Algorithms and Complexity

With In-Class Exercises

Koç University COMP 305
Instructor: Deniz Yuret
Duygu Sezen Islakoglu

- interval scheduling
  - single resource multiple requests. ✓
  - each requests i has (s_i, f_i) start-end time. ✓
- NP: You can check its validity in polynomial time.
  - ↳ Hamiltonian path.
- Is NP $\stackrel{?}{=}$ P
- NP completeness: the ones where all NP's can be converted to this

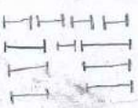
VERSION 1

- Maximize the # of requests scheduled.
- Greedy Algorithm
  1. Pick a request
  2. Eliminate the ones which cause overlapping
  3. Go to 1

CLASS EX ① Suggest 3 heuristics to pick next requests.

- a: min overlapping
- b: earliest start time
- c: " finish time
- d: min length

② Find counter examples where each fails over 1-4

a:  b:  c: we cannot d: 

INDUCTION

k^* : num of requests in optimal solutions

Prove that if k^* is optimal ⇒ #c will find it.

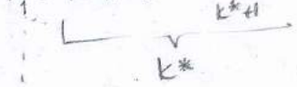
Base: If $k^* = 1$ ⇒ #c will find it.

Inductive: If true up to k^* → solve for $k^* + 1$

Assume claim is true up to k^*

Take a config whose opt. sol is (k^*+1)

Applying #c gives us a solution of length x.

Opt sol: 

#c: 

- We know $f(1') \leq f(1)$

Construct an alternative problem w/ all intervals after $f(1')$ (P') What is the opt sol for P'? k^*.

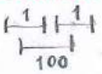
⇒ #c can find the opt sol. for P'.

3rd

VERSION 2

- Each request has weight w_i in addition to (s_i, f_i) find sol that max Σw_i

③ Find counter example for #a-d



④ Write DP algorithm

Let S be a subset of requests.

$$opt(s) = \max_{i \in S} (w_i + opt(\text{following}(i)))$$

meaning $j \in R$ that $s(j) \geq f(i)$

⑤ Complexity $O(n^2)$ → assume opt(following) is order 1. max operation = n, n subproblems

HW

① Find n log n DP for this

② What if there are two rooms?

LEC 2

DIVIDE AND CONQUER

3 Examples: Merge sort / Matmul / Strassen

3 Methods: Tree / Subst and Induction / Master Theorem

Given a problem of size n, divide into A subproblems of size n/B

$(A \geq 1, B > 1)$ Solve each recursively. Merge solutions

- Time complexity

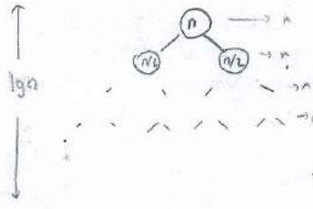
$T(n) = A \cdot T(\frac{n}{B}) + [\text{cost of merge}]$

Merge Sort

$T(n) = 2\,T(\frac{n}{2}) + n$

Tree Method

What functions of $T(n)$ solves this equation?



① Draw tree
② Write merge cost
③ How many levels
④ What is the cost of each level?

$\Rightarrow T(n) = \sum_{i=0}^{\lg n} n = \lg n \times n$

Subst Method: Guess $T(n) = O(n \lg n)$

Prove this by induction. Assume true for $m < n$

$T(m) \leq c \cdot m \lg m$ for all $m < n$ } inductive assumption

Original Exp: $T(n) \leq 2 \cdot c\frac{n}{2} \lg \frac{n}{2} + n$ (because $\frac{n}{2} < n$)

$= cn\lg n - cn + n$

$= cn\lg n - (c-1)n$ if $c \geq 1$

$\leq cn\lg n$

Master Theorem

Subproblems should have same size.

$T(n) = a\,T(\frac{n}{b}) + n^c$ $\quad a^{\log n} = n^{\log a}$ $\quad (\log_b a)\log_b(a) \neq \log_b n)\log_b(a)$

$1 - c < \log_b(a)$ → leaves cost $\longrightarrow T(n) = O(n^{\log_b a})$

Root cost

$2 - c = \log_b(a) \longrightarrow T(n) = O(n^{\log_b a} \lg n)$

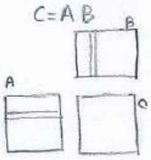
$3 - c > \log_b(a) \longrightarrow T(n) = O(n^c)$

▶ Merge sort

$a = 2 \quad b = 2 \quad c = 1 \quad \log_b a = 1$

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

Matrix Multiplication

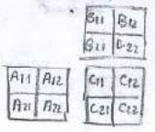


Algorithm 1 Bunch of for loops

$T(n) = n^3$ (3 n loops) (2 for A and B, 1 for dot product)

$n \times n$ matrices

D&C Method



$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

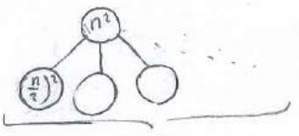
$T(n) = \overset{\text{8 pieces}}{8}\,T(\frac{n}{2}) + n^2$ → addition

CE ① Find it by methods.

- $a = 8 \quad b = 2 \quad c = 2$

$\log_b a = 3$

$T(n) = O(n^3)$

- 

level $0 = n^2$

level $1 = 8(\frac{n}{2})^2$

level $i = 8^i(\frac{n}{2^i})^2 = n^2(2^i)$

level $\lg n = n^2 2^{\lg n} = n^3$

$\sum_{i=0}^{\lg n} n^2 2^i = n^2 \sum 2^i = n^2(2^{1+\lg n} - 1)$

$= O(n^3)$

- Assume $T(m) \leq cm^3 \quad \forall m < n$.

Show $T(n) \leq cn^3$.

$T(n) = 8\,T(n/2) + n^2 \leq 8 \cdot c(\frac{n}{2})^3 + n^2$

$\leq cn^3 + n^2$

Does not work ← We do not want n^2.

Do more powerful assumption.

$T(m) \leq cm^3 - m^2 \quad \forall m < n$ → This was the problematic one.

Show $T(n) \leq cn^3 - n^2$

$T(n) = 8\,T(n/2) + n^2 \leq 8\,(c\frac{n^3}{8} - \frac{n^2}{4}) + n^2$

$= cn^3 - 2n^2 + n^2$

$= cn^3 - n^2$

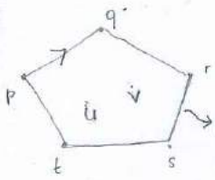
as desired

CONVEX HULL

Given n points in plane

$S = \{(x_i, y_i) \mid i = 1, \dots n\}$ Find smallest polygon containing all points : $CH(S)$

(Assume points have distinct x, y coords and no 3 on a line)



$\rightarrow CH(S) = p, q, r, s, t$

(clockwise double linked list)

3 APPROACHES

1- Take $\binom{n}{3}$ triangles

Each has an easy convex hull. How do we merge?

2- Connect any two points

If all other points are one on side of the line $\Rightarrow$ it is part of CH

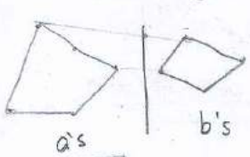
How many line segments? $\binom{n}{2} \sim O(n^2)$

How long does it take to test one? $O(n)$

$\Rightarrow$ Total cost : $O(n^3)$

Divide and Conquer Approach

3- Sort points by x coordinate. Split into 2 halves.

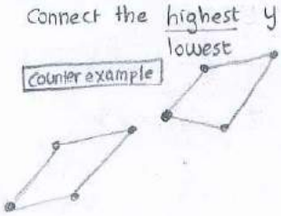


How to merge?

IDEA 1

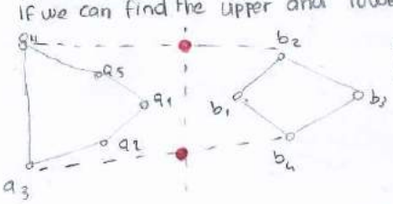
Connect the highest y coordinate a and b points.

Counter example



IDEA 2

If we can find the upper and lower tangents.



upper tangent: (a_4, b_2)

lower tangent: (a_3, b_4)

MERGE ALGORITHM

- Start with $a_i b_j$ upper tangent
- Keep going in the (clockwise) b-list until you reach lower tangent.
- Follow Lower tangent back to a-list, keep going until you reach the beginning.

$(a_1\ a_2\ a_3\ a_4\ a_5)$ $(b_1\ b_2\ b_3\ b_4)$ Clockwise sorted

Clockwise sorted

$(a_4\ b_2\ b_3\ b_4\ a_3)$

not all them is needed!

Merge idea 2

Pick points closest to the boundary (a_1, b_1)

FINGER

- Go clockwise in a until ~~the boundary intersection gets lower~~ $a_i b_j$ ~~does not cross~~ any shape
- Go counter clockwise in b until $a_i b_j$ ~~does not cross~~ any shape
- Repeat until no crossing $\Rightarrow$ found L.T. (lower tangent)

COSTS

- Sorting : $n \log n$
- UT and LT : n
- Construct solution : n (red one)

} Merge cost

$T(n) = 2\,T(\frac{n}{2}) + O(n)$

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

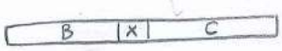
BACK TO **MEDIAN FINDING!**

Find middle number in array if it was sorted.

$T(n) = T(n/2) + O(n)$

Select (S, i) # return the i'th smallest elt in S

- Pick $x \in S$ (randomly but cleverly)
- Compute $k = \text{rank}(x)$ # number of elements $\leq x$



$k = i$ return x

$k > i$ Select (B, i)

$k < i$ Select $(C, i-k)$

Polynomials, Operations, representations, divide and conquer
complex numbers, collapsing sets

LEC 4

FFT

Polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$

OPERATIONS

① Evaluation: What is $A(x)$?

Horner's Rule: $a_0 + x(a_1 + x(a_2 + \dots + x(a_n)))$ $O(n)$

② Addition: $C(x) = A(x) + B(x)$: $c_k = a_k + b_k$ $O(n)$

③ Multiplication: $C(x) = A(x)B(x)$ $\forall x$: $C_k = \sum_{j=0}^{k} a_j \, b_{k-j}$ $O(n^2)$

for $0 \le k \le dg(A) + dg(B)$

USE DIVIDE AND CONQUER

REPRESENTATIONS

① coeff vector $\langle a_0, a_1, \dots \rangle$

② $A(x) = (x-r_0)(x-r_1)\dots(x-r_{n-1})\cdot C$ } DOES NOT WORK

Fundamental Theorem of Algebra

$C(x) = (x-r_0)(x-s_0)(x-r_1)(x-s_1)\dots c\,d$ $O(n)$

(from B)

Conversion $O(n^2)$ (2)

For n case

③ Samples $(x_0, y_0)(x_1, y_1)\dots(x_{n-1}, y_{n-1})$

with $A(x_i) = y_i$ $\forall i$ and x_i's distinct

uniquely determine $A(x)$

operations

Multiplication: multiply each y_i

Addition: add each y_i

Evaluation: requires interpolation

samples → coeff

MATRIX NOTATIONS:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & & & \\ 1 & \vdots & & & \\ 1 & x_n & x_n^2 & & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}$$

V = Vandermonde Matrix

Coeffs → Samples: matrix vector product $Y = VA$

Samples → Coeffs: $V \backslash Y = A$

→ matlab, Julia } solves

$V^{-1}Y = A$ } $O(n^3)$

OR PRECALCULATE V^{-1} ONCE, REUSE: $O(n^2)$

Can we find a special set of x's that make this cheap?

YES! Collapsing set of nth roots of 1.

DIVIDE AND CONQUER

① Divide into even and odd coefficients

$A_{even}(x) = \langle a_0, a_2, \dots \rangle$

$A_{odd}(x) = \langle a_1, a_3, \dots \rangle$

② Take $X^2 = \{x^2 \mid x \in X\}$ Call Aeven and Aodd recursively

③ Merge: $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$ $\forall x \in X$

Dividing to 2set

$T(n, |X|) = 2T(n/2, |X|) + O(n)$

So can we get $|X|$ smaller as well?

Is there a set X s.t. the set X^2 is half sized?

$X = \{-5, -4, \dots, 0, 1, \dots 5\}$ $|X^2| = 6$

→ not collapsing

DEF: X is a collapsing set if $|X^2| = \frac{|X|}{2}$

AND X^2 IS ALSO COLLAPSING

COMPLEX NUMBERS

nth roots of 1

Take 2^n of these

$|X^2| = |X|/2$

} Take $n = 4$

Best n

Worst n^2

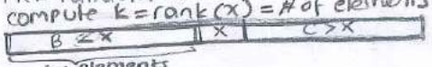
HW MEDIAN FINDING

Select (a, i) #returns the ith smallest element in a

$O(1)$

$O(n)$ Pick random $x \in a$

compute $k = \text{rank}(x) = \#$ of elements $\le x$



k-1 elements

If $i = k$ ⇒ return x

$i < k$ ⇒ Select(B, i)

$i > k$ ⇒ Select$(C, i-k)$

$T(n) = 2T(n/2) + O(n)$ → WRONG

LEC 5

S · E · L · F

summary
bit vector
$x = i\sqrt{u} + j$
high(x) low(x)

B Trees : memo hierarchy (Disk to cache)
- B ≤ #of child ≤ 2B
- B-1 ≤ # of keys ≤ 2B-1 (keys: 17 | 20)
- B: Branching Factor
- All leaves have same depth
- B depends on cache length

DELETION
- merge needed : move parent to down, merge childs

2 options : ① Having siblings that can donate to you
②

VAN EMDE BOAS

(# of presents in bit vector)

- Data struct with ints
- Goal: Maintain n elts in $\{0,1,\dots,u-1\}$ s.t all ops in $O(\lg\lg u)$
- APP: Network Routing Tables

If $u=32$, $\lg\lg u=5$
If $u=n^c$ or $u=n^{\lg^c n} \Rightarrow \lg\lg n$

- How do we get lgn?

$$T(n) = T(n/2) + O(1)$$

1 → 1 → 1 } $\lg n \Rightarrow O(\lg n)$

- How do we get lglgn?

$$T(\lg u) = T\left(\frac{\lg u}{2}\right) + O(1) \quad \& \; T(\lg u/2)$$

$$\equiv$$

$$T(u) = T(\sqrt{u}) + O(1)$$

VERSION1 Bit vector : V[x] = is x in the set

V = [0 0 1 | | 0 0 0 | | 1 0 1]

Insert	O(1)
Delete	O(1)
Successor	O(u)
Predecessor	O(u)

VERSION2 Use $\sqrt{u}$ clusters of size $\sqrt{u}$

[0 1 0 0 | 0 0 0 0 | 0 1 1 0 | 0 0 0 1]
1 0 1 1

V.cluster[0]

PROBLEM Given x which cluster and which position do I look?

