Dynamic Programming

→ Used for optimization Problems.

Optimization Problem : Limited resources → for parameters maximization & minimization.
E.g:- 2 hours study, max. no.s

→ All problems cannot be solved by DP.
→ Traveling Sales Person Problem. (Hamatonyar path)
↳ minimum distant
↓
Do not visit vertex again.

Ilar → Do not visit edge again
path

```
Fib (n)
{
  if n=0
      return 0
  if n=1
      return 1
  else
      return (Fib(n-1)+Fib(n-2))
}
```

Overlapping Computation

$T(n) \leq 2^n$

proof = $(1.6)^n$

Sol:- IF we store same recursive calls at one place. ↑ result.

```
Fib(n)
{
    a = Fib(0)
    b = Fib(1)
    for i = 2 to n
    {
        c = a+b
        a = b
        b = c
    }
}
```

O(n)

→ Recursion is top down approach.
→ Loop is faster than recursion.
→ In recursion sub-problems are not independent.

2nd-December-2021 Thursday

Lecture # 16

i	1	2	3	4	5	6
w_i	2	3	5	8	13	16
p_i	1	2	3	5	7	10

0/1 Knapsack Problem

Bag Capacity
Weight of things
Money of things

→ We have n items, for which we know their weights w_i & profit p_i. In addition to this there is a capacity of bag 'M'.

Maximize $\sum_{i=1}^{n} x_i p_i$, where $x_i = 0$ or 1.

Such that

$$\sum_{i=1}^{n} x_i w_i \leq M$$

Max Profit P

→ capacity of bag.

1- $P(n, M) = P(n-1, M)$ if $w_n > M$

↓ no. of items

$P(n, M) = P(n-1, M)$ if $w_n \leq M$, if $x_n = 0$

~~$P(n-1, +p_n$~~ if $w_n \leq M$, if $x_n = 1$

$= P(n-1, M - w_n) + p_n$

$$P(n, M) = \max \begin{cases} P(n-1, M) & \text{if } x_n = 0 \\ P(n-1, M-w_n) + p_n & \text{if } x_n = 1 \end{cases}$$

Recursive equation

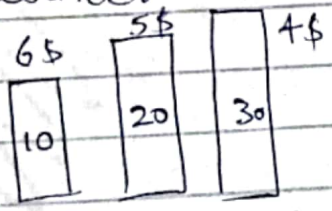
Base Case :-

$P(1, M) = 0$ if $w_1 > M$

↓ reduce M

$P(1, M) = p_1$ if $w_1 \leq M$

Counter Example :-



60$ 100$ 120$

160 220

M=50.

Profit ratio.

weight

↓ Greedy stategy does not work always

$\frac{22}{5} = 5.22$

$\frac{60}{10} = 6$

Example:

M = 30.

i	1	2	3	4	5	6
w_i	2	3	5	8	13	16
P_i	1	2	3	5	7	10

Using bottom up approach, for reuse of nodes. computation.

	M: 0	1	2	3	4	5	6	7	8	9	10	.....	30
1	0	0	1	1	1	1	1	1	.....				1
2	0	0	1	2	2	3	3	.....					3
3													
4													
5													
6													18

