Subject:  Date: 3rd Sept. 2

# * Counting $\underbrace{\text{Basic}}$ Operations:

Simplification for analysis

Constants = $C_1, C_2, C_3 \ldots$

only for loop will take n+1 time.

```
Somefunc ( )
  for ( j=1 to n)  ——  $C_1 \times (n+1)$
  {     i = j  ——————  $C_2 \times (n)$
        while (i>0) — $C_3 \times \underbrace{\sum_{j=1}^{n}}_{\text{the time for which the for loop will run.}} \underbrace{(j+1)}_{\text{1 while loop}}$
        {  i = i-1  ——  $C_4 \times \sum_{j=1}^{n} (j)$
        }
  }
```

$$T(n) = C_1(n+1) + C_2(n) + C_3\sum_{j=1}^{n}(j+1) + C_4\sum_{j=1}^{n}(j)$$

as $\sum_{j=1}^{n} j = n$

$$\text{''} = C_1(n+1) + C_2(n) + C_3\frac{(n+1)(n+2)}{2} + C_4\frac{n(n+1)}{2}$$

$$\text{''} = an^2 + bn + c \quad \text{where } a, b \,\&\, c \text{ are constants.}$$

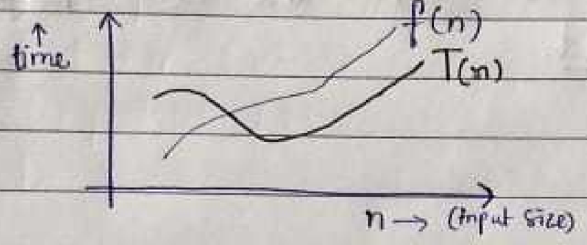
Let's say we have $T_1(n) = pn^2 + qn + r$ then $T(n)$ and $T_1(n)$ will have same

1. Get rid of lower order terms
2. Ignore leading constants.

Subject: ____________ Date: ____________

* $O(f(n))$ = $\{T(n)$ : these are constants $c>0$ , $n_0>0$ such that $0 \leq T(n) \leq cf(n)$ for all $n \geq n_0\}$

$T(n)$ is $O(f(n))$



$T(n) = an^2 + bn + c$

$T(n) < an^2 + bn^2 + cn^2$

$T(n) < (a+b+c)\,n^2$

$T(n) < c_5(n^2)$

* $T(n)$ is in $O(n^2)$

or $T(n) \underset{\substack{\uparrow \\ \text{not equality}}}{=} O(n^2)$

or

$T(n) \in O(n^2)$

e.g

$T(n) = 3n^2 + 5n + 2$ $c, n_0 = ?$

$an^2 + bn + c$

$(a+b+c)\,n^2$

$(3+5+2)\,n^2$

$(10)n^2$

$0 < T(n) \leq 10n^2$

$n_0 = 1$ or $n_0 = 2$

$3(1) + 5(1) + 2 \leq 10(1)^2$

$3+5+2 \leq 10$

$10 \leq 10$ no valichated

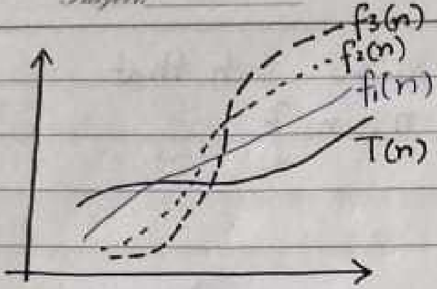
$3(2^2) + 5(2) + 2 \leq 10(4)$

$12 + 10 + 2 \leq 40$

$24 \leq 40$ no validiated

"WORST CASE ANALYSIS"

* $T(n)$ is the maximum time on any input of size $n$.



$T(n)$ is in $O(f_1(n))$

$T(n)$ is in $O(f_2(n))$

$T(n)$ is in $O(f_3(n))$

But we'll choose $O(f_1(n))$ only; this is because we want the closest / tight upper bound.

InsertionSort ( )

```
for (j=2 to n)
{
      Key = A[j]
      i = j-1
      while ( i>0  &&  A[i] > key)
      {
          A[i+1] = A[i]
          i = i-1
      }
}
```

$T(n) = 2^{2n}$   $T(n)$ in $O(2^n)$   True / [✓ false]

$T(n) = 2^n \cdot \cancel{2^n} \le c\cancel{2^n}$

$2^n \le c$

$T(n) = \log_2(n^2)$   $T(n)$ is $O(\log_2(n))$   [✓ True] / false.

$2\log_2(n) \le c \log_2(n)$

$\Rightarrow T(n) = ?$

```
func A ( int n)
{
    for (i=1 to n)
    {   funcB(n)
    }
  func C (n)
}
```

func B takes $O(\log(n))$ time

func C takes $O(n \log(n^2))$ time

func A takes $O(n \log(n))$ time

$\overbrace{n \times C_1 \cdot \log(n)}^{\text{funcB}} + \overbrace{2 \times n\log(n)}^{\text{func C}} = n\log(n)$

- All logs grow slower than polynomials
- All polynomials grow slower than exponentials.

* $n! >> 2^n >> n^3 >> n^2 \gg n\log(n) >> n >> \log(n) >> 1$

* Sort in Increasing order of growth rate:

$T_1(n) = \pi^{n-1}$ → exponential

$T_2(n) = n^4$

$T_3(n) = 4^{\pi}$ → constant

$T_4(n) = \underset{\text{Same as } n}{\underset{\downarrow}{2^{\log(n)}}}$ → linear

$4^{\pi} < 2^{\log(n)} < n^4 < \pi^{n-1}$

Subject

Wednesday
11th Sep. 2019
Date:

* $T(n) = a^2n^2 + bn + d$

$C = a+b+d$ such that $T(n) \leq C \cdot (n^2)$ but there can be another constant $C_1$ such that $C_1 \cdot (n^2) \leq T(n) \leq C \cdot (n^2)$

$\therefore$ $T(n)$ is $\underbrace{\Theta(n^2)}$ → Tight Bound

e.g. Insertion Sort    Input: any n integer numbers.

$T(n)$ is in $O(n^2)$ (in worst case)

$T(n)$ is in $\Omega(n)$ (When list is already sorted)

Q// If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n)$ is $O(h(n))$

Q// If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n) + g(n)$ is $O(h(n))$
becoz one of them will have lower growth so we will ignore that one.

* Logarithms:

$T_1(n) = \log_a(n)$    a & b are constants

$T_2(n) = \log_b(n)$

$$\log_b(n) = \frac{\log_a(n)}{\underbrace{\log_a(b)}_{\to \text{constant}}}$$ so their growth rates will be same!

le can say, $T_1(n)$ is in $\Theta(T_2(n))$ and $T_2(n)$ is in $\Theta(T_1(n))$

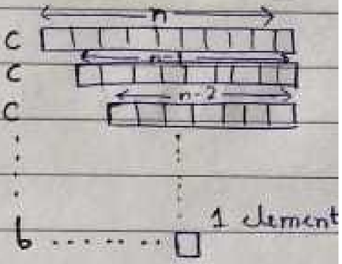
Algorithms having polynomials are the efficient.

Subject ______ Date: ______

```
            1....n
            ↑
RecSum (A, n) → will take T(n)
if (n==1) { val = A[n] ; return val}  ← Base Case → Time to execute is 'b' where b is a constant
else {
        val = RecSum (A, n-1)     } this will take
        return  val + A[n]        }   c + T(n-1)
    }
```

$T(n) = c(n-1)+b \longrightarrow T(n)$ is in $O(n)$

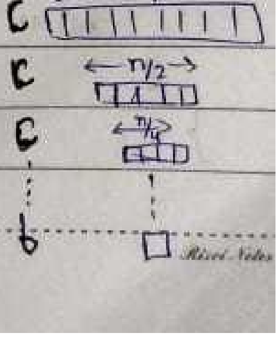
Recurrence Tree

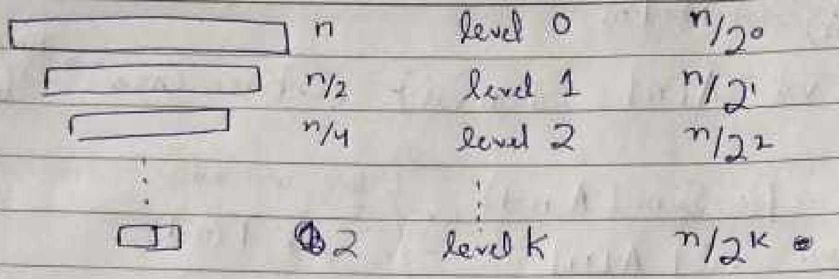


```
* BinarySearch (A[1.....n], value, low, high) → takes T(n) time
  {
      If (high < low)  return NULL; //not found  → Base Case will take 'b'

      mid = (low + high) / 2

      If A[mid] > value
          return  BinarySearch (A, value, low, mid-1)      } c + T(n/2)

      else if A[mid] < value
          return  BinarySearch (A, value, mid+1, high)     }

      else  return mid  // found  → will take 'b'
  }
```

writing sub time in te
of total time
T(

Recurrence Tree





$\frac{n}{2^k} = 2$ ↓ 2 elements.