- If $u=2^k$ then x is a k bit-number

x: [1 0 0 1]
2 1

x goes to V.cluster[2][1]

Define low(x) = x mod $\sqrt{u} \Rightarrow j$: position in cluster
high(x) = $\lfloor x/\sqrt{u} \rfloor \Rightarrow i$; cluster #
index(i,j) = $i\sqrt{u}+j$; x

VERSION2. INSERT O(1)

Set V.cluster[high(x)][low(x)] O(1)
Mark cluster high(x) nonempty O(1)

. SUCCESSOR $O(\sqrt{u})$ ——— $O(\sqrt{u})$

- Look w/in cluster high(x) ——— $O(\sqrt{u})$
- else find next nonempty cluster i ——— $O(\sqrt{u})$ (Baştan sona taradın diyelim)
- Find min element in that cluster ——— $O(\sqrt{u})$ (Her elemana baktın)
- return index(i,j) ——— Total = $O(\sqrt{u})$

VERSION3

Use a recursive data struct ⇒ V is vEB of size u
⇒ V.cluster[i] is vEB.size $\sqrt{u}$

Q3 **. INSERT** (V,x)

Insert (V.cluster[high(x)], low(x)) $T(\sqrt{u})$
Insert (V.nonempty, high(x)) $T(\sqrt{u})$

$T(u) = 2T(\sqrt{u}) + O(1)$

. SUCCESSOR (V,x) $O((\lg u)^{\lg 3})$ Q4

```
i = high(x)
j = successor(V.cluster[i], low(x))
if not found:
    i = successor(V.nonempty, i)
    j = successor(V.cluster[i], -∞)
return index(i,j)
```

$$T(u) = 3T(\sqrt{u}) + O(1)$$

$$a = \lg u$$

$$T'(a) = 3T'\left(\frac{a}{2}\right) + O(1) \quad a=3 \; b=2 \; c=0$$

$$\Rightarrow O((\lg u)^{\lg 3})$$

LEC 6

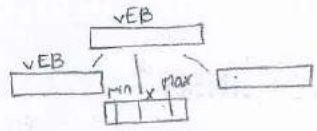
	INSERT	SUCCESSOR
V1 Bit arrays	O(1)	O(u)
V2 $\sqrt{u}$ clusters of size $\sqrt{u}$	O(1)	$O(\sqrt{u})$
V3 D&C Recursive data struct	O(lgu)	$O((lgu)lg^3)$

V4 Keep min/max of vEB structs	O(lgu)	O(lglgu)
V5 Don't recurse for min	O(lglgu)	O(lglgu)

GOAL

Store n keys from $\{0, \dots, u-1\}$ with lglgu ops.

VERSION 4 *Keep min/max*



x is here
look at this cluster
If x < max successor is in there

```
Successor (V, x)
    i = high(x)
    if low(x) < V.cluster[i].max
        j = Successor (V.cluster[i], low(x))
    else i = Successor (V.nonempty, high(x))
         j = V.cluster[i].min
    return index(i, j)
```

$T(u) = T(\sqrt{u}) + O(1)$
$= O(lglgu)$

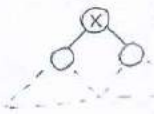
VERSION 5

```
Insert (V, x)
  if (v.min = none) V.min = V.max = x, return ⇐
  if (x < V.min) swap(x, V.min)
  if (x > V.max) V.max = x
  if (V.cluster[high(x)].min = none)
      Insert (V.nonempty, high(x)) ⟹ full recursive call
  Insert (V.cluster[high(x)], low(x)) ⟹ O(1) if empty
                                        full if nonempty
```

WATCH

B-TREES

Binary tree



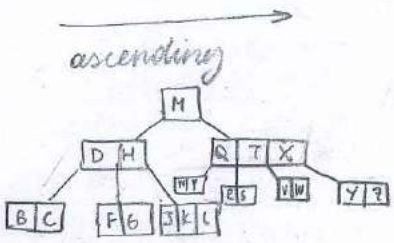
Keep tree balanced for O(lg n) ✓

x : B-tree node ✓
x.n : # of keys $\in [t-1, 2t-1]$
x.n+1 : # of children $\in [t, 2t]$
All leaves at same depth
$t \geq 2$ min degree, x.leaf is a boolean : true if leaf
x.key[1:x.n] array of keys (sorted) ✓
x.c[1:x.n+1] array of children ✓

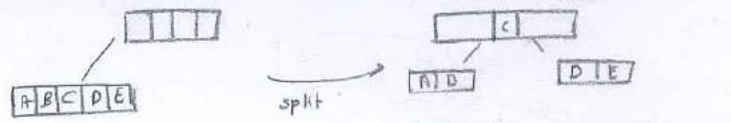
root also has ONE VALUE TWO CHILDREN

DO NOT APPLY • LOWER LIMIT to ROOT NODE

If k_i is any key in $x.c_i$
$k_1 \leq x.key_1 \leq k_2 \leq x.key_2 \leq \dots$



INSERT



IDEA proactively split full nodes going down

PROBLEM 1 : Search(x,k)
PROBLEM 2 : B-tree height

LOOK AT THE NOTES

LEC 7
LEC 8
LEC 9

AMORTIZED ANALYSIS

+Average case (like contrast:

- Worst case, overestimating real cost does not work. Why? WATCH
- Expected case: ▶ Sorting. Probabilistic, it may fail. Random oldugu için $T(n)$ diyemiyoruz. Input: worse case + algorithm random kes
- Amortized: Average cost per operation in the worst case (no probabilities)

You do not invent $\hat{c}$, you invent potential

TECHNIQUES

☑ Aggregate: Find total cost of n ops, $T(n)/n$ → Assumes that every operation has same cost.

☑ Accounting: Avg cost / op.type, credit individual objects in DS.

☑ Potential: Avg cost/op.type, credit entire DS

c_i = actual cost of op i.

$\hat{c}_i$ = amortized cost of op i.

$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \quad \forall n$ → fake costs; you can even get negative costs

1. AGGREGATE ANALYSIS

▶ Stack, push, POP
+Multipop (S, k) — stack

$c_i = \begin{cases} \min(k, s) & \text{for the multipop} \\ 1, & \text{push/pop} \end{cases}$

What is the cost of n operations?

→ Most expensive op: MPOP
→ at the end with $O(n)$ cost
→ $O(n^2)$ w.c cost for n operations → multipop all elements every time
→ $O(n)$/op w.c. cost → there are n operations

Summary: Find upper bound on total cost, divide by number of op.

Aggregate Analysis

→ # of pushes $\leq n$
→ cost of pops $\leq$ # of pushes $\leq n$.
→ $T(n \text{ operations}) = \Theta(n) \Rightarrow T(\text{1 op}) = T(n)/n = O(1)$

→ The amortized cost/op $= O(1)$

▶ Binary counter

$A[0, \dots, k-1]$ bit array

A_{k-1}	A_{k-2}	...	A_0
0	0	...	0000
0	0	...	0001

LEC7 Q1 LEC7 Q2

2. ACCOUNTING

Can assume diff amort cost $\hat{c}_i$ for diff ops.
Always keep $\sum \hat{c}_i \geq \sum c_i$

$c_i = \begin{cases} 1 & \text{push} \\ 1 & \text{pop} \\ \min(s,k) & \text{mpop} \end{cases}$

$\hat{c}_i = \begin{cases} 2 & \text{push} \\ 0 & \text{pop} \\ 0 & \text{mpop} \end{cases}$

Same question

Show $\sum \hat{c}_i \geq \sum c_i$
⇒ Amort cost

Accounting

Each time bit turns to 1, put $2 ($\hat{c}_i$)
Each time bit turns to 0, no cost.
⇒ while loop is free amortized = ∞ cost
⇒ each incr costs 2 //

$\sum \hat{c}_i \geq \sum c_i$ $O(1)$ LEC7 Q3

3. POTENTIAL METHOD

Potential function for D_i data structure (previously was the piece of data structure) after the i^{th} operation s.t.

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Only constraints: $\Phi(D_0) = 0 \quad \Phi(D_n) \geq 0$.

Multi-pop

$\Phi(D_i)$ = # of elements on stack D_i.

$\Phi(D_i) \geq 0 = \Phi(D_0)$ LEC8 Q1 LEC8 Q2 LEC8 Q3

$\Phi(D_i)$ = # of ones.

DYNAMIC TABLE / ARRAY

- Support insert
- Table starts with initial size
- When table gets full, next insert doubles table, copies elements. LEC8 Q4 LEC8 Q5 LEC8 Q6

$O(1)$ $O(n)$

Like 1 2 1 4 1 1 1 8 ... example

RANDOMIZED ALGORITHMS

make decisions based on random numbers.

☑ On the same input they may:
- Run for a different # of steps ⇒ Expected run time
- Produce different outputs ⇒ Prob of correct output

RUN TIME X
- ☐ Monte Carlo: always runs in poly time. Prob(correct) ↑
- ☐ Las Vegas: always produce correct output, expected poly run-time

MATRIX PRODUCT CHECKER

Someone claims $C = A \times B$
Check if this correct in $O(n^2)$ → Better than BRUTE FORCE $O(n^3)$

If $A \times B = C$, then Prob(YES) = 1
If $A \times B \neq C$, then $\frac{1}{2} \leq$ Prob(NO) ≤ 1 LEC9 Q1

Different seed

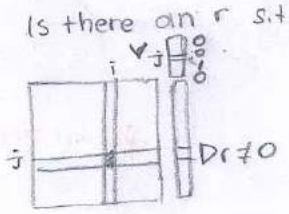
FREIVALD'S ALGORITHM

For $n \times n$ matrices, pick random binary vector $r[1 \dots n]$, $P_r(r_i = 1) = 1/2$ LEC9 Q2

If $A \cdot B \neq C$, then $P[ABr \neq Cr] \geq 1/2$

PROOF

Let $D = AB - C \Rightarrow D \neq 0$
Is there an r s.t $Dr \neq 0$?



We can have liar r.

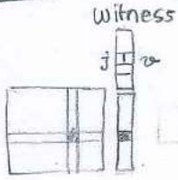
We need
$Pr(ABr \neq Cr) \geq \frac{1}{2}$
$Pr(ABr - Cr \neq 0) \geq 1/2$
$\Rightarrow Pr(Dr \neq 0) \geq \frac{1}{2}$

For a random chosen r.

LEC 10

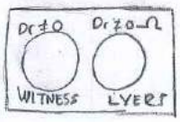
If $D \neq 0$, $Dr = \emptyset \Rightarrow r$ is lying.
$Dr \neq \emptyset \Rightarrow r$ is a witness.

PLAN Take a lying r, show there is a matching witness r 1-1 correspondence.



If r is a lying
$r' = r + v \mod 2$
$Dr' = D(r+v)$
$= Dr + Dv$

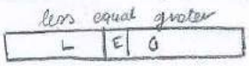
LEC 9 Q4



Random choosing
You get $\frac{1}{2} \leq P \leq 1$

LEC 3 Q5 Q6

QUICKSORT (RANDOMIZED)



Rearranging array around pivot = $O(n)$

- ☐ Pick pivot
- ☐ Arrange into L, E, G
- ☐ Call recursively on L, G

WORSTCASE

- You always pick the smallest (or largest) element as pivot $L = \emptyset$ or $G = \emptyset$

$T(n) =$ cost of L + cost of G + cost of split
$O(1) + T(n-1) + O(n)$

LEC 9 Q5 HW

$O(n^2)$

BEST CASE LEC 9 Q6

- You can find median in $O(n)$ and use as pivot.

$T(n) = T(n^2) + T(n^2) + O(n)$
$T(n) = n \log n$

FASTER

But you can use pick-pivot method

PARANOID QUICKSORT

- ☐ choose pivot at random $O(1)$
- ☐ Perform partition $O(n)$
- ☐ If partition bad (L or $G < \frac{1}{4} n$ elts) REPEAT
- ☐ Recursive call

WORSTCASE infinite → you can choose always bad partion → USELESS → Then we need to invent new analysis

Expected run time?

Good split: $L \& G \geq 1/4$

Bad split L or $G < 1/4$

Q1 Q2

$T(n) = \max_{\frac{n}{4} \leq i \leq \frac{3n}{4}} [T(i) + T(n-i)] +$ [# of iterations × $c \cdot n$] (# of iterations: random)

max when $i = \frac{n}{4}$ or $\frac{3n}{4}$

Split and arrange

Let look at the expectation

$E(T(n)) = \max_{\frac{n}{4} \leq i \leq \frac{3n}{4}} [T(i) - T(n-i)] + E[\#\text{ of iterations}]$

Q3 WATCH 09.10

MEDIAN FINDING

Back

Define more general SELECT function

SELECT(A, i) # return the ith element

- ☐ Pick pivot randomly $O(1)$
- ☐ Arrays into $\boxed{<x \mid x \mid >x}$ $O(n)$

If $|L|+1 = i \Rightarrow$ return x
If $|L|+1 > i \Rightarrow$ return SELECT(L, i)
If $|L|+1 < i \Rightarrow$ return SELECT(G, $i - |L| - 1$)

ANALYSIS OF RANDOMIZED SELECT

Define E_i: event of splitting into $|L| = i$

Expected $T(n) = n + \sum_{i=0}^{n-1} P(E_i) \max(T(i), T(n-i-1))$ ($P(E_i) = 1/2$)

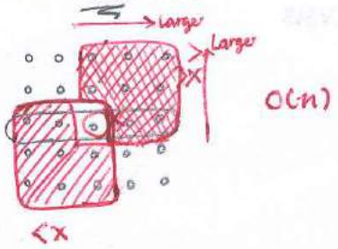
The difference from Quicksort
Recurse into L and G, not both.

