Lecture # 15 — Monday 21st October 2019

"Stable Matching Problem"

* Self-enforcing Process: where we take care of preferences.

"Perfect Match" everybody in one set is connected to somebody in other set.

* **Matching:** A matching 'S' is a set of ordered pairs, each from $I \times E$ with property that each member of I and each member of E appears in atmost one pair in S.

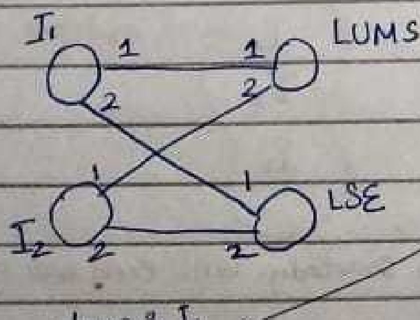
* **Perfect Matching:** S' is a perfect matching with property that each member of I & each member of E appears in exactly one pair in S'.

• We are assuming that we have same number of objects in both sets $n \times n$.

*Also,
→ Ties in preference
→ Ranking can not change once denied.

e.g. $n = 2$

I_1 (Intern 1) : LUMS, LSE (1, 2) LSE: I_1, I_2 (1, 2)
I_2 (Intern 2) : LUMS, LSE LUMS: I_1, I_2



• One possible perfect match is ~~$S = \{(I_1, LSE), (I_2, LUMS)\}$~~
$S = \{(I_1, LUMS), (I_2, LSE)\}$

~~S7~~

if LSE offers extra scholarship to I_1, I_1 will not change preference (self-enforcement) becoz its first preference is LUMS.

If I_2 offers money to LUMS, LUMS will not change preference.

• lums & I_1 ~~both prefer each other~~ so No instability

* Matching 'S' is stable, if,
  -> it is perfect
  -> there is no instability w.r.t S

$S_2 = \{ (I_1, LSE), (I_2, LUMS) \}$

$(I_1, LUMS)$ is an instability w.r.t S_2. becoz it does not belong to S_2.

* (I_x, E_y) is an instability w.r.t a matching 'S', if I_x prefers E_y to it's current partner and E_y prefers I_x to its current partner.

=> GALE-SHAPLEY ALGORITHM:

INTERNS

* I_1	~~E_C~~ Day 2	~~E_B~~ Day 3	E_E	E_A	E_D
I_2	~~E_A~~ Day 1	E_B	E_E	E_C	E_D
I_3	E_D	E_C	E_B	E_A	E_E
I_4	~~E_A~~ Day 1	E_C	E_D	E_B	E_E
I_5	E_A	E_B	E_D	E_B	E_C

EMPLOYERS

* E_A	I_3	I_5	I_2	I_1	I_4
E_B	I_5	I_2	I_1	I_4	I_3
E_C	I_4	I_3	I_5	I_1	I_2
E_D	I_1	I_2	I_3	I_4	I_5
E_E	I_2	I_3	I_4	I_1	I_5

$n=5$

Employers	Day 1	Day 2	Day 3	Day 4
E_A	$\cancel{I_2}, \cancel{I_4}, I_5$	I_5	I_5	I_5
E_B		I_2	$\cancel{I_1}, I_2$	I_2
E_C	I_1	$\cancel{I_1}, I_4$	I_4	I_4
E_D	I_3	I_3	I_3	I_3
E_E				I_1

Stayed back becoz only employees

becoz favoured by E_C

-> Stable matching

* What is the "Progress Measure"? At end of day, somebody will cross out a candidate.
  ~~Degree~~ no. of cross-outs

- It will take atmost $O(n^2)$ $n \times (n-1) \approx n^2 - n$

① Show G-S Algo produces a perfect matching.

⇒ <u>Proof by Contradiction</u>: Assume G-S does not produces a perfect matching.
- Some intern I which is rejected by every employer.
- Every employer has a better candidate than I.
- contradiction.

② Show G-S algo produces a <u>stable</u> matching.

⇒ <u>Proof: by Contradiction</u>: (I, E) is a pair not in S.
Show (I, E) is not an instability

<u>Case I</u>: Employer E must have rejected I.

case II: I have interviewed at E. ← I went to E' which is preffered over E.

- Article in Tardos

16 was mid :-

lecture 17

Monday
28th Oct 2019

I_1	E_2	→ if we do not take preference of employees, then we will get a stable match, but it includes Unfairness.
I_2	E_3	
⋮	⋮	
I_n	E_1	

* In a Bipartite graph we are sure to get a stable match.

* Divide and Conquer:

• Collaborative Filtering:

→ Similarity metric.

n set of movies 'n'	M_Z	M_A	M_B	M_Y	M_P
Person x	1	2	3	4	5
Me	3	1	2	5	4

← sorted rating of person x.

a_1 . . . . a_n ← 'a' numbers.

"Inversions" $a_i < a_j$, where $i < j$

suppose a_1 .... $\underbrace{\overset{i}{5} \; ... \; \overset{j}{2}}_{\text{inversion}}$ ... a_n

The inversion is w.r.t person x. (becoz person x's data is sorted)

How many inversions in above example? Easier way to check it:

"No. of crossings or intersections are inversions in our data"

→ There are 3 inversions; (1,3), (2,3), (4,5) aesthatic pan

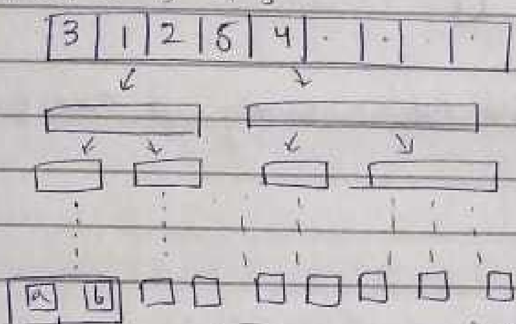
my algo is better than yours

* Brute Force will take $O(n^2)$ to do this task. We can reduce this time to $O(n.\log n)$ by using divide and conquer.

e.g

indices is given ← 1 2 3 4 5

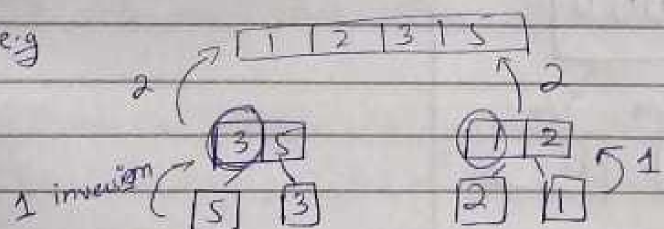
it's data



compare these if $b < a$ then there is inversion, just deal with it

→ do this for all $\underline{n}$ things.

e.g



→ take heads & compare.

1 is less than 3

So 1 has inversions equal to the no. of elements in other array (whose head is 3)

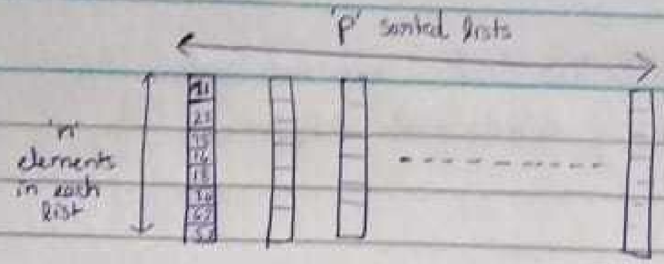
in this case, it's 2.

similarly for 2 it's 2.

