval Scheduling

- single resource
- requests $R = \{r_1, \dots, r_n\}$ as intervals
- compatibility: r_i comp. $r_j \iff r_i \cap r_j = \emptyset$
- goal: find a largest set $\subseteq R$ of compatible requests

↳ greedy algorithm: earliest finish time

- use exchange/hybrid argument to compare S_{OPT} with S_{GREEDY} and do swapping to realize they have same cardinalities
- sorting takes $O(n \log n)$ → linear scan → $O(n \log n)$ total

weighted Interval Scheduling

- intervals now have weights

↳ dynamic programming

$$W(R) = \max \{ W(R \setminus \{r_n\}),\ w_{r_n} + W(R \setminus INC(r_n)) \}$$

(weight: w_{r_n}; incompatibility set: $INC(r_n)$)

$W(R)$: sorted by end times

$R \setminus \{r_n\}$, $R \setminus INC(r_n)$ are both prefixes of $\{r_i\}$

$R \setminus INC(r_n) = \{r_1, r_2, \dots, r_i\}$ for some $i < n$

finding i takes $O(\log n)$ via binary search

→ total time $O(n) \times O(\log n) = O(n \log n)$.

LECTURE 2 2/8/24 11AM

Divide & Conquer.

Median Finding, Integer Multiplication

↳ split into a problems of part $\frac{n}{b}$

$\Rightarrow T(n) = a \cdot T(\frac{n}{b}) + \{$ time to combine, etc. $\}$

Median Finding

input: set S of n distinct #s

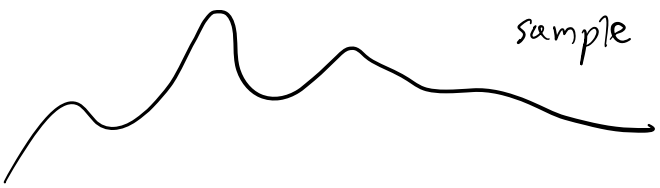
$\text{rank}(x) := \#$ elements in $S \leq x$

$\cap S$

upper median = rank $\lceil \frac{n+1}{2} \rceil$ element } same if n odd

lower median = rank $\lfloor \frac{n+1}{2} \rfloor$ element }

motivation (Robust Statistics)



sampling from dist.

sample mean ✓✓ but

if noisy/corrupted

median is robust

methods sort then find → $O(n \log n)$

check all elements → $O(n)$ per element ⇒ $O(n^2)$.

modified Brute Force - compute $L(x) = \{k \mid k < x, k \in S\}$

$G(x) = \{k \mid k > x, k \in S\}$

for $x \in S$. Then search in bigger bucket for median

↳ but worst-case $= O(n^2)$ too

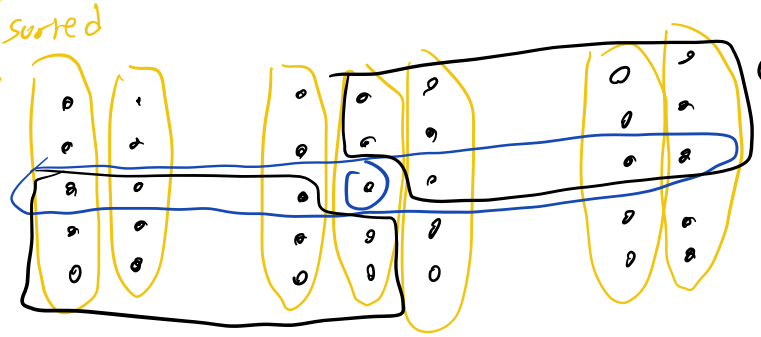
Def Say $x \in S$ is c-balanced if $\max\{|L(x)|, |G(x)|\} \leq c \cdot n$

Goal Find some c-balanced element in time $O(n)$

Subroutine for finding a c-balanced element

1. divide S into $n/5$ buckets of size 5.
2. compute median of each bucket $\rightarrow O(n)$ time
3. compute "median-of-medians" recursively.

assume each bucket sorted



$\in G(x) \sim \frac{3n}{10}$

$\in L(x) \sim \frac{3n}{10}$

$\Rightarrow T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$

$\Rightarrow T(n) = n$

Karatsuba - $T(n) = 3T(\frac{n}{2}) + O(n) \Rightarrow T(n) = n^{\log_2 3}$

Harvey, Van der Hoeven $\rightsquigarrow T(n) = O(n \log n)$

LECTURE 3 2/13/24 (Virtual, did not attend)

Monte Carlo
- always efficient
- $P(\text{right}) \rightarrow$ high

Las Vegas
- efficient in expectation
- always right

average-case (input)

expected-case (random choice)

Matrix Products

$C = A \times B \rightarrow O(n^3)$ multiplications for $n \times n$.

Strassen's $[2 \times 2] \times [2 \times 2]$ in 7 multiplications

$\Rightarrow O(n^{\log_2 7}) \rightarrow O(n^{2.81})$

Matrix Product Verification

- given A, B, C, check $AB \stackrel{?}{=} C$.

↳ can we do better than matrix-multiply ??

Freivald's Algorithm WLOG $A, B, C \in \mathbb{F}_2^{n \times n}$

- choose random binary vector $r \in \mathbb{F}_2^n$

If $\begin{cases} A(Br) = Cr & \text{YES} \\ \text{else} & \text{NO} \end{cases}$ $\Rightarrow O(n^2)$

easy to see if $A \times B = C \Rightarrow$ YES

$A \times B \neq C \Rightarrow$ NO with high probability?

why is

Claim If $C \neq AB$, $\Pr[ABr \neq Cr] \geq \frac{1}{2}$.

Proof If for some r we have

$(C-AB)r = 0$, we have $(C-AB)(r + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix})$,

$(C-AB)(r + \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}), \ldots, (C-AB)(r + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix})$

one of these $\neq 0$.

let WLOG $(C-AB)(r + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}) \neq 0$.

Then we have a 1:1 for each r s.t. $(C-AB)r = 0$, $r' = r + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ has $(C-AB)r' \neq 0$.

so $\Pr[ABr \neq Cr] \geq \frac{1}{2}$. □

recall rank-finding from Lecture 2.

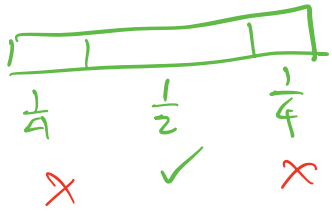
worst-case to find rank is $O(n^2)$

↳ how can we "cleverly" pick pivots to ensure rank of pivot $\geq \frac{1}{4}n$, $\leq \frac{3}{4}n$??

Quickselect

random pivot is enough, $\frac{1}{2}$ chance!

"Las Vegas Algorithm"



Paranoid Quickselect

keep repicking pivot until rank $\geq \frac{1}{4}n$, $\leq \frac{3}{4}n$ before divide+conquer

$\Rightarrow$ expected $\leq 2cn = O(n)$

$\frac{1}{2}$ chance each try

$T(n) = O(n) + T(\frac{3}{4}n)$

$\Rightarrow O(n)$.

Quicksort

1. Pick a pivot → basic - pick 1st or last; randomized; median-find
2. sort elements before and after pivot
3. recurse on two halves L, G.

} $O(n^2)$ worst-case, $O(n \log n)$ avg case

} $O(n \log n)$ all cases $\Rightarrow$ impractical against mergesort though

Paranoid Quicksort using Paranoid Quickselect

- $O(n)$ paranoid quickselect

$T(n) \leq \max [T(i) + T(n-i)] + O(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + O(n)$

(expected; expected)

$\Rightarrow O(n \log n)$ expected

Union Bound

$P(A_1 \cup A_2 \cup \dots \cup A_n) \leq \sum P(A_i)$

Markov's Inequality

$P[X \geq c\mathbb{E}[X]] \leq \frac{1}{c}$

Las Vegas w/ exp time T —run for cT→ Monte Carlo w/ success chance $\geq 1 - \frac{1}{c}$

why does randomized quicksort give sorted w.h.p. in $O(n \log n)$?

OR, with recursion depth L:

let $Y_{i,k}$ = indicator of if container with a_i at k^{th} recursion chooses bad pivot.

$$\mathbb{E}\left[\sum_{i,k} Y_{i,k}\right] \leq \sum_i 0.5L \leq 0.5L$$

(0.5 per level)

$$\mathbb{P}\left[\sum Y_{i,k} \geq (1+0.2) \cdot 0.5L\right] \leq e^{-0.04 \cdot 0.5L/3} = e^{-L/150}$$

let $L = 300 \log n$ ⟶ n^{-2}

$$\mathbb{P}[(\text{any } \textstyle\sum Y_{i,k} \text{ over all } i) \geq 0.6L] \leq \frac{1}{n^2} \cdot n = \frac{1}{n}$$

↳ at least $0.4L$ good pivots

$$\Rightarrow n(3/4)^{0.4L} = n \cdot \left(\frac{3}{4}\right)^{120 \log n} < n^{-33} \leq 1.$$

so $\mathbb{P}[\text{termination}] \geq 1 - \frac{1}{n} \rightsquigarrow 1$ for n large □.

↑ so choice of L is good!

LECTURE 4 2/15/24 11AM **Amortized Analysis**

Setting Data structure, member functions called repeatedly.

Can we more accurately capture cost of using the data structure?

Case Study Union-Find / Disjoint Sets

3 Operations

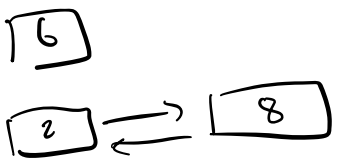
Make-set (x) → add {x} to collection

Find-set (x) → return REP[S(x)]
↑ representative of set of x.

union (x,y) → replace sets S(x), S(y) with single S(x) ∪ S(y) with arbitrary single representative

Application
connected components in graphs

Implementation 1 - doubly-linked lists



Make-set ⇒ O(1)

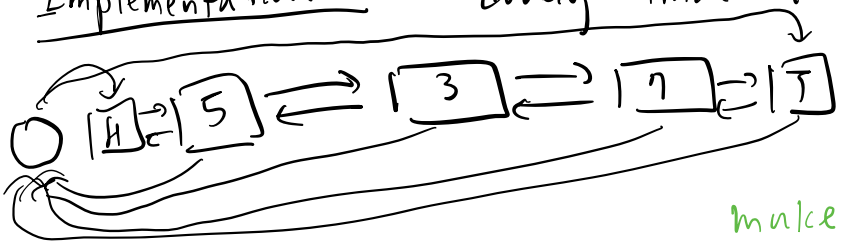
find-set ⇒ O(n), follow pointers to head

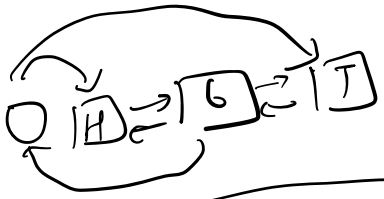
union ⇒ find tail of x
head of y, and
link ⇒ O(n)

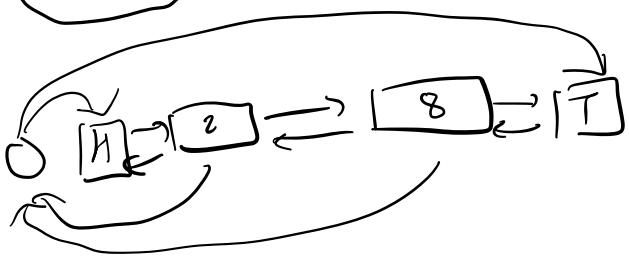
Amortized Cost

An operation has amortized cost T if any sequence of k operations has cost ≤ kT

Implementation 2 - doubly-linked lists w/ head/tails







make-set $\Rightarrow O(1)$

find-set $\Rightarrow O(1)$ by finding head

union $\Rightarrow$ still $O(n)$ if need to update all pointers of $s[r]$ to $\bigcirc \downarrow$

$O(\min\{|S(x)|, |S(y)|\})$

Claim Amortized cost of union is $O(\log n)$

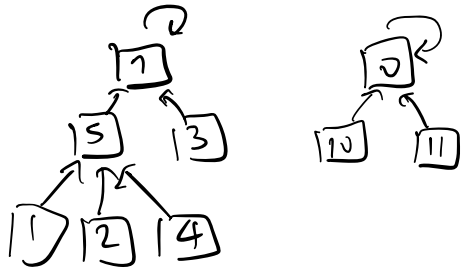
Proof WTS {total # of pointer updates} $\leq O(\log n)$

$\downarrow$

each element x's pointer updated $O(\log n)$ times.

$\Rightarrow$ but pointer only updated if set size more than doubles $|S(y)| \rightarrow |S(x) \cup S(y)| \geq 2|S(y)|$ □

Implementation 3 - trees



make-set $\Rightarrow O(1)$ worst case

find-set $\Rightarrow O(\text{height}(S(x))) \rightarrow O(n)$

union $\Rightarrow$ merge two trees by updating root of shorter one to point to root of larger

$O(\text{height}(S(x)) + \text{height}(S(y))) \rightarrow O(n)$ worst case

$\rightarrow$ union-by-rank.

ensure height of each tree $= O(\log n)$

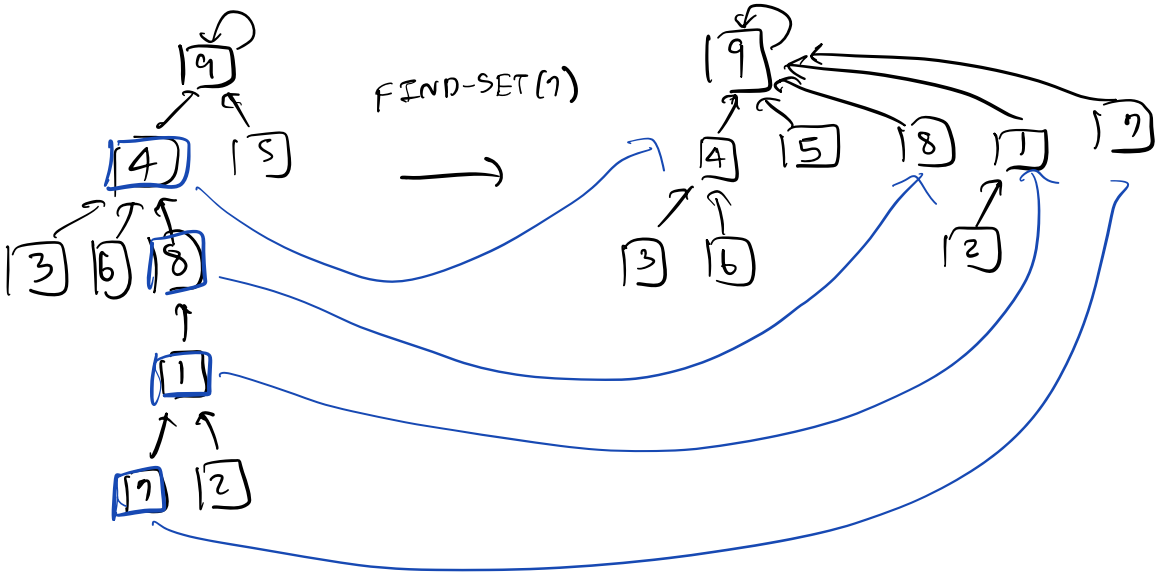
combine shorter tree into larger tree

find-set, union $\Rightarrow O(\log n)$ amortized

proof is by induction ⇒ show first that all tree's have height $O(\log n)$ after n operations.

then show each element's pointer is update depth(x) $\leq O(\log n)$ times □.

→ path compression



Claim w/o Union-by-rank, w/ path compression amortized cost is $O(\log n)$

Pf Let c_i = cost of i^{th} operation.