$K = \log_2(n) - 1$

$\frac{n}{2^k} = 2$
$n = 2^k \cdot 2$
$n = 2^{k+1}$
$\log n = \log 2^{k+1}$
$\log n = k+1$
$k = \log_2(n) - 1$

T(n) is in $O(c \cdot (\log_2(n) - 1) + b)$

```
* Mystery (Array A, Start, end)
{
    if (start == end)
        return A[start]            } b as base case
    else
    { mid = $\frac{start+end}{2}$
                                     c + T(n/2) + T(n/2)
        var1 = mystery (A, start, mid)
        var2 = mystery (A, mid, end)

        return min (var1, var2)
    }
}
```

$T(n) = c\left(1+2+4+8+\ldots\frac{n}{2}\right) + (n \times b)$

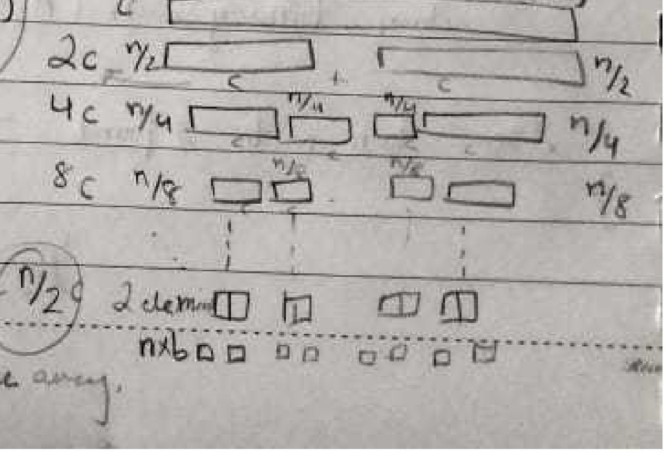
as $\sum_{k=0}^{n-1} ar^k = \frac{a(r^n - 1)}{r - 1}$

$T(n) = c\left(1+2+4+8+\ldots 2^{\log n - 1}\right) + nb$

$= c\frac{\left(2^{\log(n)} - 1\right)}{2-1} + n \times b$

$T(n) = c\,(n-1) + nb$

$T(n)$ is $O(n)$

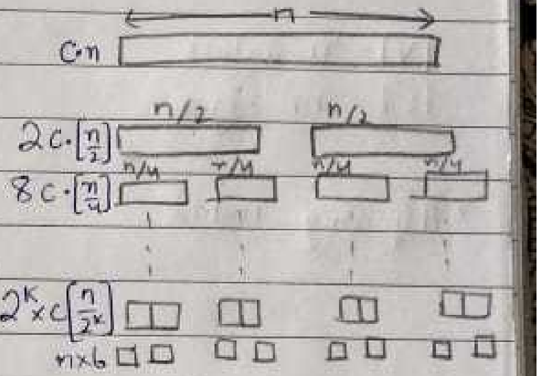


if total n elements (n size of array) so how many 2 elem arrays can we make from n size array.

Subject:

Saturday
Date: 14th Sepkmbe 2019

```
*MergeSort (A, start, end)
{
        if (start < end)   ⟶ C1        } will run
        {                              }  n times
nc          mid = ⌊ (start + end)/2 ⌋ ⟶ C2 }
+T(n/2)
+T(n/2)   MErge Sort (A, start, mid)
          Merge Sort ( A, mid+1, end)

   ⟶      return Merge (A, start, mid, end) ⟶ n·C
        }
    else
        { return A }    } 'b' constant
}
```



$\frac{n}{2^k} = 2$ elem.

$n = 2^{k+1}$

$\log n = \log 2^{k+1}$

$\log n = \log 2 \; k+1$

$k = \log n - 1$ ← ignore as const.

e.g

1, 3, 17, 23          7, 8, 18, 20

Merge will take O(n) time becoz we take each element for once, so for n elements it would take O(n).

total for

$T(n) = cn + 2c\cdot\frac{n}{2} + 4c\cdot\frac{n}{4} + \ldots + 2^k \times c \times \frac{n}{2^k} + bn$

$T(n) = c\cdot n(\log(n) - 1) + b(n)$

$= cn\log(n) - cn + bn$

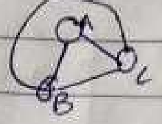
T(n) is in $O(n \log n)$

# GRAPHS :-  ⑨

Subject:  Date:

+ Graphs  $G = (V, E)$
vertex ↗ edge

→ degree of a graph

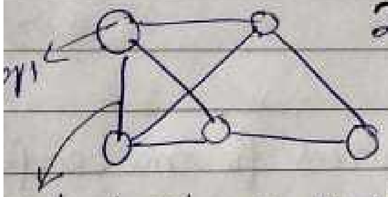


C has degree of 3

$|V| =$ 'n' vertices
$|E| =$ 'm' edges.

+ Adjacency matrix

better than matrix.
↑
+ Adjacency list
↓
by default graph exits as AL



2m+n
O(m+n)

'm' = no. of edges

2m

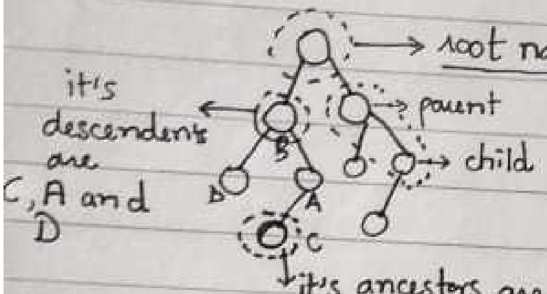
simple graph is one in which we have distinct ~~vert~~ nodes

Subject: 16th September 2019

* Notation for graphs: $|V| = n$ , $|E| = m$

⇒ Degree for undirected graph: degree $= 2|E|$

• for directed it's just E.



→ root node : Because no parent.

→ parent

→ child

it's descendents are C, A and D

it's ancestors are A, B and root as well.

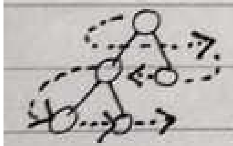
* if total nodes are 'n', then there are 'n-1' edges. [because root does not have any edge, other than that each node has an edge]

* Graph is connected, undirected, weighted graph with no cycles. How can we find the shortest path?

means it's a Tree only has one path!

⇒ As it's a tree with one path, we can use any traversal technique. BFS e.g. It's time complexity is $O(V+E)$.

• Dijkstra's Algo is for heavy weighted graphs.



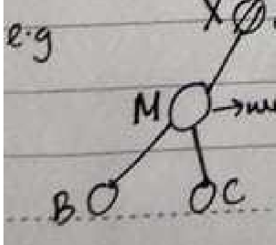
• Breadth First Search.

we do level wise traversal.

• BFS is used when we don't consider weights or weights are same.

While traversing we cross each edge twice, so it's $2m+n$

e.g



already visited node

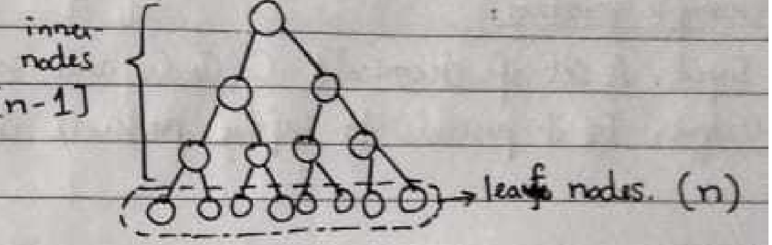
→ we are here

so M will look at all of its neighbours including X as well ∵ we have gone from the edge from X → M and M → X.

$$so * 2m+n = O(m+n) = O([n-1]+n) = O(n)$$

Subject: Date:

* Tree with 'n' leaves:



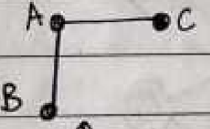
$T(n) = c \cdot (n-1) + b(n)$

* Complete Graph:

in which each pair of vertices has an edge. e.g

We have $\frac{n(n-1)}{2}$ edges in undirected comp. graphs.

* How do we find 'connected components' in an undirected graph?



just count that how many nodes are connected with A. i.e 2 we have 2 connected components.

→ we can take any vertix.