$P(2,0) = P(1,0)$

$P(2,1) = P(1,1)$

$P(2,2) = P(1,2)$

$P(2,3) = \begin{cases} P(1,3) & 1 \\ P(1,0) & 0 + 2 \end{cases}$

$P(2,4) = \begin{cases} P(1,4) & 1 \\ P(1,1) & 0 + 2 \end{cases}$

$P(6,30) = P(5,30)$

$P(5,14) + 10$

$P(5,14) = P(4,14)$

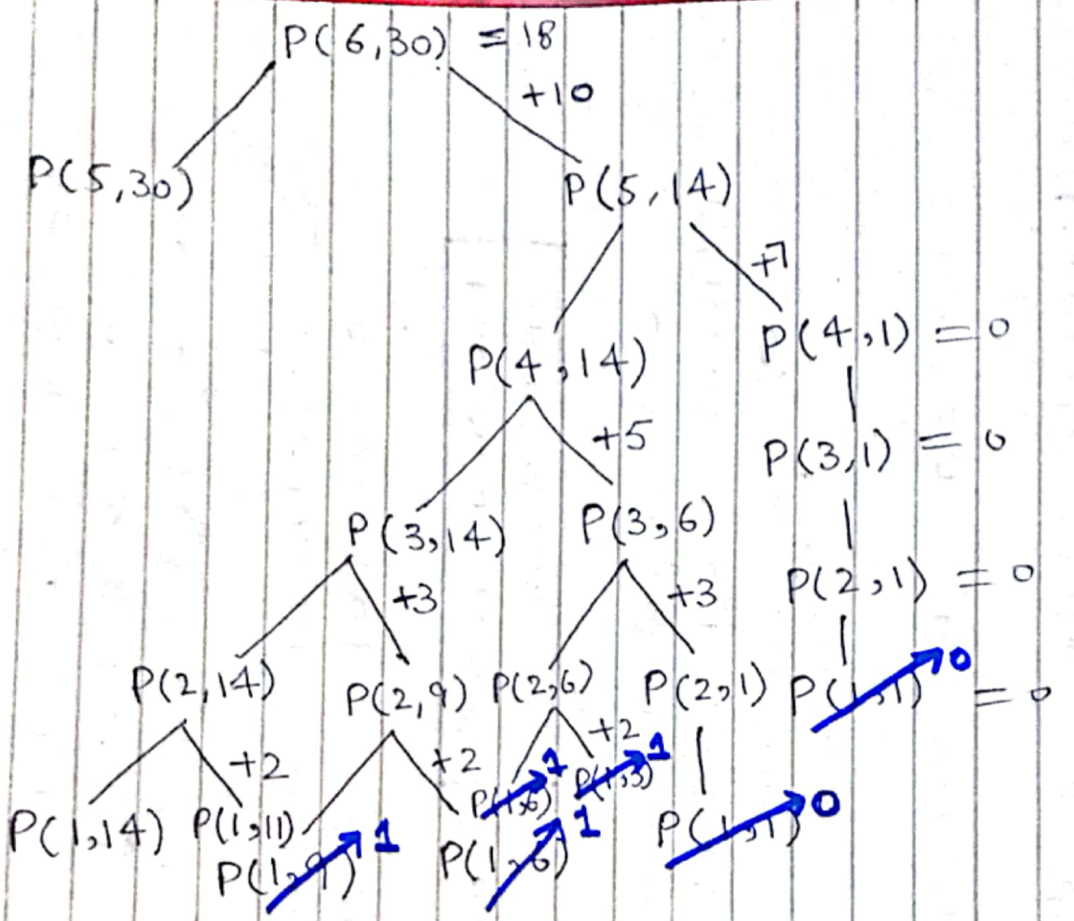
$P(4,1) + 7$

Seudo polynomial

$T(n) = O(n \cdot M)$

max, profit

Backtracking = $O(n!) = O(n)$



height = (n-1)

No. of nodes = $2^n - 1 = \boxed{O(2^n)}$

Exponential Problem.

↳ NP-Hard Problems

↳ Non-Deterministic Polynomial

<6, 4, 3>

<6, 4, 2, 1>

$2^6 = 64$

@ n.M = 6.30 = 180. → expensive.

8-12-2021 Wednesday

Lecture #17

Longest Common Subsequence Problem

1 cell DNA → billions of Length.

X = A T T C A G T T (m)

0 1 1 0 0 1 0 1 → m

Common subsequence of both: T T G T → subsequence $= 2^m$

Y = C G T G T A G A T → n

0 0 1 0 1 0 1 0 1 (n)

Comparison $= 2^m \cdot n$

Needleman Wunsch
SmithWaterman
} algorithms for L.C.S.P.

c → longest Common Subsequence

$$C(n,m) = C(n-1,m-1)+1 \quad ; \text{if } x_m == y_n$$

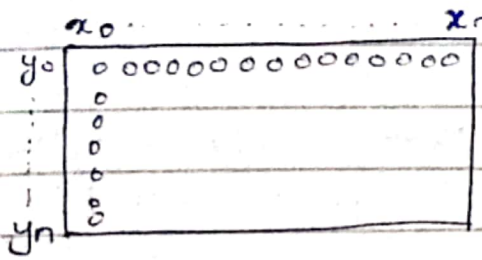
$$C(n,m) = \max \begin{cases} C(n-1,m) \\ C(n,m-1) \end{cases} \quad ; \text{if } x_m \neq y_n$$

$C(n,m) = 0$. (base case) ; if m or $n == 0$

Recursively solution $= 3^n$ or 3^m.