WTS $\forall k$, $\sum_{i=1}^{k} c_i = O(k \log n)$ → but c_i can vary.

⇒ Potential functions

Suppose we define $\hat{c}_i$ s.t. $\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} \hat{c}_i$ ↑ individually small

$\hat{c}_i = c_i + \Delta\Phi_i$

$= \Phi_i - \Phi_{i-1}$

↑ ↑ both ≥ 0

let $\Phi(D) = \sum_{x \in D} \log \text{size}(x)$ ↑ # of nodes in subtree rooted at x.

use D_0 = initial state of all singletons ⇒ $\Phi(D_0) = 0$,

make-set: $\hat{c_i} = c_i + \Delta\Phi_i \longrightarrow \hat{c_i} = O(1) + 0 = O(1)$.

($\Delta\Phi_i$: no change)

find-set: let $x, u_1, u_2, \ldots, u_h = REP(S(x))$ be the node→root path we traverse. $c_i = h$

we have $u_j.SIZE \longrightarrow u_j.SIZE - u_{j-1}.SIZE$, upon path compression

$$\Delta\Phi_i = \sum_{j=1}^{h-1} \log\left(\frac{u_j.SIZE - u_{j-1}.SIZE}{u_j.SIZE}\right)$$

$$\hat{c_i} = h + \sum_{j=1}^{h-1} \log\left(\frac{u_j.SIZE - u_{j-1}.SIZE}{u_j.SIZE}\right)$$

$$= 1 + \sum_{j=1}^{h-1} 1 + \log\left(\frac{u_j.SIZE - u_{j-1}.SIZE}{u_j.SIZE}\right)$$

note:

$$1 + \log\left(\frac{u_j.SIZE - u_{j-1}.SIZE}{u_j.SIZE}\right) \geq 0$$

$$\Leftrightarrow \log\left(1 - \frac{u_{j-1}.SIZE}{u_j.SIZE}\right) \geq -1$$

$$\Leftrightarrow \frac{u_{j-1}.SIZE}{u_j.SIZE} \leq \frac{1}{2}$$

$$\Leftarrow u_j.SIZE \geq 2\,u_{j-1}.SIZE$$

because u_j can only double at most $\log n$ times to stay under n (since u_j is monotonically increasing)

$$= 1 + \log n$$

$$= O(\log n)$$

union:

$\hat{c}_i = 1 + \log(x.size + y.size) - \log(y.size)$

$= 1 + \log\left(1 + \frac{x.size}{y.size}\right)$

$= O(\log n)$ □.

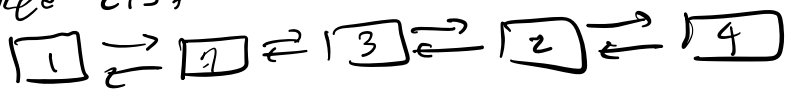
link x root to y root

together union-by-rank + path compression $\Rightarrow O(\alpha(n))$.

LECTURE 5 2/22/24 11AM

self-organizing lists

linked-list



stores n elements w/ distinct keys

one operation - ACCESS(x) = start from head of L, find element with key x.

$\Downarrow$

time/cost = $rank_L(x)$

suppose we have a sequence of N accesses

→ want to minimize total cost

min = N max = nN

but, we can also reorder (via transpositions) L after each access → each transposition costs 1

$C_A(i)$ = total cost of access(x)

= access cost + reordering cost

= $rank_L(x_i)$ + # transpositions after x_i

offline algorithm – knows all access calls to the array beforehand

ex) $L =$ [1 2 3 ... n]

$S = n, n, \ldots, n \qquad |S| = n.$

OPT: access n, move n to front, keep accessing n

(n) ($n-1$) ($|S|-1$)

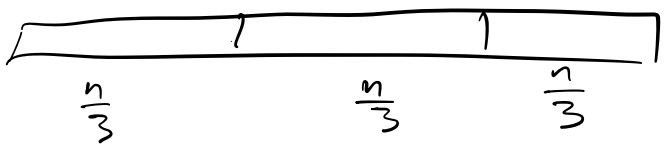
total cost $2n-1+|S|-1 = (2n-2)+N.$

worst case example

$L =$ [1 2 3 ... n]

$S = n, n-1, n-2, \ldots, 1$

divide L into



each of the first $\frac{n}{3}$ calls must either:

– be moved to first $\frac{n}{3}$ spots in list → cost $\geq \frac{n}{3}$

OR

– be accessed somewhere outside of first $\frac{n}{3}$ spots in list → cost $\geq \frac{n}{3}$

so cost is $\geq \underbrace{\frac{n}{3}}_{\text{elements}} \cdot \underbrace{\frac{n}{3}}_{\text{cost}} = \Omega(n^2)$

worst case for online

adversary asks for last element every time $\Rightarrow O(|S|n)$

↳ get access calls one at a time

Move to Front (MTF) Heuristic

after accessing x_i, move x_i to front of L

$\rightarrow$ cost = $\text{rank}_L(x_i)$ + $\text{rank}_L(x_i) - 1$

$\text{rank}_L(x_i)$: access

$\text{rank}_L(x_i) - 1$: move to front

Defn An online algorithm A is α-competitive if there is some cost c s.t. $\forall$ access sequences S,

total cost of A on $S \leq \alpha$ [total cost of OPT] + c

OPT: optimal offline

Thm MTF is 4-competitive.

Proof Let $S = x_1, x_2, \ldots, x_N$ access sequence

L_i = list that MTF has after accessing x_i

L_i^* = list that OPT has after accessing x_i

at i^{th} access x_i

$L \xrightarrow{MTF} L_1 \rightarrow L_2 \rightarrow \cdots \rightarrow L_i \rightarrow \cdots \rightarrow L_N$

MTF has L_{i-1}

$C_{MTF}(i) = 2 \cdot \text{rank}_{L_{i-1}}(x_i) - 1$

$L \xrightarrow{OPT} L_1^* \rightarrow L_2^* \rightarrow \cdots \rightarrow L_i^* \rightarrow \cdots \rightarrow L_N^*$

OPT has L_{i-1}^*

$C_{OPT}(i) = \text{rank}_{L_{i-1}^*}(x_i) + t_i$

we want a potential function to compare L_i^*, L_i.

Def distance(L_i, L_i^*) = min (# of transpositions to transform L_i to L_i^*)

proof relies on fact that

$\#\text{inversions} \le \#\text{transpositions}$

$\#\text{transpositions} \ge \#\text{inversions}$

↑ can fix ≥ 1 inversion each time

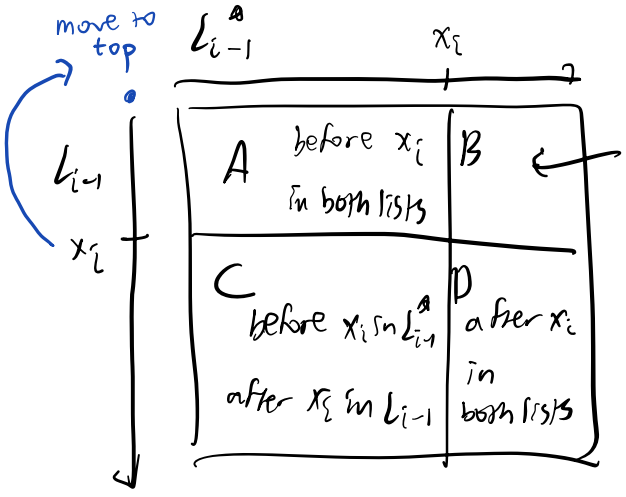
→ = # of inversions (# of pairs in L_i that have swapped order in L_i^*)

every inversion must be fixed

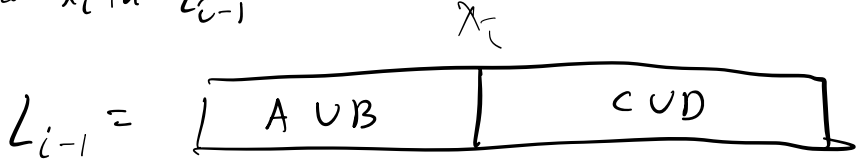
Potential Function $\Phi(i) = 2 \cdot$ # inversions between L_i & L_i^*

$$\hat{c}_{MTF}(i) = c_{MTF}(i) + \Phi(i) - \Phi(i-1)$$

recall $c_{MTF}(i) = 2 \cdot \text{rank}_{L_{i-1}}(x_i) - 1$

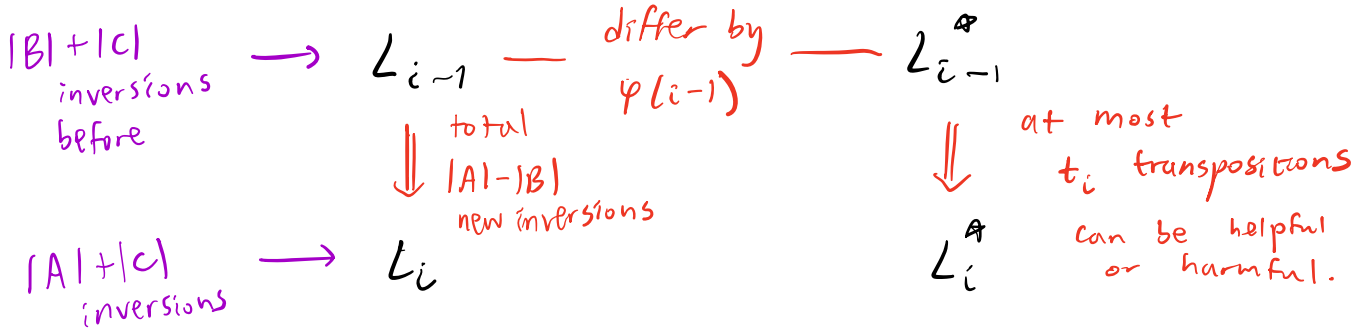


before x_i in L_{i-1}
after x_i in L_{i-1}^*



rank → $|A| + |B| + 1$

rank → $|A| + |C| + 1$



$$\Rightarrow \quad \varphi(i) - \varphi(i-1) \le 2(|A| - |B| + t_i)$$

$$\Rightarrow \hat{c}_{MTF}(i) = c_{MTF}(i) + \varphi(i) - \varphi(i-1)$$

$$\le 2|A| + 2|B| + 1 + 2|A| - 2|B| + 2t_i$$

$$= 4|A| + 2t_i + 1 \le 4\,\text{rank}_{L^*_{i-1}}(x_i) + 4t_i \le 4c^*_i$$

note $\text{rank}_{L^*_{i-1}}(x_i) = |A| + |C| + 1 \Rightarrow |A| \le \text{rank}_{L^*_{i-1}}(x_i) - 1$

so $\hat{c}_{MTF} = c_{MTF} + \varphi(|S|) - \varphi(0)$

$$c_{MTF} \le \hat{c}_{MTF} \le 4c_{OPT}$$

□

LECTURE 6 2/27/24

Hashing

Dictionary has INSERT(x), DELETE(x), SEARCH(x)

$x = (k, v)$ key-value pair

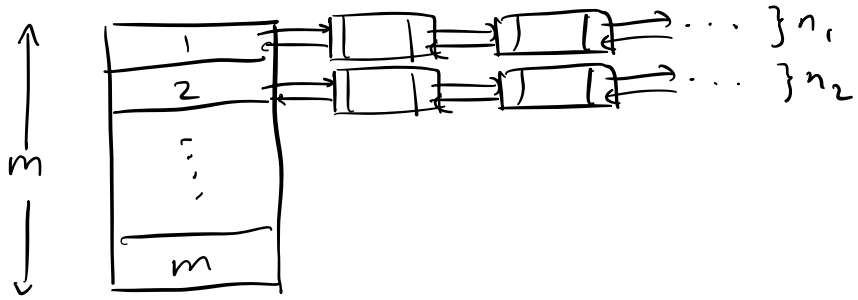
direct access

n keys
m slots
$|U|$ possible keys

SOLUTION 0 - Doubly Linked List $O(n)$ Insert/Delete, $O(n)$ Search

Direct Addressing - $O(1)$ all operations but $|U|$ space

Chaining



INSERT, DELETE $O(1)$

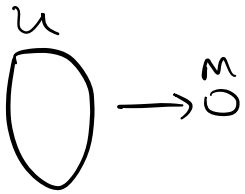
SEARCH $O(n_i)$

best case $n_i = \frac{n}{m} = \alpha$ ← load factor

worst case $n_i = \Theta(n)$

Hash Functions

$h: \mathcal{U} \to M = \{0, 1, \dots, m-1\}$ $\quad m \ll |\mathcal{U}|$



collisions - k_1, k_2 s.t. $h(k_1) = h(k_2)$

random oracle h

- when given a new key k, choose a random location, save it

↳ can give us expected load $\frac{n}{m} = \alpha$

few collisions

for random oracle h, for any $k_1 \neq k_2$, $\mathbb{P}[h(k_1) = h(k_2)] = \frac{1}{m}$

Family of Hash Functions $\quad \mathcal{H} = \{h : \mathcal{U} \to M\}$

UNIVERSAL HASHING for any $k_1 \neq k_2$, $\mathbb{P}[h(k_1) = h(k_2)] \leq \frac{1}{m}$.

how to get such a family $\mathcal{H}$?

base m representation

$\mathcal{H} = \{h_a : a \in \mathcal{U}\}$ $\quad a = (a^{(e-1)}, \dots, a^{(1)}, a^{(0)})$

for $k = (k^{(e-1)}, \dots, k^{(0)})$, $\quad h_a(k) = \sum_{i=0}^{e-1} a^{(i)} k^{(i)} \pmod m$

$$\mathbb{P}[h(k_1) = h(k_2)] = \mathbb{P}[a^T(k_1 - k_2) = 0] = \mathbb{P}[a = 0] = \frac{1}{m}$$

($k_1 \neq k_2$; mod m)

Open Addressing

→ probe sequence $h(k,p)$ = p^{th} preferred location of k

→ uniform (permutation) hashing : probe sequences of different keys are independent uniform permutations

↳ all operations are expected $O(\frac{1}{1-\alpha})$

linear probing : $h_{LIN}(k,p) = h(k) + p \mod m$

quadratic probing : $h_{QUAD}(k,p) = h(k) + c_1 p + c_2 p^2 \mod m$ for c_1, c_2

double hashing : $h_{2HASH}(k,p) = h_1(k) + p \cdot h_2(k) \mod m$

↑ both random hash functions

resizing to lower α → low amortized cost !!

LECTURE 7 2/29/24 11AM

n keys
m slots
$|\mathcal{U}|$ possible keys

	Space	Insert	Delete	Search	Build	
SOLUTION ZERO	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	
DIRECT ADDRESSING	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	
CHAINING	$O(m+n)$	$O(1)$	$O(1)$	exp $O(\alpha)$	$O(n)$	
OPEN ADDRESSING	$O(m)$	exp $O(\frac{1}{1-\alpha})$	exp $O(\frac{1}{1-\alpha})$	exp $O(\frac{1}{1-\alpha})$	exp $O(n)$	
PERFECT HASHING	$O(n)$	–	–	$O(1)$	exp $O(n)$	$m=n$
CUCKOO HASHING	$O(n)$	exp $O(1)$	$O(1)$	$O(1)$	exp $O(n)$	$m>2n$

STATIC dictionary

- no INSERTs / DELETEs
- perfect hashing scheme
  ↳ want no collisions on given set $K \subseteq \mathcal{U}$, $|K| = n$.