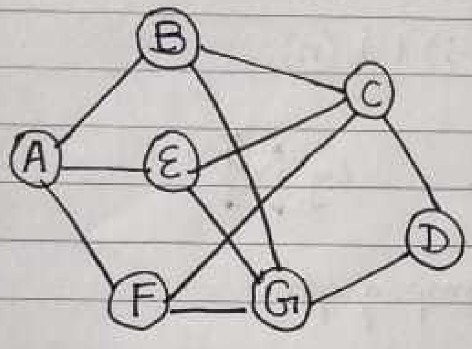
↑ comp. graph is in slides

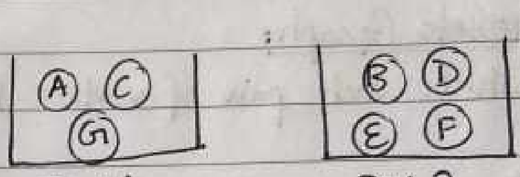
* We can use any graph traversal (BFS or DFS) for above thing.

example question:

Input: A set of chemicals $C_1, C_2, C_3$ and so. on ,and reactions among them.

Output: Is it possible to safely pack all chemicals in 2 boxes. [No chem should mix with othe

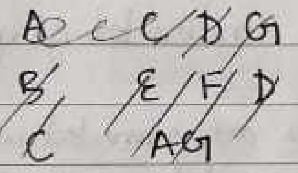


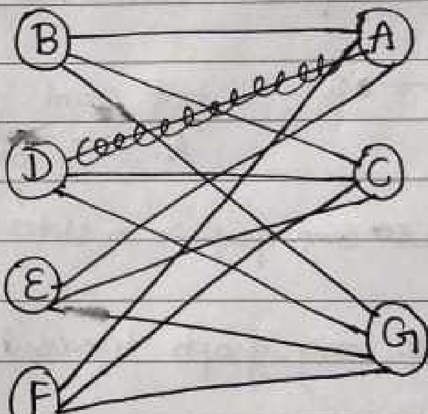


Box 1 Box 2

HOW? ☹

↓





~~EBae~~

Bipartite Graph:

s a graph $G(V,E)$ whose vertices
an be partitioned into 2 sets
$/ = A \cup B$ and $A \cap B = \emptyset$
nd there are no edges b/w vertices
f same set.

↑

$*\underbrace{\text{Bi}}\underbrace{\text{partite}}$ Graph

Two parts

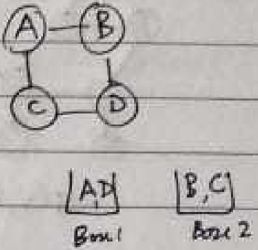
Subject:

Date:

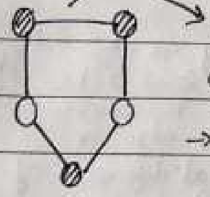
* Matching Algorithms: e.g Employers ⟷ Interns
Students ⟷ Grad Schools
Viewers ⟷ Movies

Bipartite Again 2 =̄

② A graph is bartite iff it can be coloured with 2 ~~co~~lours. (2-colo~~g~~rable graph)

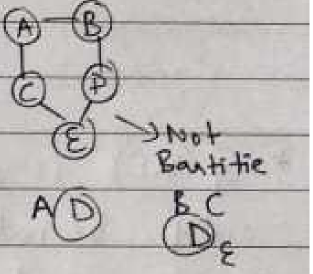


e.g



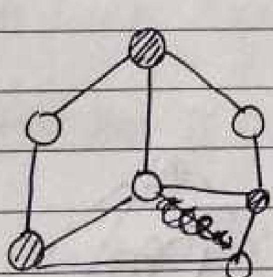
No two nodes of same colour can have an edge.

→So it's not bipartite.



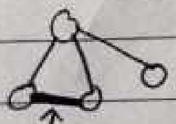
③ A graph is bipatite iff it contains no cycles of odd length.
[in other words we can't have cycle of odd number]
We can't have odd no. of edges.

e.g



* We can do BFS.
→One hop away will have same colour as root/parent.
→2 hops away will have diff. colour.

? How can we find while doing BFS that it's ~~e~~ not ~~p~~ bipartite?



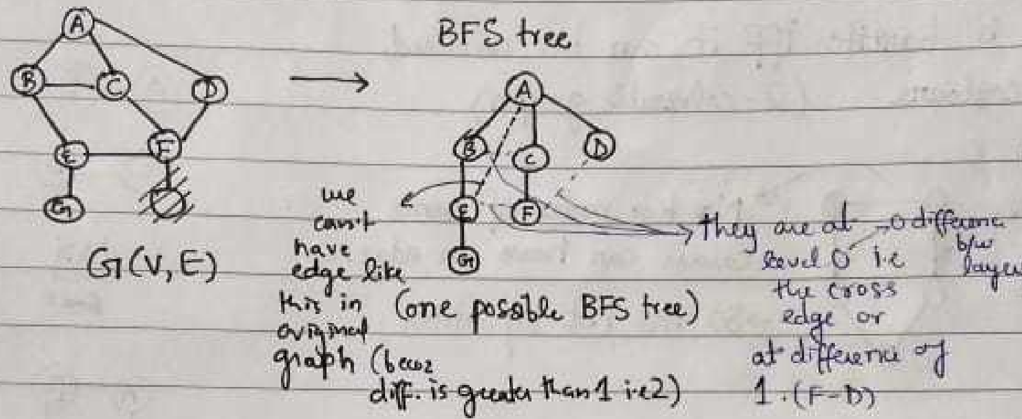
ONLY applicable for undirected graph.

~~if we have any~~ "cross edge" b/w any 2 vertices, then it's not bipartite.
*of same level

# "Properties of Traversal Trees" (14)

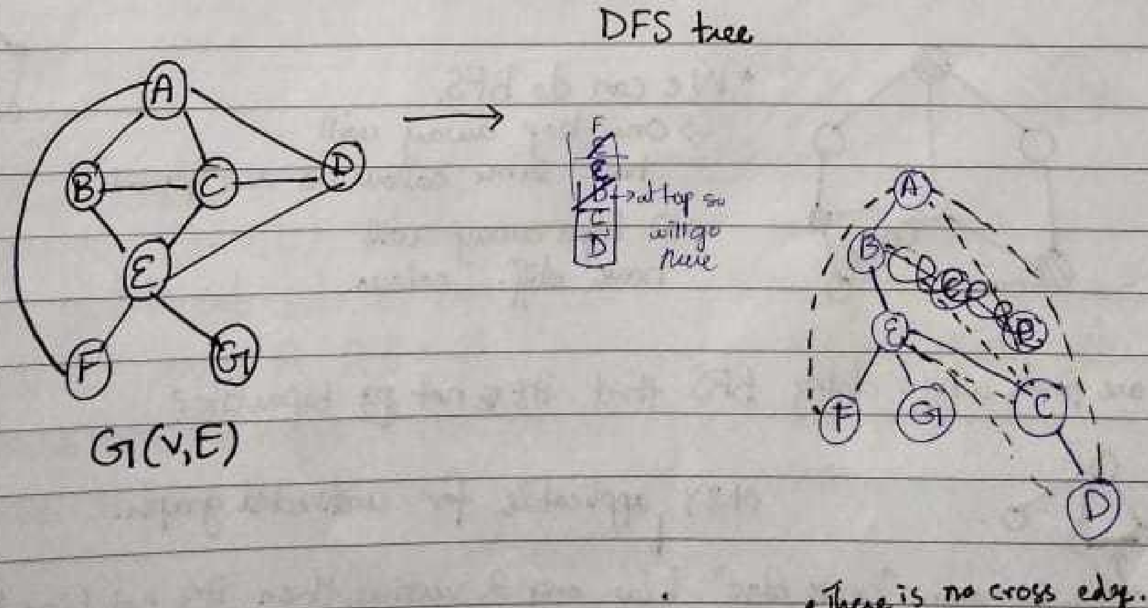
Wednesday
18th September 2019

* G(V,E) is an undirected graph. Let 'T' be a BFS tree of G. Let x,y be nodes in T belonging to layers Li and Lj and let ~~edt~~ (x,y) be an edge of G. Then i and j differ by at most 1.

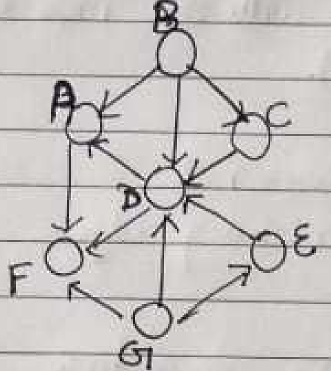

A
B
C
D
E
F
G
G(V,E)
BFS tree
A
B
C
D
E
F
G
we can't have edge like this in original graph (becoz diff. is greater than 1 i.e2)
(one possible BFS tree)
they are at level 0 i.e the cross edge or at difference of 1. (F-D)
0 difference b/w layers

T is subset of G so it will have all vertices V but some edges E'
T(V, E')

G(V,E) is an undirected graph. Let T be a DFS tree of G. Let x and y be nodes in T and let (x,y) be an edge of G that is not an edge of T, than one of x and y is ancestor of the other.



• There is no cross edge.