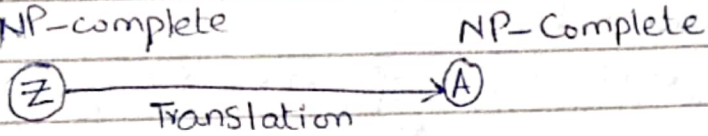
Bottomup approach $= O(n \cdot m)$

10-12-2021 Friday



Lecture # 18

Problems in DP are dependent.



Greedy Algorithms

20-12-21 Monday

Lecture # 19

Activity Selection

→ At a time one activity Schedule.
→ Activity must run to completion.
→ Every greedy algorith has recursive solution.
Complexity = $O(n \log n)$

Huffman Code

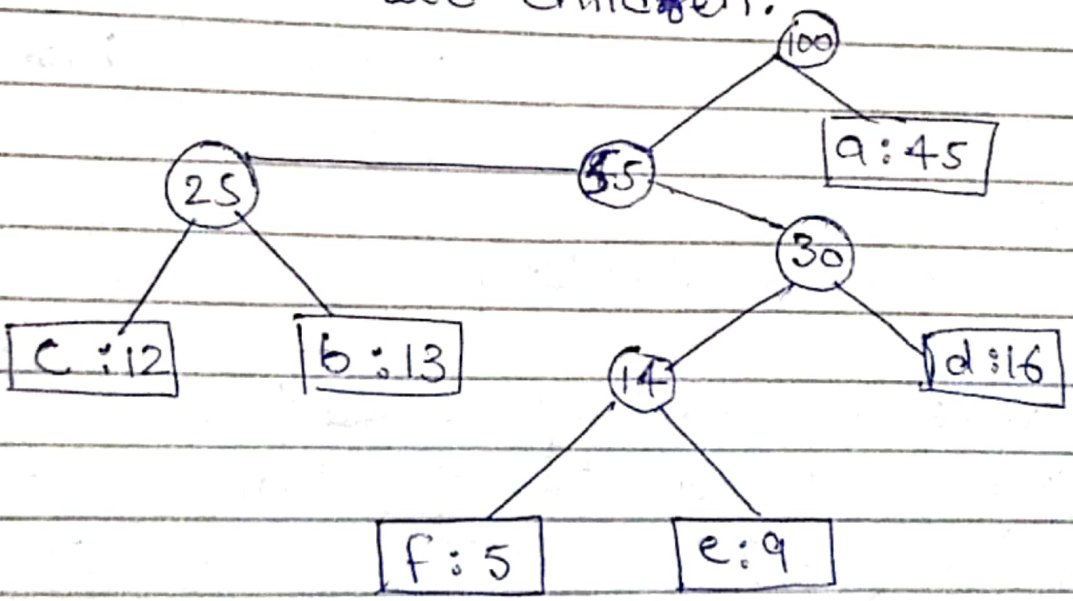
* Lossy Compression
  E.g: Image Processing

* Non-Lossy Compression
  E.g:- document compression.

→ prefix code : no code word is prefix of another codeword.

→ All characters will be leaf nodes. = prefix code

→ Optimal tree :- Each internal node must have two children.



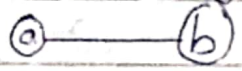
Complexity = O(n log n)

21-12-2021 Tuesday

Lecture# 20

Graph Algorithms.

Simple graph → undirected graph.



(a,b) (b,a)

directed → undirected ; not possible
undirected → directed ; possible.

Represent graph

- Adjacency Matrix → for dense graph
- Adjacency List. → for non-dense graph.

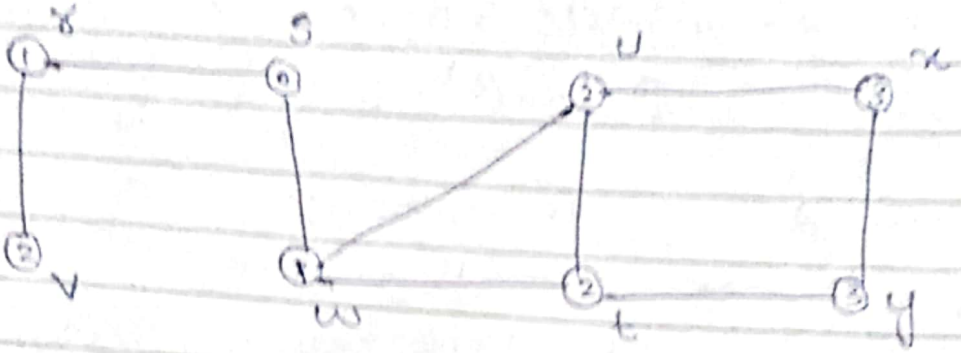
Find Shortest Path.