NO COLLISIONS

Take universal hash family $\mathcal{H} = \{h : \mathcal{U} \to M\}$

find: $\mathbb{P}[\text{no collisions on } K \subseteq \mathcal{U}]$

given two keys, $\mathbb{P}[\text{collision}] \leq \frac{1}{m}$

over all pairs of keys, $\mathbb{P}[\text{collision}] \leq \frac{\binom{n}{2}}{m} \leq \frac{n^2}{2m}$

make $\boxed{m = n^2}$ $\rightarrow$ $\mathbb{P}[\text{no collisions}] \geq \frac{1}{2}$

expected ≤ 2 random calls to $\mathcal{H}$ for no collisions.

FKS Perfect Hashing

① Pick h_1 from universal hash family $\mathcal{H}_1 = \{h_1 : \mathcal{U} \to M\}$

- hash all keys in K with h_1
- let n_i = # keys with $h_1(k) = i$

(1.5) if $\sum_{i=0}^{m-1} n_i^2 \geq 4n$, resample (expected 2 samples)

② For each i, pick $h_{2,i} \in \mathcal{H}_{2,i} = \{h_{2,i} : \mathcal{U} \to M_i\}$ ← $M_i = \{0, 1, 2, \ldots, n_i^2 - 1\}$

- resample until no collisions

↳ Space used $= O\left(m + \sum_{i=0}^{m-1} n_i^2\right)$ set $m = n$. → expected $O(m+n) = O(n)$

let N_i = random variable for n_i

$$\mathbb{E}\left[\sum_{i=0}^{m-1} N_i^2\right] = \sum_{(k_1,k_2) \in K^2} \mathbb{E}\left[\chi_{h_1(k_1)=h_1(k_2)}\right]$$

indicator for if $h_1(k_1) = h_1(k_2)$

↑ sums over all ordered pairs of collisions

$$= \sum_{k \in K} 1 + \sum_{(k_1,k_2) \in K^2} \mathbb{P}[h_1(k_1) = h_2(k_2)]$$

$$\leq n + \frac{1}{m} \cdot 2\binom{n}{2} \leq n + \frac{2n^2}{2m} \leq n + \frac{2n^2}{2n} \leq 2n.$$

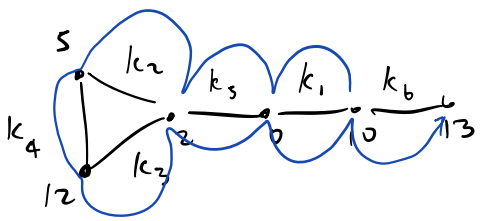
Markov $\mathbb{P}\left[\sum_{i=0}^{m-1} N_i^2 > 4n\right] < \frac{1}{2}$, so $\mathbb{P}\left[\sum_{i=0}^{m-1} N_i^2 \leq 4n\right] > \frac{1}{2}$

so add step (1.5) above → then worst-case SEARCH is $O(1)$. build is exp $O(n)$ space is $O(n)$

Cuckoo Hashing

- two hash functions h_1, h_2
- make list of length $> 2|K|$
- every key k to $h_1(k)$ or $h_2(k)$
  option 0 option 1
- add keys 1 at a time and if there's a collision, kick the old one out. the old one of in option i goes to option $1-i$ and repeat

we can represent this as a graph, where keys k are edges between $h_1(k)$ and $h_2(k)$.



if we start with
$k_1 \to 10$
$k_5 \to 0$
$k_3 \to 12$
$k_4 \to 5$
$k_2 \to 2$

then adding k_6 with $h_1(k_6) = 10$, $h_2(k_6) = 13$ makes a path on our cuckoo graph

time for INSERT = length of path

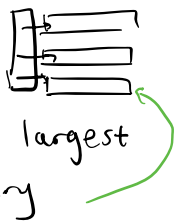
cannot exist, rehash expected $O(1)$ times to obtain valid cuckoo graph

if we bound the number of cycles and lengths of paths between two keys $h_1(k_1)$ $h_2(k_1)$ $h_1(k_2)$ $h_2(k_2)$ we get

$O(\frac{1}{m})$ expected probability of a collision
$\Rightarrow$ expected cost of insert $= O(1)$

→ more hash functions to allow for more slots gives better load factor restrictions than $\frac{1}{2}$.

2-WAY CHAINING



- 1-way chaining gives $\Theta(\frac{\log n}{\log\log n})$ = size of largest dictionary
- 2-way chaining: take 2 oracles, hash to less-full dictionary
  $\Rightarrow \Theta(\log\log n)$

LECTURE 8 3/5/24 11AM

trees

spanning tree = tree of (connected $G = (V, E)$ with vertex set V and edge set $E' \subseteq E$

(1) # edges $= |V| - 1$

(2) not necessarily unique

(3) unique path from u to v $\forall$ edges (u,v) not in T

$P_e \rightarrow P_e \cup \{e\}$ is a fundamental cycle ← e

(4) for every non-tree edge $e = (u,v)$, edge $e' \in P_e$

$T' = T \cup \{e\} \setminus \{e'\}$ also a tree

Minimum spanning tree — not necessarily unique

- Minimum: smallest total weight
- spanning: every vertex reachable
- tree: no cycles, connected

Meta-Greedy-MST

$A = \phi$ (empty)

for $|V|-1$ iterations:

find a "safe" e to add to A

$A \cup \{e\}$ maintains invariant

$A \leftarrow A \cup \{e\}$

return A

runtime: (greedy, irreversible)

$(n-1) \cdot$ (cost of finding safe edge)

invariant

e is safe if $A \cup \{e\}$ is a subset of edges of some MST.

minimum-weight edge e^* of G is always safe!

A cut of a graph G is a partition $S \cup V \setminus S$.

An edge (u,v) crosses the cut if $u \in S$, $v \in V \setminus S$ or vice versa.

A cut respects A if no edge of A crosses the cut (i.e. all on one side)

An edge crossing a cut is a light edge if it has minimum weight among all edges crossing the cut.

Thm Let A be any set of edges of $G = (V, E)$ in some MST. Let $(S, V \setminus S)$ be any cut respecting A. Let e be a light edge for the cut. Then e is safe.

Pf Consider an MST T containing A. Let $e^* = (a,b)$ be our light edge.

Let $p_{(a,b)}$ be the path in T from a to b.

must have some edge e crossing the cut.

Take $T' = T \setminus \{e\} \cup \{e^*\}$ which is still a spanning tree.

lighter

□

Kruskal's greedy step

Find minimum weight edge not forming a cycle.

Proof The edge is a light edge between a tree and $G \setminus \{\text{tree}\}$.

→ union-find can give

$$O(m \log m + n \cdot T_{make\text{-}set} + n \cdot T_{union} + m \cdot T_{find\text{-}set})$$

- $m \log m$: Sort E in non-decreasing order
- $n \cdot T_{make\text{-}set}$: each vertex starts as a "tree"
- $n \cdot T_{union}$: combine two trees
- $m \cdot T_{find\text{-}set}$: to connect trees

$$O(\text{sort}(m) + m\alpha(n))$$

- $O(m \log n)$ otherwise
- $O(m\alpha(n))$ if E sorted OR E can be radix sorted

Prim's Greedy step

Find minimum weight edge incident to growing tree.

Proof The edge is a light edge between growing tree and rest of graph

→ Dijkstra's can give $O(m + n \log n)$ via Fibonacci Heaps.

when to use each??

Kruskal's if E's are sorted / easy to sort

or if m is small enough → $m \leq O(n \log n)$

LECTURE 9 3/7/24 11AM

Max-Flow

Input: A network = directed graph $G=(V,E)$

source $s \in V$ sink $t \in V$

edge capacities $c: E \to \mathbb{R}^{\geq 0}$

let $c(u,v)=0$ if $(u,v) \notin E$

(CLRS) flow

→ inconvenient for flow cycles we care about net flow

let "gross flow" $g: E \to \mathbb{R}^{\geq 0}$

$g(e)$: flow along $e \in E$

feasibility: $0 \leq g(u,v) \leq c(u,v) \quad \forall (u,v) \in E$

flow conservation: $\forall \ v \neq s,t$

$$\sum_{(x,v) \in E} g(x,v) = \sum_{(v,y) \in E} g(v,y)$$

flow in = flow out

(6.1220) Flow

- don't assume s, t are source/sink

let "net flow" $f: V \times V \to \mathbb{R}$ ← can be <0

feasibility: $f(u,v) \leq c(u,v)$ → $(u,v), (v,u) \notin E$ ⇒ $f(u,v)=0$.

if $(u,v) \notin E$, $f(u,v) \leq 0$

flow conservation: $\sum_{u \in V} f(u,v) = 0 \quad \forall \ v \in V \setminus \{s,t\}$

skew-symmetry: $f(u,v) = -f(v,u) \quad \forall \ u,v \in V$

↳ $f(u,u)=0$

↳ $f(u,v)$ = net flow from u to v

↳ for every $v \neq s,t$ $\sum_{u: f(u,v)>0} f(u,v) = \sum_{x: f(v,x)>0} f(v,x)$

since $\sum_u f(v,u) = 0$

positive flow in = positive flow out

Let value of flow f $|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$

Max Flow Problem

Given $G = (V, E, s, t, c)$ we want to find a flow of max value.

Claim: Any flow can be decomposed into a collection of s-t flow paths and flow cycles

Formally, let $\text{supp}_f(G)$ = subgraph of G of edges with $f(u,v) > 0$

Proof: Induct on # edges of $\text{supp}_f(G)$.

If no edges, then $|f| = 0$ and we are trivially done.

Now suppose we have $L > 0$ edges, and that all f' with $|f'| \geq 0$ whose $\text{supp}(G)$ has $< L$ edges can be decomposed.

If we find a cycle C in $\text{supp}_f(G)$

Let $f_{min}(C) = \min_{e \in C} f(e)$.

$$\text{new } f'(u,v) = \begin{cases} f(u,v) - f_{min}(C) & \text{if } (u,v) \in C \\ -f'(v,u) = -(f(v,u) - f_{min}(C)) & \text{if } (v,u) \in C \\ f(u,v) & \text{otherwise} \end{cases}$$

then f' → skew-symmetric by definition

→ flow conservation still works

→ $f'(u,v) \leq f(u,v) \leq c(u,v)$ $\forall$ (u,v) with $(v,u) \notin C$

→ $f'(u,v) = f_{min}(C) - f(v,u) \leq 0 \leq c(u,v)$ for $(v,u) \in C$

consider $\sum_v f'(s,v) = |f'|$.

If C doesn't touch s, $|f'|$ unaffected.

If C goes into and out of s, $|f'| = |f|$ by skew-symmetry

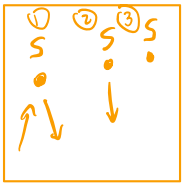
→ we've removed an edge from $\text{supp}_f(G)$ $(\text{argmin}_{e \in C} f(e))$!

Finding the cycle

since all vertices except s, t have positive flow in = positive flow out,

- if $\exists (u,v) \in \text{supp}_f(G)$ with $v \neq t$, $\exists (v,u') \in \text{supp}_f(G)$
- if $\exists (u,v) \in \text{supp}_f(G)$ with $u \neq s$ $\exists (v',u) \in \text{supp}_f(G)$

3 cases for $|f| \geq 0$



DFS from s, we will either find t → cycle ✓ or

edge out of t ↕ symmetric

① then if there's an edge into s, backward DFS to get cycle ✓ or return to t ↓ then we have cycle too ✓

guarantees

③ if no edges into or out of s (and also no edges into/out of t) then DFS from any other $v \in \text{supp}_f(G)$ to get a cycle ✓ without s or t

② otherwise $\exists$ a path P out of s to t. (and no cycles including s or t)

let $f_{min}(P) = \min_{e \in P} f(e)$

new $f'(u,v) = \begin{cases} f(u,v) - f_{min}(P) & \text{if } (u,v) \in P \\ -f'(v,u) = -(f(v,u) - f_{min}(P)) & \text{if } (v,u) \in P \\ f(u,v) & \text{otherwise} \end{cases}$ → still a flow as before!

no edges into s $\Rightarrow$ $f(s,v) \geq 0 \quad \forall \ v \in V$

note that $f'(s,v) = f(s,v) \geq 0 \quad \forall \ v$ except first edge in P: (s,v')

then $f'(s,v') = f(s,v') - f_{min}(P) \geq 0$ so

$|f'| = \sum_v f(s,v) \geq 0$ as desired.

→ In both a cycle and path scenario, we've removed edges to induct. □

Now let f^* be a max flow in G (not necessarily unique)

Let $F^* = |f^*|$. How to determine if $F^* > 0$?

Let G^* = subgraph of G with edge of capacities > 0

$F^* > 0$ $\Longleftrightarrow$ (G^* has an $s \to t$ path)

only paths, not cycles, contribute to flow

can send positive flow along the path

But how do we show that $F^* = 0$, or $\nexists$ $s \to t$ paths in G^*?

→ A cut !

Let $\bar{S} = \{v \in V \mid \exists\ s\text{-}v$ path in $G^*\}$

$t \notin \bar{S}$ $\longleftrightarrow$ no s-t path $\longleftrightarrow$ $F^* = 0$

Def An **s-t cut** is a cut $(S, V \setminus S)$ s.t $s \in S$, $t \in V \setminus S$

Let $c(S) = c(S, V \setminus S) = \sum_{u \in S} \sum_{v \in V \setminus S} c(u, v)$

be the capacity of the cut.

↑ only from $S \to V \setminus S$, not other direction

$c(S) = 0$ $\Longleftrightarrow$ cut separates s from t.

$\Rightarrow$ $F^* = 0 \Leftrightarrow \nexists$ s-t path in $G^* \Leftrightarrow \exists$ s-t cut with $c(S) = 0$

Minimum s-t cut problem

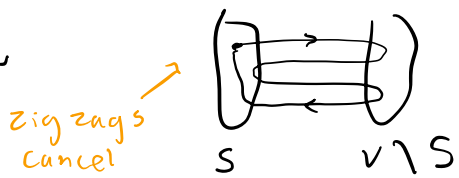
Given $(G = (V, E), s, t \in V, c)$ find an s-t cut of minimum capacity

Trivially $f(S) \leq c(S)$ for any s-t cut S by feasibility.

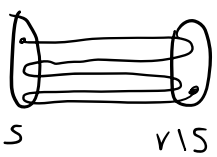
Claim For two s-t cuts S, S' we have $f(S) = f(S')$.

Proof Separate f into flow cycles + s-t flow paths.

→ cycles contribute 0 to flow for both $f(S)$ and $f(S')$



→ paths cancel out except for final crossing which contributes exactly the flow of the s-t path to both $f(S)$ and $f(S')$



□

Note that $\{s\}, V \setminus \{s\}$ is an s-t cut with flow $= |f|$ so $\forall$ s-t cuts, $f(S) = |f|$.

$\Rightarrow$ Max flow $\leq$ min-cut

F^* S^*

$$F^* = |f^*| = f^*(S^*) \leq c(S^*)$$

LECTURE 10 3/14/24 11 AM

Given a flow f, how to increase its value? (or conclude $F^* = |f|$?)

Idea Just try to find an s-t path to push more flow through t.

Residual Network given $G = (V, E, s, t, c)$ and flow f

$$G_f = (V, E_f, s, t, c_f)$$

↳ $c_f(u,v) = c(u,v) - f(u,v) \geq 0$

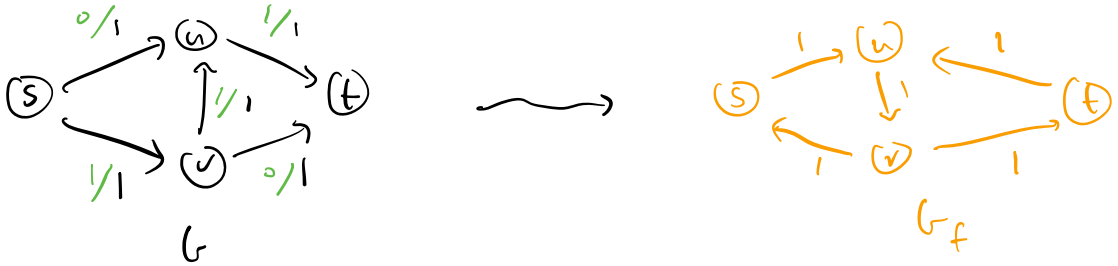
$$E_f = \{(u,v) \mid c_f(u,v) > 0\}.$$

If $(u,v), (v,u) \notin E$, neither has any capacity or flow

⇒ neither is in E_f

⇒ E_f may consist only of edges in G and reverses of edges in G.

ex)



Max Flow Min Cut Theorem

TFAE.

