Module - 4

Dynamic programming

Nithin Kumar
Asst. prof Dept of CSSE
VVCE, Mysuru

We know that divide-and-Conquer technique is used to solve the problems that can be divided into independent subproblems. On the otherhand, dynamic programming is one such strategy that can be used to solve the problems having dependent subproblems.

That is, in case of some problems, their subproblems are shared & they cannot be solved independently.

→ Consider a problem of finding n^{th} fibonacci number. The formula is given by

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

with initial conditions $F(0) = 0$ & $F(1) = 1$

Hence, if we try to solve $F(n-1)$ that will contain a term $F(n-2) + F(n-3)$ so $F(n)$ & its subproblem $F(n-1)$ are sharing another subproblem $F(n-2)$. Thus calculating these repeated terms is simply waste of time.

→ Instead of solving overlapping subproblems again & again, Dynamic programming suggests solving each of the smaller subproblems only once & recording the results in a table from which we can obtain a soln for original problem.

* Multistage Graphs :-

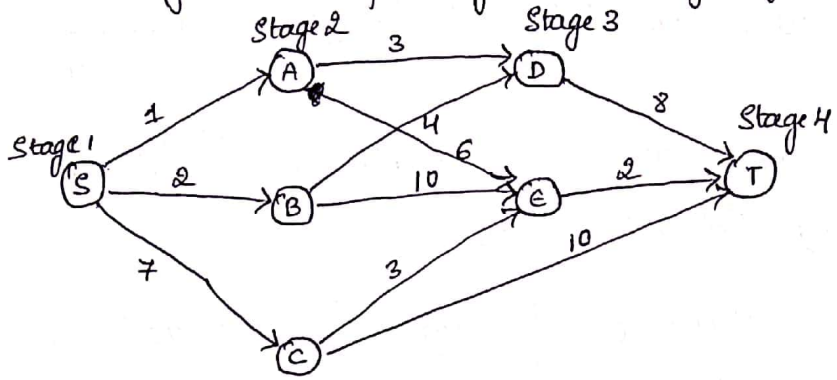
Multistage graph is a directed graph in which the nodes can be divided into a set of stages (k stages) such that all edges are from a stage to next stage only.

In this graph all the vertices are partitioned into the "k" stages where $k >= 2$.

→ In Multistage graph problem we have to find the Shortest path from "Source" to "Sink".

The Cost of a path from Source (denoted by S) to Sink (denoted by T) is the Sum of the Costs of Edges on the path.

→ Consider the foll Example of Multistage graph :



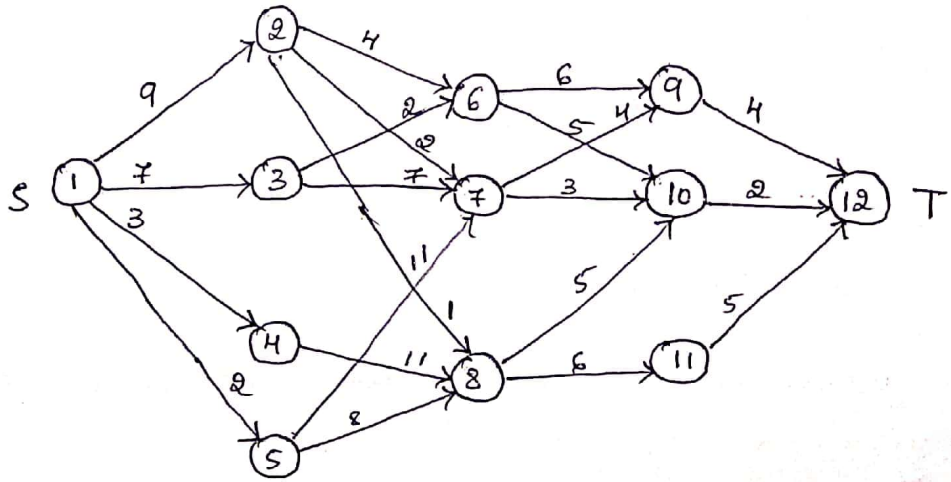
S = Source
T = Sink

→ Multistage graph problem can be solved to find Shortest path from Source to Sink Using

* Forward Approach
* Backward Approach

→ Using dynamic Approach, the Multistage graph problem is Solved. This is because in Multistage graph problem we obtain the Minimum path at Each Current Stage by Considering the path length of Each Vertex obtained in Earlier Stage.

* Solve the given Multistage graph problem using forward Approach to find the Shortest path from Source to Sink.



→ Firstly, make a table to indicate "vertex v", "distance of vertex cost(min)" & "reachable through vertex 'd'"

V	1	2	3	4	5	6	7	8	9	10	11	12
cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2/3	7	6	8	8	10	10	10	12	12	12	12

Cost of Sink i.e 12 is 0.

→ Next, take Every Individual vertex at a time & update Min distance in table with through which vertex (d).