Q4

Can we pick the pivot clearly? Pick-pivot

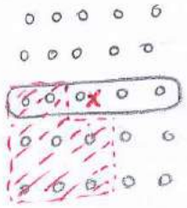
- ☐ Arrange A into columns of 5 elements
- ☐ Sort each column. Total cost in $O(n)$ ($O(1)$ but there is $\frac{n}{5}$ $O(1) = O(n)$)
- ☐ Find median of medians with recursive call.

Takes $T(n/5)$ time



Q5 Show pick-pivot is $O(n)$.

Rewatch



LEC 11

Analysis Methods

I Worst case
II Amortized
III Expected
IV W.H.P
With high probability

DEF Something true (e.g. cost=$o(n)$) WHP.
means $Pr[\text{cost} \div o(n)] \to 1$ as $n \to \infty$

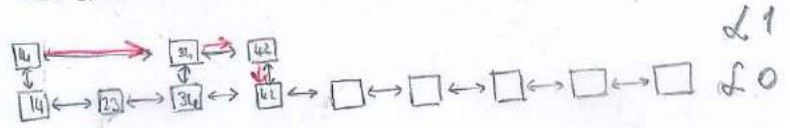
Today: Analyze skip lists with this approach. WHP is stronger than "expected"

SKIP LISTS

☐ Insert, delete, search in $\lg(n)$? (WHP)
☐ Easier to implement than balanced tree

Regular Linked List (sorted)
cost of search $= o(n)$ in worst case

TWO LISTS



◆ TWO LIST SEARCH (x)

- First search in L_1 until going right would goo too far
- Then go down to bottom list L_0
- Walk right until element found (or not)

✓ INTUITIVE

Q1

▽ If bottom list has $|L_0|$ elements and a random $|L_1|$ selected for the top list ⇒ Search cost?

▽ 2 sorted lists with $|L_0|$ and $|L_1| \Rightarrow |L_1| + \frac{|L_0|}{|L_1|}$

What size L_1 would you pick? $|L_1| = \sqrt{|L_0|}$

More linked lists — Cost → search cost
2 sorted lists ⇒ $2\sqrt{n}$
3 sorted lists ⇒ $3\sqrt[3]{n}$
k sorted lists ⇒ $k\sqrt[k]{n}$
$\lg n$ sorted lists ⇒ $\lg n \sqrt[\lg n]{n} = 2$

Q2

Balanced:
L_2: $\sqrt[2]{n}$
L_1: $\sqrt[3]{n}\sqrt[3]{n}$
L_0: $\sqrt[4]{n}\sqrt[4]{n}\sqrt[4]{n}\sqrt[4]{n}$

They provide you with approx, not precise answer.

◆ DATASTRUCTURE IS PROBABILISTIC,
so we do not controll all the time.

W A T C H

INSERT(x)

- Search(x) to see where it fits in L_0
- Always insert into L_0.
- Insert x to same lists above. HOW?

◇ Flip fair coin: If heads ⇒
promote next level up & repeat
else ⇒
stop

Warmup Lemma: # levels in an n-element skip list is $c \cdot \lg n$ WHP

Q3

PROOF

$P[> c\lg n \text{ levels}]$ (failure)

$= P[\text{some element got promoted} > c\lg n \text{ times}]$

$\leq n \cdot P[\text{element } x \text{ get promoted} > c\lg n \text{ times}]$

~~UPPER~~ UNION BOUND

$P(A \cup B \cup C) \leq P(A) + P(B) + P(C)$

$\leq n \left(\frac{1}{2}\right)^{c\lg n} = \frac{n}{n^c} = \frac{1}{n^{c-1}}$ ☺

THEOREM Any search in an n-elt skiplist costs $O(\lg n)$ WHP
(IT WAS OUR MOTIVATION)

Cool Idea: LOOK AT SEARCH BACKWARDS

bsearch(x): start from x. If you can go up, do it.
- if not move left
stop when you reach top left

Claim: bsearch(x) and search(x) have same # of steps

Probability of any node having up arrow $= 1/2$
because of the insert operation.

◇ bsearch going left (tails) or up (heads) because like independent coin tosses.

◇ Warmup lemma ⟶ total # of heads $\leq c\lg n$ WHP

LEC 12

HASHING

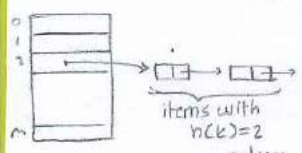
Universal hash functions
Perfect hashing

n: # of keys stored in table
m: # of slots in table
u: # of possible keys
α: n/m load factor

PROBABILITY SUMMARY ⑥ $P(x > (1+\delta)\mu) \leq e^{-\delta^2 m/3}$ for $\delta \geq 1, E(x) = \mu$ (x binomial)

① Linearity of expectations $E[X_1 + \dots + X_n] = E[X_1] + \dots E[X_n]$
② Union bound: $P[X_1 \cup X_2 \cup \dots X_n] \leq P(X_1) + \dots + P(X_n)$
③ Markov inequality: $P[x \geq a] \leq E[x]/a$ for $x, a \geq 0$
④ Chebyshev Inequality $P[|X-\mu| \geq c] \leq \sigma^2/c^2$ if $E[x] = \mu$, $Var[x] = \sigma^2$
⑤ Chernoff " $P[x > E[x] + r] \leq e^{-2r^2/m}$ for binomial x with m trials

HASHING W/ CHAINING



Items with $h(k)=2$ → key
Given a key k, compute hash fn $h(k) \in [0, m-1]$

$O(1)$ for { search, delete, insert
$O(n)$ space

complexity for op → look up, → load factor

$\Theta(1+\alpha)$

Assuming simple uniform hashing

For $\Theta(1+\alpha)$ to be true
Either:

◇ Assume inputs are random
(Average case analysis)
◇ Choose $h \in \mathcal{H}$ randomly
$\mathcal{H}$ is a family of a hash functions
(Expected case analysis)

DEF
Universal hash family $\mathcal{H}$

$$\Pr_{h \in \mathcal{H}} \{ h(k) = h(k') \} \leq \frac{1}{m} \quad \forall k \neq k'$$

There exist such a family!
▶① $\mathcal{H}$ = {all functions $h: \{0, \dots, u-1\} \to \{0, \dots m-1\}$}
problem: h takes $O(u)$ space to store since it is arbitrary.

▶② From CLRS (BOOK). Choose prime $p \geq u$ (once)
Define $h_{ab}(k) = [(ak+b) \bmod p] \bmod m$
UNIVERSAL ← $\mathcal{H} = \{ h_{ab} \mid a, b \in \{0, 1, \dots, u-1\} \}$

▶③ Dot product hash family
: assume m is prime
: assume $u \leq m^r$ for integer r
: view keys in base m.
$k = k_0 + k_1 m + \dots + k_{r-1} m^{r-1}$
: pick a random key $a = a_0 + a_1 m + \dots a_{r-1} m^{r-1}$

$h_a(k) = \sum a_i k_i \pmod m$

THEOREM ③ is universal

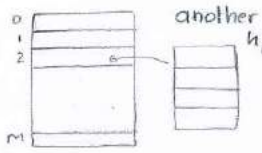
PROOF Take any two keys $k \neq k'$
they differ in at least one digit $k_d \neq k_d'$

$$\Pr_a \{ h_a(k) = h_a(k') \}$$
$$= \Pr \{ \sum a_i k_i = \sum a_i k_i' \pmod m \}$$
$$= \Pr \{ (\sum_{i \neq d} a_i k_i) + a_d k_d = (\sum_{i \neq d} a_i k_i') + a_d k_d' \pmod m \}$$
$$= \Pr \{ \sum_{i \neq d} a_i (k_i - k_i') + a_d (k_d - k_d') = 0 \pmod m \}$$
$$= \Pr \{ a_d = \dots \pmod m \} = 1/m$$
(non zero)

PERFECT HASHING

Guaranteed $O(1)$ search
Worst case $O(n)$ space
Possible for a fixed set of keys
Polynomial build time WHP

IDEA Two level hashing



(we have it since we want $O(n)$ space)

① Pick $h \in \mathcal{H}$ for $m = \Theta(n)$
hash all items with chain

② For each slot j:
- ℓ_j = # of items in slot j
- Pick h_j for this slot.
- Create table w/ size $\Theta(\ell_j^2)$
- Use this instead of chain.

(1.5) space $= n + \sum \ell_j^2$ → you can find h functions easier in this way
If space $> c \cdot n$ then redo step 1

(2.5) If there is a collision on the i^{th} hash table, redo step 2.

LEC 13

watch until 08:40

SUMMARY:

- PERFECT HASHING
  - No collusion WHP
  - O(n) space
- SKIP LISTS
  - O(nlogn) height WHP
  - O(nlogn) search WHP

PERFECT HASHING CN'D

How many times do I repeat 1.5 and 2.5? What is the total build time?

(2.5) P(collision if ℓ_j elts hash to ℓ_j^2 slots)

$= P(\text{at least one of } \binom{\ell_j}{2} \text{ pairs collide})$

$\left(\text{Union Bound}\right) \leq \binom{\ell_j}{2} \underbrace{P(\text{one specific pair collides})}_{\frac{1}{\ell_j^2} \text{ (universal hash function)}}$

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

EXPECTED # OF REPETITIONS FOR MINI TABLE J

≤ 2

MAX # OF REPETITIONS IN ANY CELL

O(lg n) WHP

Assume m=n

- P(any cell repeats $\geq c \lg n$ times)

  $\leq n \cdot P(\text{one cell repeats} \geq c \lg n \text{ times})$ — IMPORTANT

  $\leq n\left(\frac{1}{2}\right)^{c \lg n} = \frac{1}{n^{c-1}} \xrightarrow[n \to \infty]{} \infty$ if $c > 1$ ✓

TOTAL COST OF 2.5 $m = O(n)$

$O(n) \cdot O(\lg n) \cdot \ell_j$ — $O(\lg n)$: We need to consider now.

- $O(n)$: Size of big table
- $O(\lg n)$: max # of repetitions
- ℓ_j: cost of one repetition = # of elements in a small table

- Prove $\max \ell_j = O(\lg n)$ WHP (with using ⑤)

Q1

- What is binomial variable?

$O(n \cdot \lg^2 n)$ is total cost of #2.5

TOTAL COST OF 1.5

If size $> c \cdot n \Rightarrow$ repeat

- $E[\text{size}] = E\left[\sum_{j=0}^{m-1} \ell_j^2\right] = E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} I_{i,j}\right]$

  $I_{i,j}$: indicator — 1 if i, j collide, 0 otherwise

$\ell_j = 2$ (1, 2) $\ell_j^2 = 4$

$1 = I_{11}, I_{12}, I_{21}, I_{22}$

$\ell_j = 3$ (1, 2, 3) $\ell_j^2 = 9$ } There are 9 cases s.t. $I_{i,j} = 1$

$= \sum\sum E[I_{i,j}]$

$= \underbrace{\sum_{i=1}^{n} E[I_{i,i}]}_{1} + 2 \underbrace{\sum_{i>j} E[I_{i,j}]}_{1/m}$

$= n + 2 \cdot \frac{1}{m}\binom{n}{2}$ → # pairs

$= \Theta(n)$ $\left(\frac{O(n^2)}{O(n)}\right)$ expected

Using Markov (③)

$P(\text{size} > cn) \leq \frac{\Theta(n)}{cn}$

If we make c large enough I can make $P(\text{size} > c \cdot n) < \frac{1}{2}$

Total cost $O(n)$

TOTAL BUILD COST — Polynomial

LEC 14
LEC 15

Data Structure Augmentation

Can we add/maintain new fields to a data structure to support new ops?

- Rank/select in BST's
- Finger search trees
- Range trees

Rank(x): find x's index in sorted order

Select(i): find element of rank i.

Select(Rank(x)) ⇒ x

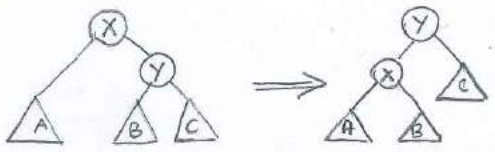
Q1

After insertion, you need to update the ranks : O(n)
Nice TRY

ADD Field size of subtree

Q2

Q3



1 Easy Tree Augmentation is when we can compute x.f from x.left.f and x.right.f (f = field)

CONSIDER NEW FIELD

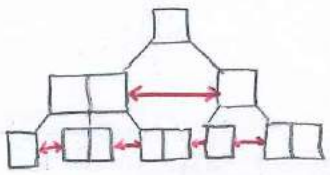
① DEPTH : Distance from root : If rotation at top, O(n) depths need to change ✗ (not working)

② HEIGHT : Distance from leaf : ✓

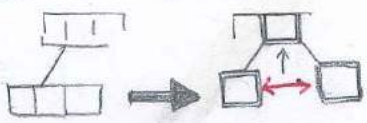
FINGER SEARCH TREES

GOAL If already found y, search (x from y) should only take $O(\lg|rank(x)-rank(y)|)$

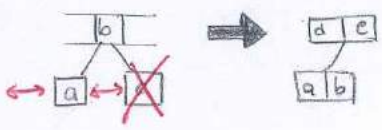
IDEA Level linked 2-3 trees



SPLIT



MERGE

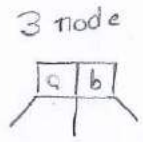


CONT.

- ☑ Easy Tree Augmentation
- ☐ Finger search SEARCH(x from y) ~ $O(\lg|rank(x)-rank(y)|)$
- ☐ Range search

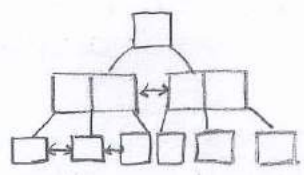
IDEA 1 Use 2-3 Trees

Either 2 or 3 children, 1 or 2 keys } at each node

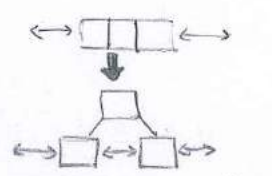


logn

IDEA 2 Add level links



SPLIT Feature:



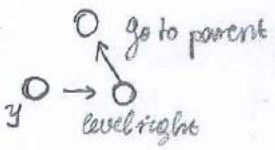
merge is similar

IDEA 3 Store all keys in leaves

non-leaf nodes do not store keys but they keep min/max of the subtree.