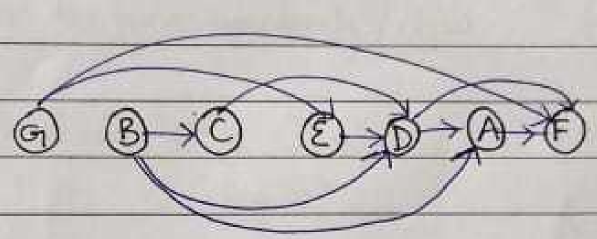
- we can not have cycles.
- it's a directed graph

* Course - Pre Requisite Problem.

What are the minimum number of Semesters to finish all of the courses.

| 1 | 2 | 3 | 4 | 5 | ← semesters |
|---|---|---|---|---|---|
| B<br>G | C<br>E | D | A | F | ← courses. |

"DAG" Directed Acyclic Graph



Topological Ordering / Linearization

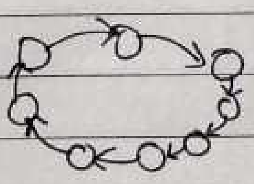
* Every DAG has a topological ordering !

Minahil

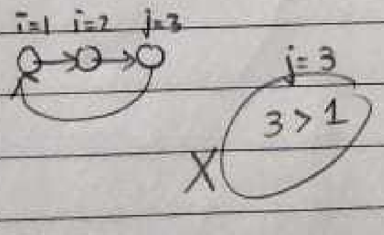
CLAIM: If G has a topological ordering, then G is DAG

PROOF By Contradiction: If G has a topological ordering & G also has a cycl

$v_i \ldots v_j$



if j > i then it's a contradiction. It means we are going in opposite direction. Because normally it is from i → j so i < j



* Every DAG has at least one souce (node with zero increasing edge and at least one sink (zero outgoing edge).

⇒Proof by Contradiction: DAG 'G' has no Souce node.
Every node has some incoming edge.

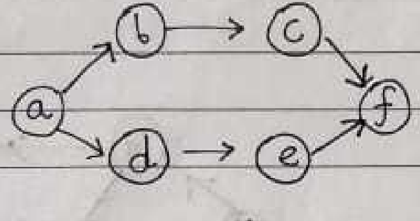
O(m+n) ← lineartime for graph.

Subject: Date:

## * Time complexity of finding topological ordering:

⇒ Exercise: How many topological ordering possible?

Start from source node ↑ which has no incoming edge



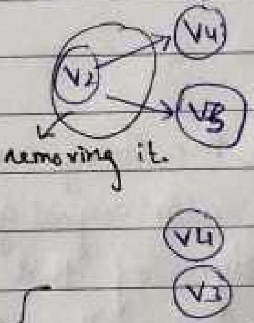
• there are 6 possible top. orderings.

"Pre-processing" reduces the time complexity / effort.

| v1 | v2 | v3 | v4 | v5 | v6 |
|---|---|---|---|---|---|
| 2 | 0 | 1 | 3 | 2 | 0 |

→ Binary Search has linear time $O(n)$ and sublinear time $O(\log n)$

$S = \{v_2, v_6\}$



this required constant time

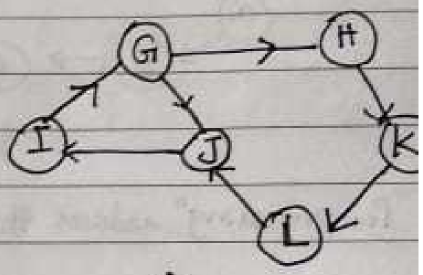
| v1 | v2 | v3 | v4 | v5 | v6 |
|---|---|---|---|---|---|
| 2 | 0 | 0 | 2 | 2 | 0 |

But in total we have $O(m+n)$

* Strong Connectivity:

Nodes u & v are mutually reachable

from slides



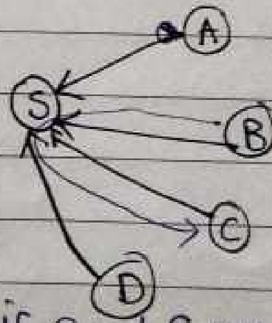
↑ Strongly connected

if
- 'u' and 'w' are mutually reachable

and
- 'w' and 'v' are mutually reachable
- then 'u' and 'v' are also mutually reachable.

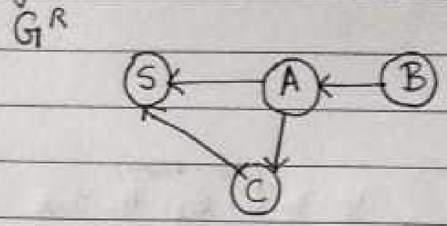
* How to determine the strong connectivity?
Run BFS/ DFS with original graph & then reverse all the edges of that graph and run BFS / DFS again, from the same node.

*

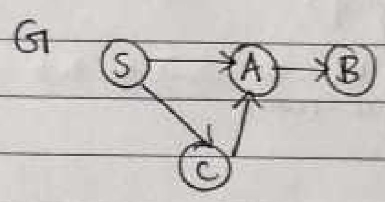


If B and S are mutually connected then B & C are also. (via S)

Subject *e.g 1 Date: 25th Sep. 2019



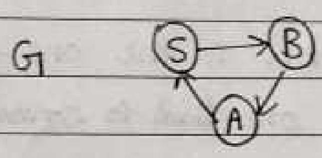
set of vertices reachable from S. {ϕ}



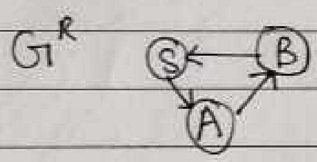
set of vertices reachable from S {A, B, C}

∴ NOT strongly connected

*e.g 2



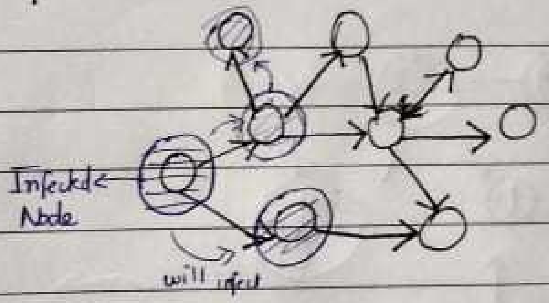
Nodes reachable from S = {A, B}



Nodes reachable from S = {A, B}

∴ Strongly connected graph.

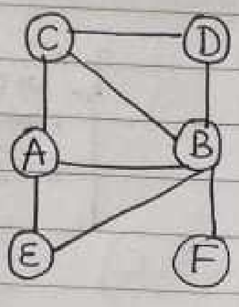
*Directed graph:



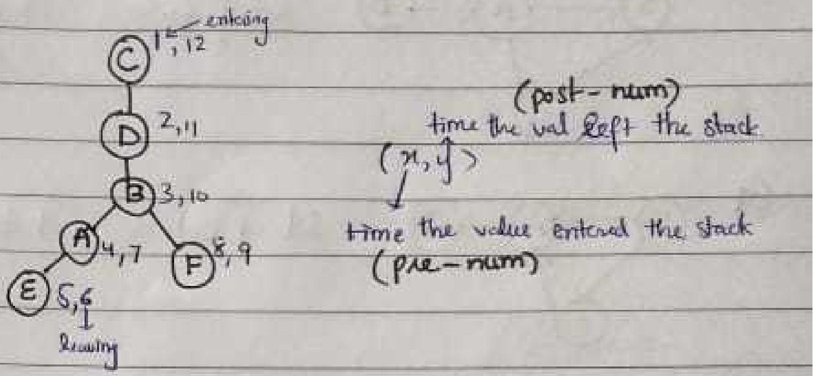
→ Is there any node that is infated and can effect /infect the entire network

What we are trying to determine is "Reachability" from a vertex 'v'.

*



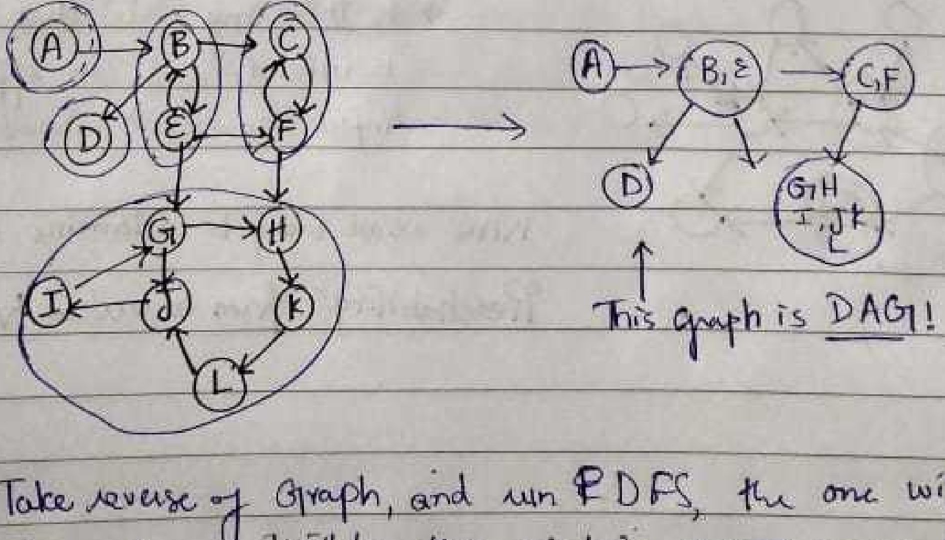
Running DFS now,



* The one which gets the highest post number is the one from where we should start running DFS / BFS and the one ~~that is~~ ~~[illegible]~~ from where we can reach to each vertix.