*Stage 4

* Cost(4,9) = 4 * cost(4,10) = 2 * cost(4,11) = 5

*Stage 3

* Cost(3,6) = min{ Cost(6,9) + cost(9), Cost(6,10) + Cost(10)}
= min{ 6+4, 5+2 } = 7// ∵ d = 10

* Cost(3,7) = min{ Cost(7,9) + cost(9), Cost(7,10) + Cost(10)}
= min{ 4+4, 3+2 } = 5// ∵ d = 10

* Cost(3,8) = min{ Cost(8,10) + Cost(10), Cost(8,11) + Cost(11)}
= min{ 5+2, 6+5 } = 7// ∵ d = 10

*Stage 2

* Cost(2,2) = min{ Cost(2,6) + cost(6), Cost(2,7) + cost(7), Cost(2,8) + Cost(8)}
= min{ 4+7, 2+5, 1+7 } = 7// ∵ d = 7

* Cost(2,3) = min{ Cost(3,6) + cost(6), Cost(3,7) + Cost(7)}
= min{ 2+7, 7+5 } = 9// ∵ d = 6

* Cost(2,4) = min{ Cost(4,8) + Cost(8) }
= min{ 11 + 7 } = 18// ∵ d = 8

* Cost(2,5) = min{ Cost(5,7) + cost(7), Cost(5,8) + Cost(8)}
= min{ 11 + 5, 8 + 7 } = 15// ∵ d = 8

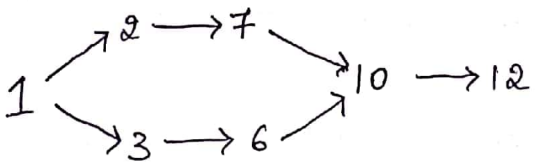
* Stage 1

* Cost (1,2) = min { cost(1,2) + cost(2), cost(1,3) + cost(3), cost(1,4) + cost(4)
, cost(1,5) + cost(5) }

= min { 9+7, 7+9, 3+18, 2+15 }

= 16 // ∵ d = 2 & 3

→ finally, Backtrack the path from Source '1' using final 'd' values i.e '2' & '3'

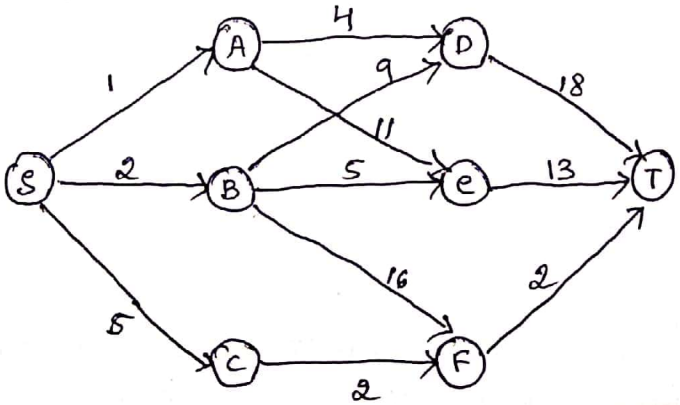


& Total Cost = 16 //

* Algorithm Multistage Graph (Forward Approach)

```
Multistage Graph (er, k, n)
{
  cost[n] = 0.0;
  For (int j = n-1; j >= 1; j--)
      {
        cost[j] = cost[j][r] + cost[r]
        D[j] = r
      }
}
```

* Apply forward Approach to find Shortest path from 'S' to 'T' for given graph



→ Firstly, make a table to indicate "Vertex V", "distance (min cost)" & "Reachable Vertex d"

V	S	A	B	C	D	E	F	T
cost	9	22	18	4	18	13	2	0
d	C	D	E/F	F	T	T	T	T

Cost of Sink (T) is '0'.

→ Next, take Every Individual Vertex at a time & update Min distance in table with Reachable Vertex (d).

* Stage 3 :-

* Cost (3, D) = 18// * Cost (3, E) = 13// * Cost (3, F) = 2//

* Stage 2 :-

* Cost (2, A) = min { Cost (A, D) + Cost (D), Cost (A, E) + Cost (E) }
= min { 4 + 18, 11 + 13 } = 22// ∵ d = D

* Cost (2, B) = min { Cost (B, D) + Cost (D), Cost (B, E) + Cost (E), Cost (B, F) + Cost (F) }
= min { 9 + 18, 5 + 13, 16 + 2 } = 18// ∵ d = E & F

* Cost (2, C) = Cost (C, F) + Cost (F)
= 2 + 2 = 4// ∵ d = F

* Stage 1 :-

* Cost (1, S) = min { Cost (S, A) + Cost (A), Cost (S, B) + Cost (B), Cost (S, C) + Cost (C) }
= min { 1 + 22, 2 + 18, 5 + 4 } = 9// ∵ d = C

→ Finally, Backtrack the path from Source 'S' using final 'd' Value 'C'

S → C → F → T & Total Cost = 9//

* Knapsack problem using Dynamic programming :-

Consider a knapsack problem of finding the most valuable subset of n items of weights $w_1 \ldots w_n$ & values $v_1 \ldots v_n$ that fit into a knapsack of capacity 'M'.