SEARCH (x FROM y):

v : leaf containing y (given)
if v.min ≤ x ≤ v.max
  do top-down search in v.
if x < v.min v = v.level left
elif x > v.max v = v.level right
v = v.parent
repeat



How many times go up

- Every step up ×2 ×3 more nodes in subtree
- There are rank(x)-rank(y) keys between x and y
  we will reach v.min ≤ x ≤ v.max in $O(\lg|rank(x)-rank(y)|)$ steps.
- search is also $O(\log|rank(x)-rank(y)|)$

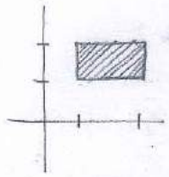
RANGE SEARCH : Put n points $\in \mathbb{R}^D$ into a data structure s.t. support range queries

- Find (count) points in given axis-aligned box

  $min_1 - max_1$ 1st dimension
  $min_2 - max_2$ 2nd //

LEC 15 CONT

IN 1D : range
IN 2D : rectangle
3D : prism



1D : sorted array + binary search → successor / predecessor
(binary search: log n)

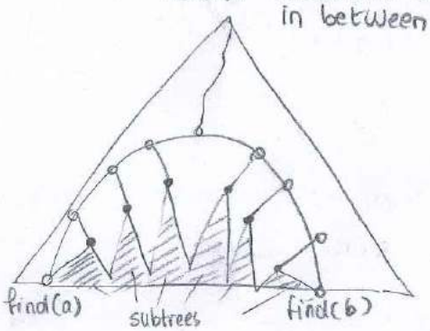
$O(\lg n)$ search for x.min and its index
// // // x.max // // //

Finger search Tree

Find x.min from $-\infty$ ($O(\lg n)$)
Go right in $O(1)$ k times
For counting also keep indices with keys $\geq \lg n$

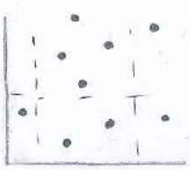
1D range tree — Complete BST (static)
AVL (dynamic)

range query (a, b)
- search (a) # lgn
- search (b) # lgn
- trim common prefix
- return nodes and subtrees in between.

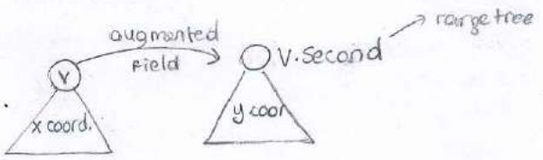


- We can represent implicit answer by listing subtrees,
  $O(\lg n)$ subtrees
- If insist on listing keys $\in [a,b]$
  $O(\lg n + k)$ where k is # keys $\in [a,b]$
- If only count $O(\lg n)$ with size augmentation.

2D IDEA 1 Use a 1D range tree for the x. coordinates



IDEA 2 Every node v in x-tree stores a secondary 1D range tree for y coordinates of the elements in v.



you store so many trees.

Space

x-tree $O(n)$ space

How many secondary trees appear in?
= how many parents which list a $\geq$ a secondary tree?

$O(\lg n)$

y-tree $O(n \lg n)$

Search and list $O((\lg n)^d + k)$ (d: dimension)

DYNAMIC PROGRAMMING

- DP vs D&C
- Bottom-up vs Top Down
- Extracting the solution
- Examples

DC Total cost = Σ all merge costs in tree

① Divide problem into subproblems
② Compute subproblems recursively
③ Merge results

▶ Merge sort

DP Total cost = Σ all merge costs for unique subproblems

① ② ③ ✓

DIFFERENCES
- Subproblems repeat
- caching answers help

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

1 1 2 3 5 ...

Naive recursive implementation

$T(n) = T(n-1) + T(n-2) + \Theta(1) \sim O(2^n)$

If we "memoize" the results:

$T(n) \sim O(n)$ → O(1) for computing each element

Runtime complexity in DP

NUM OF UNIQUE SUBPROBLEMS
X
THE COST OF MERGE

BOTTOM-UP: create a table Fib [1|1|2|3|5|...] } LOOP. look-up to the table

TOP-DOWN: function fib(n)
if $n \leq 2$ return 1 } base case
elseif fibtable[n] is defined
return fibtable[n]
else
ans = fib(n-1) + fib(n-2)
fibtable[n] = ans
return ans

N = subproblems, only unique

a dictionary

Q1
Q2
Q3

If Blocks = ∅
return 0
else max_i { RB(C[i]) + h_i }

unique subproblems ~ unique C[i] ~ n ✓

cost of each: n | To compute C[i] O(n)

visit all to check

Bonus nlogn?

Take 20

mincoins(20)
min(mincoins(N-1)+1, min coins(N-5), mincoins(N-10))

mincoins(19) mincoins(15)
mincoins(10) + 1
↳ min(minco(10-1), minco(10-5), minco(10-10))

LEC16
LEC17 DYNAMIC PROGRAMMING 2

1. Recursively define the value of an optimal solution based on optimal solutions of subproblems
2. Use top-down (recursion & memoization) or bottom-up (table based) implementation
3. Total cost = # unique subproblems × cost of merge

Problem 1 Longest Palindrome Subsequence

subsequence should be non-contigrous

c h a r a c t e r

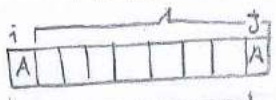
Q1 : underqualified → deified
: turbaventilator → rotator

DP Solution: L(i,j) length of LPS of X[i:j]

if i==j return 1

Q2 Rest of the implementation

Assume L[i+1, j], L[i, j-1] known



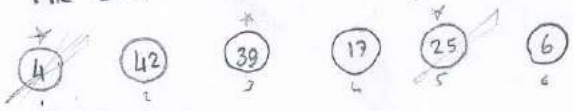
if i > j return ∅
if X[i] == X[j], return 2 + L(i+1, j-1)
else max(L(i+1, j), L(i, j-1))

▶ A ... TOT ... A

cost: # of unique steps O(n²) } CHOICE OF i,j
merge cost O(1) max
(if memoized)

Problem 2 Alternating Coins

Row of n coins of values $V_1 .. V_n$ (n even)
In each turn, next player takes first or last coin
The one with more money at the end wins.



odd: 68
even: 65

Simple Strategy

Compare odds $V_1 + V_3 + \cdots$
// even $V_2 + V_4 + \cdots$

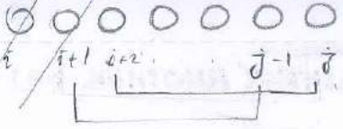
1st player can always take from always odd or always even. ✓

To maximize winning:

V(i,j) max value we can definitely win if it is our turn and only coins $V_i \cdots V_j$ remain

V(i,i) = just pick, get v_i

$V(i, i+1) = \max(v_i, v_{i+1})$



$V(i,i+2)$ ...
$V(i,i+3)$

Q3 Recursive algorithm

opponent

$V(i,j)$:

if $i==j \Rightarrow v_i$

else $\max \begin{cases} < \text{Pick } v_i > \\ < \text{Pick } v_j > \end{cases}$

$v_i + \min \{ V(v_{i+2}, j), V(v_{i+1}, j-1) \}$

$v_j + \min \{ V(i+1, j-1), V(v_i, v_{j-2}) \}$

If I pick V_i, I get V_i

$V_{i+1} \cdots V_j$ is remaining

Opponents turn :

Opponent's choices
① V_{i+1}
② V_j

If $V_{i+1} \Rightarrow$ I get $V(v_{i+2}, j)$

If $V_j \Rightarrow$ I get $V(v_{i+1}, j-1)$

Q4 cost

how many i,j pairs $\} n^2$

merge cost: $\times$ $\} O(1)$

OPTIMAL BINARY SEARCH TREES

Keys $k_1 < k_2 < \dots < k_n$

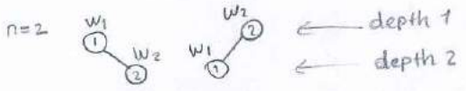
weights $w_1, w_2, \dots, w_n$

Find tree that minimizes

$$\sum_{i=1}^{n} w_i \cdot \text{depth}(k_i)$$

example motivation

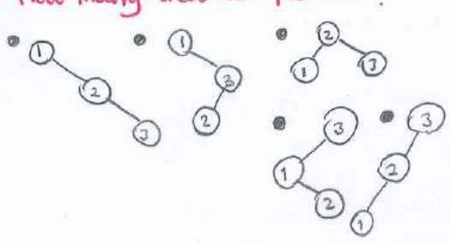
why ~ Dictionary search problem
w_i: how often particular word is inserted
if frequent, put it to top



cost $= w_1 + 2w_2$

$w_2 + 2w_1$

Q5 How many trees are possible?



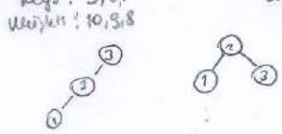
If r has the biggest weight

Pick biggest weight in $\{ < r \}$ | Pick biggest weight in $\{ > r \}$

Does it work?

Counter example:

Keys: 3, 2, 1
Weights: 10, 9, 8



$9+2.8+2.10 = 45$

$10+2.9+3.8$
$= 52$

$e(i,j)$ = cost of optimal BST on $\{k_i, k_{i+1}, \dots, k_j\}$

If $i==j$: w_i

// what we need to do is picking the root

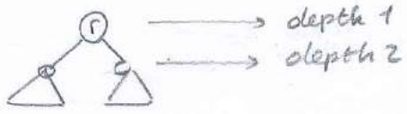
$$\min_{i \le r \le j} \left(w_r + \text{cost of } \{i \dots r-1\} + \text{cost of } \{r+1, \dots, j\} \right)$$

$w_r + \sum_{k=i}^{j} w_k$

root node

$\text{cost of } \{i \dots r-1\} \rightarrow e(i, r-1) + \sum_{k=i}^{r-1} w_k$

$\text{cost of } \{r+1, \dots, j\} \rightarrow e(r+1, j) + \sum_{k=r+1}^{j} w_k$



Final answer $= \min \left(w(i,j) + e(i, r-1) + e(r+1, j) \right)$

LEC 19 (UE DISMISSED)

SHORTEST PATHS

- ☐ REVIEW
  - ☐ General Properties
  - ☐ Single pair S.P.
  - ☐ Single source S.P.
- ☐ ALL PAIRS S.P.
  - ☐ Floyd-Warshall
  - ☐ Johnson

REVIEW

Given weighted directed graph $G=(V,E,w)$

$p = \{u_1, u_2, \dots, u_n\}$ is a path $u_1 \sim u_n$ if $u_i, u_n \in E$

$w: E \to \mathbb{R}$ edge weights

$\delta(s,v)$ = shortest path weight $\left(\sum w(u_i, u_{i+1}) \right)$

(s: vertex)

SPSP: Given s v find $\delta(s,v)$

Source SSSP: Given s find $\delta(s,u)$ $\forall u \in V$

APSP: Given G find all $\delta(s,v)$ $\forall s,v \in V$

GENERAL OBSERVATION

① 

Given a shortest path
all subpaths
must be shortest

② Cycles: Can a shortest path have cycle?

NO

a) Positive weights (+) Not allowed
b) Negative weights (−) not
c) ∅ cycles

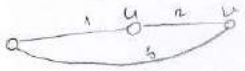
③ If no cycles, how many edges can SSP has?

$|V|-1$

④ RELAXATION

We start with $d(u,v)$ estimate $\geq \delta(u,v)$

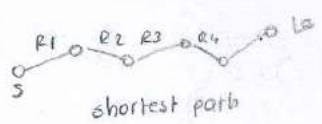
Relax(s,u,v):



If $d(s,v) > d(s,u) + w(u,v)$

$d(s,v) = d(s,u) + w(u,v)$

⑤ Belman Ford $O(V,E)$ work with negative edges



Relax R1, R2, R3 ... in that order

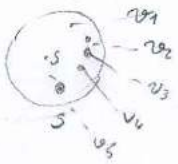
I get $d(s,u) = \delta(s,u)$ $\forall u$ in SP

BF If I relax every edge $|v|-1$ times

I should get $d(s,u) = \delta(s,u)$ $\forall u \in V$

how expensive

⑥ DIJKSTRA . $O(V \lg V + E)$



Queue

Maintain a priority-p that keeps shortest path to S $d(s,v)$ from every vertex outside S.

WHILE $S \subset V$

V = Extract min(Q)

$S = S \cup \{v\}$

for u in V.neighbors

decrease key(u);

$O(v)$ times extractMin

$O(E)$ times decrease key

APSP

Non-neg weights $\Rightarrow |V| \times$ Dijkstra — General Run Time $O(V^2 \lg V + VE)$

General $\Rightarrow |V| \times$ B-F (Belman Ford) — $O(V^2 E)$

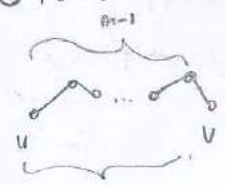
	Dense $E \sim O(V^2)$	Sparse $E \sim O(V)$
$O(V^3)$	$O(V^2 \lg V)$	
$O(V^4)$	$O(V^3)$	

DYNAMIC PROG FOR APSP restrict all to m edges

① subproblems $d_{uv}^{(m)}$ = weight of S.P. $u \to v$ that uses $\leq m$ edges

② Guess What is the last edge (x,v)?

③ Recurse $d_{uv}^{(m)} = \min_{x \in V} \left[d_{ux}^{(m-1)} + w(x,v) \right]$



$d_{uv}^{(0)} = \begin{cases} \infty & \text{if } u \neq v \\ 0 & \text{if } u = v \end{cases}$

$O(V^4)$ { Initialize $d^{(0)}$

for m in 1:(|V|-1) → like 3 for operation

For $u \in V, v \in V, x \in V$

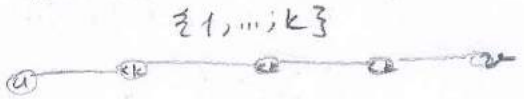
if $d_{uv}^{(m)} > d_{ux}^{(m-1)} + d_{xv}^{(1)}$ } relaxation } you can omit superscripts

$d_{uv}^{(m)} = d_{ux}^{(m-1)} + d_{xv}^{(1)}$

FLOYD WARSHALL (ANOTHER DP)

① subproblem Number of vertices 1:(|V|-1)

$c_{uv}^{(k)}$ = weight of S.P. $u \to v$ whose intermediate vertices come from $\{1, \dots, k\}$



not optimal but subproblem

② Guess

Do we use k in S.P?