(1) $\exists$ s-t cut $(S, V \setminus S)$ s.t. $c(S) = f(S) = |f|$

(2) f is a max flow

(3) There is no s-t path in G_f

Pf ((1) ⇒ (2)) Let S^* be s.t. $(S^*, V \setminus S^*)$ is a min s-t cut.

let f^* be a max flow

$$|f| \leq |f^*| \leq c(S^*) \leq c(S) = f(S) = |f| \Rightarrow |f| = |f^*|.$$

(max flow; weak duality; min cut; all s-t cuts S)

↳ max flow = min cut

(③ ⇒ ①) Suppose no s-t path in G_f.

$\Rightarrow \exists$ a cut $(S, V \setminus S)$ s.t. $c_f(S) = 0$

$c(S) = |f| = f(S)$

$\Rightarrow 0 = c_f(S) = \sum_{u \in S} \sum_{v \in V \setminus S} c_f(u,v) = \sum_{\substack{u \in S \\ v \in V \setminus S}} c(u,v) - f(u,v) \Rightarrow \sum_{\substack{u \in S \\ v \in V \setminus S}} c(u,v) = \sum_{\substack{u \in S \\ v \in V \setminus S}} f(u,v)$

(NOT ③ ⇒ NOT ②) contrapositive

Suppose P is an s-t path in G_f.

Let $c_f(P) = \min_{e \in P} c_f(e) > 0$ by definition of G_f.

we can push flow $c_f(P)$ along P as we've done before!

↳ $f'(x,y) = \begin{cases} f(x,y) + c_f(P) & \text{if } (x,y) \in P \\ -f(y,x) - c_f(P) & \text{if } (y,x) \in P \\ f(x,y) & \text{otherwise} \end{cases}$

- skew-symmetric
- feasible

(check for all 3 cases $(x,y) \in P$, $(y,x) \in P$, otherwise)

now $|f'| = \sum_x f'(s,x) = \sum_{x \neq u} f(s,x) + f'(s,u) = |f| + c_f(P) > |f|$

(s,u) = first edge of P

↳ so ① is false! □

Thus, we can improve the flow of and G iff there is an s-t path in G_f.

Ford-Fulkerson Algorithm

Start with f = all 0 flow

While $\exists$ s-t path in G_f:

  $P \leftarrow$ s-t path in G_f

  augment f via $c_f(P)$ along P] $O(|E|)$ via BFS/DFS

return f] f is a max flow!

→ $O(|E| \cdot \text{iterations})$ overall!

Runtime: If all capacities $\in \{0, 1, \ldots, C\}$ integers, then each $c_f(e) \geq 1$ so at most $O(mF^*) \leq O(mnC)$ pseudo polynomial

F^*: at most $n-1$ edges from s

Flow Integrality Theorem

If all capacities $\in \mathbb{Z}$, then there is an integral max flow.

→ rational capacities still finite runtime → real capacities may take forever

Two Variants!

1) Max Bottleneck Path Algorithm (MBP)

- find P in G_f from s to t that maximizes $c_f(P)$
  - ↳ doable in $O(m \log n)$
- then augment along P.

Runtime $O(m \log |f|) = O(m^2 \log n \log nC)$

2) Edmonds-Karp Algorithm

Find p with minimum # of edges. Augment along P.

Runtime $O(m^2 n)$ → each edge part of at most $\frac{n}{2}$ augmenting paths → $O(mn)$ augments
→ BFS to find shortest path in $O(m)$

Strongly Poly Runtime ⟶ no dependence on edge weights (fast, even for real weights)
⇝ real inputs

Edmonds Karp $O(m^2 n)$ Best $O(mn)$

weakly Poly Runtime ⟶ polynomial dependence on $\log C$ (fast for $C \sim \text{poly}(n)$)
⇝ integer inputs

MBP $O(m^2 \log n \log nC)$ Best $O(m^{1+o(1)} \log C)$

pseudo Poly Runtime ⟶ polynomial dependence on C (only fast for C small)

Ford-Fulkerson $O(m|f|) \leq O(mnC)$

LECTURE 11 3/19/24 11AM

Linear Programming

max / min $c_1 x_1 + \cdots + c_n x_n$ ← (linear) objective function

subject to m+n constraints

$$\left.\begin{aligned} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n &\{\leq, \geq, =\} b_1 \\ &\vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n &\{\leq, \geq, =\} b_m \end{aligned}\right\} \text{linear constraints}$$

$x_1, x_2, \ldots, x_n \geq 0$ ← non-negativity constraints

n decision variables

Standard Form

can make $\min\ c_1x_1 + \cdots + c_nx_n \rightarrow \max\ (-c_1)x_1 + \cdots + (-c_n)x_n$

can make $A_{i,1}x_1 + \cdots + A_{i,n}x_n \geq b_n \rightarrow (-A_{i,1})x_1 + \cdots + (-A_{i,n})x_n \leq (-b_n)$

can make $A_{i,1}x_1 + \cdots + A_{i,n}x_n = b_n \rightarrow \begin{cases} A_{i,1}x_1 + \cdots + A_{i,n}x_n \leq b_n \\ (-A_{i,1})x_1 + \cdots + (-A_{i,n})x_n \leq (-b_n) \end{cases}$

can represent LP as → $\max\ c^T x$ subject to $Ax \leq b$, $x \geq 0$

$A = m \times n$, $b = m \times 1$, $c = n \times 1$, $x = n \times 1$

feasibility - feasible region $= \{x \in \mathbb{R}^n \mid Ax \leq b \wedge x \geq 0\}$

infeasible if region $= \emptyset$, feasible otherwise

boundedness - feasible LP is either bounded (↑ optimum $< \infty$) or unbounded (↑ optimum $= \infty$)

feasible region is either bounded (↑ fits into finite ball) or unbounded (↑ not in any finite ball)

bounded feasible region → bounded optimum → unique optimal value

degeneracy - bounded LP can have multiple optimal solutions

Geometry of an LP

$n=1$: feasible region is $[\cdot,\cdot]$ or $[\cdot,\infty)$

① feasible region is single continuous interval

② optimal objective function is at corner point

lower bound if $c_1 < 0$
0 if $c_1 = 0$
upper bound if $c_1 > 0$

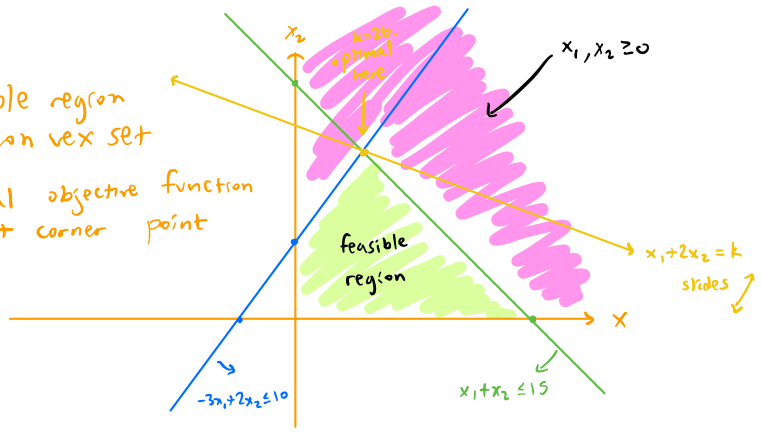
$n=2$:

ex) P_5: max x_1+2x_2

$-3x_1+2x_2 \le 10$
$x_1 + x_2 \le 15$
$x_1, x_2 \ge 0$

① Feasible region is convex set

② optimal objective function is at corner point



$n=3$: planes bound a space.

Solving Linear Programs → doable in poly time.

Duality

false P_5: max x_1+2x_2 again. Try $x_1+2x_2 \le 2(x_1+x_2) \le 30$.

$-3x_1+2x_2 \le 10$
$x_1 + x_2 \le 15$
$x_1, x_2 \ge 0$

$$x_1+2x_2 \le \frac{1}{6}x_1+2x_2$$
$$= \frac{1}{6}(-3x_1+2x_2) + \frac{5}{3}(x_1+x_2)$$
$$\le \frac{10}{6} + \frac{25}{3} = \frac{80}{3}$$

$$x_1+2x_2 \le \frac{1}{5}(-3x_1+2x_2) + \frac{8}{5}(x_1+x_2)$$
$$\le 2 + 24 = 26 \quad \text{as desired.}$$

LECTURE 12 3/21/24 (I did not attend)

Primal		Dual
Max $c^T x$	$\rightleftharpoons$	min $b^T y$
subject to $Ax \le b$		subject to $A^T y \ge c$
$x \ge 0$		$y \ge 0$

note dual of the dual = primal.

ex) max x_1+2x_2
$-3x_1+2x_2 \le 10$
$x_1+x_2 \le 15$
$x_1, x_2 \ge 0$

$y_1(-3x_1+2x_2) \le 10y_1$
$y_2(x_1+x_2) \le 15y_2$
$-3y_1+y_2 \ge 1$
$2y_1+y_2 \ge 2$
min $10y_1+15y_2$.

beyond the standard form...

Primal	Dual
constraint ($\leq, \geq, =$)	decision variable ($\geq, \leq$, unrestricted)
decision variable ($\geq, \leq$, unrestricted)	constraint ($\geq, \leq, =$)
coefficients of objective	constraint bounds
constraint bounds	coefficients of objective

Weak Duality

$$\begin{aligned} \text{Max} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \quad \rightleftharpoons \quad \begin{aligned} \text{min} \quad & b^T y \\ \text{subject to} \quad & A^T y \geq c \\ & y \geq 0 \end{aligned}$$

Let $\vec{\tilde{x}}$, $\vec{\tilde{y}}$ be primal and dual feasible solutions.

Then $c^T \vec{\tilde{x}} \leq b^T \vec{\tilde{y}}$

PF

$$c^T \tilde{x} = \tilde{x}^T c \leq \tilde{x}^T A^T y \leq b^T y.$$

↳ primal unbounded → dual infeasible

↳ dual unbounded → primal infeasible

↳ primal bounded → dual bounded OR infeasible

↳ dual bounded → primal bounded OR infeasible

> both primal AND dual may be infeasible

Strong Duality

primal bounded ⟷ dual bounded.

intuition - ball with coords $(y_1, y_2, \ldots)$ dropped into feasible region of dual space. moves to optimal solution where normal forces of walls (primal constraints) cancel gravity

magnitudes x_i

points normal to objective function

optimal dual solution → primal (bordering) constraints maxed → optimal primal solution

note - not all walls will be used.

suppose only some walls i will be used

↳ in dual solution this means $\underbrace{A^{(i)T}}_{\text{column vector}} y = c_i$ (on wall)

↳ all other constraints are slack ⟺ $x_i = 0$ (no normal force for non-touching walls)

"complementary slackness"

So at optimality, $x_i (A^{(i)T} y - c_i) = 0 \quad \forall i \in \{1, 2, \ldots, n\}$

(x_i: 0 for no wall; $A^{(i)T} y - c_i$: 0 for yes wall)

LECTURE 13 4/2/24 (Did not attend)

P, NP, NP-Completeness

poly-time	exp-time, faster unknown
unweighted shortest path in $O(\|E\|)$	unweighted longest path in $O(1.66^{\|V\|})$ undirected, $O(2^{\|V\|} \cdot \|E\|)$ directed
maximum bipartite matching in $O(\|E\|^{1+o(1)})$	maximum tripartite matching in $O(3.52^{\|V\|})$
linear programming in $O(n^{2.372})$ $n \times n$ matrix A	integer programming (integer vector) $n \times n$ integer matrix A → exp-time even if $A, b, c, x \in \{0,1\}$

		optimization	search	decision
s-t shortest path	I	G, s, t	G, s, t, k	G, s, t, k
	O	min-weight s-t path	simple path of weight $\leq k$	YES if ∃ path of weight $\leq k$ NO otherwise
s-t longest path	I	G, s, t	G, s, t, k	G, s, t, k
	O	max-weight s-t path	simple path of weight $\geq k$	YES if ∃ path of weight $\geq k$ NO otherwise
max bipartite matching	I	G	G, k	G, k
	O	maximum pairing of vertices	pairing of $\geq 2k$ vertices	YES if ∃ pairing of $\geq 2k$ vertices NO otherwise

solving optimization → search → decision

↳ no efficient algorithm for decision

⇓

no efficient algorithm for optimization, search

A decision problem π is solvable in poly time if $\exists$ algorithm A and constant c s.t. $\forall$ inputs x of size n,

↳ $A(x)$ runs in $O(n^c)$

↳ $\pi(x) = \text{YES}$ iff $A(x)$ returns 1.

P = All decision problems solvable in poly time. (deterministic)

NP = decision problems that are easy to verify in poly time.

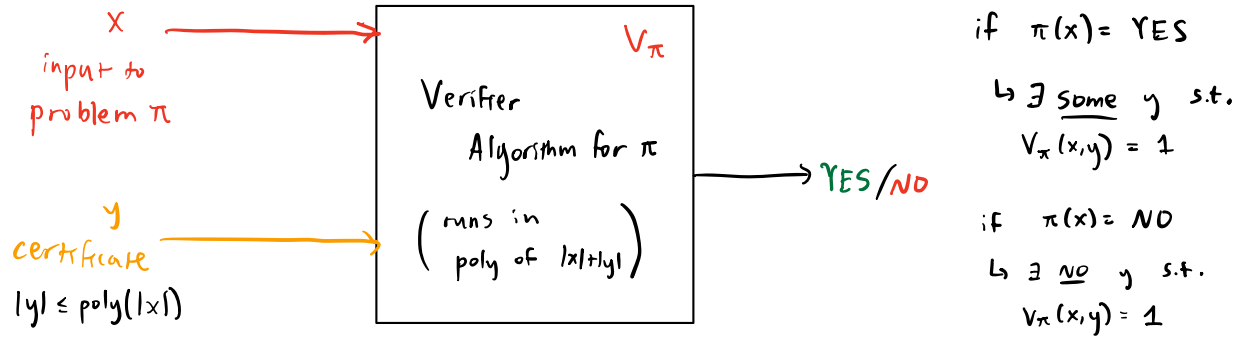
Formally, $\pi \in NP$ if $\exists$ verification algorithm V_π and constants c, c' s.t.

↳ V_π takes two inputs x, y where $|y| \leq n^{c'}$

↳ $V_\pi(x,y)$ takes $O((|x|+|y|)^c)$ time

↳ for every input x of size n,
$\pi(x) = \text{YES}$ iff $\exists$ y of length $\leq n^{c'}$
s.t. $V_\pi(x,y) = 1$.

y is a certificate proving that x is a YES instance of π



y is usually just the decision problem itself

remember, only decision problems can be in P, NP, etc.

no-long path - given $G=(V,E)$, $s,t \in V, k$, is there no path from s-t of length $\geq k$? $\in$ co-NP unknown if $\in NP$

Thm $P \subseteq NP$.

Also known

$P \subsetneq EXPTIME \subsetneq R$ (↑ decidable problems)

Pf Take any $\pi \in P$.

Let A_π be the poly-time algorithm for π.