→ The dynamic programming strategy for solving this problem requires to derive a recurrance relation that expresses a soln to an instance of knapsack problem in terms of soln to its smaller sub instance.

→ Consider an instance of a problem with first 'i' items having weights $w_1 \ldots w_i$ & values $v_1 \ldots v_i$ & knapsack capacity is 'M'

We have the following two conditions

$$* V[i,j] = \begin{cases} \max\{V[i-1,j],\ v_i + V[i-1, j-w_i]\} & \text{if } j - w_i \geq 0 \\ V[i-1,j] & \text{if } j - w_i < 0 \end{cases}$$

With the initial condition

$* V[0,j] = 0$ & $* V[i,0] = 0$

Our requirement is to find $V[n, M]$ based on the above relation.

* Algorithm Dynamic Knapsack (n, M)

```
while (V[i,j] ≠ 0)
{
    if j < wi
        Val ← V[i-1, j]
    else
        Val ← max{V[i-1, j], vi + V[i-1, j - wi]}
    V[i,j] ← Val
Return V[i,j]
```

* Consider the following problem with three items & the knapsack of capacity W=4, the weights & values are as shown

Item	Weight	Value
A	3	25
B	1	20
C	2	40

⇒ M=4 $w_1=3$ $w_2=1$ $w_3=2$ $v_1=25$ $v_2=20$ $v_3=40$

→ Firstly, Create a table in which dimension i → indicates no of objects including '0' & j → indicates capacity of knapsack including '0'

i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	25	25
2	0	20	20	25	45
3	0	20	40	60	(60)

fill the first row & column with '0's, As we know that V[0,j] = 0 & V[i,0] = 0

→ Now for remaining values perform computation based on Relation

$$*V[i,j] = \begin{cases} \max\{V[i-1,j],\ v_i + V[i-1, j-w_i]\} & j-w_i \geq 0 \\ V[i-1,j] & j-w_i < 0 \end{cases}$$

* **i=1 :-** i=1 & $w_1=3$ $v_1=25$

* V[1,1] = V[i-1,j] = V[0,1] = 0 // ∵ $j-w_i = 1-3 < 0$

* V[1,2] = V[i-1,j] = V[0,2] = 0 // ∵ $j-w_i = 2-3 < 0$

* V[1,3] = max{V[i-1,j], v_i + V[i-1, j-w_i]} ∵ $j-w_i = 3-3 = 0$
= max{V[0,3], 25 + V[0,0]} = 25 //

* V[1,4] = max{V[i-1,j], v_i + V[i-1, j-w_i]} ∵ $j-w_i > 0$ 4-3 = 1
= max{V[0,4], 25 + V[0,1]} = 25

* $i=2$:-

$i=2 \quad w_2=1 \quad v_2=20$

* $V[2,1] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[1,1], 20+V[1,0]\} = \max\{0, 20+0\}$
$= 20$ //

* $V[2,2] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[1,2], 20+V[1,1]\} = \max\{0, 20+0\}$
$= 20$ //

* $V[2,3] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[1,3], 20+V[1,2]\} = \max\{25, 20+0\}$
$= 25$ //

* $V[2,4] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[1,4], 20+V[1,3]\} = \max\{25, 20+25\}$
$= 45$ //

* $i=3$:-

$i=3 \quad w_3=2 \quad v_3=40$

* $V[3,1] = V[i-1,j] = V[2,1] = 20$ //

* $V[3,2] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[2,2], 40+V[2,0]\} = \max\{20, 40+0\}$
$= 40$ //

* $V[3,3] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[2,3], 40+V[2,1]\} = \max\{25, 40+20\}$
$= 60$ //

* $V[3,4] = \max\{V[i-1,j], v_i + V[i-1, j-w_i]\}$
$= \max\{V[2,4], 40+V[2,2]\} = \max\{45, 40+20\}$
$= 60$ //

→ Here, we have

$V[n,W] = V[3,4] = 60$

∴ By observing the table, the soln is $\{2,3\}$ or $\{B,C\}$ with the profit 60.

* Bellman - Jord Algorithm :-

Bellman - Jord algorithm Solves the Single - Source Shortest path problem In the general case In which Edge weight may be negative.

→ Given a weighted, directed graph $G = (V, E)$ with Source S & weight function w, the Bellman - Jord algorithm returns a boolean value Indicating whether or not there Is a negative weight cycle that Is reachable Jrom the Source.

→ If there Is Such negative weight cycle, the algorithm Indi -cates that no Solution Exists.

If there Is no Such cycle, the algorithm produces the Shor -test paths & their weights.