③ Recursion

$c_{uk}^{(0)} = w(u,v)$

$c_{uv}^{(k)} = \min \{ \underbrace{c_{uv}^{(k-1)}}_{\text{don't use } k}, \underbrace{c_{uk}^{(k-1)} + c_{kv}^{(k-1)}}_{\text{use } k} \}$

$c = w(u,v)$ $\forall u,v$ Initialization

for k=1: |V| -1

for u in V

for v in V

if $c_{uv} > c_{uk} + c_{kv}$

$c_{uv} = c_{uk} + c_{kv}$

} $O(V^3)$

Relaxation

LEC 20

- ☐ All pairs shortest paths
  - ☐ Johnson's Algorithm
- ☐ Maximum Spanning Tree
  - ☐ Prim's Algorithm
  - ☐ Kruskal's Algorithm

Last Time

① Floyd Warshall : Clever DP for APSP

Gives $O(V^3)$ compared to $O(V^4)$ for naive DP

② Dijkstra Fast $O(V \lg V + E)$ Algo for SSSP

But cannot handle (-) weights

③ Johnson's Algorithm: IDEA get rid of (-) w's and use Dijkstra V times

JOHNSON

① Find function $h: V \to \mathbb{R}$ s.t

$w_h(u,v) = w(u,v) + h(u) - h(v) \geq 0$

getting rid of negative edges

② Run Dijkstra on (V, E, w_h) from every $s \in V$.

$\Rightarrow \delta_h(u,v)$ for all $u,v \in V$

③ Claim $\delta(u,v) = \delta_h(u,v) - h(u) + h(v)$

Q1 Prove #3

Look at any path $p = v_0 \to v_1 \to \cdots \to v_k$

and show that $w_h(p) = w(p) + h(v) - h(z)$

$$w_h(p) = \sum_{i=1}^{k} w_h(v_{i-1}, v_i)$$

$$= \sum_{i=1}^{k} [w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)]$$

$$= \underbrace{\left(\sum w(v_{i-1}, v_i)\right)}_{w(p)} + h(v_0) - h(v_k)$$

★ w_h does not change shortest paths.

HOW TO FIND h?

Add a new vertex s to G and add weight 0 edges $s \to v$ $\forall v \in V$

$s \to z$ path now exist $\forall z$

$\delta(s, z)$ is finite $\forall z$

(assuming no (-) cycles)

Assign $h(z) = \delta(s, z)$ — shortest path weight

Q2 Why does this work?

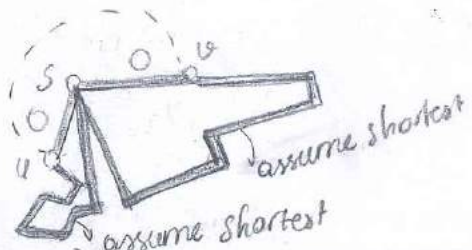
(Use triangle inequalities)

Claim: $w_h(u, z) \geq 0$

$w(u, z) + h(u) - h(z) \geq 0$

$w(u, z) + \delta(s, u) - \delta(s, z) \geq 0$ ✓

$\delta(s, z) \leq \delta(s, u) + w(u, z)$



ANALYSIS — relax every edge

① B-F from s: $O(VE)$

② Reweight all edges: $O(E)$

③ Run Dijkstra V times: $O(V^2 \lg V + VE)$

\+ ④ Reweight all pairs (go to original pair) $O(V^2)$

$O(V^2 \lg V + VE)$

if dense O^2

$d_h \to \delta$ $\forall u, v$ — min total weight

MINIMUM SPANNING TREE

Tree: connected graph with no cycle

Q3 Find MST for the graph

GREEDY ALGORITHMS

① OPTIMAL SUBSTRUCTURE (like DP)

② GREEDY CHOICE PROPERTY: LOCALLY OPTIMAL CHOICES LEAD TO GLOBALLY OPTIMAL SOLUTION (literally one choice comparing to DP)

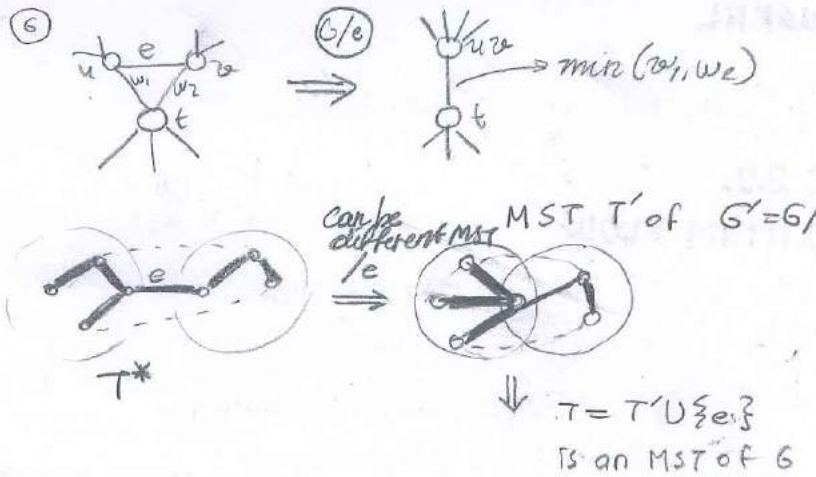
OPTIMAL SUBSTRUCTURE FOR MST

If $e = (u, v) \in$ some MST of G

contract edge e merge vertices $u, v \Rightarrow G'$

If T' is an MST of $G' = G/e$

then $T = T' \cup \{e\}$ is an MST of G



$\Downarrow$ $T = T' \cup \{e\}$ is an MST of G

PROOF

T^* be MST of G that contains e.

$\Rightarrow T^*/e$ is a spanning tree of G' — we do not know it is min.

T' is an MST of G'.

$\Rightarrow w(T') \leq w(T^*/e)$

$\Rightarrow w(T) = w(T') + w(e) \leq w(T^*/e) + w(e) = w(T^*)$ — MST

So it is also MST.

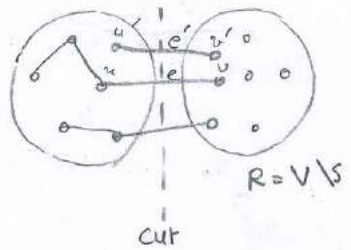
LEC 21 : GREEDY ALGORITHMS

① Find heuristic
② Prove correctness
③ Efficient implementation
④ Complexity analysis

- ☐ MST: Prim's Algorithm (one vertex)
- ☐ MST: Kruskal's Algorithm
- ☐ Other Problems

MST HEURISTIC

The least weight edge across a cut is always part of an MST.



PROOF cut & paste argument (common)

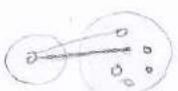
- Considering an MST T of $G = (V, E)$
- Assume the least weight edge $e(u, v) \notin T$
- T must have some other path $u \to v$
- $T' = T \setminus \{e'\} \cup \{e\}$ (cut and paste)
- T' is a spanning tree & $w(T') \leq w(T) \Rightarrow T'$ is an MST

Q1 Q2 Q3 Q4 Q5

PRIM

Start with $|s| = 1$ & grow.

☑ Priority Queue Q on R where $v.key = \{\min(w(u, v)) \mid u \in S\}$



☑ Initialize $S = \emptyset$, Pick starting vertex u, $u.key = 0$, $v.key = \infty$ $\forall v \neq u$

☑ Until Q is empty

- $u = extract\text{-}min(Q)$
- add u to S
- For $v \in Adj(u)$

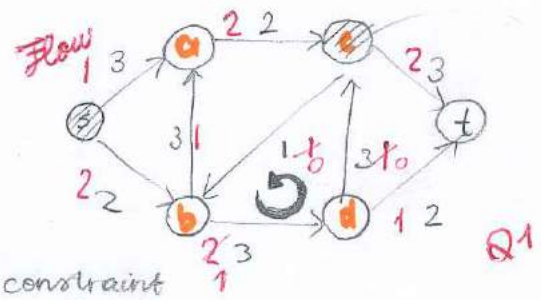
If $v \in R$ & $w(u, v) < v.key \Rightarrow v.key = w(u, v)$

KRUSKAL O

Takes globally min edge
Add if the two vertices not already connected.

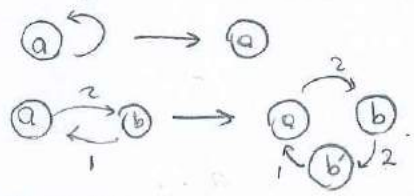
LEC 22
MAXIMUM FLOW

- Flow Network: $G(V,E)$ with 2 distinguished vertices source s, sink t. (directed)
- Each edge $(u,v) \in E$ has a ≥ 0 capacity $c(u,v)$ A flow $f(u,v)$ can go through on edge if $f \leq c$



constraint
f_{in} = flow except for s and t

- Some cycles are not allowed.



Flow Properties

- $f(u,v) = -f(v,u)$ Skew Symmetry

Implicit summation notation

$$f(s,V) \equiv \sum_{v \in V} f(s,v)$$

single vertex, set of all vertices

Notation

$$|f| = \sum_{v \in V} f(s,v) = f(s,V)$$

PROOF of $|f| = f(V,t)$

$$f(s,V) = f(V,V) - f(V-s,V)$$
$$= f(V,V-s)$$
$$= f(V,t) + \underbrace{f(V,V-s-t)}_{0}$$

CUTS A cut (S,T) of G ← IT IS A CUT
is a partition V s.t. $s \in S$, $t \in T$.

Flow across a cut $= f(S,T)$

Q2 WARNING — DIFFERENT GRAPH —

capacity of a cut

$$c(S,T) = \sum_{x \in S} \sum_{y \in T} c(x,y)$$

Q3

THEOREM The value of any flow is bounded above by the capacity of any cut.

$$|f| = f(S,T)$$
$$= \sum_{u \in S} \sum_{v \in T} f(u,v)$$
$$\leq \sum_{u \in S} \sum_{v \in T} c(u,v)$$

RESIDUAL NETWORK Given G, f, define $G_f(V, E_f)$ is the graph with >0 residual capacities
Forward: Difference (Residue)
Backward: −Flow

Q4

Augmenting Path Any path from s to t in G_f the min capacity along path can be added to the original flow to increase it.

FORD-FULKERSON MAX FLOW ALGORITHM

$$f(u,v) = \emptyset \quad \forall u,v \in V$$

while an augmenting path in G_f exists:
do augment f by $c_f(p)$

Last time **LEC 24 (NO 23)**

- Flow value : $|f| = f(s,v)$
- cut: partition (S,T) of V s.t. $s \in S$ $t \in T$
- $|f| = f(S,T)$ for any cut (S,T)
- $|f| \leq c(S,T)$ for any cut (S,T)
- Residual Graph of G for f : G_f
- Augmenting path : Any path from $s \to t$ in G_f.

★ *Ford Fulkerson Algorithm*

- Initialize $f(u,v) = 0$ $\forall u,v \in V$
- Compute G_f
- Residual Capacity $c_f(u,v) = c(u,v) - f(u,v)$
  of a path: $c_f(P) = \min_{(u,v) \in P} c_f(u,v)$
- If there is an augmenting path in G_f : P
  Augment f by $c_f(P)$
  Repeat

MAX FLOW MIN CUT THEOREM

TFAE

(i) $|f| = c(S,T)$ for some cut (S,T)

(ii) f is a max flow.

(iii) f admits no augmenting paths.

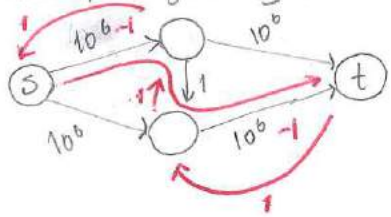
PROOF

Q1

COMPLEXITY OF FORD FULKERSON

Computing G_f from G, f : $O(E)$

Find if augmenting path : $O(E)$



EDMONDS-KARP : Eliminate inefficiencies by using BFS to find shortest augmenting paths (min number of edges)

① $\delta(v) = \delta_f(s,v)$ shortest distance $s \to v$ in G_f
$f(e)$ never decreases during Edmonds-Karp

② The # of augmentations is $O(VE)$

$\Rightarrow O(VE^2)$

VERTEX COVER

Given Graph $G=(V,E)$, $S \subseteq V$ covers G if for every $(u,v) \in E$, S contains u or v.

Find the minimal $|S|$ (NP-hard)

BIPARTITE GRAPH

$G=(V,E)$, $V = L \cup R$ $u \in L$ $v \in R$ $\forall (u,v) \in E$

Solution

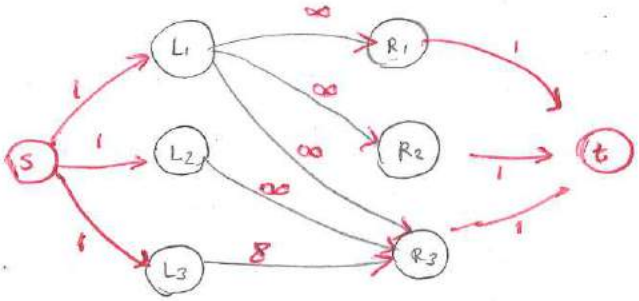
— Given G, define flow network H.

— Create a new source vertex s with cap 1 edges to L

— // // // t // // // // R

— Direct all edges E from $R \to L$ with cap ∞.