Then let V_π just run A_π and ignore y.

Runs in $poly(|x|) \leq poly(|x|+|y|)$. □

Thm $NP \subseteq EXPTIME$ ← decision problems solvable in $2^{poly(n)}$

Pf Sketch Just run verifier on all strings y of length n^c

→ $exp(n)$

NP-Complete "hardest problems in NP"

① Must be in NP

② Must be NP-hard

↳ "poly-time algorithm for π can be used to solve any NP problem"

Let Q and π be decision problems.

A **many-one polynomial time reduction** from Q to π is an algorithm R that takes input = instance x of Q,

- runs in $poly(|x|)$ time
- returns instance y of π

s.t. $Q(x) = YES \rightarrow \pi(y) = YES$

$Q(x) = NO \rightarrow \pi(y) = NO$

→ y will have length $poly(|x|)$ since R runs in $poly(|x|)$

$Q \leq_p \pi$

"many-one" because many instances x can go to one instance y

Claim If $Q \leq_p \pi$ and $\pi \in P$, $Q \in P$.

Pf Let A be algorithm for π, let R be the many-one poly time reduction of Q to π. Given instance x of Q, take $A(R(x))$ for π, which has same answer as x for Q.

Runtime: A, R both run in poly time of input → $Q \in P$.

LECTURE 14 4/19/24 (Did not attend)

Claim $A \leq_p B$, $B \leq_p C \rightarrow A \leq_p C$

Pf R_1 reduces A to B in $O(n_1^c)$

R_2 reduces B to C in $O[n_2^{c'}]$ (n_2)

$\rightarrow$ $R_2 \circ R_1$ reduces $A \mapsto C$ in $O((n_1^c)^{c'}) = O(n_1^{cc'}) = \text{poly}(n_1)$ □

$A \Leftrightarrow B \mapsto C$

YES YES YES

So if H is NP-complete

$\forall Q$ with $H \leq_p Q$ and $Q \in NP \rightarrow Q$ is NP-complete.

$\forall Q \in NP \rightarrow Q \leq_p H$

$A \leq_p B$ and $B \leq_p A \Leftrightarrow A, B$ equivalent.

↳ All NP-complete problems are equivalent

Def CNF Formula on $x_1, \ldots, x_n$ (Boolean literals) is of the form (conjunctive normal form)

$$\bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{p} \ell_{ij} \right)$$

every $\ell_{ij} = \neg x_k$ or x_k for $k \in [n]$.

Def k-SAT : $p = k$ for every clause in CNF Formula

1SAT, 2SAT $\in P$ $k \geq 3$ gives NP-complete

Def A vertex cover of $G = (V, E)$ is a subset $S \subseteq V$ s.t. $\forall (u,v) \in E$, either u or v or both in S. Size $= |S|$

VERTEX-COVER

Input : $G = (V, E)$, integer k

Output : does there exist a vertex cover of G of size $\leq k$?

VERTEX-COVER $\in NP$

↳ V_π checks $y = S \subseteq V$ s.t. $|S| \le k$ and every edge $\in E$ has at least one endpoint in S ↳ poly-time

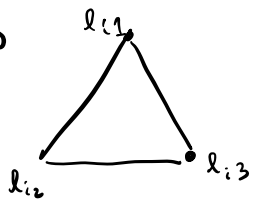
Thm $3\text{-SAT} \le_p$ VERTEX-COVER.

First construct problem from instance of 3-SAT,

Given $\Psi = (\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge (\ell_{21} \vee \ell_{22} \vee \ell_{23}) \wedge \cdots \wedge (\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})$

↑ literals x_k or $\neg x_k$

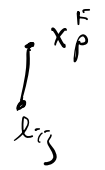
① for every x_k → 

② for every clause → 

$k = n + 2m$

③ for each ℓ_{ij} in clause i

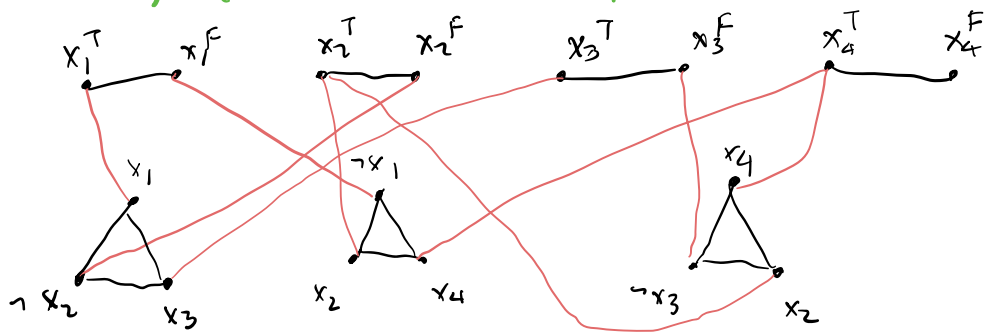
if $\ell_{ij} = x_p$ 

else $\ell_{ij} = \neg x_p$ 

ex) $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_4 \vee \neg x_3 \vee x_2)$

$x_1, \ldots, x_n$

$\ell_{i1}, \ell_{i2}, \ell_{i3}$
$1 \le i \le m$



YES instance of 3SAT → YES instance of VERTEX-COVER

(a) → put $x_k^T \in S$ for $x_k = 1$ else $x_k^F \in S$

(b) → pick one satisfied literal (from 2 or 3 if needed) per clause, and put other two literals $\in S$.

obviously ①, ② type edges are covered by S, $|S| = n + 2m$.

③: if x_k^T — x_k if neither is covered, $x_k = 1$ by (b)
but then $x_k^T \in S$ by (a) → contradiction

x_k^F — $\neg x_k$ if neither is covered, $x_k = 0$ by (b)
but then $x_k \in S$ by (a) → contradiction □

YES instance of VERTEX-COVER → YES instance of 3SAT

read off $x_k = 1$ if x_k^T, else $x_k = 0$.

note we need $\geq n \in S$ in the ① type edges

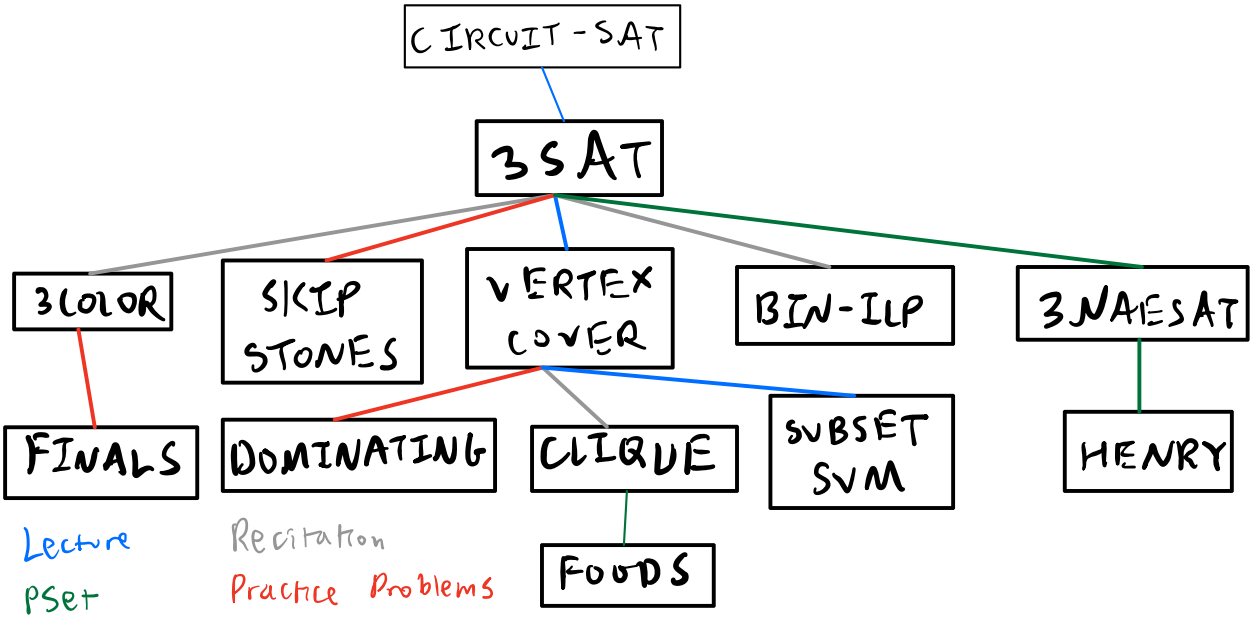
note we need (disjoint) $\geq 2m \in S$ in the ② type edges

so $|S| = n + 2m$ exactly.

↳ exactly one of x_k^T or x_k^F is covered.

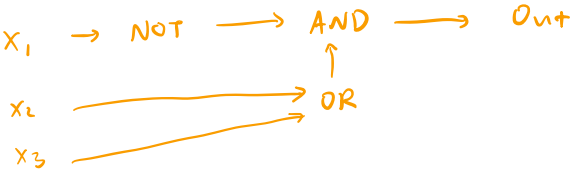
for each clause

if all three literals weren't satisfied, then entire triangle $\in S$ → contradiction since $k = n+2m$ → EXACTLY 2 in S. □



LECTURE 15 4/9/24 (Did not attend)

Circuits: AND OR NOT



$\longleftrightarrow$ $\neg x_1 \wedge (x_2 \vee x_3)$

can evaluate assignment in linear time via topological sort then Dynamic Programming.

CIRCUIT-SAT: given circuit C on n inputs $x_1, \ldots, x_n$,

YES iff ∃ assignment of x_i's to output 1 on C.

↳ CIRCUIT-SAT ∈ NP.

Cook-Levin Theorem CIRCUIT-SAT is NP-complete.

WTS ∃ many-one poly-time reduction from every NP problem to CIRCUIT-SAT.

↳ Use existence of verifier algorithm.

↳ make following assumptions on machine model

① program is stored in memory as sequence of instructions
- operations to perform
- addresses of operands
- addresses to store results

② program counter maintains where we are in program
↳ branching for loops, etc.

③ memory holds entire state of program
↳ program counter, program, storage, bookkeeping

④ runtime $O(p(n))$ means we use $O(p(n)\log(p(n)))$ space
(pointer size in RAM)

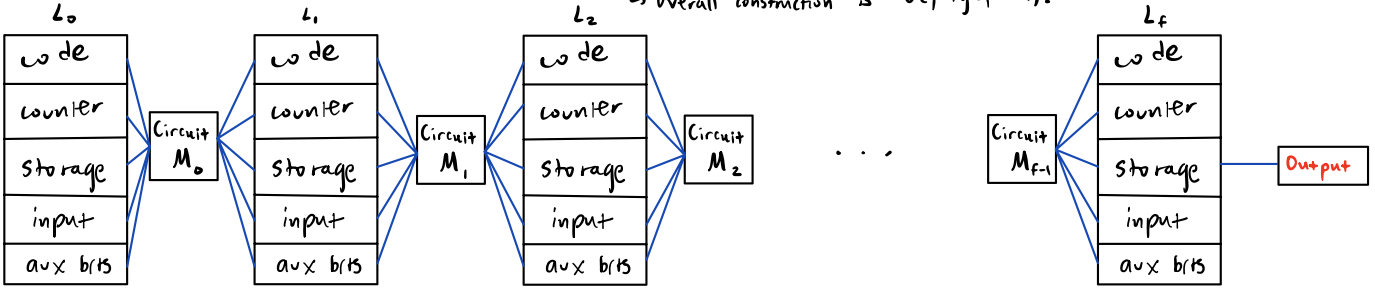
⑤ Main Assumption Let L_i be memory at step i of algorithm. From each $L_i \to L_{i+1}$, ∃ Boolean circuit taking L_i to L_{i+1}, constructable in $\text{poly}(p(n))$ time

then with these assumptions, we get

Thm Suppose Q is a decision problem that can be solved by a program P in $p(n)$ time on inputs of size n. Then for fixed n ∃ Boolean circuit C_n constructable in $O(\text{poly}(p(n)))$ time s.t. ∀ inputs $x_1, \ldots, x_n$,

$$Q(x_1, \ldots, x_n) = C_n(x_1, \ldots, x_n).$$

Pf Idea $f = p(n)$ steps, each M_i constructable in $O(\text{poly}(p(n)))$ time
↳ Overall construction is $O(\text{poly}(p(n)))$.



Proof of Cook-Levin

Let A be any problem $\in NP$ w/ verifier V_A running in poly time.
Then $\exists$ circuit C_N $\forall$ input of size N computable in poly(N) time

if

$V_A(x,y)$ can be represented as $C_N(x_1, \dots, x_n, y_1, \dots, y_{p(n)})$

where $N = n + p(n)$, just hardcode x_i's into C_N

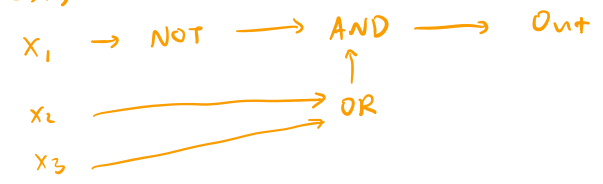
YES-instance of $A \Leftrightarrow \exists y$ for $V_A \Leftrightarrow$ satisfying instance of C_N □

Now we need CIRCUIT-SAT $\leq_p$ 3SAT.

Pf Encode each gate using a variable

(reduction) gate i: $g_i \Leftrightarrow \neg x$ $g_i \Leftrightarrow x \wedge y$ $g_i \Leftrightarrow x \vee y$

ex)



$(g_1 \Leftrightarrow \neg x_1) \wedge (g_2 \Leftrightarrow x_2 \vee x_3)$

$(g_3 \Leftrightarrow g_1 \wedge g_2) \wedge g_3$

↑ final output should be true.

$g_i \Leftrightarrow \neg x$ $g_i \Leftrightarrow x \wedge y$ $g_i \Leftrightarrow x \vee y$

↓ ↓ ↓

$(g_i \vee x) \wedge (\neg g_i \vee \neg x)$ $(\neg g_i \vee a \vee b) \wedge (g_i \vee \neg a) \wedge (g_i \vee \neg b)$

$(g_i \vee \neg a \vee \neg b) \wedge (\neg g_i \vee a) \wedge (\neg g_i \vee b)$

Can be used to create exact 3SAT instance!

□

LECTURE 16 4/11/24 11AM

$\mathcal{P}$ - class of minimization/maximization problems

$P \in \mathcal{P}$ - one instance

x = solution to some P, $Q_P(x)$ outputs quality of x

$OPT(P)$ = quality of optimal solution to P

For any algorithm A, $A(P)$ - quality of A's solution to P

Def $\mathcal{P}$ - class of minimization problems

$\alpha \geq 1$ - approximation factor

We say A is an α-approximation for $\mathcal{P}$ if and only if $\forall P \in \mathcal{P}$

$$[OPT(P) \leq] \; A(P) \leq \alpha \cdot OPT(P)$$

(by def)

max: $\alpha \geq 1$ s.t.

$$[OPT(P) \geq] \; A(P) \geq \frac{1}{\alpha} OPT(P)$$

Maximum Matching - max nonoverlapping pairs of vertices in G connected by edges

- ↙ max flow for bipartite
- ↓ solvable in poly time in general

greedy algorithm - take edges and add to matching if possible.

Claim 2-approximation. ⇝ tightness (upper bound is optimal) @

PF If M is output of our algorithm, M^* is optimal, then $|M^*| \leq 2|M|$.

For $e \in M$, $e^* \in M^*$, add label "e^*" to e if e "blocks" e^* i.e. they share a vertex

Then each edge $e \in M$ can have at most two labels.