Total inversions $= \underline{6}$

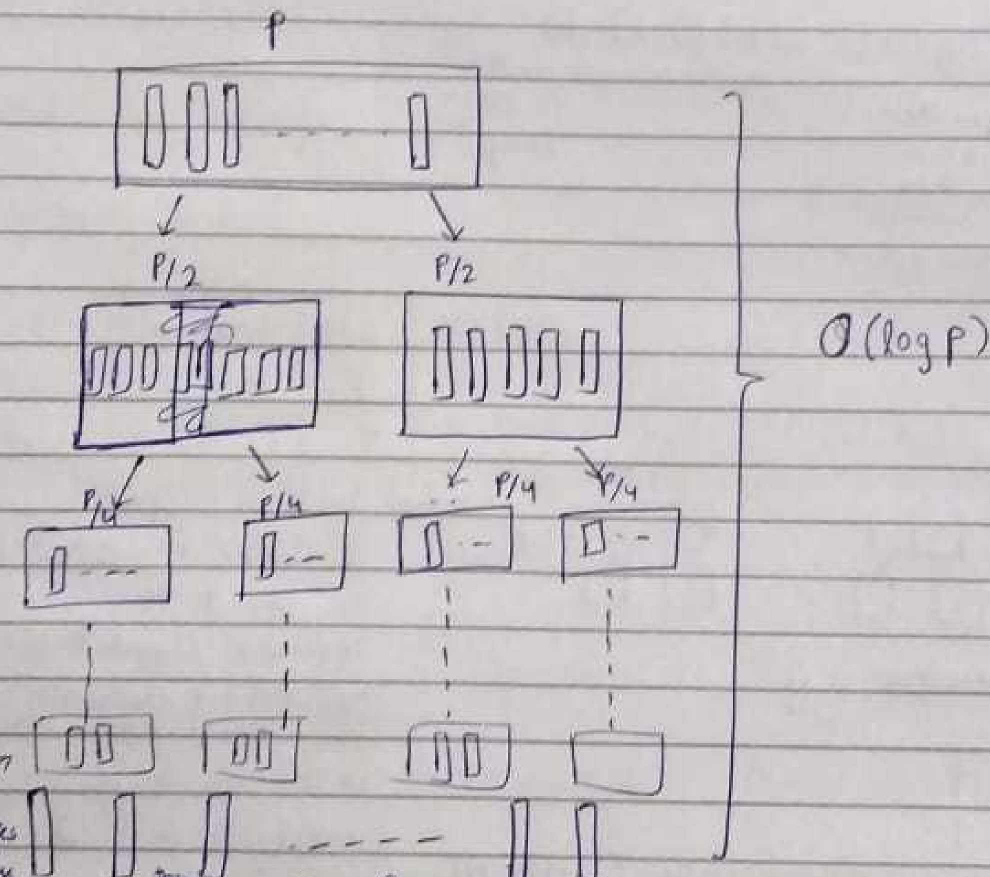
↑

Takes linear time $O(n \log n)$



→ We want to make a big sorted list containing all these p sorted lists.

• size would be "pn"



P/2 merges, each merge takes 2n so ~~each~~ level takes $P/2 \times 2n = pn$

each level takes np so total time is $\boxed{O(np \log p)}$

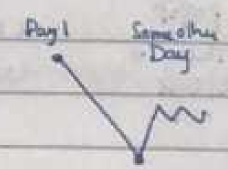
Lecture 18 — 30th October 2019, Wednesday

* Data that has fluctuations:

e.g. Stock price fluctuations

S_1 = Min Stock Price
S_2 = Max Stock Price

• Brute force time complexity $= O(n^2)$.



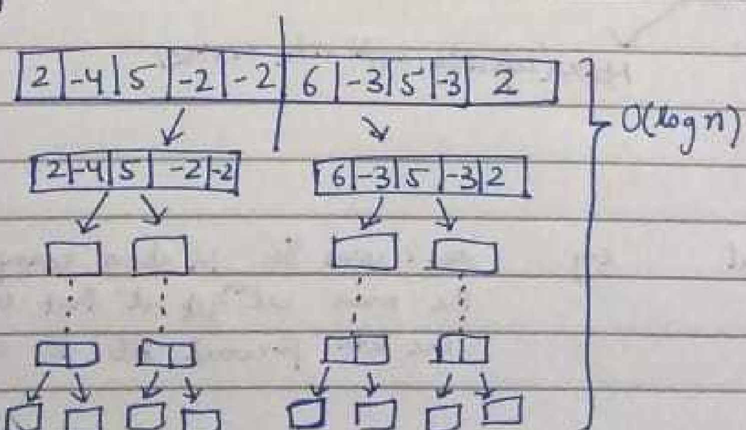
e.g. Price = { . . . . }
Change = {2, −4, 5, −2, −2, 6, −3, 5, −3, 2}

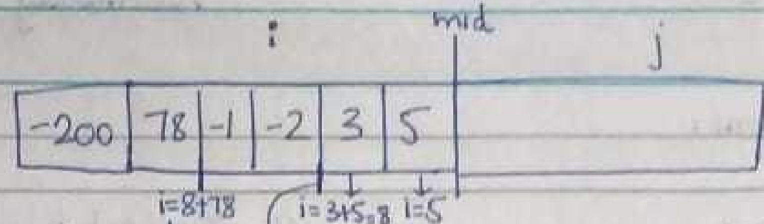
• if we are given set of prices & changes in price i & j where i < j then either we can get the difference of our extreme prices as our profit; and if we are looking at changes then we have to add all of the changes from max price to min price to get our profit.

→ In this problem, we are picking the two ~~prices~~ Days (one → when will we purchase it and one → when will we sell it) and therefore 2 prices

• Divide and Conquer will give us better than brute force but not the best. i.e $O(n \log n)$ but we can do this in linear time as well.

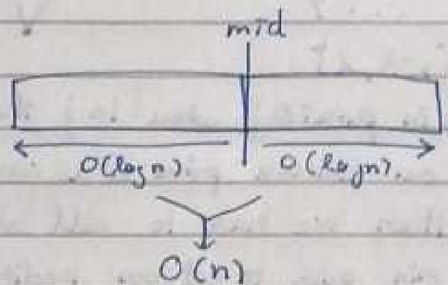
Array for Changes:

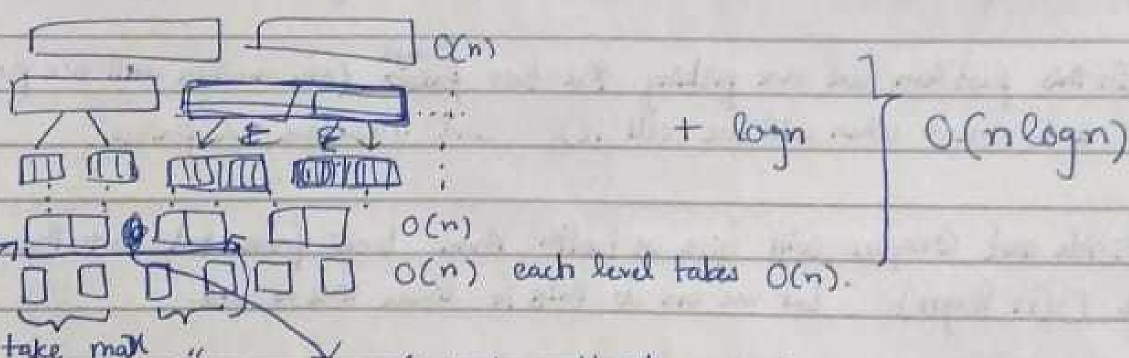




update the value of profit if there is a increase else not.

after this point I am not sure that whether my profit value would increase or decrease. so I just place a marker here (and will ~~stop~~ stop updating my value) and continue





take max of this "

and this would answer to above level

↓

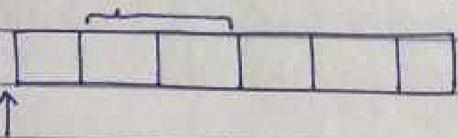
this continues for each level

Max (leftside, right side, across)

e.g. our i was 86 in above example & that's the max we can get at that stage so we will forward 86 to above level.

* <u>In linear time</u> :

some -ve values → we will not add them in our max sum . . . ignored!



Start from one end.

→ Max sum, → that is being updated only if we have increase in value of profit
→ Current sum, → adds up all the values

It will take $O(n)$.