BFS → (O(V+E))

color
distance
parent.

22-12-2021 Wednesday

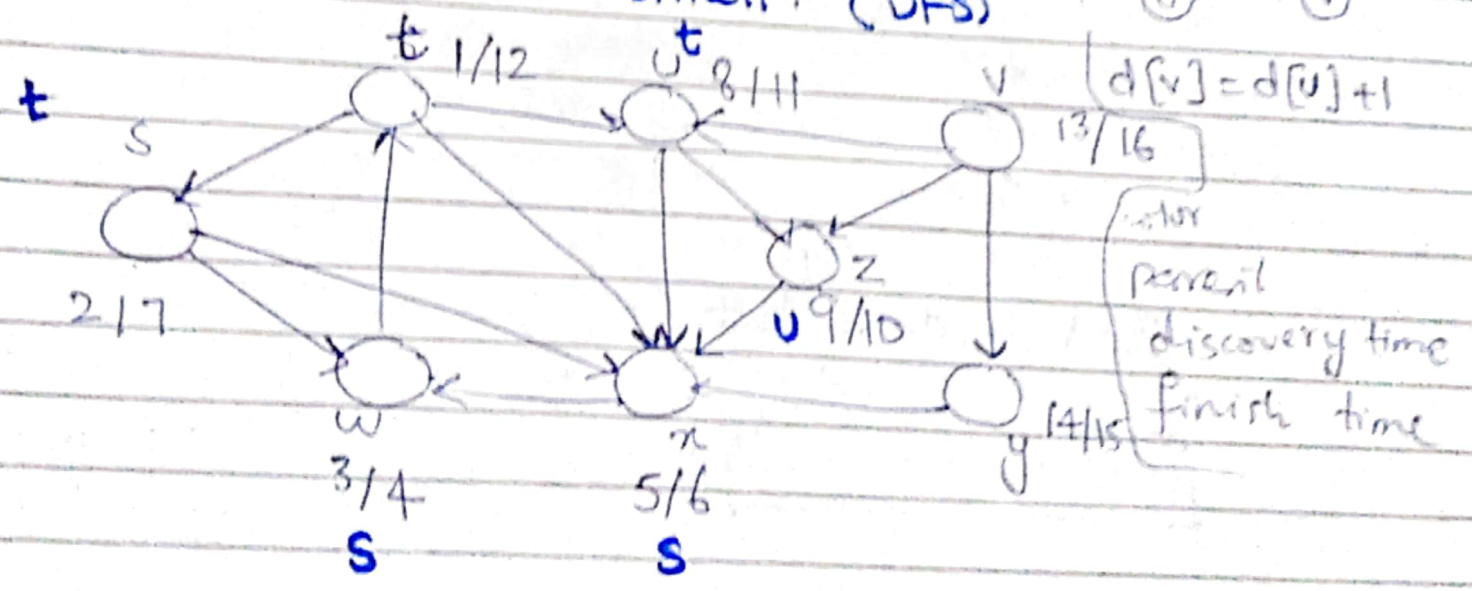
Lecture #21



$G_\pi = (V_\pi, E_\pi)$

$\delta(s,v)$ = shortest path.

DEPTH-FIRST-SEARCH. (DFS)



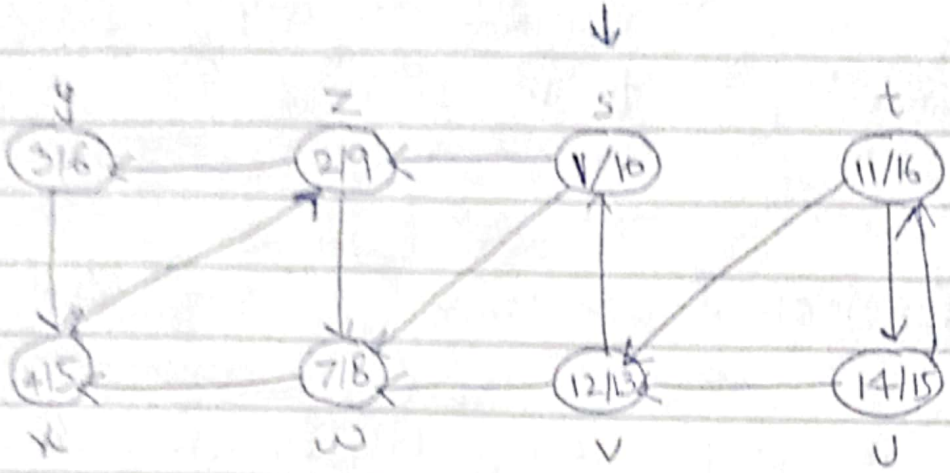
DFS (E)