But also every edge $e^* \in M^*$ must be blocked by some $e \in M$ (or just add e^* to M).

$$2|M| \geq \sum_{e \in M} \#\text{labels} \geq |M^*|$$

□

Vertex Cover

S
$\downarrow$

greedy algorithm - find maximal matching, take all endpoints.

Claim 2-approximation ~ tightness @ 

S is a vertex cover. If $\exists$ some edge e not covered by S, M should have found it.

$|M| \leq |S^*|$ because just consider edges in M → S^* must contain one vertex for each $e \in M$.

↳ $|S| = 2|M| \leq 2|S^*|$

note: we can't get much better than 2, currently $2 - O\left(\frac{1}{\sqrt{\log|V|}}\right)$

$\sqrt{2} - \varepsilon$ is best if $P \neq NP$, but no known algorithm.

Weighted Vertex Cover

$G = (V, E, w)$ → we want minimum weighted vertex cover $\sum_{v \in S} w(v)$

$w: V \to \mathbb{R}^{\geq 0}$ → LP rounding...

LP Rounding

integer program → relax to LP → solve LP → round solution.

let x_v for $v \in V$ be indicator variables for keeping/excluding v in S

want $\min \sum_{v \in V} x_v$ where

$x_a + x_b \geq 1 \quad \forall \{a, b\} \in E$

$x_v \in \{0, 1\} \quad \forall v \in V$

↳ $0 \leq x_v \leq 1 \quad \forall v \in V$

solve to get $x_v^* \ \forall \ v \in V$.

$$x_v := \begin{cases} 0 & x_v^* < \frac{1}{2} \\ 1 & x_v^* \geq \frac{1}{2} \end{cases}$$

Claim S is a cover.

Pf Each $\{u, v\} \in E$ has $x_u^* + x_v^* \geq 1$ → at least one is $\geq \frac{1}{2}$

↳ $\max(x_u, x_v) = 1$ □

Now we have

$$\sum_{v \in S} w(v) = \sum_{v \in V} w(v) x_v \leq 2 \sum_{v \in V} w(v) x_v^* \leq 2 \sum_{v \in S^*} w(v)$$

2-approximation!

Max k-SAT Given k-CNF formula ϕ, find an assignment satisfying the max # of clauses. (Assume literals in each clause distinct)

↑ decision is NP-complete

optimization is harder

Algorithm Independently set each x_i T or F

↳ each clause has $1-2^{-k}$ chance of being satisfied

↳ approximation ratio $\alpha = \frac{1}{1-2^{-k}}$.

Now derandomize.

$$\mathbb{E}_{x_1,\dots,x_n}[\#SAT(\phi)] = \frac{1}{2}\mathbb{E}_{x_2,\dots,x_n}[\#SAT(\phi) \mid x_1=T] + \frac{1}{2}\mathbb{E}_{x_2,\dots,x_n}[\#SAT(\phi) \mid x_1=F]$$

assign x_1 the one with better conditional expectation

↳ deterministic $\alpha = \frac{1}{1-2^{-k}}$-approximation.

KNAPSACK n items, (v_i, w_i) need to max $\sum v$ with $\sum w \le W$.

Greedy Algo 1

Best $\frac{v_i}{w_i}$ first.

But if we have $K=2C$, items $(2C, 2C), (2, 1)$

our approximation ratio is $\frac{2C}{1} \to 2C$.

Greedy Algo 2

Take one item with best v_i.

But if we have $W = \frac{C(C+1)}{2}$ and items $(1,1), (2,1) \dots, (C,1)$ then $\alpha = \frac{\frac{C(C+1)}{2}}{C} = \frac{C+1}{2}$.

LP Relaxation → Fractional Algorithm

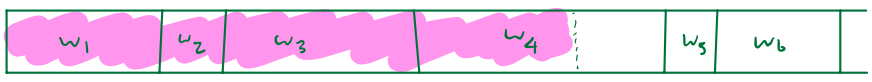
$$\max \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le W$$

$$x_i \in \{0,1\} \quad \forall i \in [n]$$

$$\Downarrow$$

$0 \le x_i \le 1$ relaxation



decreasing order of $\frac{v_i}{w_i}$

↑ take largest fraction of first non-fitting item and add that to x_i

Final 2-approx Algo

take max(Greedy Algo 1, Greedy Algo 2)

Pf relaxed LP optimal $\geq$ integer LP optimal

but Greedy Algo 1 + Greedy Algo 2 $\geq$ relaxed LP $\geq$ integer LP $=$ OPT

→ contains all non-fractional items compared to relaxed LP

→ value > fractional item in relaxed LP

∴ our algorithm gets value $\geq \frac{OPT}{2}$

LECTURE 17 4/16/24 11AM

Online Learning (Part 2)

→ LEARNER, set of actions $a \in A$ ← known to LEARNER & ADVERSARY

→ at each time step t

(1) pick some probability distribution p_t on A

(2) pick action according to p_t

(3) feedback $c_t(a)$, c_t is cost/loss function

↑ known to adversary

known to learner after a picked

Goal

$$\text{Loss} = \mathbb{E}[C] = \sum_{t=1}^{T} \sum_{a \in A} p_t(a)\, c_t(a)$$

Benchmark B ← random variable

"no regret" if $\lim_{T\to\infty} \frac{1}{T}(\mathbb{E}[C] - \mathbb{E}[B]) = 0$

"best" for $B = \sum_{t=1}^{T} \min_{a\in A} c_t(a)$

↳ infeasible

"ok": $B = \min_{a\in A} \sum_{t=1}^{T} c_t(a)$

At time step t

- take weighted majority of n experts' predictions

for each expert i. $w_i^0 = 1 \ \forall i$.

correct: $w_i^{t+1} = w_i^t$

incorrect: $w_i^{t+1} = (1-\varepsilon) w_i^t$

$$W^t := \sum_{i=1}^{n} w_i^t$$

$$W^{t+1} = \sum_{i=1}^{n} w_i^{t+1} = W^t - \varepsilon \sum_{i\text{th expert incorrect}} w_i^t \leq \left(1 - \frac{\varepsilon}{2}\right) W^t$$

$\geq \frac{1}{2}$ of experts wrong if majority wrong

only if LEARNER wrong

let i^* = best expert, makes m^* mistakes over T steps

$$w_{i^*}^T = (1-\varepsilon)^{m^*}$$

now if our method makes m mistakes over T steps

$$W^T \leq \left(1 - \frac{\varepsilon}{2}\right)^m n$$

we have

$$w_{i^*}^T \leq W^T \Rightarrow (1-\varepsilon)^{m^*} \leq \left(1 - \frac{\varepsilon}{2}\right)^m n$$

$$m^* \ln(1-\varepsilon) \leq m \ln\left(1 - \frac{\varepsilon}{2}\right) + \ln n$$

VI, IA — for $0 \leq \varepsilon \leq \frac{1}{2}$

$$m^*(-\varepsilon - \varepsilon^2) \leq -m\frac{\varepsilon}{2} + \ln n$$

external regret → $m - m^* \leq (1+2\varepsilon)m^* + \frac{2\ln n}{\varepsilon}$

but $\lim_{T\to\infty} \frac{1}{T}\left((1+2\varepsilon)m^* + \frac{2\ln n}{\varepsilon}\right) \geq 1$ is possible,

so no vanishing regret

Randomized Multiplicative Weights

At time step t

- pick expert i based on distribution $(w_1^t, w_2^t, \dots, w_n^t)$

for each expert i

correct	incorrect
$w_i^{t+1} = w_i^t$	$w_i^{t+1} = (1-\varepsilon) w_i^t$

Define M_i^t = indicator for expert i making mistake at time t

M^t = indicator for LEARNER making mistake at time t

$M = \sum_{t=1}^{T} M^t$ = # of mistakes of LEARNER by time T

$$W^{t+1} = \sum_{i=1}^{n} w_i^{t+1} = \sum_{i=1}^{n} (1-\varepsilon M_i^t) w_i^t = W^t - \varepsilon \underbrace{\sum_{i=1}^{n} M_i^t \frac{w_i^t}{W^t}}_{\mathbb{E}[M^t]} W^t \le e^{-\varepsilon \mathbb{E}[M^t]} W^t$$

($\frac{w_i^t}{W^t}$ = $\mathbb{P}[i^{th}$ expert picked at time $t]$)

then $W^T \le e^{-\varepsilon \mathbb{E}[M]} W^0$

$w_i^T = (1-\varepsilon)^{m_i}$ $\Rightarrow (1-\varepsilon)^{m_i} \le e^{-\varepsilon \mathbb{E}[M]} n$

$m_i(-\varepsilon-\varepsilon^2) \le m_i \ln(1-\varepsilon) \le -\varepsilon \mathbb{E}[M] + \ln n$

$$\mathbb{E}[M] \le m_i(1+\varepsilon) + \frac{\ln n}{\varepsilon}$$

↳ $\mathbb{E}[M] - m_i = m_i \varepsilon + \frac{\ln n}{\varepsilon}$

$\lim_{T\to\infty} \frac{1}{T}\left(m_i \varepsilon + \frac{\ln n}{\varepsilon} \right) \le \varepsilon$

↑ can control to get vanishing regret!!

Multiplicative Weights Update

Same as RMW but now $M_i^t \in [-1, 1]$ continuous

cost

M = total cost of LEARNER

At time step t

- pick expert i based on distribution $(w_1^t, w_2^t, \dots, w_n^t)$

for each expert i

$$w_i^{t+1} = (1-\varepsilon M_i^t) w_i^t$$

$$\mathbb{E}[M] \le m_i(1+\varepsilon) + \frac{\ln n}{\varepsilon}$$

$\forall\, i \in \{1, 2, \dots, n\}$

where $m_i = \sum_{t=1}^{T} M_i^t$

= total cost of i^{th} expert

LECTURE 18 4/23/24 (Did not attend)

Casino: start w/ \$20, ± \$5 each time w/ ½ chance

Def A **Stochastic process** is a sequence of random variables $\{X_t : t \in \mathbb{N}\}$ indexed by time

Process is a **Markov Chain** / has **Markov Property** if X_{t+1} only depends on X_t $\forall\, t \in \mathbb{N}$

* Let $\mathbb{D}$ be our state space

ex) $D = \{\$0, \$5, \ldots, \$40\}$

$X_t \in D$ = amount of $ you have after t rounds of play

Graphical Representation

$\forall t \in \mathbb{N}$, $G_t = (D, E_t, w_t)$ with $E_t = \{(u,v) \in D^2 \mid \Pr[X_{t+1} = v \mid X_t = u] > 0\}$

$w_t(u,v) = \Pr[X_{t+1} = v \mid X_t = u] \quad \forall u,v \in D$

time-homogeneous if G_t independent of t.

Walk Matrix Representation

$W \in \mathbb{R}_{\geq 0}^{D \times D}$ $\quad W_{uv} = \Pr[X_{t+1} = v \mid X_t = u]$

need W + initial distribution

$\vec{x}^{(0)} \in [0,1]^{|D|}$ (row vector)

s.t. $\sum_{i=1}^{|D|} x_i^{(0)} = 1$

Given a weighted directed graph we can also build a Markov chain via random walk

$$W_{uv} = \frac{c(u,v)}{\sum_{x \in D} c(u,x)}$$

row sums to 1

$\vec{x}^{(1)} = \vec{x}^{(0)} W$, $\vec{x}^{(t)} = \vec{x}^{(0)} W$

stationary distribution - $\vec{x} W = \vec{x}$ (usually can just solve!)

DEFINITIONS

strongly connected = irreducible

SCC = communicating class

period(v) = gcd(len(all directed cycles in its SCC))

v is periodic if period(v) > 1, else aperiodic

an SCC is periodic if its vertices are periodic, else aperiodic

a graph is periodic if some SCC is periodic, else aperiodic

in undirected graphs, note that v is periodic iff its SCC is bipartite

an SCC is periodic if it is bipartite, else aperiodic

a graph is periodic if some SCC is bipartite, else aperiodic

Fundamental Theorem of Markov Chains

Let G_x be the Markov chain for a time-homogeneous Markov process.

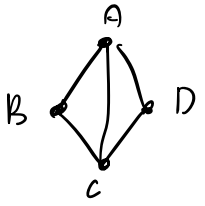
There is always a stationary distribution for G_x (for finite D).

→ if G_x is strongly connected, it is unique

→ if G_x is aperiodic, it converges to some stationary distribution

intuition. only consider recurrent (no outgoing edges) SCCs since transient (yes outgoing edges) SCCs can lose mass.

Remark If G_x is undirected/unweighted before normalization, Stationary distribution is proportional to degree of each vertex
(↑ connected, ↖ not bipartite)



$\Rightarrow \left[\frac{3}{10}, \frac{1}{5}, \frac{3}{10}, \frac{1}{5}\right]$

α-lazy random walk → do nothing with probability α, normal with probability $1-\alpha$.

↳ creates aperiodicity.

LECTURE 19 4/25/24 (Did not attend)

Sampling Problem How do we draw samples from a probability distribution π?

e.g. if we want to estimate expectation

example card shuffling - riffle takes ~7 shuffles for uniform.

Idea Design walk matrix W s.t. π is its stationary distribution.

Alg: Sample/pick X_0 arbitrary.

Then for $t=1, \dots, T$

sample $X_t \sim W(X_{t-1}, \cdot)$.

then

$X_T \sim \vec{x}^{(0)} W^T \approx \pi$

for big T.

π is usually given to us as a bunch of weights $w(x)$ for $x \in D$ s.t. $\pi(x) = \frac{w(x)}{\sum_{y \in D} w(y)}$, but we can't add up since $|D|$ is HUGE.

Def W is **reversible** if it satisfies detailed balance equations

$\exists \pi$ s.t. $\pi(u) W_{uv} = \pi(v) W_{vu} \quad \forall u, v \in \Omega$

i.e. for $\{X_t : t \in \mathbb{N}\}$, assuming $X_0 \sim \pi$, $(X_s, \dots, X_T)$ has same distribution as $(X_T, \dots, X_s)$

* For a symmetric walk matrix, we can have a uniform stationary distribution.

$$\pi(y) = \pi(y) \sum_{x \in \Omega} W_{yx} = \sum_{x \in \Omega} \pi(y) W_{xy} = \sum_{x \in \Omega} \pi(x) W_{xy}$$ as desired.

Motivating Problem - Sampling a K-coloring of G efficiently given that $K >$ maximum degree $+ 1$ (extra flexibility) AND given a valid k-coloring

idea → pick a random vertex, change its color to one of the K colors so that new coloring is still valid.

→ repeat T times

↳ Markov process, states are valid colorings.

① self-loops since we can pick same color

② strongly-connected.

Pf take two colorings $\vec{c}$, $\vec{c}'$.

First vertex: if $c_1 = c_1'$, ok.

Else, set v to an unused color (exists since $K >$ max $+1$)

then color all neighbors of v in $\vec{c}$ that had color c_1' to a different color

Then set v to c_1'. Induct.

↳ we never need to worry about changing earlier vertices because $\vec{c}'$ is valid, i.e. desired coloring has no same-color edges. □

③ uniform distribution is stationary

Pf Note $W_{\vec{c}', \vec{c}} = W_{\vec{c}, \vec{c}'} \quad \forall \vec{c}, \vec{c}' \in \Omega$ → symmetric □

⇒ our idea, if run long enough, gives us a uniformly random coloring!

Metropolis-Hastings

estimate $\nu : D \to \mathbb{R}$ given $\bar{\nu} = K \cdot \nu$ for large (nonenumerable) K

idea pick a **known** Markov chain $g : D^2 \to \mathbb{R}$

sample based on g but accept/reject using info in $\bar{\nu}$

ex) $\nu(\vec{C}) = \begin{cases} \frac{1}{N} & \text{if } \vec{C} \text{ valid} \\ 0 & \text{otherwise} \end{cases}$ $\bar{\nu}(\vec{C}) = \begin{cases} 1 & \text{if } \vec{C} \text{ valid} \\ 0 & \text{otherwise} \end{cases}$

$K = N = \#$ valid colorings

$g(\vec{C}', \vec{C})$ is nonzero iff $\vec{C}'$ differs from $\vec{C}$ by one vertex or fewer in color.