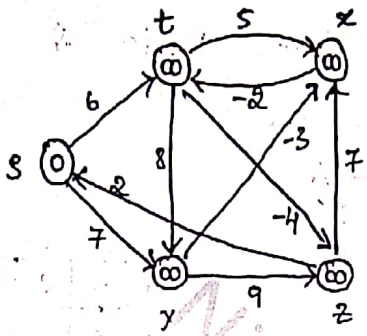
* BELLMAN_FORD (G, w, S)

1. Initialize_Single_Source (G, S)
2. Jor $i = 1$ to $|G.V| - 1$
3. Jor Each Edge $(u,v) \in G.E$
4. RELAX (u, v, w)
5. Jor Each Edge $(u,v) \in G.E$
6. If $v.d > u.d + w(u,v)$
7. return FALSE
8. return TRUE.

The Algorithm Relaxes Edges, progressively decreasing an Estimate $v.d$ on the weight of a Shortest path δ Jrom the Source S to Each vertex $v \in V$ until It achieves the actual Shortest path weight $\delta(S, V)$.

→ The Algorithm returns TRUE If and only If the graph Contains no negative - weight cycle Jrom Source.

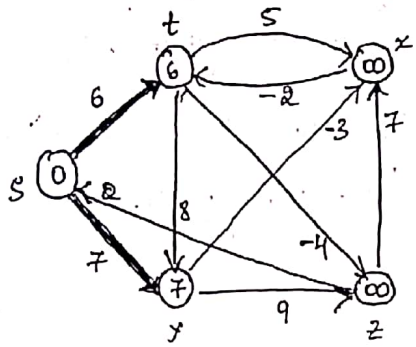
→ Consider the Joll Example.



	s	t	x	y	z
d	0	∞	∞	∞	∞
π	/	/	/	/	/

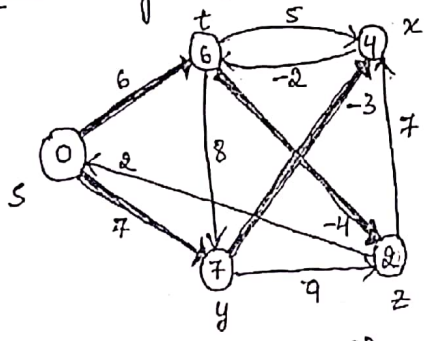
Using vertex 's' as the source, we Initialize all other distance to ∞

* Iteration 1:- Edge (s,t) & (s,y) relax updating the distances 6 & 7



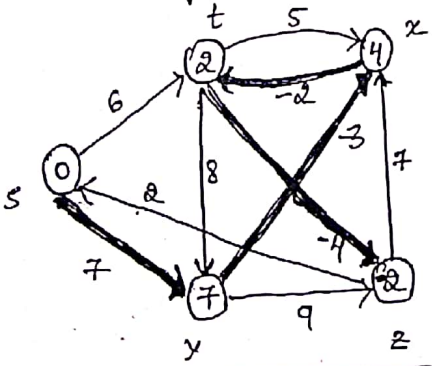
	s	t	x	y	z
d	0	6	∞	7	∞
π	/	s	/	s	/

* Iteration 2:- Edges (y,x) & (t,z) relax updating distances 4 & 2



	s	t	x	y	z
d	0	6	4	7	2
π	/	s	y	s	t

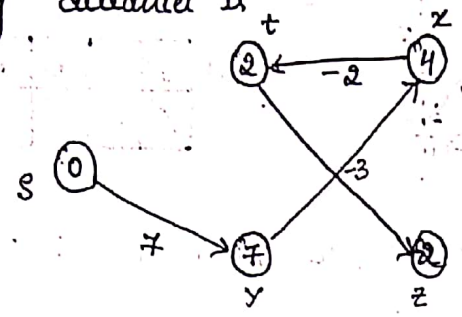
* Iteration 3:- Edge (x,t) & (t,z) relax updating distance 2



	s	t	x	y	z
d	0	2	4	7	-2
π	/	x	y	s	t

* Iteration 4: No Edges relax

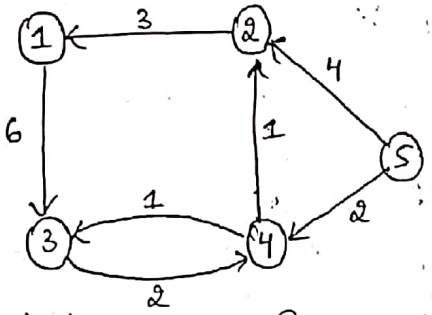
→ The final Shortest path from vertex S with corresponding distances is



	S	t	x	y	z
d	0	2	4	7	-2
π	/	x	y	S	t

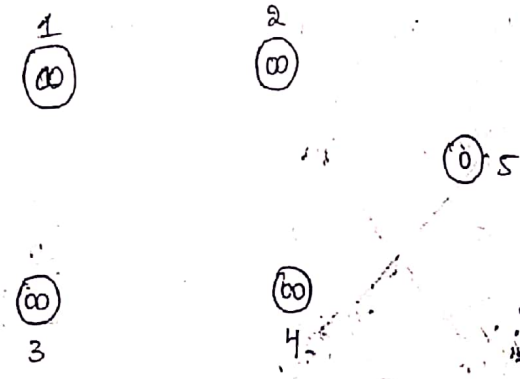
We now check the relaxation condition. one additional time for each edge. If any of the checks pass then there exists a negative weight cycle in the graph.