* Recurrence Time with <u>Substitution</u>:

$$T(1) = b$$
$$T(n) = T(n/2) + cn$$

$$T(m) = T(m/2) + cm$$ → basic template

where $m = n/2$

$$T(n/2) = T\left(\frac{n/2}{2}\right) + \frac{cn}{2}$$
$$T(n/2) = T(n/4) + \frac{cn}{2}$$

now $m = n/4$

$$T(n/4) = T\left(\frac{n/4}{2}\right) + \frac{cn}{4}$$
$$T(n/4) = T(n/8) + \frac{n}{4}c$$

$$T(n) = \underbrace{T(n/2)}_{\text{put value of}} + cn$$

$$T(n) = \underbrace{T(n/4)}_{\text{put value of}} + \frac{cn}{2} + cn$$

$$T(n) = T(n/8) + \frac{cn}{4} + \frac{cn}{2} + cn$$

$\vdots$

$$T(n) = \cancel{T(n/j)}\ T\left(\frac{n}{2^j}\right) + \left[\frac{cn}{2^{j-1}} + \ldots + \frac{cn}{2} + cn\right]$$

$$'' = T(1) + cn\left(\frac{1}{2^{j-1}} + \cdots \frac{1}{2} + 1\right)$$

$$'' = cn\left(\frac{1-1/2^j}{1-1/2}\right) + b$$

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

$$'' = 2c\cancel{n}\left[\frac{n-1}{\cancel{n}}\right] + b$$

$$'' = 2cn - 2c + b$$

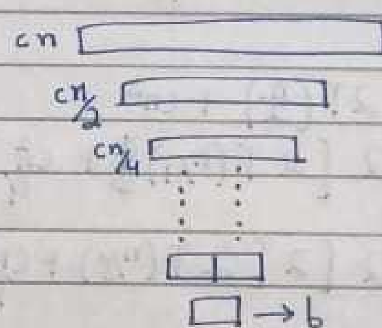
$$T(n) = O(n)$$

$$T(n) = 2c(n-1) + b$$

$$T(n) = cn + \underbrace{cn - 2c} + b$$

The first level takes almost half of the time and

rest take lesser & lesser time as we go down



this thing is called 'Decaying Recurrence Time'

or "Decaying Geometric Series"

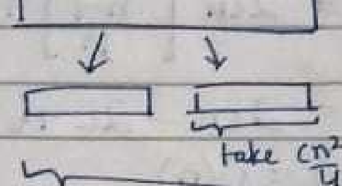
* e.g. $T(n) = 2T\left(\frac{n}{2}\right) + cn^2$

$T(2) = b$

$T(m) = 2T(m/2) + cm^2$ where $m = n/2$

$T(n/2) = 2T\left(\frac{n/2}{2}\right) + c(n/2)^2$

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



take $\frac{cn^2}{4}$

takes $\cancel{1}2 \times \frac{cn^2}{\cancel{4}2} = \frac{cn^2}{2}$

$m = \frac{n}{4}$

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n/4}{2}\right) + c\left(\frac{n}{4}\right)^2$

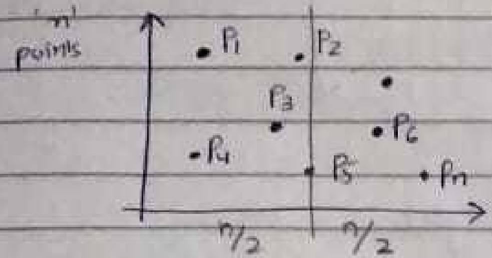
$T(n/4) = 2T(n/8) + \frac{cn^2}{16}$

$T(n) = 2T\left(\frac{n}{2}\right) + cn^2$

$T(n) = 2\left[2T(n/4) + \frac{cn^2}{4}\right] + cn^2$

$T(n) = 2\left[2\left[2T(n/8) + \frac{cn^2}{16}\right] + \frac{cn^2}{4} + cn^2\right.$

4th Nov. 2019
Monday



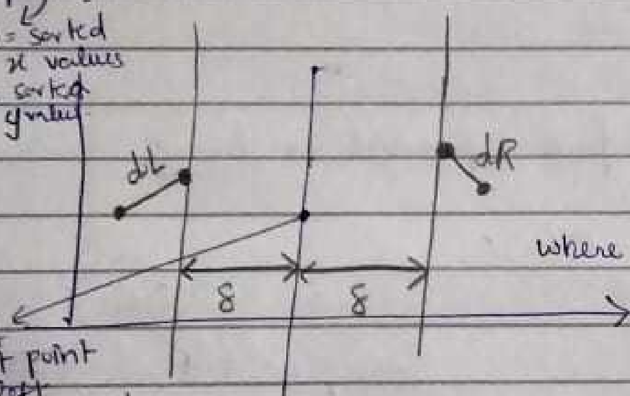
By brute-force : $O(n^2)$

Choose points p_i, p_j such that we minimize the Euclidean dist. (P_i, P_j).

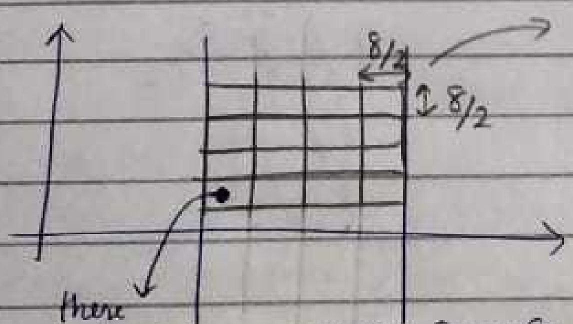
- We want to reduce the time to $O(n \log n)$. (firstly)

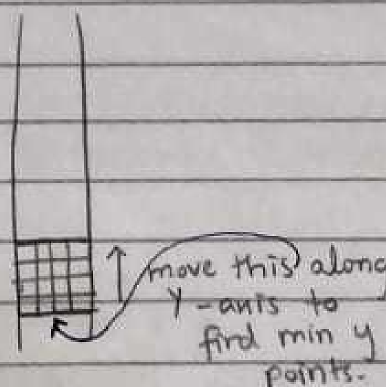
for that we need our combining operation to be of $O(n)$.

$p(x,y)$
p_x = sorted x values
p_y = sorted y values



where $\delta = \min(d_L, d_R)$

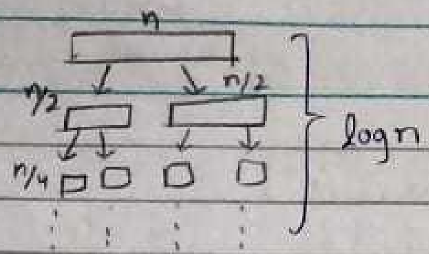


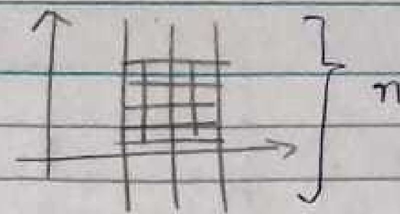


move this along Y-axis to find min y points.

there can be just one point in $\delta/2 \times \delta/2$ square.

because if we will have 2 points in one square (then their length would be lesser than δ → contradictory thing





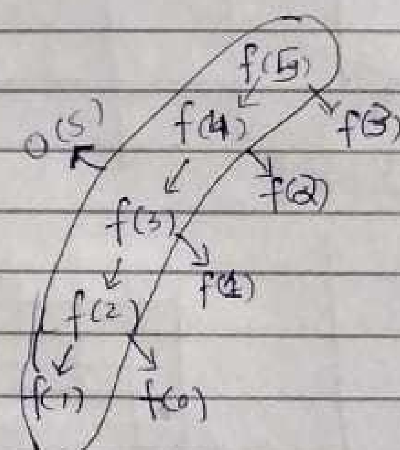
∴ total time is $n \times \log n \approx O(n \log n)$

record
Memo

* **Memoization** : ⟶ reducing time from exponential to linear

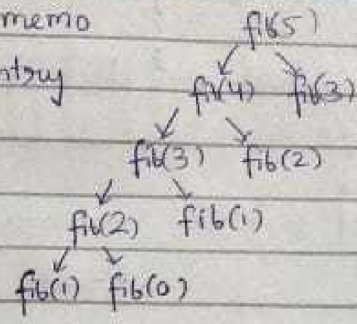
For fibonacci : Memo filling needs $O(n)$

Total time is also $O(n)$.



lecture 20 — Wednesday 6th November 2019

*Running time = Number of entries in the memo
× time to compute each entry



time for each entry

i.e $n \times O(1) = O(n)$

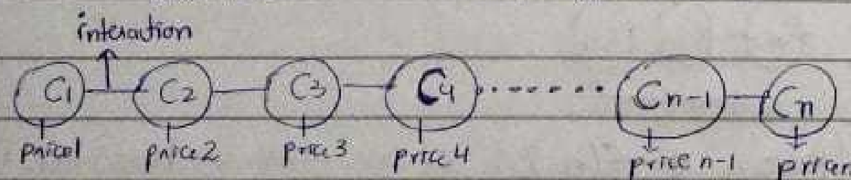
no. of entries

*Dynamic Programming: Problems that can be tackled using DP have two characteristics:

① Optimal Substructure
② Overlapping sub problems.