Formally we design a walk matrix W

$$W_{xy} = g(y|x) \cdot p_{acc}(y,x) \qquad p_{acc}(y,x) = \min\left\{1, \frac{\bar{\nu}(y)}{\bar{\nu}(x)} \cdot \frac{g(x|y)}{g(y|x)}\right\}$$

$g(y|x) = \mathbb{P}(X_{t+1} = y \mid X_t = x)$ $\qquad \frac{\nu(y)}{\nu(x)}$

$$W_{xx} = 1 - \sum_{y \neq x} W_{xy} = \Pr[\text{no move / reject}]$$

→ strongly connected **and** aperiodic if

$g : D^2 \to \mathbb{R}^{>0}$ and $\bar{\nu} : D \to \mathbb{R}^{>0}$

Claim W has stationary distribution ν.

Pf We prove W is reversible: $\nu(x) W_{xy} = \nu(y) W_{yx}$.

So we need to show

$$\nu(x)\, g(y|x) \min\left\{1, \frac{\nu(y)}{\nu(x)} \cdot \frac{g(x|y)}{g(y|x)}\right\} = \nu(y)\, g(x|y) \min\left\{1, \frac{\nu(x)}{\nu(y)} \cdot \frac{g(y|x)}{g(x|y)}\right\}$$

casework on this vs. this shows this indeed. □

Rate of Convergence

$\{X_t : t \in \mathbb{N}\}$ $X_0 \sim \vec{x}^{(0)}$. want to understand how fast $X_t \sim \vec{x}^{(0)} W^t \to \pi$

Now imagine another Markov process $\{Y_t : t \in \mathbb{N}\}$ s.t. $Y_0 \sim \pi$

and $\Pr[Y_{t+1} = v \mid Y_t = u] = W_{uv}$ $\forall u, v, t$.

Coupling: Correlate X_t and Y_t s.t. $\mathbb{E}[T]$ is minimized.

Here, $T = \min\{t : X_t = Y_t\}$ (random walk collides with stationary distribution)

↑ mixing time

↳ after $O(\mathbb{E}[T])$ steps the distributions are "close"

LECTURE 20 4/30/24 (Did not attend)

multiplying polynomials → $p(x) = a_0 + \cdots + a_{n-1}x^{n-1}$

$q(x) = b_0 + \cdots + b_{n-1}x^{n-1}$

takes $\Theta(n^2)$ to do naively.

Assume we can multiply real/complex numbers precisely in $O(1)$ time.

↳ More on this later.

coefficient representation

we can also describe polynomials of degree $n-1$ as a set of n values at n inputs (then interpolate)

then $p(x_i)q(x_i)$ for $0 \le i \le n-1$ is easy to compute.

→ but is this useful? how to convert ⇔ coefficient representation?

if we carefully choose x_i, turns out that we can do this all in $O(n \log n)$.

Fast Fourier Transform

- can take $\Theta(n)$ to evaluate degree-n polynomial at x_i, but we can choose related x_i to do less work

observe that (for n even)

$$p_{even}(x) := a_0 + a_2 x + \cdots + a_{n-2} x^{\frac{n}{2}-1}$$
$$p_{odd}(x) := a_1 + a_3 x + \cdots + a_{n-1} x^{\frac{n}{2}-1}$$

$$p(x) = a_0 + a_2 x^2 + \cdots + a_{n-2} x^{n-2} + a_1 x + a_3 x^3 + \cdots + a_{n-1} x^{n-1}$$
$$= p_{even}(x^2) + x\, p_{odd}(x^2)$$

↳ $p(-x) = \underbrace{p_{even}(x^2)}_{\text{even}} - x \underbrace{p_{odd}(x^2)}_{\text{even}}$

- define size of degree-$(d-1)$ polynomial as d = # coefficients
- let $S^k = \{x^k \mid x \in S\}$ for sets S.

idea

- Compute $\{p_{even}(x^2) \mid x \in S\} \Leftrightarrow \{p_{even}(y) \mid y \in S^2\}$ $\quad T(\frac{n}{2})$
- Compute $\{p_{odd}(x^2) \mid x \in S\} \Leftrightarrow \{p_{odd}(y) \mid y \in S^2\}$ $\quad T(\frac{n}{2})$
- Combine $p(x) = p_{even}(x^2) + x\, p_{odd}(x^2)$ for $x \in S$ $\quad O(n)$

$T(\text{size } n \text{ poly @ } n \text{ pts}) = 2\,T(\text{size } \frac{n}{2} \text{ poly @ } \frac{n}{2} \text{ pts}) + O(n) \rightarrow O(n \log n)$.

for this we need $|S^{2n}| = \frac{|S^n|}{2}$ ↗ to reduce computations.

↳ use roots of unity! (first pad n s.t. $n = 2^k$)

then $S = \{\omega \mid \omega^n = 1\}$ works

so we can now apply...

Discrete Fourier Transform (coefficient → point-value)

start with coefficients $(a_0, \ldots, a_{n-1})$ → values at $(1, \omega, \omega^2, \ldots, \omega^{n-1})$

↳ $(y_0, y_1, \ldots, y_{n-1})$

$$y_k = \sum_{i=0}^{n-1} a_i \omega^{ik}$$

FFT $((a_0, \dots, a_{n-1}), n)$: (size $n = 2^k$)

if $n = 1$ → return a_0

else:

values at $1, \omega^2, \dots, \omega^{n-2}$ → $y_{even} \leftarrow FFT((a_0, a_2, \dots, a_{n-2}), \frac{n}{2})$

$y_{odd} \leftarrow FFT((a_1, a_3, \dots, a_{n-1}), \frac{n}{2})$

for $j \in \{0, 1, \dots, \frac{n}{2} - 1\}$:

$y[j] \leftarrow y_{even}[j] + \omega^j y_{odd}[j]$

$y[j + \frac{n}{2}] \leftarrow y_{even}[j] - \omega^j y_{odd}[j]$

return y

But how do we go backward, i.e. from $(y_0, \dots, y_{n-1}) \to (a_0, a_1, \dots, a_{n-1})$?

- $O(n^3) \sim O(n^{2.373})$ by solving system of equations on $p(a_i) = y_i$
- $O(n^3) \sim O(n^2)$ using Lagrange Interpolation.

LECTURE 21 5/2/24 (Did not attend)

note that

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

$\vec{y}$ W $\vec{a}$

And that $\frac{1}{\sqrt{n}} W$ is unitary. So $\vec{a} = \frac{1}{n} W^\dagger \vec{y} = \overline{\frac{1}{n} W \vec{y}} = \overline{W \frac{\vec{y}}{n}}$.

↳ apply FFT on $\frac{\vec{y}}{n}$ then take conjugate.

Convolution: ignore polynomials, just think of sequences $(a_0, a_1, \dots, a_m)$ $(b_0, b_1, \dots, b_n)$

$(c_0, \dots, c_{m+n}) := c_k = \sum_{i+j=k} a_i b_j$

↳ solvable in $O((m+n)\log(m+n))$.

Minkowski Sum: given $X, Y \subseteq \{0, 1, \dots, m-1\}$, compute $X+Y = \{x+y \mid x \in X, y \in Y\}$.

↳ naively takes $O(|X||Y|)$. → ok if $m \gg n$.

↳ let $(x_0, \dots, x_{m-1})$ s.t. $x_i = 1$ if $i \in X$, else 0. same for $(y_0, \dots, y_{m-1})$.

→ convolve to get $(c_0, \dots, c_{2m-2})$, then $i \in X+Y$ iff $c_i > 0$. ⇒ $O(m \log m)$

String Matching in $\{0,1\}$ for now. Let $m \le n$.

search string $S = (s_0, s_1, \dots, s_{n-1})$, target string $P = (p_0, \dots, p_{m-1})$

want to find $M := \{k \mid s_{k+i} = p_i \ \forall i \in [m]\}$

↳ naively takes $O(mn)$ → quadratic if $m = \frac{n}{2}$

⇒ construct $p^{rev'} = (p'_{m-1}, p'_{m-2}, \dots, p'_1, p'_0)$ where $s'_i = \begin{cases} 1 & s_i = 1 \\ -1 & s_i = 0 \end{cases}$ $p'_i = \begin{cases} 1 & p_i = 1 \\ -1 & p_i = 0 \end{cases}$

convolve $p^{rev'} * S'$ to get $L = (l_0, \dots, l_{m+n-2})$

↳ $l_{i+m-1} = m$ iff $i \in M$

Proof

$$l_{i+m-1} = \sum_{j+k=i+m-1} p'_{m-1-k} s'_j = \sum_{j-i=x} p'_x s'_j = \sum_{j-i=x} \begin{cases} 1 & p_x = s_j \\ -1 & p_x \neq s_j \end{cases} = m - 2(\text{mismatches})$$

let $m-k-1 = x$

⇒ can also easily check how many substrings have $\ge m-k$ characters in common: $l_{i+m-1} \ge m-2k$

LECTURE 22 5/7/24 (did not attend)

Sublinear Algorithms - read a fraction of input, hope to get an approximately correct answer

(1) classical approximation
→ answer within α of correct one → optimization, not useful for decision problems.

(2) property testing

YES if it is or is close to a YES-instance
NO if it is far from a YES-instance (and sometimes if close)

(1) **diameter of point set** - given $N = n \times n$ distance matrix D

↳ $D_{ij} = D_{ji}$ ↳ $D_{ij} \le D_{ik} + D_{kj}$ $\forall i,j,k$,

estimate $d^* = \max_{i,j} D_{ij}$

Algorithm
→ pick 1 point p, return $\max_i D_{ip}$ → $O(n)$

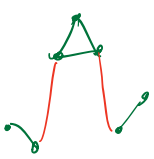
Pf if $D_{i^*j^*}$ is best, then
$2 \max(D_{i^*p}, D_{j^*p}) \ge D_{i^*p} + D_{j^*p} \ge D_{i^*j^*}$ → 2-approx.

can't do better,
e.g. D = all 1's vs.
D' = all 1's except $D'_{ij} = D'_{ji} = 2$ for some i,j
need $\Omega(n^2)$ to find i,j

2a graph connectedness (use adjacency list representation)

$\varepsilon > 0$ small number.

Def Graph G with n vertices, max degree d is **ε-close to connected** if adding $< \varepsilon \cdot d \cdot n$ edges will connect it

ex) 

$n = 7$
$d = 2$
$\varepsilon < 7$

gives ε-close to connected.

if $\varepsilon \leq \frac{1}{dn}$, $\varepsilon dn \leq 1$ so G is already connected! So let $\varepsilon >> \frac{1}{dn}$.

Algorithm (d, n, G, ε given)

Repeat $\frac{c}{\varepsilon d}$ times:

pick a random point, DFS from it.

if $< \frac{2}{\varepsilon d}$ vertices in connected component, output NO.

if $\geq \frac{2}{\varepsilon d}$ vertices, continue.

output YES.

$\varepsilon >> \frac{1}{dn} \Rightarrow \frac{1}{\varepsilon d} << n$

Runtime is $\frac{c}{\varepsilon d} \cdot \frac{2}{\varepsilon d} \cdot d = O\left(\frac{1}{\varepsilon^2 d}\right) = O\left(\frac{1}{\varepsilon}\right) \cdot O\left(\frac{1}{\varepsilon d}\right)$ ($\frac{1}{\varepsilon}$: constant) $\Rightarrow$ sublinear.

→ obviously YES if connected.

→ if NOT ε-close to connected, at least $\varepsilon \cdot d \cdot n$ edges needed to connect G

↳ $\geq \varepsilon \cdot d \cdot n$ CC's.

↳ $\geq \frac{\varepsilon \cdot d \cdot n}{2}$ CC's with $< \frac{2}{\varepsilon d}$ vertices (else too many vertices)

↳ $\geq \varepsilon \cdot \frac{d \cdot n}{2}$ vertices in CC's of $< \frac{2}{\varepsilon d}$ vertices ← each CC has ≥ 1 vertex

So $\Pr[\text{NO}] = 1 - (\text{never find "small" CC}) \geq 1 - \left(1 - \frac{\varepsilon d}{2}\right)^{\frac{c}{\varepsilon d}}$

$\geq 1 - \left(\left(1 - \frac{1}{2/\varepsilon d}\right)^{\frac{2}{\varepsilon d}}\right)^{\frac{c}{2}}$

$\geq 1 - e^{-\frac{c}{2}}$

$\geq \frac{3}{4}$

for large enough c.

(if ε-close to connected but not connected, we don't care)

(25) list sortedness

Def A list of length n is ε-close to sorted if $\leq \varepsilon n$ elements can be removed to sort the list

Idea 1

Random x_i, x_{i+1} check if sorted.

↳ NO, ex) $1, 3, 5, \eta, 9, \ldots, \frac{n}{2}, 2, 4, 6, \ldots$

only one pair tells us NO.

$Pr \approx \frac{1}{n}$ of discovery

Idea 2

Random $i<j$, check if $x_i < x_j$.

↳ NO ex) 2 1 4 3 6 5 8 7 ...

again only $\frac{n}{2}$ pairs tell us NO.

$Pr \approx \frac{1}{n}$ of discovery

Algorithm

Repeat $\frac{c}{\varepsilon}$ times.

pick random i

binary search for x_i in L as if it was sorted

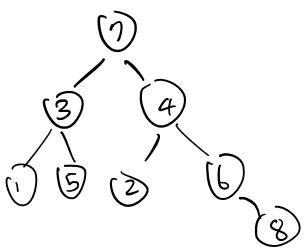
if you find inconsistency or don't end up in location i, output NO

output YES.

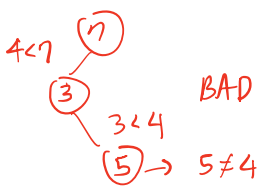
"bad" indices

Runtime is $\frac{c}{\varepsilon} \cdot O(\log n) = O\left(\frac{\log n}{\varepsilon}\right) << n$ for large enough n ε.

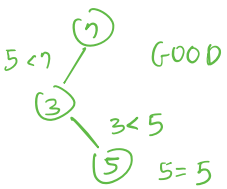
bad index examples ex) $L =$ 1 3 5 7 2 4 6 8



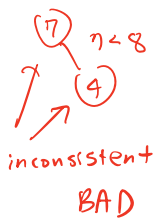
1) $i=5$, $x_i = 4$



2) $i=2$, $x_i = 5$



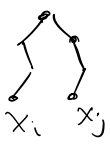
3) $i=7$, $x_i = 8$



Claim If $i<j$ and i, j are both GOOD, then $x_i < x_j$.

Pf Consider just the paths from root to x_i and x_j.

these two paths must be consistent so $x_i < x_j$.



So removing all BAD nodes will leave us with a sorted list!

"$\leq \varepsilon n$ BAD indices ⇒ list is ε-close to sorted"

⇕

"list isn't ε-close to sorted ⇒ $> \varepsilon n$ BAD indices"

Algorithm Analysis:

→ obviously YES if sorted.

→ if not ε-close to sorted ⇒ $> \varepsilon n$ BAD indices.

$\Pr[\text{NO}] = 1 - (\text{we never find BAD index}) \geq 1 - (1-\varepsilon)^{\frac{c}{\varepsilon}} \geq 1 - e^{-c} \geq \frac{3}{4}$ for $c \geq 2$.

LECTURE 23 5/9/24 (Did not attend)

Sketching!