* pre(u) < pre(v) < post(v) < post(u)

*



This graph is DAG!

DAG is NOT ~~a~~ a Tree

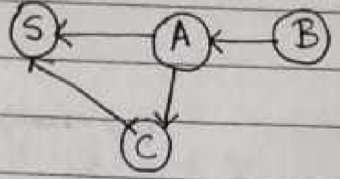
Every Tree (they is a dag

* Take reverse of Graph, and run ~~R~~ DFS, the one with high post num Will be the sink in ~~[illegible]~~ original Graph.

Subject: *e.g 1: Lecture #9

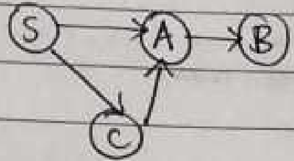
Date: 25th Sep 2019

$G^R$



set of vertices reachable from S. {$\phi$}

$G$

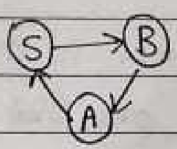


Set of vertices reachable from S {A, B, C}

∴ NOT Strongly connected

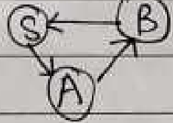
*e.g 2

$G$



Nodes reachable from S = {A, B}

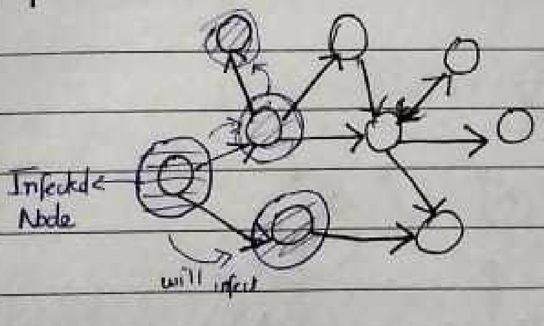
$G^R$



Nodes reachable from S = {A, B}

∴ Strongly connected graph.

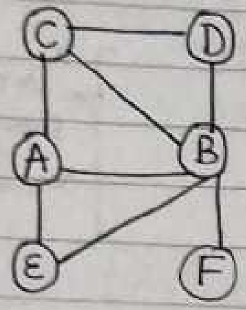
*Directed graph:



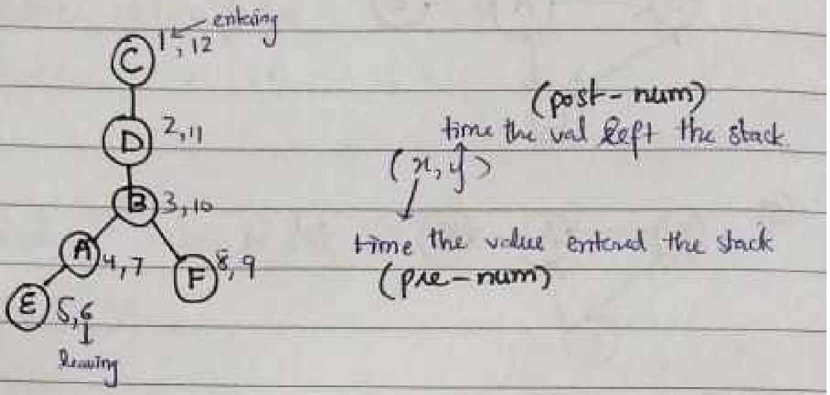
→ Is there any node that is infected and can effect /infect the entire network

What we are trying to determine is "Reachability" from a vertex 'v'.

*



Running DFS now,

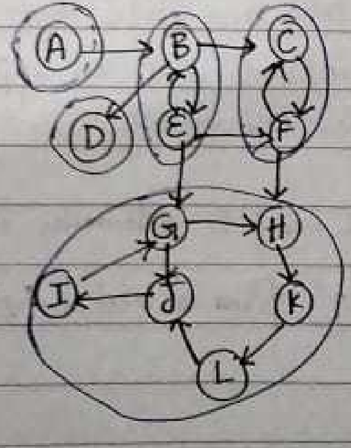


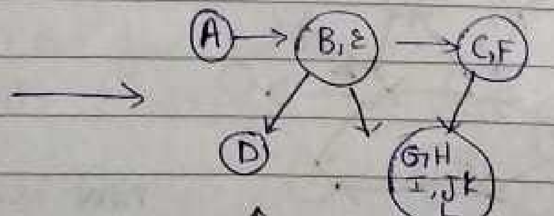
(post-num)
time the val left the stack
$(x, y)$
time the value entered the stack
(pre-num)

* The one which gets the highest post number is the one from where we should start running DFS / BFS and the one ~~that is connect~~ from where we can reach to each vertix.

* $pre(u) < pre(v) < post(v) < post(u)$

*





This graph is DAG!

DAG is NOT ~~a~~ a Tree.

Every Tree (Graph) is a dag.

* Take reverse of Graph, and run ~~R~~ DFS, the one with high post num will be the sink in ~~[illegible]~~ original Graph.

# "INTERVAL SCHEDULING"

→ We are given ONE classroom, and we want to:

- Schedule non-overlapping classes
- Maximize the no. of Classes that we can schedule

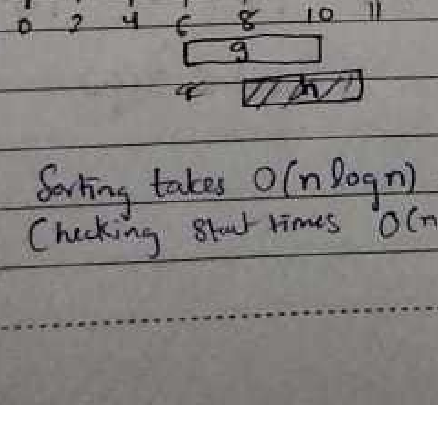
We are also given start and end times of classes.

***GREEDY Algorithms:** (short-sighted greed in the design of algos) ← we avoid/don't look at long term things.
it builds up a solution in small steps, short-sightedly choosing a decision at each step to optimize some underlying criterion.

- There can be more than one correct optimal solutions.

⇒ Local Optimal Choice -Leads to→ Global Optimal Solution

Greedy algo basically looks at the best option acc. to current situation & then assumes that it is correct all in all (globally). becoz it's observing short sightedly.



b, e, h → max classes that do not overlap.

How can we go about it?
take
- shortest width interval
↓
NON OPTIMAL Greedy strategy
↓
becoz In that case we can not get maximum classes.

⋮

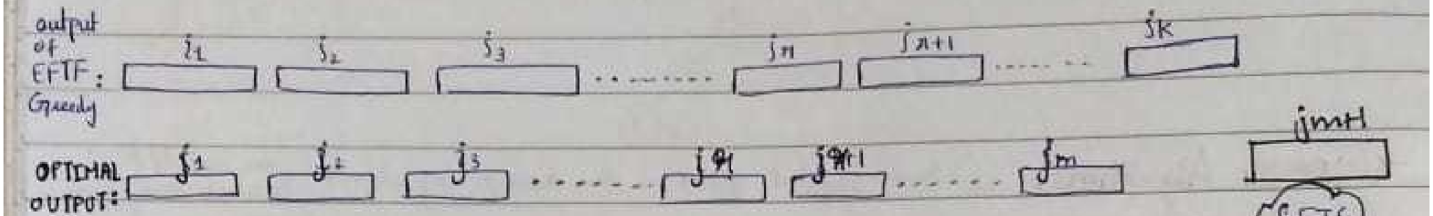
! Sort the thing according to finish time.

Sorting takes $O(n \log n)$ → if we use merge sort.
Checking start times $O(n)$ →
} $O(n \log n) + O(n)$
$\Rightarrow O(n)$

Subject: Date:

* Is that algo (earliest finish time--) optimal?

⇒ "STAYS- AHEAD" Argument. $r \rightarrow$ set of intervals and $k \leq r$



EFTE → earliest finish time first.

We have to show that $k=m$.

* Proof by induction: for all $r \leq k$ show that $f(i_r) \leq f(j_r)$ where f is finish time → at any given point ~~&~~ i would either finish before j the or at the same time as j.