① "Ordering" of sub problems — so that I can combine my smaller problems to get a bigger
② Recurrance relation

e.g 'n' chemicals — Graph Type: Undirected Chain Graph



We have to choose a ^sub^ set of chemicals that have no interaction and we ~~will~~ ^want to^ get the max price. (sum of prices to be max)

They have to be placed in only 1 box.

$S =$ Subset of chemicals such that $\sum_{x \in S} \text{prices}_x$ are maximised.

one sol proposed was choose prices at odd & even indices.
$\{C_1, C_3, C_5 \cdots\}$ and $\{C_2, C_4, C_6 \cdots\}$

↓

Not correct → Counter example

1 2 3 4
(5)—(2)—(2)—(10)

odd even
(5)—(10) gives max price = 15

odd odd
(5)—(2) = 7 → not max

even even
(2) (10) = 12 → not max

$\overbrace{C_1 - C_2 - C_3 \cdots\cdots C_n}^{I}$

* By using "Brute Force" — trying all possible subsets, we will get
Power Set $(I) = 2^n$

* DP would be an "Intelligent Brute Force"

$C_1 - C_2 - C_3 \cdots\cdots C_{n-2} - C_{n-1} - C_n$

←

$$f(n) = \max \begin{cases} \text{price}_n + f(n-2) & \leftarrow \text{case where we have } C_n \text{ includ} \\ f(n-1) & \leftarrow \text{when we } \underline{\text{DO NOT}} \text{ have } C_n. \end{cases}$$

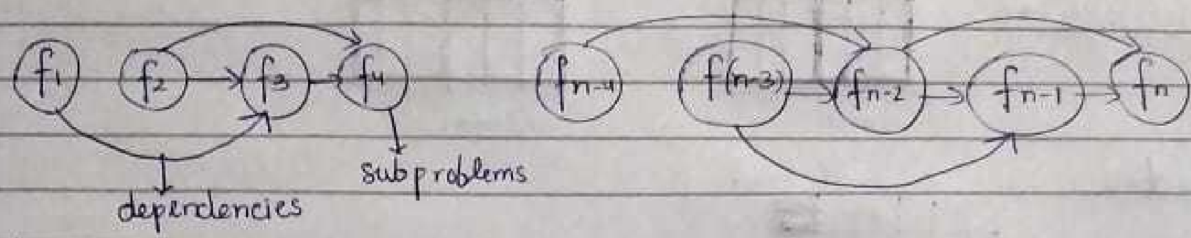
$$f(n-2) = \max \begin{cases} \text{Price}_{n-2} + f(n-4) & \leftarrow \text{compare} \\ f(n-3) \end{cases}$$

⋮

Still will give $O(2^n)$ → we will do memoization to get $\underline{\underline{O(n)}}$!!

* Base case would be $f(0) = 0$

$f(1) = price_1$



↑DAG → implicit

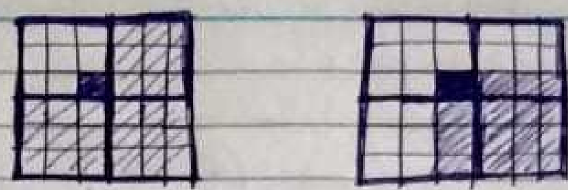
$O(n)$

11th November 2019
Monday

Lecture 21

hw solution

$T(n) = 3T(n/2) + c$



greater lesser.

<u>liner time</u> :

* How do we find set of n chemicals?

$n=5$

e.g

Memo [0]	Memo [1]	Memo [2]	Memo [3]	Memo [4]	Memo [5]
0	1	8	8	9	15

code on slide

It takes $O(n)$

* Overlapping Subproblems: when a recursive algo, re-visits the same stage/step again and again. pbl

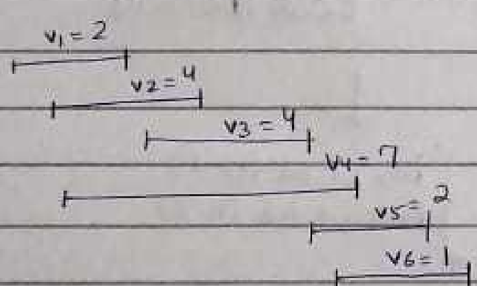
① What are your subproblems? → normally of polynomial size, should NOT be too large e.g in exponential
② Define an ordering.
③ Write recurrence relation.

* 3 steps to Dynamic Programing:

① Define the recurrence relation to get your answer.
② Keep your space small → do not use larger no. of subproblems.
③ Specify the order.

$f(0) - f(1) \cdots\cdots f(n)$

* Weighted Interval Scheduling:



$p(1) = 0$ ← number of classes that are compatible with 1 & comes earlier than 1.
$p(2) = 0$
$p(3) = 1$
$p(4) = 0$
$p(5) = 3$
$p(6) = 3$

$$f(n) = \max \begin{cases} V_n + f(p(n)) & \to \text{if } V_n \text{ included in my final answer} \\ f(n-1) & \to \text{if not} \end{cases}$$

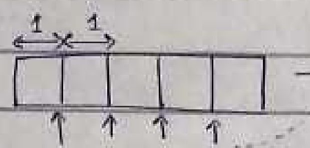
* Base case: $f(0) = 0$, $f(1) = v_1$

→ That would be an exponential algo, so do memoization to do it in linear time i.e $O(n)$
• with sorting $(O(n \log n))$.

* Rod Cutting Problem :

We are given rods of length 'n' inches, and prices relevant to the lengths

e.g. $n = 5$

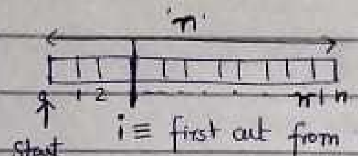


→ we can cut it at 4 places

→ 2^4 possible ways to cut.

There $\boxed{2^{n-1}}$ ways to cut the rod.

→ Goal is to maximize the revenue.



$i \equiv$ first cut from LHS (left hand side)

length of rod

$$\text{cut}(n) = \text{Max}\{\text{price}_i + \text{cut}(n-i)\} \quad (1 \le i \le n)$$

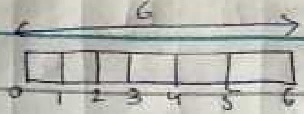
$$\max \begin{cases} \text{Price } 1 + \text{cut}(n-1) \\ \text{Price } 2 + \text{cut}(n-2) \\ \vdots \\ \vdots \\ \text{Price } n + \text{cut}(n-n) \end{cases}$$

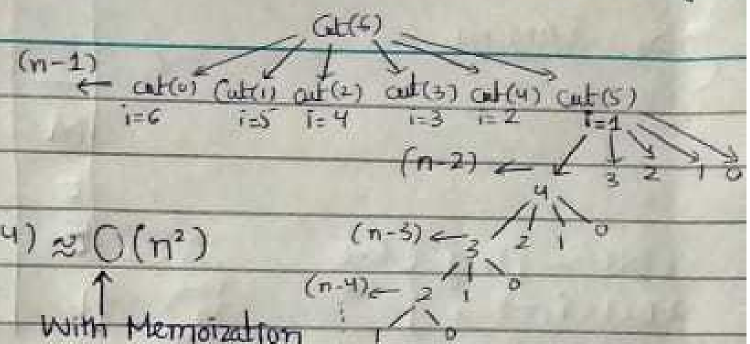
Base case: $\text{cut}(0) = 0$

Lecture 22

13th November 2019
Wednesday

e.g. $n=6$





$(n-1)+(n-2)+(n-3)+(n-4) \approx O(n^2)$

↑ With Memoization

$O(n^2)$ because at each stage we are taking max of $n-1$ memos.