Counting - string of n 1's, fed 1 at a time

- want to find # of 1's (i.e. n)

↳ OPTION 1: just count → $O(\log n)$ memory

optimal space for exact answer.

but can we estimate n in $o(\log n)$ space?? (excluding output?)

Estimation of θ with $\tilde{\theta}$ (ε, δ)-estimate → close to 0 as possible

→ want $\tilde{\theta} \in [(1-\varepsilon)\theta, (1+\varepsilon)\theta]$ with $\Pr \geq 1-\delta$.

Morris's Algorithm ~$O(\log \log n)$ space

"count fewer as you go further along"

$X = 1^n = n$ 1's $\quad f(X) = n$ ← function to be computed $\quad C(X)$ ← sketch.

→ initialize $C(X) = 0$

→ on input of 1, $C(X) \mathrel{+}= 1$ with $\Pr = \frac{1}{2^{C(X)}}$

→ output $\tilde{f}(X) = 2^{C(X)} - 1$.

{ toss fair coin $C(X)$ times and ONLY if all heads

only $O(C(X))$ space.

induct on $n = k$.

$\mathbb{E}[\tilde{f}(X)] = \mathbb{E}[2^{C(X)}] - 1$

($X = 1^k$)

$= \sum_{j=0}^{k-1} \Pr[C(1^{k-1}) = j] \cdot \mathbb{E}[2^{C(1^k)} \mid C(1^{k-1}) = j] - 1$

$$= \sum_{j=0}^{k-1} \Pr[C(1^{k-1}) = j] \cdot \left(2^{j+1} \cdot \frac{1}{2^j} + 2^j \cdot \left(1 - \frac{1}{2^j}\right)\right) - 1$$

$$= \sum_{j=0}^{k-1} \Pr[C(1^{k-1}) = j] \cdot \left(2^j + 1\right) - 1$$

$$= \sum_{j=0}^{k-1} \Pr[C(1^{k-1}) = j] \cdot 2^j + 1 - 1$$

$$= (k-1) + 1 = k$$

induct on $n = k$ to get $\mathbb{E}\left[2^{2C(x)}\right] = \frac{3}{2}k^2 + \frac{3}{2}k + 1$

$$\Rightarrow \mathbb{V}[\tilde{f}(X)] = \mathbb{E}\left[(2^{C(x)} - 1)^2\right] - \mathbb{E}\left[2^{C(x)} - 1\right]^2$$

$$= \frac{3}{2}k^2 + \frac{3}{2}k + 1 - 2(k+1) + 1 - k^2$$

$$< \frac{1}{2}k^2 \quad \Rightarrow \mathbb{E}[\tilde{f}] = n, \ \mathbb{V}[\tilde{f}] < \frac{n^2}{2}.$$

Chebyshev. $\hookrightarrow \mathbb{P}\left[|\hat{f} - n| \geq \varepsilon n\right] \leq \frac{\frac{n^2}{2}}{(\varepsilon n)^2} \leq \frac{1}{2\varepsilon^2}$

$\mathbb{P}[\text{in interval}] \geq 1 - \frac{1}{2\varepsilon^2}$ $\quad \delta = \frac{1}{2\varepsilon^2}$

Can we do better? (For general algorithm A with $\mathbb{E} = \Theta$, $\mathbb{V} = \sigma^2$, space $= S$)

→ run several times and average (in parallel)

$A^m(\varepsilon, \delta)$: let $R = \left\lceil \frac{\sigma^2}{\varepsilon^2 \Theta^2 \delta} \right\rceil$

run A R times and take average $\tilde{\Theta} = \frac{\sum_{i=1}^{R} \tilde{\Theta}_i}{R}$.

we get $\mathbb{E}[\tilde{\Theta}] = \Theta$ $\quad \mathbb{V}[\tilde{\Theta}] = \frac{1}{R^2} \sum_{i=1}^{R} \sigma^2 = \frac{\sigma^2}{R} \leq \varepsilon^2 \Theta^2 \delta$

so $\Pr\left[|\tilde{\theta} - \theta| \geq \varepsilon\Theta\right] \leq \frac{\varepsilon^2 \Theta^2 \delta}{\varepsilon^2 \Theta^2} = \delta$.

but this takes $O(RS) = O\left(\frac{\sigma^2}{\varepsilon^2 \Theta^2 \delta} \cdot S\right)$ space. ← $O\left(\frac{1}{\varepsilon^2 \delta} \cdot \log\log n\right)$ for Morris.

⇒ MEDIAN of means of estimates.

A^{mom} : let $R = \lceil 36 \ln\left(\frac{1}{\delta}\right) \rceil$

run $A^m(\varepsilon, \frac{1}{3})$ R times and take their median.

Claim If $> \frac{1}{2}$ of A^m's are in $[(1\pm\varepsilon)\theta]$ then so is our result.

$\Pr[\geq \frac{1}{2} \text{ of } A^m\text{'s are out of range}] = \Pr[\text{fails} \geq (1+\frac{1}{2})\mathbb{E}[\text{fails}]]$

"fails" $\leq e^{-\frac{1}{4}\cdot\frac{1}{3}R\cdot\frac{1}{3}} \leq e^{-\ln(\frac{1}{\delta})} = \delta.$

$\mathbb{E}[\text{fails}] = \frac{1}{3}R \Rightarrow \Pr[\text{overall success}] \geq 1-\delta.$

overall space is now

$$O\left(S \cdot \frac{3\sigma^2}{\varepsilon^2\theta^2} \cdot \ln\left(\frac{1}{\delta}\right)\right) = O\left(\frac{\sigma^2 S}{\varepsilon^2\theta^2}\ln\left(\frac{1}{\delta}\right)\right) \text{ for } A^{mom}.$$

(S: original; $\frac{3\sigma^2}{\varepsilon^2\theta^2}$: means; $\ln(\frac{1}{\delta})$: median)

A^{mom}: $O\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\log\log n\right)$

we actually get $\log\log\left(\frac{n}{\varepsilon\delta}\right)$ by showing no counter exceeds $O\left(\log\frac{n}{\varepsilon\delta}\right)$ with $\Pr > 1-\delta$.

$N :=$ total # of counters in our algorithm

$$= \left\lceil\frac{\sigma^2}{\varepsilon^2\theta^2\cdot\frac{1}{3}}\right\rceil \cdot \left\lceil 36\ln\left(\frac{1}{\delta}\right)\right\rceil = O\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right)$$

↳ if any counter exceeds $\log\left(\frac{Nn}{\delta}\right)$

then $\Pr[\text{increase}] \leq \frac{\delta}{Nn}$ ← over n time steps and N counters

$\Pr[\text{any counter increases beyond } \log\left(\frac{Nn}{\delta}\right)] \leq \delta$

$$\log\left(\frac{Nn}{\delta}\right) = O\left(\log\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\cdot n\cdot\frac{1}{\delta}\right)\right) = O\left(\log\left(\frac{n}{\varepsilon\delta}\right)\right)$$

↳ overall $O\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\log\log\left(\frac{n}{\delta\varepsilon}\right)\right)$

$O(\log\log(\frac{n}{\delta\varepsilon}))$ space

time complexity.

→ each $A^m(\varepsilon, \frac{1}{3})$ takes $O\left(\log\left(\frac{n}{\varepsilon\delta}\right)\cdot n\right)$

($\log(\frac{n}{\varepsilon\delta})$: max counter value ($\to$ max flips of coin per step); n: numbers to read

⇒ run all R in parallel

→ median of $O(\log(\frac{1}{\delta}))$ items double in $O(\log(\frac{1}{\delta}))$

→ overall $O\left(n\log\left(\frac{n}{\varepsilon\delta}\right) + \log\left(\frac{1}{\delta}\right)\right)$

Distinct Elements

count # of distinct elements in a stream.

FM83 Algorithm - Suppose we have a truly random $h: \mathcal{U} \to [0,1]$

↳ initialize $C(X) = 1$

↳ after x_i, compute $h(x_i)$, update $C(X) = \min\{C(X), h(x_i)\}$

↳ output $\tilde{f}(X) = \frac{1}{C(X)} - 1$.

$$\mathbb{E}[C(X)] = \frac{1}{d+1} \quad \text{based on } d \text{ uniform values } \in [0,1]$$

$$\mathbb{E}[C(X)^2] = \int_0^1 \Pr[C(X)^2 \geq y]\, dy = \int_0^1 \Pr[C(X)^2 \geq \sqrt{y}]\, dy$$

$$= \int_0^1 (1-\sqrt{y})^d\, dy = \int_0^1 2x(1-x)^d\, dx$$

$y = x^2$

$dy = 2x\,dx$

$$= \int_0^1 2(1-x)x^d\, dy = \frac{2}{d+1} - \frac{2}{d+2} = \frac{2}{(d+1)(d+2)}$$

$$\mathbb{V}[C(X)] = \frac{2}{(d+1)(d+2)} - \frac{1}{(d+1)(d+1)} = \frac{d}{(d+1)(d+2)}$$

but we don't have a truly random h at our disposal!:

↳ t-wise independent hashing: $\forall$ distinct $k_1, \ldots, k_t \in \mathcal{U}$

not-necessarily distinct $i_1, \ldots, i_t \in \mathcal{M}$

$$\Pr_{h \in_R \mathcal{H}}\left[\bigwedge_{j=1}^{t} h(k_j) = i_j\right] = \frac{1}{|\mathcal{M}|^t}$$

can achieve with $\mathcal{H} = \{h_a : a \in \mathcal{M}^t\}$, $h_a(k) = \sum_{\ell=0}^{t-1} a_\ell k^\ell \pmod m$

$h_a(k)$: time, space both $O(t \log m)$, $m = |\mathcal{M}|$, to store a.

KMV Algorithm

↳ pick random $h: \mathcal{U} \to M = \{0, 1, \ldots, u^3\}$, $|\mathcal{U}| = u$, from 2-wise independent family

↳ $k = \lceil \frac{24}{\varepsilon^2} \rceil$, initialize $C(X) = \emptyset$, $f(X) = d$, input stream $(x_1, \ldots, x_n)$

↳ at each step let $C(X) = \min_k \{C(X) \cup \{h(x_i)\}\}$

$$\text{output } \tilde{f}(X) = \begin{cases} |C(X)| & \text{if } |C(X)| < k \\ \frac{k u^3}{\max\{C(X)\}} & \text{otherwise} \end{cases}$$

old $C(X) \approx \frac{\max\{C(X)\}}{k u^3}$ ← protect from high variance

↑ k^{th} smallest ↑ scale to $[0,1]$

$\Pr[\tilde{f} \in [(1 \pm \varepsilon) d]] \geq \frac{2}{3}$

↳ any $1 - \delta$ if using mom.

time = n evaluations of h + n updates of $C(X)$

$= n \log u + n \log k = O(n \log uk) = O\left(n \log \frac{u}{\varepsilon^2}\right)$

space = storing h + storing $C(X)$

$= \log u + k \log u = O(k \log u) = O\left(\frac{\log u}{\varepsilon^2}\right)$

Similarity Search – n sets $A_1, A_2, \ldots, A_n \subseteq \mathcal{U}$, assume $|A_i| \leq d$

Given $A \subseteq \mathcal{U}$, detect if $\exists A_i$ s.t. $J(A, A_i) = \frac{|A \cap A_i|}{|A \cup A_i|} \geq s$.

Linear Scan takes $\Omega(nd)$

Fast Approximate Linear Scan

Modify so that

→ if $J(A, A_i) \geq s$ for some A_i, YES
→ if $J(A, A_i) < s'$ for all A_i, NO

} both satisfied with $> \frac{1}{2}$ chance.

use randomness. t hash functions $h_1, h_2, \ldots, h_t \to [0,1]$

let $\sigma_h(A_i) = (\sigma_{h_1}(A_i), \ldots, \sigma_{h_t}(A_i))$, $\sigma_{h_j}(A_i) = \min_{a \in A_i} h_j(a)$

note $Pr[\sigma_h(X) = \sigma_h(Y)] = J(X,Y)$ for random $h \to [0,1]$

estimate $Sim(\sigma_h(A), \sigma_h(A_i)) = \frac{|\{j : \sigma_{h_j}(A) = \sigma_{h_j}(A_i)\}|}{t}$

YES if $Sim_i \geq s - \varepsilon$, NO otherwise.

if $t = \frac{c}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)$ $\Pr_h[|Sim(\sigma_h(X), \sigma_h(Y)) - J(X,Y)| > \varepsilon] \leq \delta$

using Chernoff.

$\delta = \frac{1}{2n} \to t = O\left(\frac{\log n}{\varepsilon^2}\right)$, then with probability $\geq 1 - \frac{n}{2n} = \frac{1}{2}$ (all A_i simultaneously)

we get YES if $\exists\, J(A,A_i) \geq s$ and NO if no $J(A,A_i) \geq s - 2\varepsilon$

runtime $O(ndt)$ preprocessing of $\sigma_h(A_i)$'s

$O((n+d)t)$ { $O(dt)$ computing $\sigma_h(A)$
$O(nt)$ finding how many j: $\sigma_{h_j}(A) = \sigma_{h_j}(A_i)$ for all $1 \leq j \leq t$, $1 \leq i \leq n$

Sublinear Nearest Neighbor Search

→ initialize $h_1, h_2, \ldots, h_L$ s.t.

$$\sigma_{h_\ell}(A) = (\sigma_{h_{\ell,1}}(A), \ldots, \sigma_{h_{\ell,t}}(A))$$ where $\sigma, h_{\ell,j} \to [0,1]$ are from before

→ initialize L hash tables.

→ store set A_i in hash table ℓ with key $\sigma_{h_\ell}(A)$ for all $1 \leq \ell \leq L$

for A, calculate $\sigma_{h_\ell}(A)$ then compute $J(A,A_i)$ $\forall\, i$ where $\sigma_{h_\ell}(A) = \sigma_{h_\ell}(A_i)$ in some hash table. If $\geq s'$, return YES. Else NO.

If $J(A,A_i) \geq s$ for some $1 \leq i \leq n$, then

$$Pr[\sigma_{h_\ell}(A) = \sigma_{h_\ell}(A_i)] \geq s^t \quad \text{for each } \ell.$$

→ probability of failure $\leq (1-s^t)^L$. (always NO if $\nexists A_i$; YES with $Pr \geq 1-(1-s^t)^L$ if $\exists A_i$)

Expected runtime: if all $J(A,A_i) \leq s'$

preprocessing by computing $\sigma_{h_\ell}(A_i)$ $\forall\, i, \ell$ → $O(Lndt)$

exp $O(Ld(t+ns't))$

$\Pr[\sigma_{h_\ell}(A) = \sigma_{h_\ell}(A_i)] \le s't$, so $O(L\,n\,d\,s't)$ check time

(L: total # hash tables; all i; $J(A, A_i)$ compute; probability A_i is checked)

computing key $\sigma_{h_\ell}(A)$ $\forall\ 1 \le \ell \le L$ takes $O(L\,d\,t)$

ex) $s = \frac{1}{2}$, $s' = \frac{1}{4}$ $\rightarrow$ $t = \frac{\log n}{2}$; $L = \sqrt{n}$

$$\Pr[\text{fail}] \le \left(1 - \left(\frac{1}{2}\right)^{\frac{\log n}{2}}\right)^{\sqrt{n}} = \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}} \le e^{-1} \approx 0.37.$$

expected time

$$O\left(\sqrt{n}\,d\left(\log n/2 + n \cdot \left(\frac{1}{4}\right)^{\frac{\log n}{2}}\right)\right) = O\left(\sqrt{n}\,d\log n + n\sqrt{n}\,d \cdot \frac{1}{n}\right) = O(\sqrt{n}\,d\log n)$$

sublinear!!