* Apply the Bellman-Ford algorithm to find Single Source Shortest path for the following directed graph.



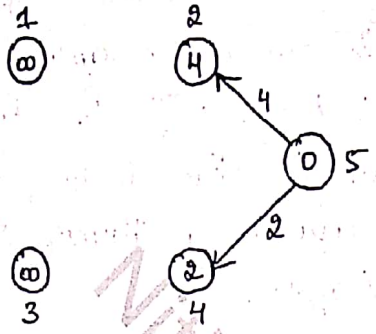
Using vertex 'S' as the Source Vertex

⇒ Using vertex S as the Source, we initialize all the other distances to ∞



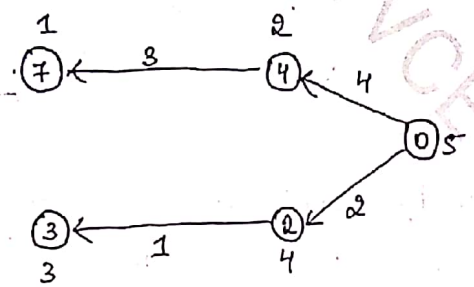
	1	2	3	4	5
d	∞	∞	∞	∞	0
π	/	/	/	/	/

→ Iteration 1 :- Edges (5, 2) & (5, 4) relax updating the distances to 2 & 4.



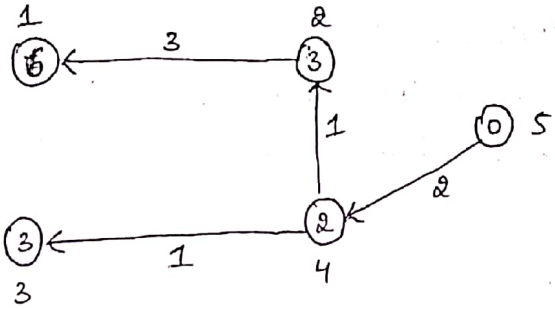
	1	2	3	4	5
d	∞	4	∞	2	0
π	/	5	/	5	/

→ Iteration 2 :- Edges (2, 1) & (4, 3) relax updating the distances to 3 & 1



	1	2	3	4	5
d	7	4	3	2	0
π	2	5	2	5	/

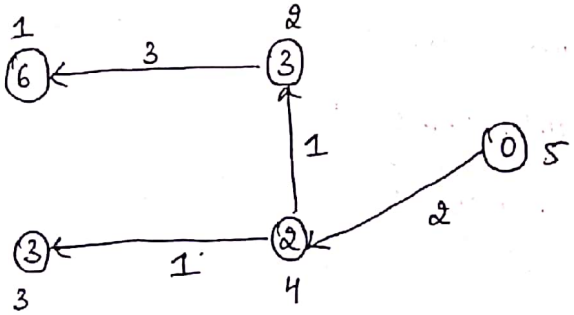
→ Iteration 3 :- Edges (4, 2) & (2, 1) relax updating 1 & 3



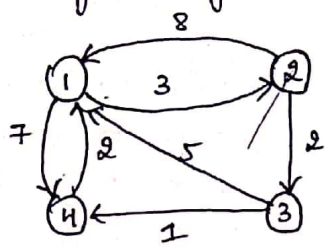
	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

→ Iteration 4 :- No Edges relax

→ The final Shortest path from vertex 5 with corresponding distance is



* Apply* floyd's Algo to find all pair Shortest path



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$A^k[i,j] = \min \{ A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j] \}$

* A^1 =
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix} \end{array}$$

$A[2,3] = \min \{ 2, A[2,1] + A[1,3] \} = \{ 2, 8+\infty \} = 2$
$A[2,4] = \min \{ \infty, A[2,1] + A[1,4] \} = \{ \infty, 8+7 \} = 15$ //
$A[3,2] = \min \{ \infty, A[3,1] + A[1,2] \} = \{ \infty, 5+3 \} = 8$ //
$A[3,4] = \min \{ 1, A[3,1] + A[1,4] \} = \{ 1, 5+7 \} = 1$
$A[4,2] = \min \{ \infty, A[4,1] + A[1,2] \} = \{ \infty, 2+3 \} = 5$
$A[4,3] = \min \{ \infty, A[4,1] + A[1,3] \} = \{ \infty, 2+\infty \} = \infty$