What does implicit DAG look like?

here $n-2$ edges are going

here $n-1$ edges are going.

*[Figure out the no. of subproblems and then relate time to compute each subproblems] ← to get time complexity

* EDIT DISTANCE:

Given 2 strings: cost function is how many changes I need to transform one string to another.

Separate / Separete ← substitution — 1 change require

for → four ← insertion — 1 change

your → you ← deletion

write } needs more
right } than one change

"TYPES OF CHANGES"

$\rightarrow\downarrow$ $2+1=3$

$\downarrow\rightarrow$ $1+2=3$ $\underbrace{\quad}_{\text{min is } 2}$ $\searrow$ $1+1=2$

$j \longrightarrow$ insertion

C1 Null string

eg

S_A = LAMENT

S_B = MEET

R1

	_	L	A	M	E	N	T
_	0	1	2	3	4	5	6
M	1	1	2	2	3	4	5
E	2	2	2	3	2	3	4
E	3	3	3	3	3	3	4
T	4	4	4	4	4	4	$\boxed{3}$

$\downarrow i$ deletion

$\rightarrow$ min cost to change SA to SB

we can reach here by 3 ways.

$$\left.\begin{array}{l} EDIT(-, j) = j \\ EDIT(i, -) = i \end{array}\right\} \text{Base case}$$

$EDIT(i, j) = ?$

* $\searrow$ diagonal represents substitution
* $\rightarrow\downarrow$ represents one insertion and then one deletion
* $\downarrow\rightarrow$ represents one deletion and then one insertion

for R1 converting null to null will take 0 change
converting null to L will take 1 change (insert)
converting ~~..~~ null(–) to –LA will ~~..~~ take 2 changes and this goes on.
converting null to –LAMENT takes 6 changes

for C1 converting M to Null takes 1 change
converting –ME takes 2 changes and so on.

given,
- INSERTION COST = 1
- DEL. COST = 1
- SUBS. COST = 1

$$EDIT(i,j) = \min \begin{cases} EDIT(i-1, j-1) + SUB\_COST, & \leftarrow \text{diagonal} \\ EDIT(i, j-1) + INS.\ COST, & \leftarrow \ j{-}1 \ \ j \\ EDIT(i-1, j) + DEL.\ COST & \leftarrow \ i{-}1,\ i \uparrow \end{cases}$$

$\downarrow$

missing one thing e.g

	_	A	C
_			
A	0	1	
B	1	$\boxed{\ }$	

$\longrightarrow$ it would be zero

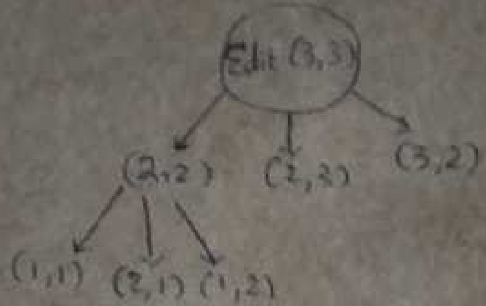
so we need to cater the case where we ~~need~~ have same characters

• introducing MATCH COST = 0

$$EDIT(i,j) = \min \begin{cases} EDIT(i-1, j-1) + \begin{cases} 0 \text{ if } S_A[i] == S_B[j] \\ \text{else SUBS. COST} \end{cases} \\ EDIT(i, j-1) + \cancel{\begin{cases} 0 \text{ if } S_A[i] == S_B[j] \\ \text{else INS. COST} \end{cases}} \\ EDIT(i-1, j) + \begin{cases} 0 \text{ if } S_A[i] == S_B[j] \\ \text{else DEL COST} \end{cases} \end{cases}$$

Not sure!

• What's the time complexity for this?

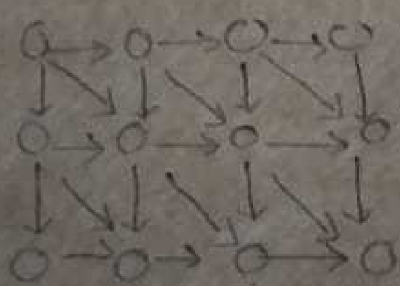


at every stage the time to add new cost is constant time.

$|S_A| = n$
$|S_B| = m$

$O(n \cdot m)$

Implicit DAG



18th November 2019
Monday

lecture 23 ①

used in comp. biology

* **Longest Common Subsequence: (LCS)**

$S_1 = a\ c\ b\ g\ h\ a\ f$
$S_2 = c\ g\ a\ b\ g\ h\ f$

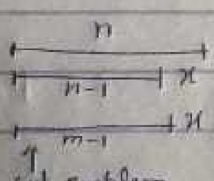
* Substrings
part of string that has no gaps b/w it.

* Subsequence
we have gaps b/w part of string.

* from S_1 one subsequence is "cb gf"
another → "c b g h f"
"a b g h f"

$|S_1| = n$ ← longest common subseq. for S1
$|S_2| = m$ ← " " " S2

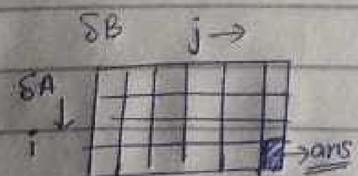
$S_A = S_{a1}\ S_{a2}\ S_{a3}$ ------ S_{an}
$S_B = S_{b1}\ S_{b2}\ S_{b3}$ ------ S_{bm}



if ~~San = Sbm~~

* Base Case:
$LCS(0,-) = 0$
$LCS(-,0) = 0$

* go with a tabular approach.



• if $S_{an} == S_{bm}$
$LCS(i-1, j-1) + 1$

• if $S_{an}\ != S_{bm}$

$$= max \begin{cases} LCS(i, j-1) \\ LCS(i-1, j) \end{cases}$$

• There are total $n \times m$ possibilities of i, j pairs.

* Reduction = use already known answer to solve new sub problem.

* Can we use EDIT DISTANCE() to find the solution for LCS?
we can change the costs, (insert_cost, delete_cost, subs_cost, match_cost)
in order to get sol. for LCS.

* In LCS, we want to maximize the match cost.
and change the subs. with delete & insert (make subs. zero.)
disallow substitutions.

* Maximum Monotonically Increasing subsequence : we can have gaps

S = 3 7 2 9 5 6 8 7

j i then $j < i$

7 9 ← one possible Monotonically ↑↑ subseq.

⇒ 2 5 6 8
⇒ 2 5 6 7

S = 3 7 2 9 5 6 8 7 → might be in our final and might not be!