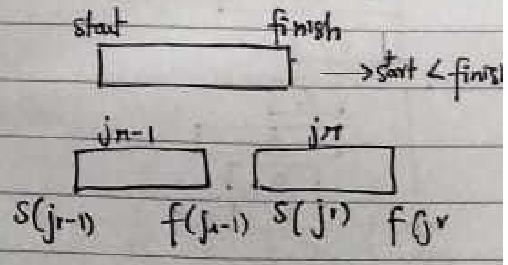
⇒ Base: $r=1 \rightarrow f(i_1) \leq f(j_1)$ True becoz greedy EFTF selects interval with min finish time at every step.

for $r>1$

⇒ Inductive Hypothesis: Assume that $(f i_{r-1}) \leq f(j_{r-1})$ holds.

$$f(j_{r-1}) \leq S(j_r)$$

from our hypothesis $f(i_{r-1}) \leq S(j_r)$



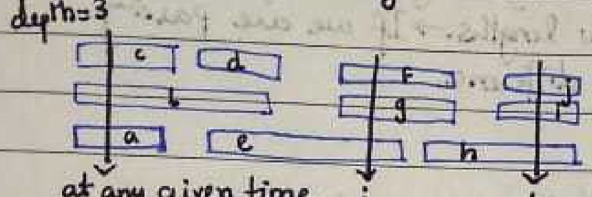
$$f(j_{r-1}) \leq S(j_r)$$

Subject: Lecture 11

Monday
Date: 7th Oct. 2019

# * Interval Partitioning Problem:

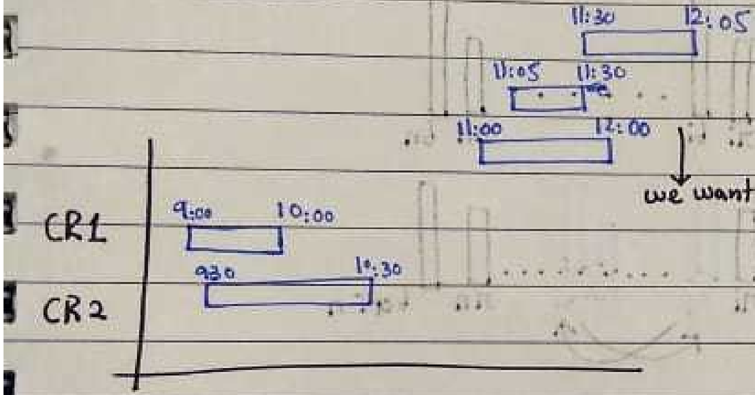
depth=3



at any given time there is max. of 3 lectures that run parallel. ........
(3 overlaps)

→ The depth of a set of intervals is the maximum no. that passes over a single point on the time line.

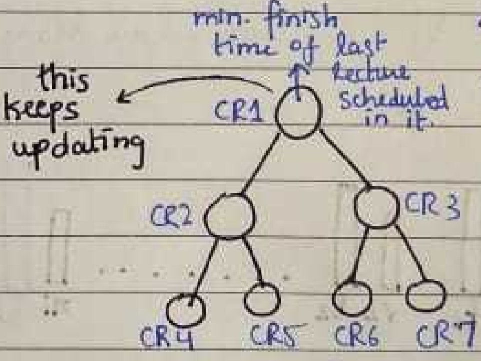
⇒ we want to minimize the gap b/w 2 classes in same room ]
this is to maximize no. of lectures in one room ]
this is to minimize the no. of class room



we want to schedule these.

Which data structure can we use?

Priority Heaps (min heap'fy).

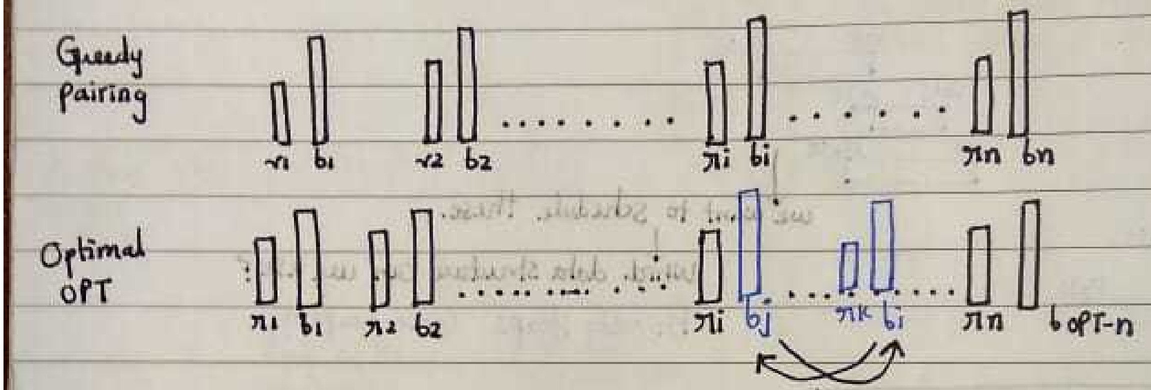


- 'n' is no. of lectures
- 'd' is no. of classrooms.

- It will take $\underline{O(n \log d)} + O(n \log$
  for initial Sorting

→ We are given 'n' red & blue sticks of abitrary hieghts / lengths, how can me minimize the avg. of the diff. b/w lengths. → If we are pairing 1 red & 1 blue and then finding their difference.

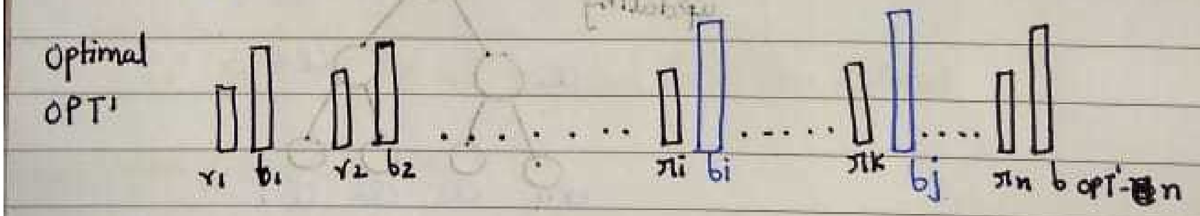
* Sort the red and blue sticks in order of their heights & pair them in sorted order. ← Optimal Greedy Strategy



Greedy pairing
$r_1$ $b_1$ $r_2$ $b_2$ ........ $r_i$ $b_i$ ...... $r_n$ $b_n$
Optimal OPT
$r_1$ $b_1$ $r_2$ $b_2$ ........ $r_i$ $b_j$ ... $r_k$ $b_i$ ... $r_n$ $b_{OPT-n}$

$b_i < b_j$ ← becoz we are listing in sorted order

$r_i < r_k$



Optimal OPT'
$r_1$ $b_1$ $r_2$ $b_2$ ........ $r_i$ $b_i$ ..... $r_k$ $b_j$ ... $r_n$ $b_{OPT'-n}$

$$\Rightarrow \text{Cost}(OPT') = \text{Cost}(OPT) + \frac{1}{n}\Big(\underbrace{|b_i - r_i| + |b_j - r_k|}_{\text{new costs added}} - \underbrace{|b_j - r_i| - |b_i - r_k}_{\text{old costs subtracted.}}$$

of previous pairs.

* **case 1**: $r_i < r_k < b_i < b_j$ ← I don't know the relative diff. b/w r's & b's.

* **Case 2**: $r_i < b_i < b_j < r_k$

* **case 3**: $r_i < b_i < r_k < b_j$

} there are total 6 possible cases.

* $\text{cost}(opt') = \text{Cost}(opt) + \frac{1}{n}\left(b_i - r_i + b_j - r_k - (b_j - r_i) - (b_i - r_k)\right)$

  $'' = \text{cost}(opt) + \frac{1}{n}(b_i - r_i + b_j - r_k - b_j + r_k\ )$ ⇒ cost opt = cost opt'

Subject:

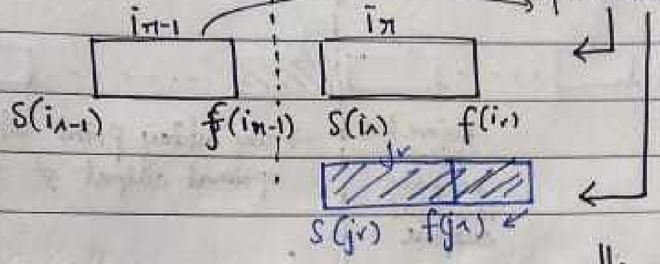
Wednesday
Date: 2nd October 2019

***Optimal Substructure Property**: we can find the final optimal solution by looking at each interval of our pblm.

if we have OSP then either we can have Greedy Algo Sol. or any dynamic pblm solution.

Continued...