— Return Max Flow

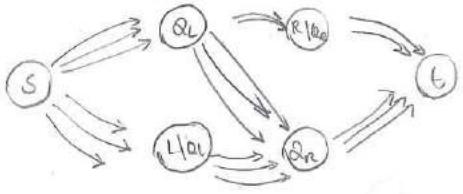


Every vertex cover Q defines a cut $C(S,T)$ *finite*

- Q is a vertex cover for G
- H is a flow network for G.
- $Q \equiv Q_L \cup Q_R$ $Q_L = Q \cap L$ $Q_R = Q \cap R$

Define cut $S = \{s\} \cup Q_R \cup (L \setminus Q_L)$

$T = \{t\} \cup Q_L \cup (R \setminus Q_R)$



$Q_L + Q_R = c(S,T)$

cuts $\rightleftarrows$ Vertex cover map

LEC 25

COMPLEXITY

P = { problems solvable in polynomial time }
size n — $n^{O(1)}$

NP = { decision problems solvable in
output = Y/N
Polynomial non-deterministic time }

in $O(1)$ time can "guess" among polynomial # of choices the one leading to "YES"

n	$O(n)$	$O(n\log_2 n)$	$O(n^2)$	$O(2^n)$
1	1	1	1	2
10	10	30	10	1024
100	100	70	10000	10^{30}
1000	1000	10000	1m	> atoms in universe

"intractable"

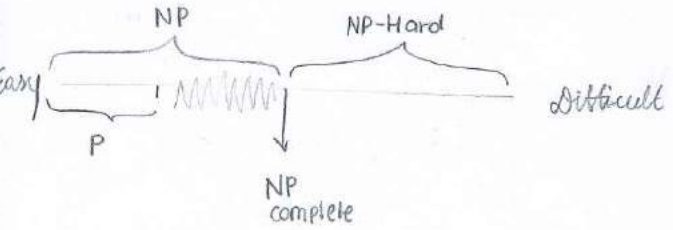
non deterministic machine: lucky machine guess

NP (alternative definition)

- Assume all guessing done in the beginning
- ⇒ equivalent to polynomial time verifier (checks the solution is correct) of polynomial-size certificates for "YES" answers (description of a solution)

☑ Beyond NP: "Proving that white always win in chess"

The description of the solution itself is exponential. (All possible moves of black.)



DEF
- Problem X is NP-hard: If every problem $Y \in NP$ can be reduced to X.
- NP-complete: If $X \in NP$ and X is NP-hard
- Reduction from problem A to B: polynomial-time algo for converting A inputs to equivalent (same answer) B inputs.

Q1

PROOF

How to prove X is NP-complete?

① $X \in NP$: via non-deterministic algorithm or certificate & verifier.

② $X \in$ NP-hard: Reduce from a known NP-complete problem Y to X.

3SAT

- Super Mario Bros
- 3D Matching
  - Subset Sum
    - Partition
      - Rect. Packing
    - 4-Partition
      - Rect Packing
        - Jigsaw Puzzles

☑ 3SAT: Given a Boolean formula of the form:

$(x_1 \vee x_3 \vee \overline{x_6}) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_7}) \wedge$

literals — clause

is there a variable assignment s.t. formula = True

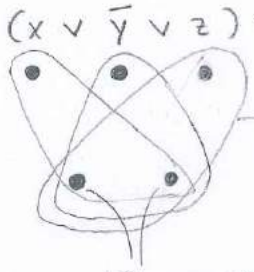
☑ 3DM: Given disjoint sets X, Y, Z each with n elements and triples $T \subseteq X \times Y \times Z$, is there a subset $S \subseteq T$ s.t. each element is in exactly one $s \in S$.

(i) 3DM $\in$ NP: List triples S (poly space). check if $S \in T$ (poly-time) check if all elements occur once (poly-time)

(ii) 3DM is NP-hard: by reduction from 3-SAT

IDEA: Use "gadgets" (pieces of graph or whatever) to represent variables, clauses etc to reduce 3 SAT to 3DM

Clause $(x \vee \overline{y} \vee z)$ All make



Garbage Collection Gadgets

of sufficient number — For extra x & x̄'s

LEC 26

REDUCTIONS To prove A is NP-complete

① First show that $A \in NP$
- ⓐ Polynomial size certificate (SOLUTION)
- ⓑ Polynomial time verification (CHECK CORRECT)

② Prove A is NP-Hard :
- ⓐ Find another NP-Hard problem B, convert B inputs to equivalent A inputs in poly-time
- ⓑ Show A solution ⇒ B solution
- ⓒ Show B solution ⇒ A solution

Q1 Hamiltonian Cycle ⇒ Hamiltonian Path

Given NP-complete (B) in this case — Prove NP complete (A)

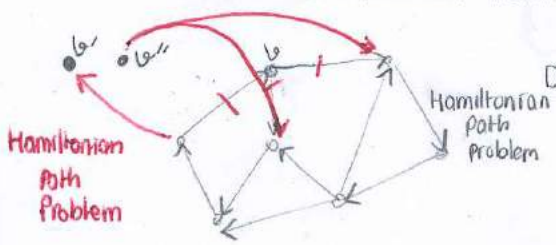
Given a directed graph $G(V,E)$, is there a cycle/path that visits every vertex exactly once.

① HP is in NP ⇒ ⓐ Sequence of vertices as solution (bound: # of vertices polynomial)

ⓑ Given $v_1, \dots v_n$ ⇒ Check each (v_i, v_{i-1}) is an edge — Hash Table $O(E+V)$
⇒ Check each vertex appears exactly once — $O(V)$
POLYNOMIAL

② ⓐ Given Ham Cycle Problem G

convert to equivalent HamPath problem G'



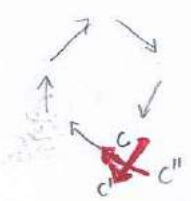
Delete v and link incomings to v' // outgoings to v''

ⓑ HamPathSol ⇒ HamCycleSol

The solution path must start with v'', and with v' (Because v'' has no incoming edges ...)

$v'' \to a \to b \to \dots \to v'$ gives me a solution for HamCycle in G.

ⓒ HamCycle ⇒ HamPath Sol



Splitting any vertex converts the cycle → path

□

Q2 Clique ⇒ Independent Set

Given NP complete — Prove NP-complete

Given a graph $G(V,E)$ and integer k,

is there a set of k vertices $C \subseteq V$ s.t.
- ★ CLIQUE all of them connected
- ★ I.SET none of them are connected

Hint: Insert the edges of G as G'

If there is an edge in G, no edge in G'

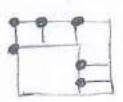
Q3 PARTITION ⇒ RECT PACKING

Partition : Given set of integers $A = \{a_1, \dots a_n\}$ is there a subset $S \subseteq A$ s.t. $\sum S = \sum A/2$

Rectangle packing: Given set of rectangles R and a large rectangle T with $A(T) = \sum A(R)$
Can we pack R into T without overlap?

ⓐ Rectangle Packing is NP.

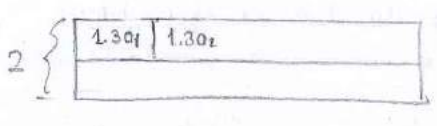
Certificate: list coordinates of R inside T



left upper (You know the sizes)

ⓑ Verification : Make sure $R_1 \dots$ do not overlap. $O(1)$
$O(n^2) \to \binom{n}{2}$ and They are all inside T $O(n)$

ⓐ Given instance A for Partition construct equivalent instance of RP



$T = (2, 3t)$ where t = the length $\sum A/2$
$R_i = (1, 3a_i)$ → guarantees horizontal direction

ⓑ RP ⇒ Partition

ⓒ Part ⇒ RP

Q4 Vertex cover ⇒ Set Cover

Vertex cover: Given undirected graph G and int k, is there a subset of k vertices $Y \subseteq V$ s.t. for each (u,v) in E, either u, or v or both in Y.

Set cover: Given a set S of n elements and m subsets $S_1 \dots S_m \subseteq S$ is there a collection of k subsets that cover all elements?

elts → edge

$US_{i_k} US_{i_j} \dots = S$

S = set of all edges S_i = the set of edges touching v_i

Q5 Subset Sum ⇒ Partition

Subset Sum = Given n ints $A=\{a_1,\dots,a_n\}$ and a target sum t, is there a subset $S\subseteq A$ with $\Sigma S = t$

(2a) Given a SSum instance (A,t) construct a Part instance $A' = A \cup \{\Sigma A + t,\ 2\Sigma A - t\}$ } Two seperate sides

$\longrightarrow \Sigma A + t, a_1, a_2 \dots$

$\longrightarrow 2\Sigma A - t, b_1, b_2 \dots$

LEC 27
APPROX. ALGORITHMS

- Vertex Cover
- Decision Vs Optimization

VC Given undirected graph $G=(V,E)$ find a subset $V'\subseteq V$ s.t. if $(u,v)\in E$ then either $u\in V'$ or $v\in V'$ or both

- DECISION Can you find $|V'|\le k$ vertices?
- OPTIMIZATION Find V' with min # elements
- HOW GOOD IS A HEURISTIC?
- APPROXIMATION SCHEME: By spending more time, can do better?

HEURISTIC

① Max-Degree : Find vertex with max degree, add to V'. Eliminate edges adjacent; REPEAT

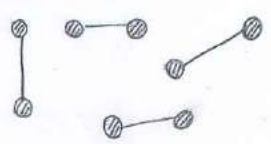
② Random-Edge : Pick random edge (u,v) add both u and v to V'; eliminate edges that touch u, v REPEAT

Performance Guarantee An algo for a problem of size n has an approx ratio $\rho(n)$ if for any input, algo produces a solution of cost C s.t.

$$\max\left(\frac{C}{C_{opt}}, \frac{C_{opt}}{C}\right) \le \rho(n)$$

"$\rho(n)$-approx algorithm"

FOR RANDOM-EDGE



} EDGES PICKED BY MY ALGORITHM $\Rightarrow |V'| = 2k$

CLAIM $C_{opt} \gg k$

k edges I picked are disjoint ⇒ need k vertices to cover.

2-approx algorithm (CONSTANT)

▶ $O\left(n^{\frac{2}{\varepsilon}}\right)$ poly in n, exp in $1/\varepsilon$ = PTAS

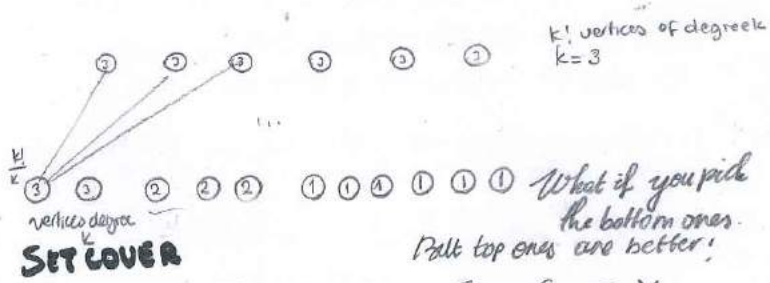
$O\left(\frac{n}{\varepsilon^2}\right)$ poly in n, poly in $1/\varepsilon$ = FPTAS

Instead of picking $C_{opt}=k!$ (top row) We could pick bottom now using Max-Degree

$$\frac{k!}{1}+\frac{k!}{2}+\dots+\frac{k!}{k} = k!\underbrace{\left(1+\frac{1}{2}+\frac{1}{3}+\dots\frac{1}{k}\right)}_{\approx \ln k}$$

Worse than

→ log(k)-approx algorithm for $n = k!(1+\ln k)$



SET COVER

Given a set X and subsets $S_1,\dots,S_m \subseteq X$

Find a minimum collection of subsets that cover all elements in X

(REDUCTION FROM VCOVER)

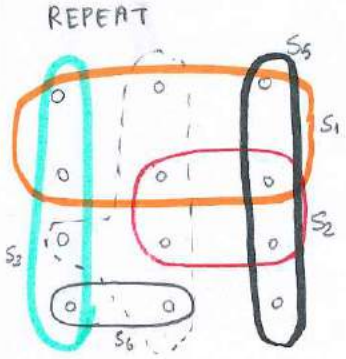
elts ⟺ edges

subsets ⟺ vertices

each edge touches two vertices but not each edge must touch two subsets

HEURISTIC

Pick largest subset eliminate the elements covered. REPEAT



SOL GIVEN BY APPROX SET-COV

S_1, S_4, S_5, S_6

OPTIMAL SOLUTION

S_3, S_4, S_5

HW Prove that this is a $\ln(n)+1$ approx

PARTITION Given a set S of n numbers partition into A and B (of equal sum) — Decision prob

$w(S) = 2L$

$$\max\left(w(A), w(B)\right)$$

w(A): total num in A; w(B): total num in B

minimizing this will get close to eq. partition

A dumb 2-approx : Put all elts into A

Approx Scheme : Polynomial in n. To get $(1+\varepsilon)$-approx algo with cost $f(n,\varepsilon)$

LEC 28

ALGORITHMS FOR NP-COMPLETE PROBLEMS

☐ Last time : APPROXIMATION ALGS

☐ TODAY : Approximation Schemes
get better with more computation

☐ TODAY : FIXED PARAMETER ALGORITHMS
Exact solution for easy version of NP complete problem (parameter=k indicating hardness)

AN APPROXIMATION SCHEME

...

PARTITION : Split a set of numbers S to two pieces A, B with equal sum (NP complete)

...