MMIS (n) = ?

returns the length of max monotonically inc. subseq.

then Fill do this

eg $\boxed{2\ 7\ 3\ 8\ 4\ 5}$ 6

6 can be part of MMIS that ends at 5
" " " at 4
" " " at 3
" " " at 2
but not that ends at 7 or 8

$\boxed{1\ 7\ 2\ 3\ 8}\boxed{4\ 5}$ 6

now check for 8 instead of 6

which should I pick 8 → 3 or 8 → 7

8 → 3 will give $\boxed{1, 2, 3, 8}$ & 8 → 7 will give $\boxed{1, 7, 8}$

longer MMIS than this

$$MMIS(n) = \max_{1 \le i < n} \left\{ MMIS(i) \quad s.t\ S[i] \le S[n] \right.$$

* <u>Base Case</u>: $MMIS(0) = 0$
$MMIS(1) = 1$

Time complexity for this is $O(n^2)$

How can we use LCS( ) to find sol. for MMIS( )?

original order
S →

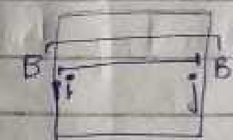
	3	9	5	6
3				
5				
6				
9				

Sorted order ↓

Lecture 24 ④ 20th November 2019
Wednesday

*Longest Palindromic Subsequence: LPS

S = C~~A~~HARACTER ($i=1$ … $i=n$)



LPS(i, j) = if S[i] == S[j]
 2 + LPS(i+1, j-1)

returns the length of LPS Longest

if S[i] != S[j]

Max { LPS(i+1, j),
 LPS(i, j+1)

Base case if i == j (only one character in S)
 return 1.

if j == i+1 (2 characters long string)
 return 1

if S[i] != S[j]
 return 1

else if S[i] == S[j]
 return 2

at this point palondromic seq of I have S "CARAC"

at this point 3 we have palondromic chars "ARA"

each column represents the subproblem S.

	j →	1	2	3	4	5	6	7	8	9
i ↓		C	H	A	R	A	C	T	E	R
i=1	C	1	1	$\boxed{1}$	$\boxed{1}$	3	5	5	5	$\boxed{5}$
i=2	H	1	1	1	$\boxed{i, j-1}$	$\boxed{x}$	3 3	3	3	3
i=3	A			1 1	$\boxed{i+1, j-1}$	$\boxed{i+1, j}$	→ 3 (2+1)			
i=4	R				1	1				
i=5	A					1	1			
i=6	C						1	1		
i=7	T							1	1	
i=8	E								1	1
i=9	R									1

→ the answer to LPS.

value at i+1 & j would be use to deduce x

back tracking

1 → 3 → 3 → 5

* But how can we construct our palindromic ~~sub~~ string after filling the table?

Back track.

R A C

5

3 → difference of 2

3

1 → difference of 2 → shows that C was added twice (at beginning & end) and A was also added twice

* **Knap Sack Problem:** Resource constrained selection problems.

Knap Sack can hold a total weight of <u>W kg</u>
'n' items of weights <u>$w_1, w_2, w_3 \cdots w_n$</u> & value <u>$v_1, v_2 \cdots v_n$</u>

Knap( ) = max value that we can achieve with the
Knap sack of capacity W and items $i = 1, 2 \cdots n$

• We need to keep track of remaining capacity. and remaining items.

$$\text{Knap}(n, w_c) = \begin{cases} \text{if } (w_n > w_c) \\ \text{Knack}(\underbrace{n-1}_{\text{go to next item}}, \underbrace{w_c}_{\text{capacity remains the same}}) \end{cases}$$

$|i_1, i_2, i_3 \ldots i_n|$
$\leftarrow$

if $(w_n \leq w_c)$ → now we can put it in our bag but we are not sure that would it go in final solution.

$$\text{Max}\begin{cases} \underbrace{v_n}_{\text{value of item added}} + \text{Knap}(n-1, w_c - w_n) & \text{if item goes inside bag} \\ \text{Knap}(n-1, w_c) & \text{if does not go in.} \end{cases}$$

<u>Base Case:</u> Knap $(0, w_c) = 0$
Knap$(i, 0) = 0$

W as <u>without</u> repitition
i.e I don't have multiple copies of same item.

7

W = 6 → capacity of Knap Sack.

Remaining capacity (W_c) ↑	i=0	items i=1, weights = 3, value = 30	i = 2, $w_2 = 3$, $V_2 = 4$	i = 3, $w_3 = 2$, $v_3 = 16$	i = 4, $w_4 = 1$, $v_4 = 9$
W = 6	0	30	34	46	55
5	0	30	30	46	46
4	0	30	30	30	39
3	0	30	30	30	30
2	0	0	0	16	16
1	0	0	0	0	
0	0	0	0	0	0

i →

path to my solution

base case row & column.

It depends on number of subproblems, and the values of Wc in each sub problem.

How?

$$\max \begin{cases} \text{Knap}(3,0) + V_4 \\ \text{Knap}(3,1) \end{cases}$$

It will take $O(nW)$ → Now my complexity depends on W e.g. we can have a very large value of W (10^{99}) and similarly very small as well W = 1. (Basically now it's not fixed)

n = no of items

W = capacity of knapsack.

"Pseudo Polynomial"

how many bits to store W? log (W) bits

e.g. n = 4 will take 2^2 bits or 2 bits

n = 8 will take 2^3 or 8 bits

$$O(nW) = O(n\, 2^{\log(W)})$$

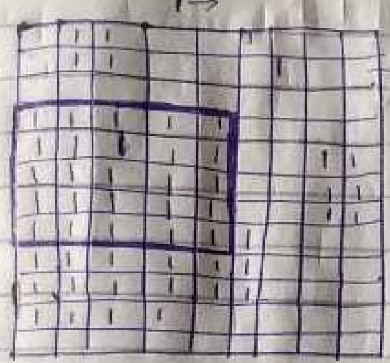
This is the time complexity based on our input.

Lecture 25 (8)

25th November 2019
Monday

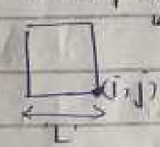
* Maximum Sized Sub Square containing all 1's :

2^{2}
2^{3}



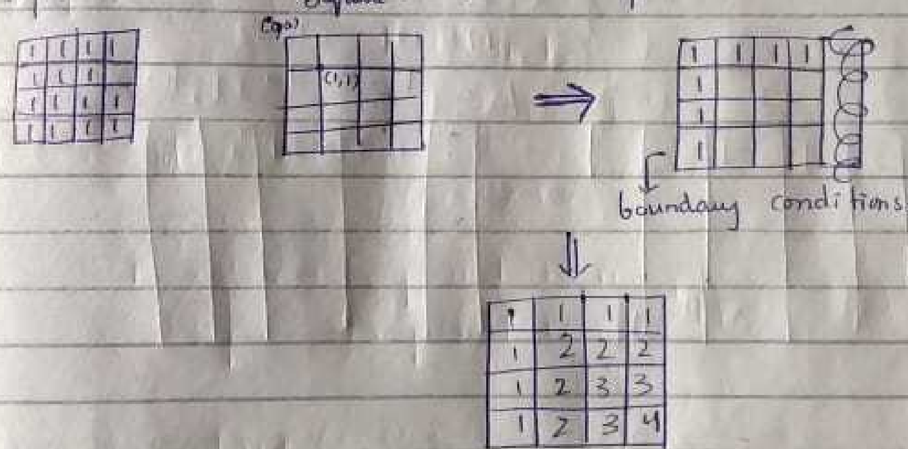
→ all the blanks are '0'

• our sub problem

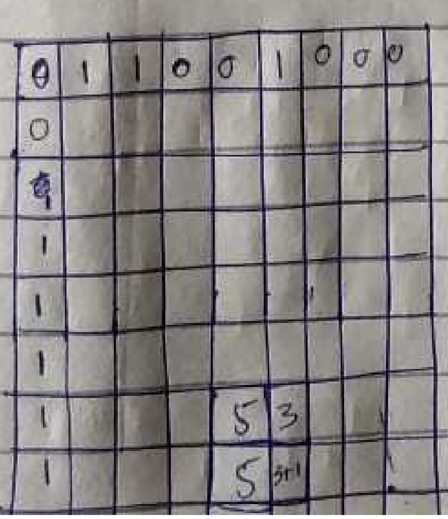


eg Input

Square ← that needs to be filled.