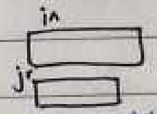
for this we have 2 possible candidates



$$\therefore \quad f(i_r) \leq f(j_r) \quad \Leftarrow$$

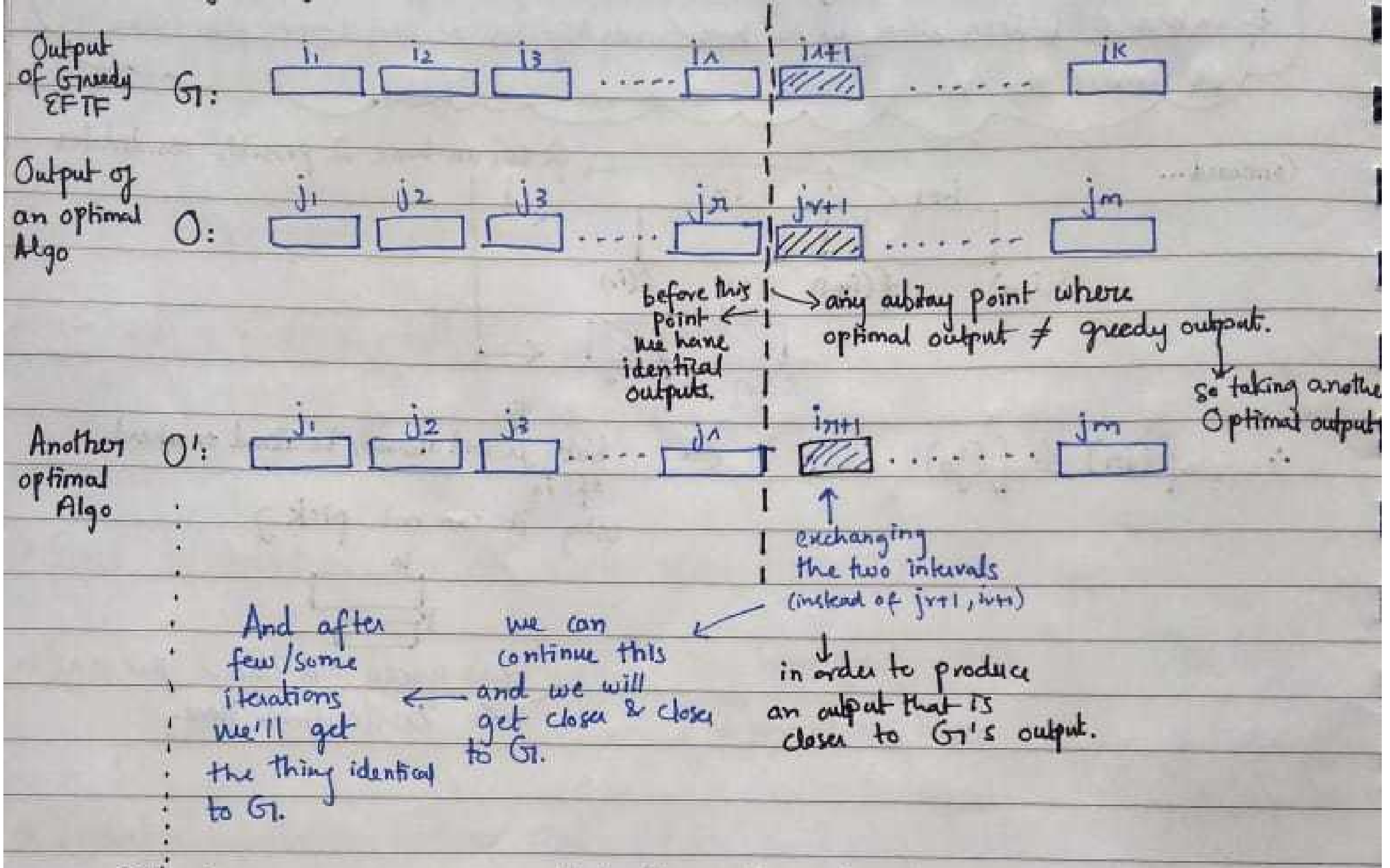
but $j_r$ will always be alined or ahead of $i_r$

why it can not pick ?



becoz greedy will choose the one with lesser finish time.

* Exchange Argument:


Output of Greedy EFTF
G:
$j_1$ $j_2$ $j_3$ ..... $j_r$ $j_{r+1}$ ....... $j_k$
Output of an optimal Algo
O:
$j_1$ $j_2$ $j_3$ ..... $j_r$ $j_{r+1}$ ....... $j_m$
Another optimal Algo
O':
$j_1$ $j_2$ $j_3$ ..... $j_r$ $i_{r+1}$ ....... $j_m$

before this point we have identical outputs.

→ any abitray point where optimal output ≠ greedy output.

So taking another Optimal output ∴

exchanging the two intervals (instead of $j_{r+1}$, $i_{r+1}$)

in order to produce an output that is closer to G's output.

we can continue this and we will get closer & closer to G.

And after few/some iterations we'll get the thing identical to G.

$O^x = G$ ∴ we can say that G is also optimal

⇒ Weighted Interval Scheduling:

- Now, I don't jus have start & finish time, but weights as well. Can we still find optimal solution through EFTF algo?

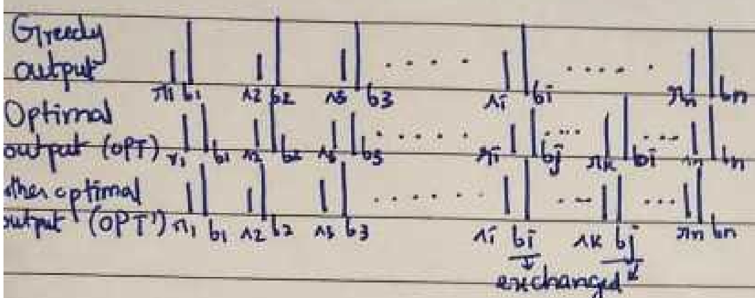
  ↓ No EFTF would not work.

-> Well, we don't have greedy algo for this. Why? You can choose some which has max. at current time, but we can have even more weigh with some other combinations.

Subject: Lecture 12 ALGO

- $b_i < b_j$
- $x_i < x_k$

Date: 9th October


Greedy output
$x_1$ $b_1$ $x_2$ $b_2$ $x_3$ $b_3$ ..... $x_i$ $b_i$ ..... $x_n$ $b_n$
Optimal output (OPT)
$x_1$ $b_1$ $x_2$ $b_2$ $x_3$ $b_3$ ..... $x_i$ $b_j$ .... $x_k$ $b_i$ ... $x_n$ $b_n$
other optimal output (OPT')
$x_1$ $b_1$ $x_2$ $b_2$ $x_3$ $b_3$ ...... $x_i$ $b_i$ ... $x_k$ $b_j$ ... $x_n$ $b_n$
exchanged

case I: $x_i < x_k < b_i < b_j$

case II: $x_i < b_i < x_k < b_j$

case III: $x_i < b_i < b_j < x_k$

case IV: $b_i < x_i < x_k < b_j$

case V: $b_i < x_i < b_j < x_k$

case VI: $b_i < b_j < x_i < x_k$

$$\Rightarrow \text{cost}(opt') = \text{cost}(opt) + \frac{1}{n}\Big(\underbrace{|b_i - x_i| + |b_j - x_k|}_{\text{pairs we get after exchange}} - \underbrace{|b_j - x_i| - |b_i - x_k|}_{\text{pairs before exchange}}\Big)$$

$$'' = \text{Cost}(opt) + \frac{1}{n}\left(\cancel{b_i} - \cancel{x_i} + \cancel{b_j} - \cancel{x_k} - \cancel{b_j} + \cancel{x_i} - \cancel{b_i} + \cancel{x_k}\right)$$

$\underbrace{\text{cost}(opt') = \text{cost}(opt)}_{\text{opt' equal opt}}$ → for case 1.

We want opt''s value to be lesser then $\overset{\text{or equal to}}{\uparrow}$ opt. (opt's value can't be greater)

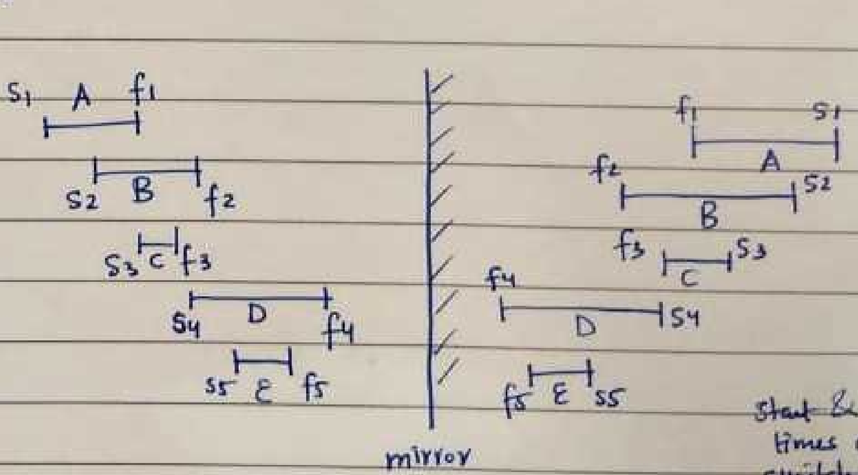
$$\downarrow$$

$$\therefore \frac{1}{n}\left(|b_i - x_i| + |b_j - x_k| - |b_j - x_i| - |b_i - x_k|\right) \rightarrow$$ should be either zero (to be lesser than opt) or -ve.