APPROX-PARTITION ALGORITHM

Define $m = \lceil \frac{1}{\varepsilon} \rceil - 1$ $\left(\varepsilon \approx \frac{1}{m+1}\right)$ 2 PHASES

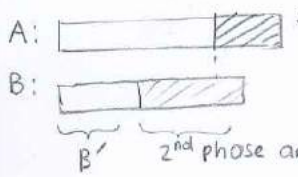
① Find optimal partition of $S_1, \dots, S_m$ ($S = \{S_1 \geq S_2 \geq \dots \geq S_n\}$) ← Takes $O(2^m)$ time

② for $i = m+1 ; n$
if $\Sigma A \leq \Sigma B$; $A = A \cup \{S_i\}$
else $B = B \cup \{S_i\}$

PROOF of $(1+\varepsilon)$-APPROX

Assume $\Sigma A \geq \Sigma B$ (WLOG)

Approx = $\frac{\Sigma A}{L}$; show $\leq (1+\varepsilon)$



S_k : last item added to A

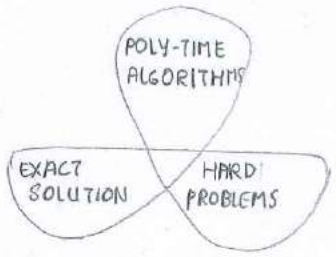
CASE 1

S_k was added to A during first phase
⇒ We have an optimal solution
Nothing added to A, we can't do better than A, B' for first m elts ⇒ we know can't do better for A, B

CASE 2

S_k was added to A in second phase
We know $\Sigma A - S_k \leq \Sigma B$
$\Sigma A - S_k \leq 2L - \Sigma A$
$\Sigma A \leq L + \frac{S_k}{2}$

Since $S_1 \geq S_2 \geq \dots S_n$
we know $S_1, \dots, S_m \geq S_k$
$2L \geq (m+1) S_k$ (since $k > m$)
Approx ratio: $\frac{\Sigma A}{L} \leq \frac{L + S_k/2}{L} = 1 + \frac{S_k}{2L} \leq 1 + \frac{1}{m+1} \leq 1+\varepsilon$ ☐



Can we do this for specific instances?

K-VERTEX COVER

Given graph G, non-neg int k, is there a subset of ≤ k vertices that "cover" all edges.

NOTE can have $k << |V|$

Can we have an exact algorithm that is poly in $n(V+E)$ but may be expt in k?

Brute Force Solution

Try all subsets $\binom{V}{k} + \binom{V}{k-1} + \binom{V}{k-2} + \dots$

⇒ Runtime = $O(E) \times O(V^k)$ (Checking coverage of all edges)

= Polynomial for fixed k } FIXED PARAMETER TRACTABLE

⇒ INEFFICIENT, define $O(n^{f(k)})$ to be bad.

BOUNDED SEARCH TREE ALGORITHM

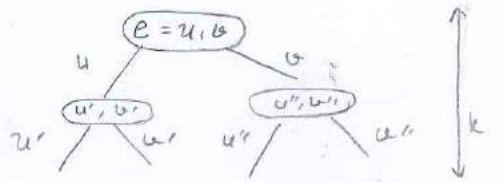
General Technique

☐ Pick arbitrary edge $e = (u, v)$

☐ know that $u \in S$ / $v \in S$ or both (S is the solution) don't know which.

☐ Try (guess) both possibilities
① add u to S, delete u and its edges, recurse with $k' = k-1$
② same thing with v.

Recursion tree



$O(E \cdot V^k)$ BAD
$O(V \cdot 2^k)$ GOOD

2^k nodes in this tree
Total cost = $O(V \cdot 2^k)$

FPT Fixed parameter tractable, if there is an algorithm with

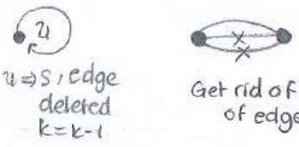
$O(f(k) \cdot n^{O(1)})$ — param; size; independent of k, n

KERNELIZATION

Polytime reduction that converts input (x, k) into small equivalent input (x', k')
$|x'| \leq f(k)$ answer(x) = answer(x')

k- Vertex Cover: Kernelization

- Remove loops and multi-edges



$\Rightarrow$ u $\Rightarrow$ S, edge deleted
k = k-1

Get rid of all but one of edges

- Any vertex with degree > k must be included (include, del edges, k = k-1)
- Remaining graph has max degree k.
- If # of remaining edges > k^2 answer is NO. Can't cover > k^2 edges by picking k vertices of degree k each.
- $|E'| \leq k^2$
- Remove isolated vertices
- $|V'| \leq 2k^2$
- $G' = (V', E')$, k' is a kernel with the same answer as $G = (V, E)$, k.

$|G'| = O(k^2)$ $\Rightarrow$ Brute force: $O(k^2 (k^2)^k + |G|)$

Bounded s.t: $O(k^2 \cdot 2^k + |G|)$

exact solution of kernel — reduction

LEC 5

Duygu Sezen ISLAKOĞLU · 54161

① Select(A,i)

$|B| = |C| = |A|/2$

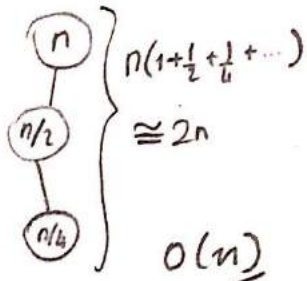
if cond1 ⟶ select(B,i)

else ⟶ select(c, k-i)

$T(n) = T(n/2) + O(n)$

There is no 2 here since there is a condition and we do not perform both

Tree Method



---

② INSERT : O(1)

DELETE : O(1)

SUCCESSOR : O(u)

PREDECESSOR : O(u)

---

③ $\frac{x}{\sqrt{u}}$ = cluster

$x - \frac{x}{\sqrt{u}}$ = position in cluster

$x = clusterNum \cdot \sqrt{u} + positionNum$

$V[x] = V.cluster[clusterNum][positronN]$

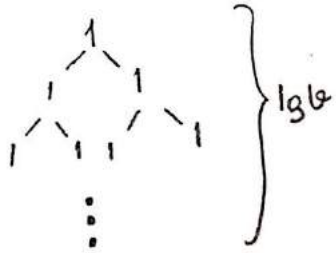
---

④ $T(u) = 2T(\sqrt{u}) + O(1)$

$v = \lg u$

$T(v) = 2T(v/2) + O(1)$

$O(v) = O(\lg u)$



VERSION 3
INSERT

---

⑤ $T(u) = 3T(\sqrt{u}) + O(1)$

$v = \lg u$

$T(v) = 3T(v/2) + O(1)$

$\log_b(a) = \log_2 3 \quad c = 0$

$O(v^{\log_2 3}) = O(\lg u^{\lg 3})$

VERSION 4
SUCCESSOR

DUYGU SEZEN ISLAKOGLU **LEC 6**

PROBLEM 1 Search (x, k)

return (node, index)

complexity in th? $O(th)$

```
i=1
while i≤x.n and k>.x.key[i]      } O(t)
      i=i+1
if i≤x.n and k==x.key[i]         } O(1)
      return (x.i)
elseif x.leaf                    } O(1)
      return NIL
else SEARCH(x.C[i],K)            } O(h)
```

PROBLEM 2 Theorem : B-Tree n keys, t degrees, h height

Show that $h \leq \log_t \frac{n+1}{2}$.

DEPTH	min nodes
0	1
1	2
2	$2t$
3	$2t^2$
⋮	
h	$2t^{h-1}$

Base Case $h=0$ $\quad 0 \leq \log_2 \frac{n+1}{2}$ ✓

Inductive Step Assume that $h \leq \log_t \frac{n+1}{2}$

We need to show that $h+1 \leq \log_t \frac{n+1}{1}$

Next Page

$$
\begin{aligned}
n &\geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\
&= 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) \\
&= 2t^h - 1 \ ,
\end{aligned}
$$

LEC 7

DUYGU SEFEN ISLAKOĞLU
54161

Q1) Increment (A) where A is bit array.

```
#Look at the rightmost.
i=0
while i≤A.len and A[i]=1:
    A[i]=0, i++
if i<A.len
    A[i]=1
```

Worst case $O(k)$

Worst case of n ops: $O(nk)$

Aggregate

- First bit flips everytime
- Second bit // half the time
- i^{th} bit // $\frac{1}{2^{i-1}}$ of the time
- Total cost of n ops $T(n)$

$= n + \frac{n}{2} + \frac{n}{4} + \cdots$

$= n\left[1 + \frac{1}{2} + \frac{1}{4} + \cdots\right]$

$= O(2n)$

$\Rightarrow$ Amortized cost/op $= O(1)$

Q2) Use agg analysis on:

$$\text{cost of op } i = \begin{cases} i & \text{if } i = 2^k \\ 1 & \text{otherwise} \end{cases}$$

consider n operations.

a) what is the worst case? b) What is the amortized cost/op?

LEC 7

a) Most expensive op: $O(n)$
n operations $O(n^2)$

b) Real costs: 1, 2, 1, 4, 1, 1, 1, 8 For 1's: $O(n)$

$$\sum_{i=1}^{\lg n} 2^i = \underbrace{1+2+2^2+\ldots+2^{\lg n}}_{2n-1}$$

11111 $1 = 2^{\lg n+1} - 1$ $O(2n)$

$O(n)$

Q3) Use Accounting to Q2

⚠ Real cost: 1 3 4 8 9 10 11 19 20

$\hat{c}_i$ = 2 2 4 6 8 10 12 14

It started good.

HW $\hat{c}_i = 3$

DUYGU SEREN ISLAK OĞLU | 54161 | LEC 8

Q1) Assume $\phi(D_0) = 0$ and $\phi(D_i) \geq 0 \ \forall i$. Show that $\sum \hat{c}_i \geq \sum c_i$

Induction:

Base step = $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$

$\hat{c}_1 = c_1 + \phi(D_1) - 0$

since $\phi(D_i) \geq 0$, $\sum \hat{c}_i \geq \sum c_i$

Inductive step: Assume $\sum \hat{c}_k \geq \sum c_k$ for some i.

We need to show that it is also true for $k+1$.

$$\sum \hat{c}_k + \hat{c}_{k+1} \geq \sum c_k + c_{k+1}$$

So since

$\hat{c}_{k+1} = c_{k+1} + Q(D_k)$

$\hat{c}_{k+1} \geq c_{k+1}$ as desired.

ANSWER

- $\sum \hat{c}_i = c_1 + \cancel{\phi(D_1)} - Q(D_0) + c_2 + Q(D_2) - \cancel{Q(D_1)} \ldots = \sum c_i + Q(D_n) - Q(D_0)$

Q2) (For multipop) $\phi(D_i)$ = # of elements on stack D.

a) $\hat{c}_{push}$

b) $\hat{c}_{pop}$

c) $\hat{c}_{multipop(k)}$

a) $\underbrace{c_{push}}_{1} + \underbrace{\phi(D_{push}) - \phi(D_{push-1})}_{1} = 2$
b) $\underbrace{c_{pop}}_{1} + (-1) = 0$
c) $\underbrace{c_{multipop(k)}}_{k} + (n-k) - n = 0$

Q3) For binary counter: What is $\hat{c}_{incr}$?

$\phi(D_i)$ = # of 1's

What is c_{inc}? (Assume t_i bits $\rightarrow$ 0, 1 bit $\rightarrow$ 1)

$$\hat{c}_{inc} = c_{inc} + \phi(D_{incr}) + \phi(D_{incr-1})$$

ANSWER

- $c_{inc} = t_i + 1$

$$\phi(D_i) = \phi(D_{i-1}) - t_i + 1$$

$$+ \quad 2$$

Real costs $c_i = \begin{cases} i, & \text{if } i-1 \text{ is } 2^k \\ 1, & \text{otherwise} \end{cases}$ Q5)

Q4) a) use dynamic table with accounting.

Find $\hat{c}$ s.t $\sum \hat{c}_i \geq \sum c_i$ or find a way to credit each item so it pays for future ops.

$\hat{C} = 3$

Why $\hat{c}_i = 3$ is enough?
- young generation
- watch the video

1 FOR INSERT
1 FOR FIRST COPY
1 FOR CARING OLD GUY

b) use dynamic table with potential method

Let • $\Phi(D_{doubling}) = 0$ (right after doubling)

• $\Phi(T) = 2 \times T_{num} - T_{size}$ (num = number of elements)

• What is $\nabla\Phi_i = \Phi(D_i) - \Phi(D_{i-1})$
What is $\hat{c}_i$?

$\hat{c}_i = c_i + \nabla\Phi_i = \begin{cases} i + \nabla\Phi_{Doubling} \\ 1 + \nabla\Phi_i \end{cases}$

$\nabla\Phi_{doubling} = \Phi(D_{Doubling}) - \Phi(D_{Doubling-1})$ ($\Phi(D_{Doubling}) = 0$)

•ANSWER

If not doubling, $c_i = 1$, $\nabla\Phi = 2$, $\hat{c}_i = 3$

If doubling $c_i = i$, $\nabla\Phi = 3 - i$ ← HOW

Q6) Table delete

- Double when full
- Halve when $\alpha(T) < 1/4$

$\alpha(T)$ = load factor = $\frac{T_{num}}{T_{size}}$

- Potential method

$\Phi(T) = \begin{cases} 2 \cdot T_{num} - T_{size} & \text{if } \alpha(T) \geq 1/2 \\ T_{size}/2 - T_{num} & \text{if } \alpha(T) < 1/2 \end{cases}$

- After expansion or contraction: $\Phi(T) = 0$, $\alpha(T) = 1/2$

$\hat{c}_{insert} = \begin{cases} — & \alpha_i \geq 1/2 \\ — & \alpha_i < 1/2 \end{cases}$

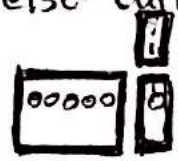
$\hat{c}_{double} = \begin{cases} — & , \alpha_i < 1/2 \text{ no contra} \\ — & , \alpha_i < 1/2 \text{ contractio} \\ — & , \alpha_i \geq 1/2 \end{cases}$

DUYGU SEZEN ISLAKOĞLU 54161 LEC 9

Q1) If $A \times B \neq C$ and $P(NO) = \frac{1}{2}$, PROB of thinking $A \times B = C$ after 10 runs?

To think, $A \times B = C$, I need Y, Y, Y, Y ...
If actually true $P = 1$.
If else $P = 1/2^{10}$

Q2) If $A(Br) = Cr \Rightarrow$ output YES
else output NO. } FRIEVALD

What is runtime?  $O(n^2)$ Runtime of mat vec mult
Runtime of Frievald includes 3 mat mult. $\Rightarrow O(n^2)$

Q3) If $AXB = C$, does always output YES?

$(AXB)xr = Cr$
$A(Br) = Cr$ (assoc)

Q4) Prove that if (one to one) $A_1 = D_{r_2} = 0$ and $r_1 \neq r_2$.

$r_1' = r_1 + v$ $r_2' = r_2 + v$

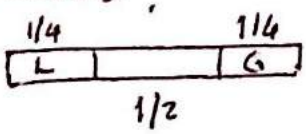
$r_1' \neq r_2'$

Q5) What is $T(n)$?
$O(n^2)$ Tree method. Notebook

Q6) Best case: $T(n) = O(n \log n)$

LEC 10 DUYGU SEZEN ISLAKOĞLU 54161

① If you pick pivot randomly P(Good)?