Back to our actual problem!
Now we'll use above example to fill in our square.



if (input [i][j] ==1)

$$\text{Square}(i,j) = \text{Min}\left[\begin{cases} \text{square}(i-1,j), \\ \text{square}(i,j-1), \\ \text{square}(i-1,j-1) \end{cases}\right] + 1$$

if input[i][j] == 0

It will take $O(n^2)$.

e.g

3	4	-1	6	

basically doing prefix sum

3	4+3	4+3-1	4+3-1+6	

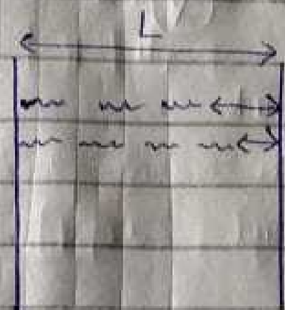
Lecture 24 (10)

"Document Layout Problem"

27th Nov. 2019
Wednesday

* Minimize Total Penality $= \sum (\text{Line Penality})^2$

$w_1 \quad w_2 \quad w_3 \quad \ldots \quad w_n$ ← words



* Line Penality = (Page Width − sum of words length of that line)

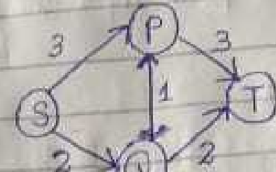
$$\text{Line}(n) = \min_{1 \le i \le n} \left\{ \underbrace{e_{in}}_{SSE} + \underbrace{C}_{\text{Penality for line segment}} + \text{line}(i-1) \right.$$

← we can use this to solve

SSE = error

"Network Flow Problems" (11)

Input: Directed graph $G(V, E)$



→ Source $S \in V$ No edges going into source

→ Sink $T \in V$ No edges coming out of sink

Integral capacity C_e $\forall$ $e \in E$ and $C_e \geq 0$

* Assign $f_e \geq 0$ $\forall e \in E$ s.t.

① $f_e \leq C_e$ $\forall e \in E$ → "CAPACITY Constraint"

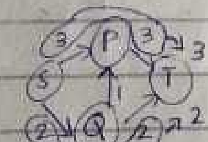
② Flow in = Flow out $\forall v \neq S, T$ → "Conservation Constraint"

(whatever goes into internal nodes, needs to get out of it)

all nodes other than source & sink are internal nodes.

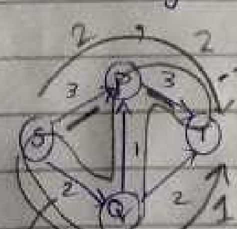
* "Objective: Maximize the flow from S to T"

greedy



→ 3+2 = 5 → maximum flow

what's the other way?



→ we also have SPT 1 out of 3 is already given by SQPT, so I am left with 2.

• in case of SQT 1 out of 2 is already given by SQPT, so we are left with 1.

out of these min. is 1 so push 1 in each edge S Q P T.

In this example, we are getting 2, 1, 1 or 2+1+1 = 4 as max flow.

Now it seems like our greedy algo. which was giving 5 was good

* **Greedy Algo:**

Initialize $f_e = 0$ for all $e \in E$

repeat

search for an s-t path such that $f_e < c_e$ for every $e \in P$

if no such path then stop with current flow f_e ($e \in E$)

else let $\Delta = \underbrace{\min_{e \in P} \underbrace{(c_e - f_e)}_{\text{room on edge } e}}_{\text{room on path } P}$

for all edges $e \in P$ increase f_e by Δ } we have Augmented the flow by Δ

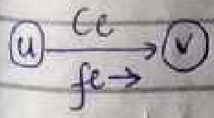
↓ e.g. in last example we changed SQT from 3 to 2 our Δ was 2 (from 3 and 2) becoz it's min.

* Greedy Picked the "wrong" path. → Implement "Undo" operation.

Ford - Fulkerson Algo:

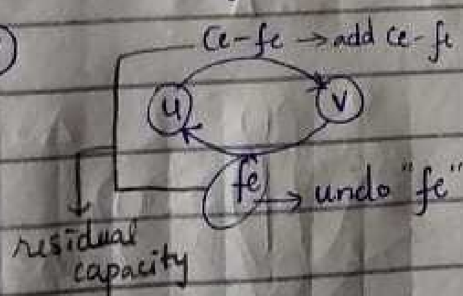
Original Graph

* $G(V, E)$



* Residual Graph → will have some extra edges which are not present in real graph G.

$G_f(V, E')$



* Ford Fulkerson ALGO (FF)

Initialize $f_e = 0$ for all $e \in E$

repeat

Search for an s-t path 'P' such that $f_e < c_e$ for every $e \in P$ in residual graph $\boxed{G_f}$

if no such path then STOP with current flow f_e $e \in E$

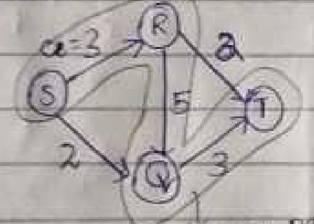
else let $\Delta = \min_{e \in P}$ (residual capacity of e in $\boxed{G_f}$)

for all edges e of $\boxed{G}$ where forward edge is in P
increase f_e by Δ

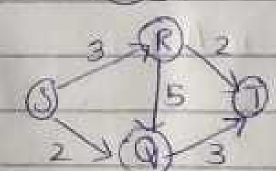
for all edges e of $\boxed{G}$ where reverse edge is in P
decrease f_e by Δ

eg

G

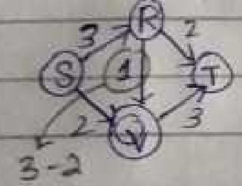


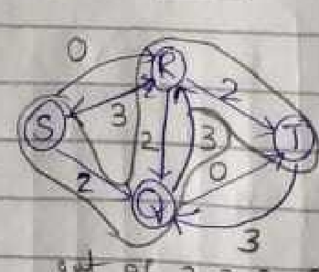
G_f



wrong path coz I can take SRT or SQT so why SRQT.

~~min. of these is 3~~ So our $\Delta = 3$

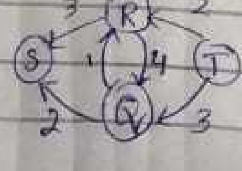




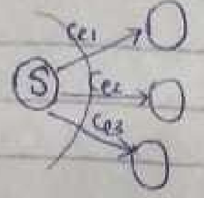
$\Delta 3 \Rightarrow P = SRQT$

$\Delta 2 \Rightarrow P = SQRT$

out of 2, 3, 2; 2 is min so $\Delta = 2$



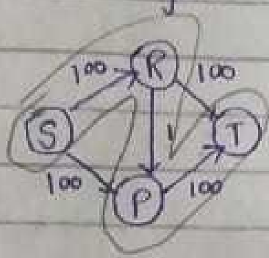
2nd December 2019
Monday



$$C = \sum_{e \in \delta(S)} c_e$$

edges coming out of S

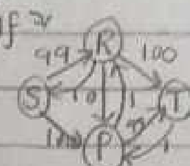
Case where FF gives a wrong maxflow?



SRPT

i.e the max flow is proportional to C but actual max flow is way lesser than this

Gf ≈



* Next time it would be "SPRT" and then again SQ, RT and this will go on till we get 0 on S→R and P→T / Q→T

Maxflow = 200

Time complexity = $O\left(C \cdot \underbrace{(m+n)}_{\text{for BFS or DFS}}\right) \approx$ ~~O(C~~ or $O(C \cdot m)$

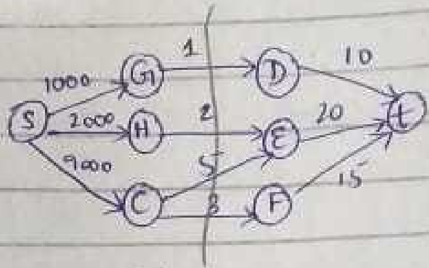
This a "pseudo polynomial" complexity because it depends on the value of our capacities.

* Edward Karp Alg (built on basis of FF) uses the path which has min. no. of hops. (from S to T)

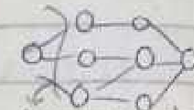
* Maximum Flow - Min cut theorem:

* S-t cut An s-t ~~g~~cut of graph $G(V, E)$ is a partition of V into sets A, B with $s \in A$ and $t \in B$.