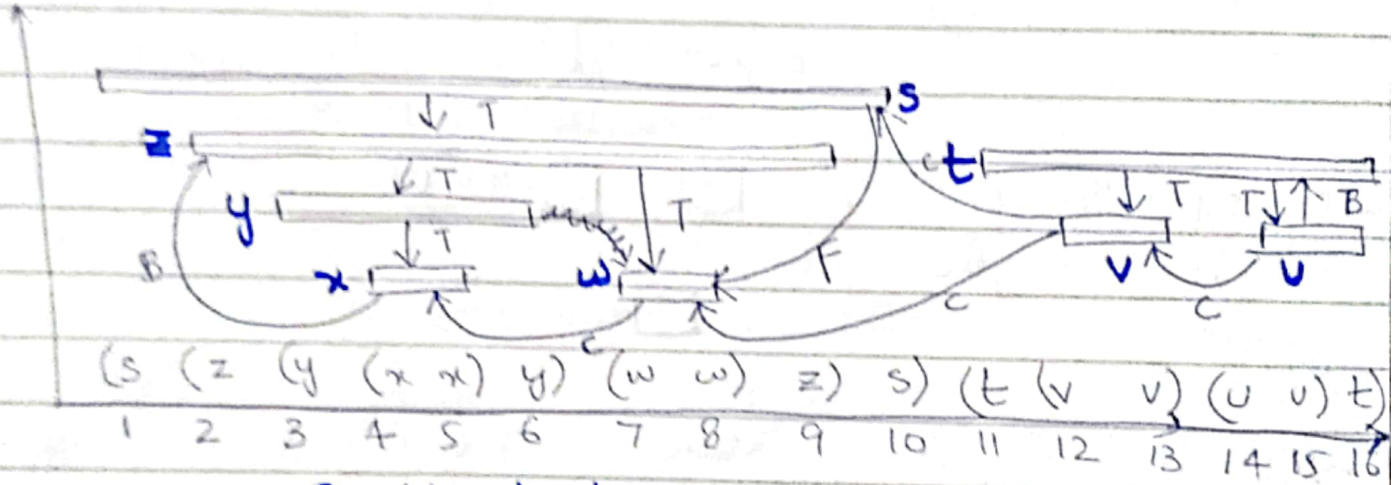
time = O(V+E)
complexity

10-01-2022 Monday

Lecture # 22

Depth first Search.





Gantt chart.

Activity complete or disjoint.

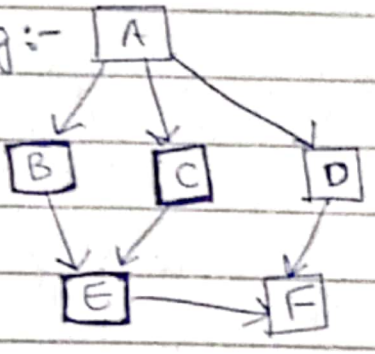
* Tree edge :- edge b/w parent & child.
* Back edge :- above x, to z, child to parent edge.
* Forward edge :- from grand parent to child, not parent to child.
* Cross edge : other edges are cross edges.

→ Back edge shows cycle.
→ There may be more than one cycle.
→ Forward edge shows, there is an alternate path between grand parent to child.

Topological Sort.

→ Directed Acyclic graph (DAG): a directed graph which does not have a cycle in it.

E.g:-



→ A dependency graph is an acylic graph.

→ Topological Sort:- Valid order + no voilation of dependencies.

E.g:- (BA) C E D F not T.S

Maximum finishing time = 2V.

DFS = (V+E) $n+k$

counting sorting = O(V)

Topological Sort = O(V+E).

12-01-2022. Wednesday

Lecture # 23

Strongly Connected Components

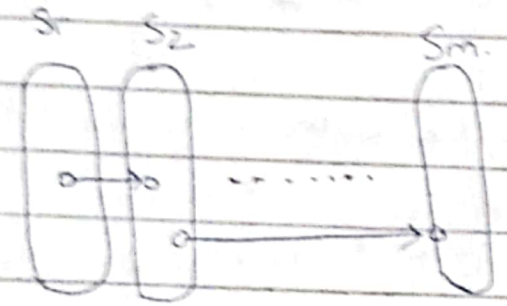
↓

acyclic

"Every directed graph is a DAG of its strongly connected components."