Imagine elements sorted 

⇒ 1/2

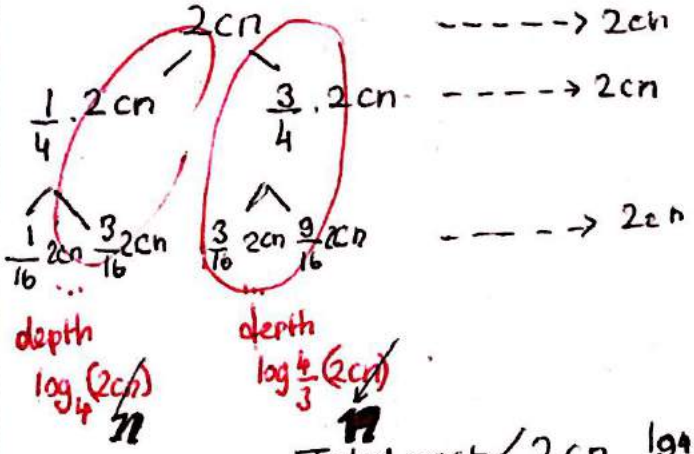
② What is the expected number of repeats? until a good pivot

Geometric form.

Expected value: $\sum_{\text{all values}}$ P(value) x val

2

③ $T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + 2cn$



Total cost $\leq 2cn \log_{\frac{4}{3}} 2(cn)$

$\Rightarrow O(n \lg n)$

$\Omega(n \log n)$ seperate proof

Lower bound

④ Show that $T(n) = O(n)$. (use induction)

Theorem 1 *Let $T(n)$ denote the expected running time of randomized select. Then $T(n) = O(n)$.*

Proof. We will show by the method of substitution. Let's say that $T(n) \leq cn$, and check that it works.

We must first check the base case. This is obvious, however, since $T(n')$ is a constant for some small constant n'.

Now let us check the inductive case. Assume that $T(k) \leq ck$ for all $k < n$, and we now want to show that $T(n) \leq cn$.

$$T(n) \leq n + \frac{2}{n} \sum_{i=0}^{n/2-1} \left(\max\left(T(i), T(n-i)\right)\right) \leq n + \frac{2}{n} \sum_{i=0}^{n/2-1} \left(\max\left(ci, c(n-i)\right)\right). \tag{4}$$

We note that that this is the same as

$$n + \frac{2}{n} \sum_{i=n/2}^{n-1} ci. \tag{5}$$

The term $\frac{2}{n} \sum_{i=n/2}^{n-1} (ci)$ is the same as $\frac{2c}{n} \sum_{i=n/2}^{n-1} i$. So we get

$$T(n) \leq n + c\left(\frac{2}{n} \sum_{i=n/2}^{n-1} i\right) \leq n + c\left(3n/4\right) = n\left(1 + \frac{3c}{4}\right). \tag{6}$$

Hence if we take $c = 4$ (which works for the case $T(1) \leq 4$ as well) we get

$$T(n) \leq n\left(1 + \frac{3 * 4}{4}\right) = n\left(1 + 3\right) = 4n, \tag{7}$$

as we wanted.

DUYGU SEZEN ISLAKOĞLU

54161

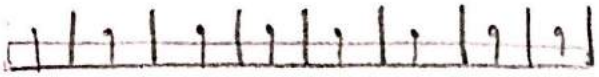
LEC 11

Q1) $100 = |L_0|$ what is the **worst case** search cost if $L_1 = 10$ vs $L_1 = 20$?

(Assume regularly spaced)

$L_1 : 20$

$L_2 : 25$



Q2) Extend the idea to 3 lists $\Rightarrow$ What size L1 L2 should you use?

What is the search cost for regular spacing in worst case?

$L_2 = \sqrt[3]{n}$

$L_1 = \sqrt[3]{n}\sqrt[3]{n}$

$L_0 = \sqrt[3]{n}\sqrt[3]{n}\sqrt[3]{n}$

cost $\sqrt[3]{n}$

Q3) What is the total num of moves (WHP)?

(HINT: CHernoff Bound)

- $\ell_j = O(\lg n)$

We will use $P(X > (1+\delta)M) \leq e^{-\delta M/3}$ for $\delta \geq 1$, $E(X) = \mu$ X binomial

① $M = 1$
② Use $\delta = \lg n$

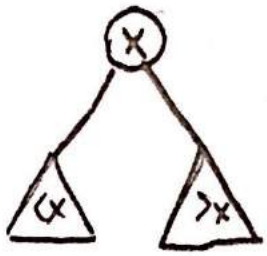
> Show ℓ_j is binomial with $p = 1/n$
>
> #trials = n
>
> Assume $m = n \Rightarrow E[\ell_j] = 1$

$$P(X > 1+\delta) \leq e^{-\delta/3} \qquad \delta = c\lg n$$

$$P(X > 1 + c\lg n) \leq e^{-\frac{c}{3}\lg n} \leq 2^{-\frac{c}{3}\lg n} = n^{-c/3}$$

DUYGU SEZEN ISLAKOĞLU **LEC 14** 54161

Q1 Starting from a BST, what field can we add to each node to support RANK/SELECT?




```
STRUCT NODE
    VALUE
    LEFT
    RIGHT
    RANK
END
```

A1 Add rank as a field to AVL

Q1A How does rank(x) work? O(lgn)

Q1B How does SELECT(i) work? O(lgn)

Q1C How do we maintain through insert/delete? O(lgn)

Q2 How does RANK/SELECT work O(lgn)

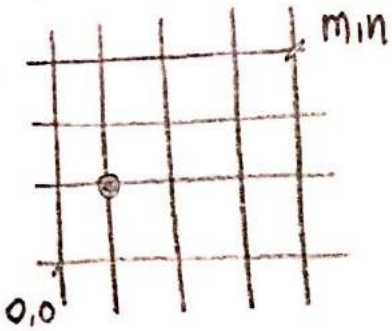
Q3 How do we maintain SIZE O(lgn)

A2 RANK(T, x)

```
IF T.VALUE=x  RETURN 1+T.LEFT.SIZE
        >x : RETURN RANK(T.Left, x)
        <x : RETURN RANK(T.right, x) +
                    1+T.Left.Size
```

DUYGU SEZEN ISLAKOĞLU LEC16
54161

Q1



Robot to move on grid from (0,0) to (m,n) only go up and right one step. How many distinct path?

- Num Paths (i,j) = (Paths $(0,0) \longrightarrow (i,j)$)

```
if i=0 or j=0
   return 1
else
   return NP(i-1,j) + NP(i,j-1)
```

- How many distinct problems ? $O(mn)$
- How much does each cost ? $O(1)$
  (assuming smaller problems solved) $\times$
  $O(mn)$

- BOTTOM UP solution?

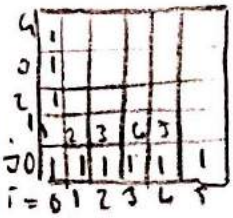


Table of answers

Q2 You have coins with values S_1 S_2 ... S_m
1kr 5kr 10kr 100kr

Make N kr with fewest coins

```
MinCoins(N):
  if N=0 : return 0
  elseif N<0 : return ∞
  else min (mincoins(N-Si)+1)
       i:1:m
```

comes from the table

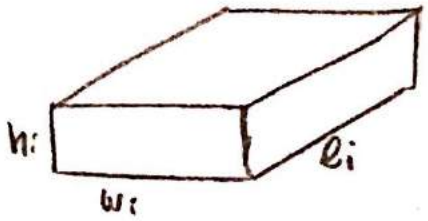
(Try all possible coin types as your first)

COST : # unique subproblems : N (unique recursive calls)

#merge cost : m

TOtal cost = m x N

Q3 RECTANGLE BLOCKS



Block i has l_i, w_i, h_i

Cannot rotate blocks

Larger blocks cannot go on smaller blocks

Can put j on i if $l_j \leq l_i$ $w_j \leq w_i$

Given a set of block measurements, what is the max height of a legal stack?

RB(1:n) #max height achieved legally using blocks 1...n

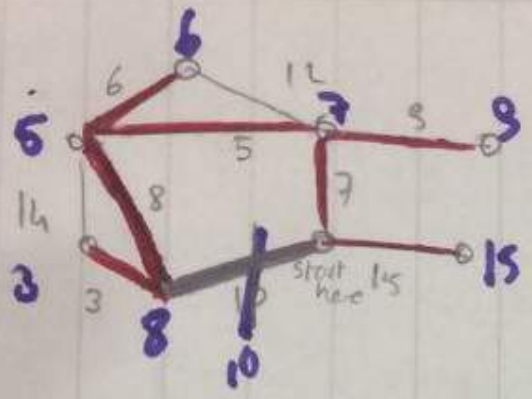
Base case : HINT : c[i] : set of blocks compatible with i.

⋮

RB(2..n) Recursive case

Q1 Solve with Prim:

DUYGU SEZEN ISLAKOĞLU
54161 LEC
LEC 21 21

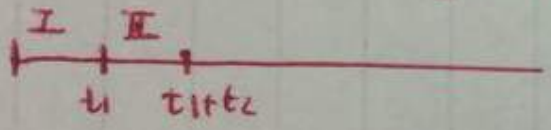


Q2 Complexity of Prim

$$\text{Time} = O(V) \times \underbrace{T_{\text{extractMin}}}_{O(\lg V)} + O(E) \times \underbrace{T_{\text{decreasekey}}}_{O(1)\ \text{Fibonacci Heap}}$$

Q3 A computer with n processes with times $t_1, \ldots, t_n$. You have to pick the order s.t. the average completion time is minimized.

$$C_n = \sum_{j=1}^{n} t_j$$



I assume that shortest job first is a solution.
Assume there is a solution other than that.
$C_n = t_1 + (t_1+t_2) + \ldots = t_1 \cdot n + t_2(n-1) + \ldots$
So swap t_1 with smaller one and apply to all.

Q4 You have a calender with n events with start time S_i, end time f_i. They may overlap & you need to attend all, so you clone yourself. What is the min # clones ?

max_overlapping − 1 will work. Assume there is more than that. So you won't need more than max_overlapping - 1 clones

Q5. You have m types of metals where type i has value S_i per kg and there is n_i kg of it total

Knapsack

Construct N $ worth of metals with least possible weight.

Start with the more valuable one. Then consider other starting points. Choose the best one.

Duygu Sezen Islakoğlu
B4161

LEC 22

Q1) $f(X,X) = ?$ 0

$f(X,Y) = ?$ in terms of $f(Y,X) = -f(Y,X)$

$f(X \cup Y, Z) = ?$ " " $f(X,Z), f(Y,Z)$

$X \cap Y = \emptyset$

$f(X,Z) + f(Y,Z)$

Q2) What is the flow across cut in example?

$f(s,a) + f(s,b) + f(c,a) + f(c,b) + f(c,d) + f(c,t) = 4$

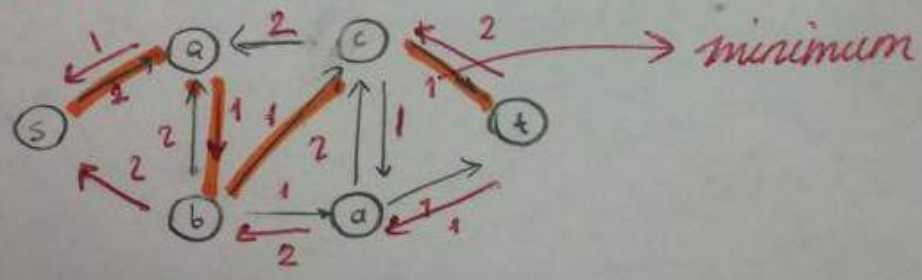
2 2 -2 0 0 2

Q3) What is $c(S,T)$?

$c(s,a) + c(s,b) + c(c,a) + c(c,b) + c(c,d) + c(c,t) = 3$

3 2 0 1 0 3

Q4) Residual Network



Duygu Seren Uslakoğlu

LEC 24

Q1.

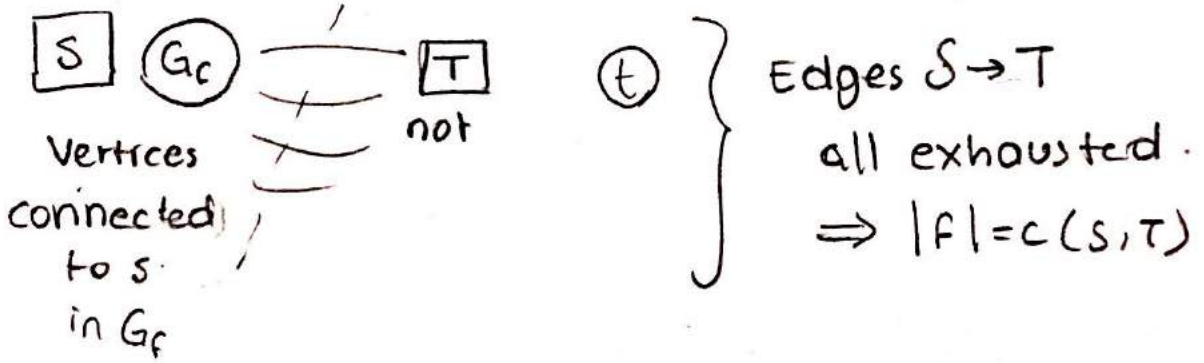
$(i \to ii)$ $|f| \leq c(S,T)$ for any (S,T)

If we find $|f| = c(S,T)$ for some (S,T)

$|f'|$ can be better.

$(ii \to iii)$ If f has any path $\Rightarrow$ f can be increased.

$(iii \to i)$



LEC 25 Duygu Sezen Işıkoğlu 54161

1) If A can be reduced to B:

a) If $B \in P \Rightarrow A \in P$

b) If $B \in NP \Rightarrow A \in NP$ (it could be P)

c) If A is NP-hard $\Rightarrow B \in$ NP-hard

(A cannot be harder than B in P/NP sense)