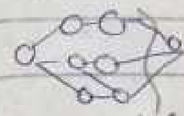
eg



S-t cut



can be a s-t cut



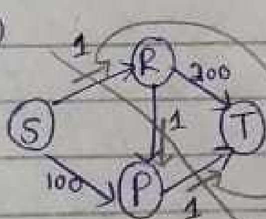
s-t cut

S-t cut can be any where but it must contain 'S' in A and 'T' in B.

* Capacity of cut = $\sum_{e \in S^{+}(A)} c_e$

edges coming out of set A

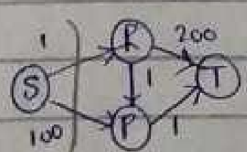
①



only two coming out of set A.

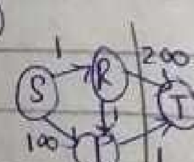
So Capacity is 2 ⟶ S→R 1, P→T 1 → 2

②



Capacity is 101

③



Capacity is 201

Maxflow for ① is 2 ... set A = {S, P} B = {R, T}

1 + 1 = 2

* <u>Theorem</u> Let 'f' be a flow in graph $G(V,E)$.
The following are equivalent (i.e all these statements hold or none of them hold)

① f is max flow of G.
② There is an s-t cut (A,B) such that the value of f equals the capacity of cut (A,B)
③ There is no s-t path (with ~~respect~~ "+ve" residual capacity) in the residual network G_f.

* <u>Proof</u>:

$$② \to ① \to ③ \to ②$$

②→① value of $f = \underbrace{\sum_{e \in \delta^+(s)} f_e}_{\text{outcoming edges.}} - \underbrace{\underbrace{\sum_{e \in \delta^-(s)} f_e}_{\text{incoming edges}}}_{\text{zero} \to \text{ becoz no edge coming in S or towards S.}} \longrightarrow ①$

$$\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = 0 \quad \text{for every } v \neq s,t \longrightarrow ②$$

Add ① & ②

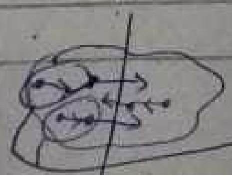
$$\text{value of } f = \sum_{v \in A} \left(\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e \right) = \sum_{e \in \delta^+(A)} f_e - \underbrace{\sum_{e \in \delta^-(A)} f_e}_{\geq 0}$$

must be +ve number or zero.

$$\sum_{e \in \delta^+(A)} f_e \;\Big\downarrow\; \leq \sum_{e \in \delta^+(A)} c_e$$

f_e cannot exceed the capacity

$$\sum_{e \in \delta^+(A)} f_e - \sum_{e \in \delta^-(A)} f_e = \sum_{e \in \delta^+(A)} c_e \quad \leftarrow \text{capacity of cut } (A,B)$$

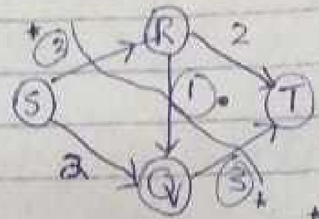


the edges that are only in A or only in B are these are not involved in calculation of value of flow.

LAST CLASS :

4th December 2019
Wednesday

* Max Flow - Min Cut Theorem continued....



capacity = 6

flow = 6 - 1

* contributes to capacity

• capacity - • contributes to flow.

2→1→3→2

* (1)→(3) (not 3 implies not 1)

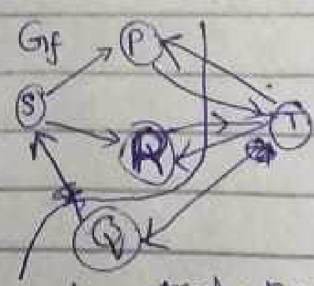
proof it by contrapositive i.e. $\neg 3 \rightarrow \neg 1$

⇒ f can be augmented so it could not previously have been the max flow.

How can I draw a min cut?

In my residual graph, I will look for the nodes that are reachable from S & these nodes will be part of Set A and they will be part of the section where we have (S).

"good example in slides"



min cut

* only P & R reachable from S.

m December 2019
Wednesday

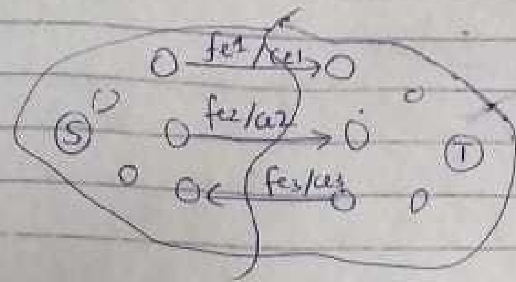
been

that

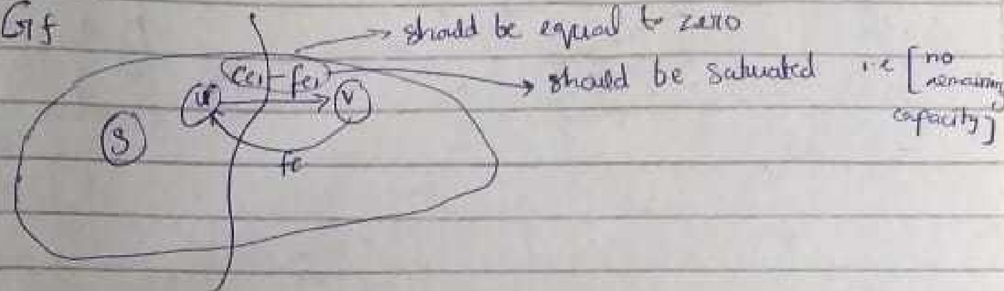
ut of

le from

* (3) → (2)



Gf

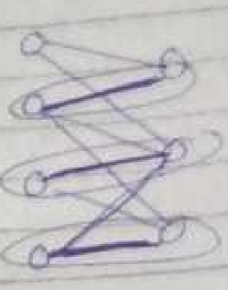


if all the edges coming out of A are saturated then we'll have no edge coming back to set A, this means that sum of ~~fe~~ f_{e} of those edges = ~~capacity~~ capacity

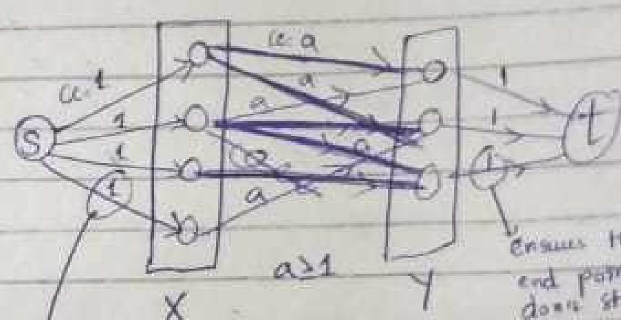
$f_e = C_e$

Applications:

* Bipartite Matching.
  Reduction!



the best match.
(we can have 3
How can we have these?

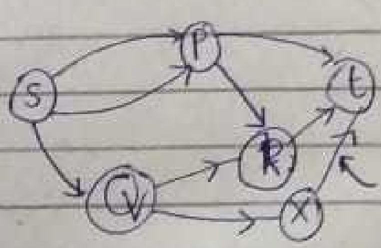


ensures that end points don't share any node.

Matching:
Set of edges where the end points don't share any node.

this shows that only 1 ~~node~~ edge [coming out of ~~these~~ x] can be selected.

* Input: $G(V,\mathcal{E})$, $s,t \in V$
  s-t edge-disjoint paths in G
  i.e no two paths share an edge, but they can share vertices

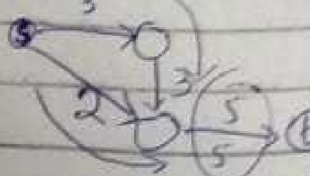


how many such paths can we have?

set capacity of each edge to 1.

How do I ensure that any edge in my path contains ~~the flow for that~~ path?

I want to avoid situations like



shared edge