* A^2 =
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A[1,3] = \min \{ \infty, A[1,2] + A[2,3] \} = \{ \infty, 3+2 \} = 5$ //
$A[1,4] = \min \{ 7, A[1,2] + A[2,4] \} = \{ 7, 3+15 \} = 7$
$A[3,1] = \min \{ 5, A[3,2] + A[2,1] \} = \{ 5, 8+8 \} = 5$
$A[3,4] = \min \{ 1, A[3,2] + A[2,4] \} = \{ 1, 8+15 \} = 1$ //
$A[4,1] = \min \{ 2, A[4,2] + A[2,1] \} = \{ 2, 5+8 \} = 2$ //
$A[4,3] = \min \{ \infty, A[4,2] + A[2,3] \} = \{ \infty, 5+2 \} = 7$ //

* A^3 =
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A[1,2] = \min \{ 3, A[1,3] + A[3,2] \} = \{ 3, 5+8 \} = 3$
$A[1,4] = \min \{ 7, A[1,3] + A[3,4] \} = \{ 7, 5+1 \} = 6$ //
$A[2,1] = \min \{ 8, A[2,3] + A[3,1] \} = \{ 8, 2+5 \} = 7$ //
$A[2,4] = \min \{ 15, A[2,3] + A[3,4] \} = \{ 15, 2+1 \} = 3$ //
$A[4,1] = \min \{ 2, A[4,3] + A[3,1] \} = \{ 2, 7+5 \} = 2$
$A[4,2] = \min \{ 5, A[4,3] + A[3,2] \} = \{ 5, 7+8 \} = 5$ //

* A^4 =
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$A[1,2] = \min \{ 3, A[1,4] + A[4,2] \} = \{ 3, 6+5 \} = 3$
$A[1,3] = \min \{ 5, A[1,4] + A[4,3] \} = \{ 5, 6+7 \} = 5$
$A[2,1] = \min \{ 7, A[2,4] + A[4,1] \} = \{ 7, 3+2 \} = 5$ //
$A[2,3] = \min \{ 2, A[2,4] + A[4,3] \} = \{ 2, 3+7 \} = 2$
$A[3,1] = \min \{ 5, A[3,4] + A[4,1] \} = \{ 5, 1+2 \} = 3$ //
$A[3,2] = \min \{ 8, A[3,4] + A[4,2] \} = \{ 8, 1+5 \} = 6$ //

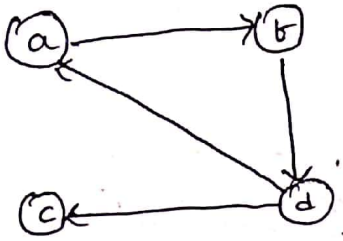
$$\therefore A = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

* **Warshall's Algo:** To find Transitive closure

$R^{(k)}[i,j] \leftarrow R^{(k-1)}[i,j]$ OR $R^{(k-1)}[i,k]$ & AND $R^{(k-1)}[k,j]$

R^0:

→ Consider the foll Example



R^0 =

	1 a	2 b	3 c	4 d
1 a	0	1	0	0
2 b	0	0	0	1
3 c	0	0	0	0
4 d	1	0	1	0

R^0 =

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	0	0	0
4	1	0	1	0

R_0 =

0	1	0	0
0	0	1	0
0	0	0	0
1	1	1	0

R^1 =

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	0	0	0
4	1	1	1	0

$R(2,2) = 0$ OR $R[2,1]$ AND $R[1,2]$
$= 0$ OR 0 AND $1 = 0$

$R(2,4) = 0$ OR $R[2,1]$ AND $R[2,4] = 0$ OR 0 AND $0 = 0$

$R(3,2) = 0$ OR $R[3,1]$ AND $R[1,2] = 0$ OR 0 A $1 = 0$

$R(3,3) = 0$ OR $R[3,1]$ AND $R[1,3] = 0$ OR 0 A 0

$R(3,4) = 0$ OR $R[3,1]$ AND $R[1,4] = 0$ OR 0 A 0

$R[4,2] = 0$ OR $R[4,1]$ AND $R[1,2] = 0$ OR 1 A 1

$R[4,4] = 0$ OR $R[4,1]$ AND $R[1,4] = 0$ OR 1 A $0 = 0$

R^2 =

	1	2	3	4
1	0	1	1	1
2	0	0	0	1
3	0	0	0	0
4	1	1	1	1

R^3 =

	1	2	3	4
1	0	1	1	1
2	0	0	0	1
3	0	0	0	0
4	1	1	1	1

R^4 =

	1	2	3	4
1	1	1	1	1
2	1	1	1	1
3	0	0	0	0
4	1	1	1	1

$R^{(0)} \leftarrow A$

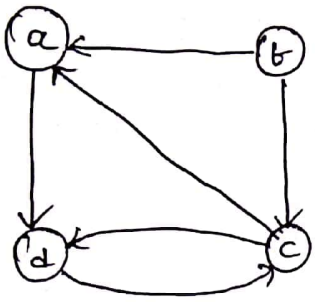
for $k \leftarrow 1$ to n do

  for $i \leftarrow 1$ to n do

    for $j \leftarrow 1$ to n do

$R^k[i,j] \leftarrow R^{(k-1)}[i,j]$ OR $R^{(k-1)}[i,k]$ AND $R^{(k-1)}[k,j]$ //

return $(R^{(n)})$

* 

$R^0 =$

	1	2	3	4
1	0	0	0	1
2	1	0	1	0
3	1	0	0	1
4	0	0	1	0

$R^1 =$

	1	2	3	4
1	0	0	0	1
2	1	0	1	1
3	1	0	0	1
4	0	0	1	0

$R[2,4] = 0$ OR $R[2,1]$ AND $R[1,4]$

$= 0$ OR 1 AND 1

$= 1$ //

$R^2 =$

	1	2	3	4
1	0	0	0	1
2	1	0	1	1
3	1	0	0	1
4	0	0	1	0

$R^3 =$

	1	2	3	4
1	0	0	0	1
2	1	0	1	1
3	1	0	0	1
4	1	0	1	1

$R[4,4] = 0$ OR $R[4,3]$ AND $R[3,4]$

$= 0$ OR 1 AND $1 = 1$ //

$R^4 =$

	1	2	3	4
1	1	0	1	1
2	1	0	1	1
3	1	0	1	1
4	1	0	1	1

//

* Traveling Salesman problem

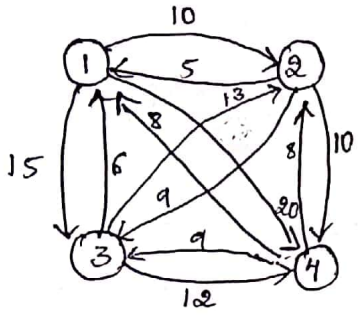
Travelling Sales Man problem consists of Salesman & a set of cities. The Salesman has to visit each city starting from home city & return to the same city.

→ The person wants to minimize the total length of the trip.

→ Let $g(i,s)$ be the length of shortest path starting at vertex i, going through all vertices in s & terminating at vertex 1

* $g(i,s) = \min_{j \in s} \{ c_{ij} + g(j, s-\{j\}) \}$ //

→ Consider the following Example



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

→ * Firstly for set $\{\phi\}$

$g\{i,\phi\} = c_{i1} \Rightarrow g\{1,\phi\} = c_{11} = 0 \quad g\{3,\phi\} = c_{31} = 6$

$g\{2,\phi\} = c_{21} = 5 \quad g\{4,\phi\} = c_{41} = 8$

→ Next for all other vertices from src vertex $\{1\}$

* $g(1,\{2,3,4\}) = \min \{ c_{12} + g(2,\{3,4\}), c_{13} + g(3,\{2,4\}), c_{14} + g(4,\{2,3\}) \}$

$= \min \{ 10+25, 15+25, 20+23 \} = 35$ //

* $g(2,\{3,4\}) = \min \{ c_{23} + g(3,\{4\}), c_{24} + g(4,\{3\}) \}$

$= \min \{ 9 + g(3,\{4\}), 10 + g(4,\{3\}) \}$

$= \min \{ 9+20, 10+15 \}$

$= 25$ //

* $g(3,\{4\}) = \min\{C_{34} + g(4,\phi)\}$
$= 12 + 8 = 20$ //

* $g(4,\{3\}) = \min\{C_{43} + g(3,\phi)\}$
$= 9 + 6 = 15$ //

→ $g(3,\{2,4\}) = \min\{C_{32} + g(2,\{4\}),\ C_{34} + g(4,\{2\})\}$
$= \min\{13 + g(2,\{4\}),\ 12 + g(4,\{2\})\}$
$= \{13 + 18,\ 12 + 13\}$
$= 25$

* $g(2,\{4\}) = \min\{C_{24} + g(4,\phi)\}$
$= 10 + 8 = 18$ //

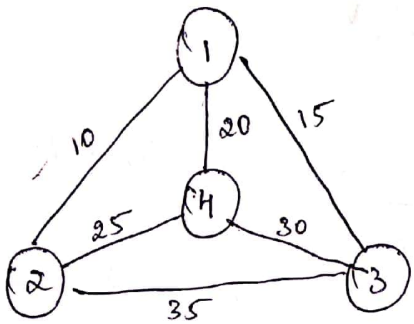
* $g(4,\{2\}) = \min\{C_{42} + g(2,\phi)\}$
$= \min\{8 + 5\} = 13$ //

* $g(4,\{2,3\}) = \min\{C_{42} + g(2,\{3\}),\ C_{43} + g(3,\{2\})\}$
$= \min\{8 + g(2,\{3\}),\ 9 + g(3,\{2\})\}$
$= 8 + 15,\ 9 + 18$
$= 23$

* $g(2,\{3\}) = \min\{C_{23} + g(3,\phi)\}$
$= 9 + 6 = 15$

* $g(3,\{2\}) = \min\{C_{32} + g(2,\phi)\}$
$= 13 + 5 = 18$

∴ $1 \to 2 \to 4 \to 3 = 35$ //



* Optimal Binary Search Trees

BST is one of the most important data Structures in Computer Science.