Again!

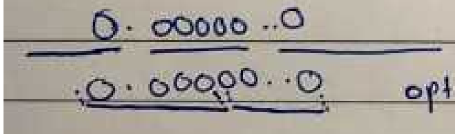
* Cost(opt') can never be greater than cost(opt).

*Class Activity: Q1.



Q2. each stick of length 1m covers 5 balls.

Start & finish times are switched. So if EFTE is optimal, then greedy is also optimal



greedy gives 3

opt. give 2.

- Some greedy algos → MST
  → Prims
  → Dijkstra

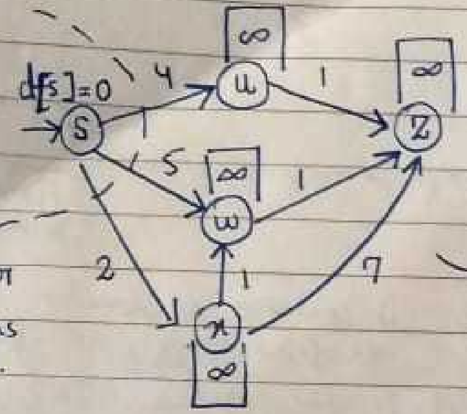
# "SINGLE SOURCE SHORTEST PATHS"

Lec # 13

Subject:

Date: 14 Oct 2019

GIVEN A directed graph G = (V,E) with designated start node 'S'. Assume 'S' has a path to every other node in G. Each edge has weight ≥ 0.

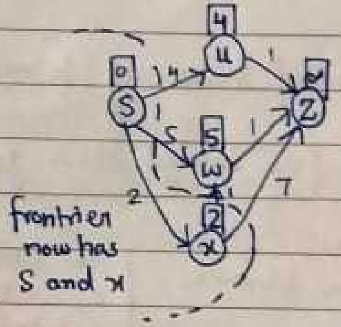


frontier
↓
only has
S rn.

* Dijskstra's don't work with "NEGATIVE VALUES/ WEIGHTS"

• we need to find the shortest path from S.

→

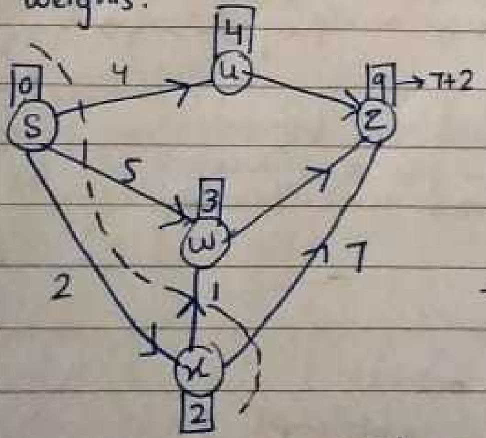


→ Now acc. to my local greedy choice i'll choose x to be the next node (becoz lowest cost)

* distance of x from 'S' → d[x] = 2

[and this is the shortest distance from S to x, becoz every other has more cost]

• Look at the edges coming out of frontier cloud, and then update the weights.



→ Now i'll choose 'W'

• Note only neighbous of X are updated.

WHY?

Subject: Date:

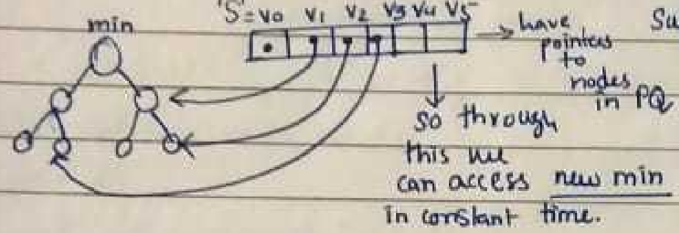
① Initialize

② Build a Priority Queue (distances from 'S' to every other node~~s~~ are keys) → all n nodes are added to the priority queue

③ The last node added to frontier → update distances of its neighbours

④ Delete minimum distance vertex from priority Q and place that inside the frontier.

For the update in PQ we need an Auxiliary data Structure → Supporting ds along with main.



→ have pointers to nodes in PQ

so through this we can access new min in constant time.

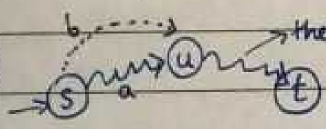
- Delete min → $O(\log n) \times n$ → i.e $O(n \log m)$
- Update distances → $\underset{\text{total no. of edges in PQ.}}{m} \times O(\log n)$ ~~i.e O(~~ i.e $O(m \log n)$

* Overall Time : $O(n\log n + m \log n) = \boxed{O(m \log n)}$

becoz we have (usually) more no. of edges in a well populated graph. then nodes.

* Optimal Substructure Property: constructing optimal solution to a problem by building it from optimal solutions to subproblems.

e.g



→ there can be nodes in between.

· shortest path from S→t is shown.

*Subpaths of shortest paths are also shortest paths.

Q. How can we claim that there is no other shortest path other than a?
If a is not shortest path, then S→t will also not be shortest, but thats not true. ∴ a is the only shortest path becoz it's subpath of shortest path S→t.

- if I have graph that has -ve distances or weights, I can add a big constant to each weight to get rid of -ve weights. Can I then run dijkstra to find shortest path?
No. → Somehow depends on no. of hops.

- But what about MST? <u>Yes</u> because MST is independent of no. of hopes.

Algo

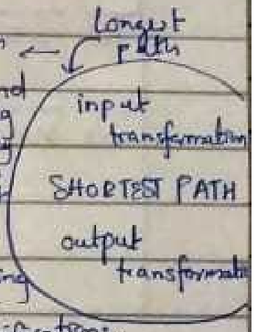
Lecture#14

Wednesday
16 October 2019

* Find single source longest paths in a DAG:

→ Reduction can be used to find one.

- Change the shortest path algo; make dist max instead of min and initialize the dist to $-\infty$.



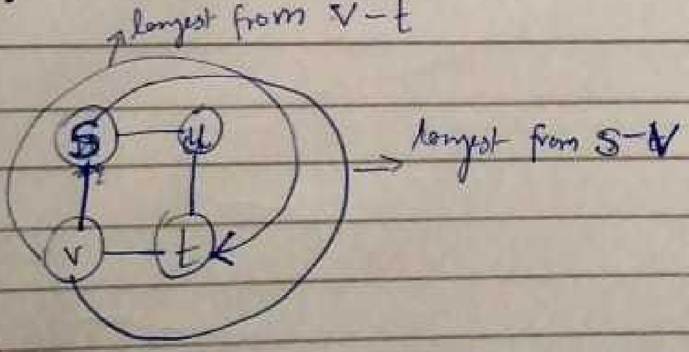
can be found using already known shortest path algo by doing slight modifications.
e.g max the distance and initialize dist to $-\infty$.

- Longest path in a general graph is a problem. This is because we can have exponential time solution but no polynomial time solution. i.e $\underbrace{n^k}$

- Also, optimal substructure property does not hold true for longest path.

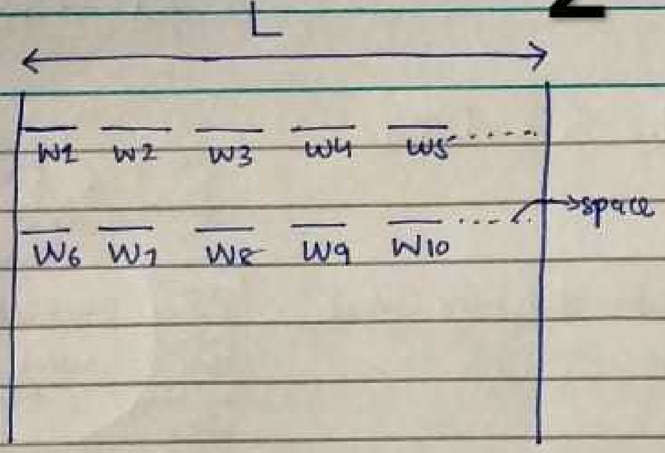
efficiency = $O(n^k)$

* NOTE: Longest path problem is not defined in the presence of +ve weight cycles. !



This forms a cycle, and this can't be done. (NOTE)

2



$\Rightarrow$ Penality $= L - \underbrace{W}_{\text{sum of widths of words on a line.}}$

•hyphenated words are not allowed.

* Find a layout which minimizes the total $\overset{\text{sum}}{\uparrow}$penality.