E.g:- Pakistan inner & outer cities & countries.

DAG :- at least 1 source, 1 sink



change all vertices of edges in reverse order, source becomes sink, & sink becomes source.

Claim:

DFS : max finish time, in source vertex.
~~vertex~~ node

→ $\exists u_1 \in S_1$

$f(u_1) > f(\forall v \in S_2)$

↓ vertex.

$f(u_1) > f(v_2)$; v_2

→ $\exists u_2 \in S_2$

$f(u_2) > f(v_3)$

Chp#22; done

DFS = $(V+E)$

Counting sort = (V)

DFS = $(V+E)$

$+$ $\boxed{O(V+E)}$

Complexity

Chp#23.

Minimum Spanning Trees (MVT).

→ Undirected
→ Connected
→ Weighted

} graphs, more than 1 (Minimum) Spanning Trees.

↳ connection of vertices with minimum cost.

Tree has minimum connected vertices.

viable set = minimum spanning tree edges.

$A = \phi$

$= \{e_1, e_2, e_3\}$

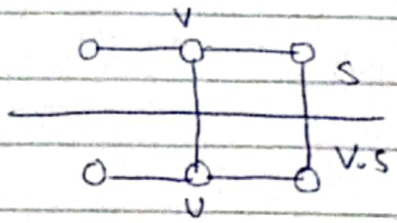
↓ safe edge

اگر ایک edge کو viable set میں add کرنے سے جو viable set بنے
وہ بھی safe edge ہو تو اسے add کی گئی edge کہتے ہیں۔

17-01-2022 Monday

Lecture #24:

MST Lemma.



respects: Edges not cut.

$w(u,v) \leq w(x,y)$

Connected, undirected graph.

Viable

Let T be a MST in which (u,v) is not present.
Let (x,y) belongs to MST 'T'.
Let make a new Tree T', where we add (u,v) & remove (x,y).

$$w(T)' = w(T) - w(x,y) + w(u,v).$$

$$w(T') \leq w(T)$$

Kruskal – Algorithm.

Disjoint set datstru

Sorting :- ElogE

Complexity :- E(V+E)

Prim's Algo.

Weighted graph } two inputs of Prim's
Starting vertex. } Algo

19-01-2022 Wednesday

Lecture # 25

chp #23.

Practice Questions:

15.4-5 , 15-5

16.1-2 , 16.1-4

16.2-7

22.2-7 , 22.2-8

22.3-5 , 22.3-6 , 22.3-13

22.4-3

22.5-3 , 22.5-6.

Prim's Algorithm.

Time Complexity :-

for loop = O(V)

Q = V[G] → O(n)

while loop → O(V) } VlogV

ExtratMin → logV }

for → 2E.

decrease key key[v] = w(u,v) → E logV

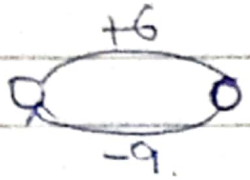
O(V) + O(V) + VlogV + E logV. = E logV (best)

Kruskal → O(ElogE)

Asymtotically = O(E logE) = O(ElogV) $E = V^2$ max edges.

Shortest Path Algorithm.

1- Single Source . Single destination SP problem.
2- Single-pair shortest-path problem Lahore to all cities
3- All-pairs // each city to every city



Optimul Substructure.
Relaxation.

24-01-2022 Monday.

Lecture # 26. Chp#24.

Bellman Ford Algorithm.

Quiz 2 | 15, 16, 22

→ More powerful than Dijkstra.

Longest simple path shortest path - ہو سکتا ہے
V بار پر جا کر اگر relax ہو لو اس کا مطلب ہے کہ
cycle-ve ہے - گراف V-1 پر relax ہو سکتا ہے تبھی اس میں
path Shortest ہے - ملے گا۔

15
15
22
23
24
final

Bellman Ford -ve edge کو
- بتا دیتا ہے path shortest

Time complexity :- O(VE)

Initialize = O(V)
1st for = (V-1)E
2nd for = E
O(V) + (V-1)E + E
VE = O(VE)

DAG

→ IF DAG no need for Bellman Ford.

(V+1) topological
V initialize
E for loop

O(V+E) = Time complexity.

Dijkstra Algorithm: no -ve edge
O(E log V)