One of its principal Applns is to Implement a dictionary, a Set of Elements with the operations of Searching, Insertion & deletion.

→ If probabilities of Searching for Elements of a Set are known, we can Minimize the Average no of Comparisons in a Successful Search by Constructing optimal BST.

* Consider the following Example & Construct optimal BST

	0	1	2	3
Node :-	10	12	16	21
Freq :-	4	2	6	3

→ Firstly, Construct a perfect Square Matrix Based on no of nodes. In the above Example we've 4 nodes

	0	1	2	3
0				
1				
2				
3				

we come across diff cases based on no of nodes/keys

* Case I :- $l = 1$, Only one node is taken at a time & fill the values

$(0,0) = 10 = 4$

$(1,1) = 12 = 2$

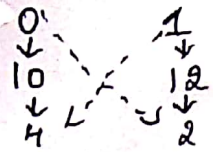
$(2,2) = 16 = 6$

$(3,3) = 21 = 3$

	0	1	2	3
0	4			
1		2		
2			6	
3				3

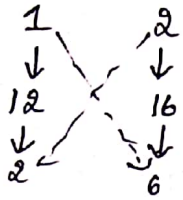
* Case 2 :- $l = 2$, choose two nodes at a time

*



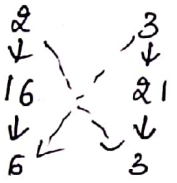
$\rightarrow 4+2+\min \begin{cases} 2 \leftarrow 0 \\ 4 \leftarrow 1 \end{cases} = 8^{(0)} = (0,1)$

*



$\rightarrow 6+2+\min \begin{cases} 2 \leftarrow 2 \\ 6 \leftarrow 1 \end{cases} = 10^{(2)} = (1,2)$

*



$\rightarrow 6+3+\min \begin{cases} 6 \leftarrow 3 \\ 3 \leftarrow 2 \end{cases} = 12^{[2]} = (2,3)$

→ Finally update the Matrix

	0	1	2	3
0	4	$8^{(0)}$	$26^{(2)}$	
1		2	$10^{(2)}$	$16^{[2]}$
2			6	$12^{(2)}$
3				3

* Case 3 :- $l = 3$, Choose three nodes at a time

*

0	1	2
↓	↓	↓
10	12	16
↓	↓	↓
4	2	6

$\rightarrow 4+2+6+\min \begin{cases} 10 \leftarrow 0 \\ 4+6 \leftarrow 1 \\ 8 \leftarrow 2 \end{cases} = 20^{[2]}$

*

1	2	3
↓	↓	↓
12	16	21
↓	↓	↓
2	6	3

$\rightarrow 2+6+3+\min \begin{cases} 12 \leftarrow 1 \\ 2+3 \leftarrow 2 \\ 10 \leftarrow 3 \end{cases} = 16^{[2]}$

* Case 4 :- $l = 4$, Choose all four nodes at a time

*

0	1	2	3
↓	↓	↓	↓
10	12	16	21
↓	↓	↓	↓
4	2	6	3

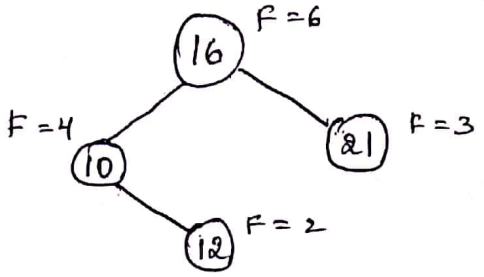
$\rightarrow 4+2+6+3+\min \begin{cases} 16 \leftarrow 0 \\ 4+12 \leftarrow 1 \\ 8+3 \leftarrow 2 \\ 20 \leftarrow 3 \end{cases}$

$= 26^{[2]}$ //

→ Finally the Matrix looks like this

	0	1	2	3
0	4	$8^{(0)}$	$20^{(1)}$	$26^{[2]}$
1		2	$10^{(2)}$	$16^{[2]}$
2			6	$12^{[2]}$
3				3

$= 26^{[2]}$ ∴ root = 2



$\Rightarrow 6\times1 + 4\times2 + 3\times2 + 2\times3$

$= 26$ //

* Construct a optimal Binary ST for the given set

key : A B C D

prob : 0.1 0.2 0.4 0.3

	0	1	2	3
0	0.1	0.4	1.1	1.7
1		0.2	0.8	1.4
2			0.4	1.0
3				0.3