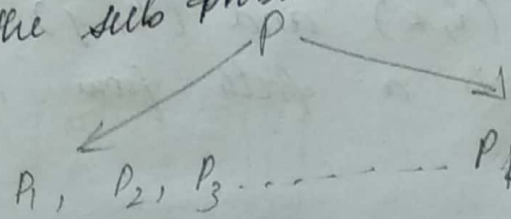
Module - 4

Dynamic Programming.

- This technique also divides the problem into sub-problems where
  (1) sub problems are dependent / overlapping
  (2) Solution of the last sub problem is the sol of original problem
  (3) Solution of every sub problem is saved which can avoid resolving of similar instance of the sub-problem.



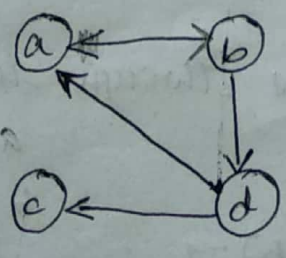
Warshall's Algorithm

This is used to find path from one node to another node. Generally, Warshall's algorithm is used to obtain path matrix (transitive closure) of a given directed, unweighted graph.

If $G = (V, E)$ is a graph of n vertices then, path matrix is a matrix of size $n \times n$

where $P[i, j] = \begin{cases} 1 & \text{if there is a path from } i \text{ to } j \\ 0 & \text{if there is no path} \end{cases}$

Example:



Adj. Matrix (4 x 4)

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

path matrix (4×4)

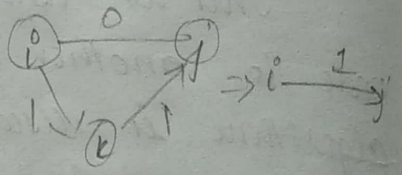
	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

Starting from A I can reach B, C, D

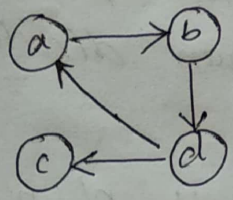
Logic behind Warshall's Algorithm

- If, $P[i, j] = 1$ it means that there is a path from i to j
- If, $P[i, j] = 0$ it means that there is no path from i to j but, if there exist a path from (i to k) (i, k) and (k, j) then it implies that there exists a path from i to j (i, j)

Eg: Apply Warshall's algorithm to obtain transitive closure for the given graph.



Graph



Adjacency Matrix is:

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

i/p is: Adjacency Matrix (first sub problem)

Step 1: Consider the paths through the vertex a

(i, k) (k, j) => (i, j)

(d, a) = 1 & (a, b) = 1 => (d, b) = 1

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

Since we are finding path through the vertex 'a' consider a^{th} row & a^{th} column from the above adjacency matrix and also consider only the pair with entries 1.

Here 'k' is nothing but vertex 'a' and also we know that if $(i,k)=1$ & $(k,j)=1$ then $(i,j)=1$

Since $(d,a)=1$ & $(a,b)=1$ then $(d,b)=1$

Step2: Consider the paths through the vertex 'b' obtained consider b^{th} row & b^{th} column of the matrix in step 1.

$(a,b)=1$ & $(b,d)=1$ $\therefore (a,d)=1$

$(d,b)=1$ & $(b,d)=1$ $\therefore (d,d)=1$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

Step3: paths through vertex 'c' consider c^{th} row & c^{th} column of the matrix obtained in step 2.

Note that $(d,c)=1$ but, all the entries in c^{th} row are 0. which means that there is no path through the vertex c

$\therefore$ The resultant matrix is unaltered.

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

Step4: path through vertex 'd'. consider d^{th} row and d^{th} column of the matrix obtained in step 3

Start from dth column, combine with row.

$(a,d)=1 \wedge (d,a)=1 \therefore (a,a)=1$
$(a,d)=1 \wedge (d,b)=1 \therefore (a,b)=1$
$(a,d)=1 \wedge (d,c)=1 \therefore (a,c)=1$
$(a,d)=1 \wedge (d,d)=1 \therefore (a,d)=1$

$(b,d)=1 \wedge (d,a)=1 \therefore (b,a)=1$
$(b,d)=1 \wedge (d,b)=1 \therefore (b,b)=1$
$(b,d)=1 \wedge (d,c)=1 \therefore (b,c)=1$
$(b,d)=1 \wedge (d,d)=1 \therefore (b,d)=1$

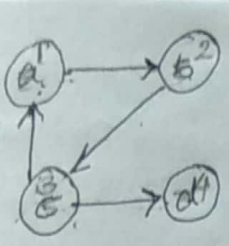
$(d,d)=1 \wedge (d,a)=1 \therefore (d,a)=1$
$(d,d)=1 \wedge (d,b)=1 \therefore (d,b)=1$
$(d,d)=1 \wedge (d,c)=1 \therefore (d,c)=1$

The matrix with the updated values

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

Finally the path matrix is given below.

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

2. 

The adjacency matrix is

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

Step1 : Consider a path through vertex '1'

(3, 1) = 1 & (1, 2) = 1 => (3, 2) = 1

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

Step2 : consider a path through vertex '2'

(1, 2) = 1 & (2, 3) = 1 => (1, 3) = 1

(3, 2) = 1 & (2, 3) = 1 => (3, 3) = 1

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

Step3 : Consider a path through vertex '3'

(1, 3) = 1 & (3, 1) = 1 => (3,3)=1 (1,1)=1

(1, 3) = 1 & (3, 2) = 1 => (3,3)=1 (1,2)=1

(1, 3) = 1 & (3, 3) = 1 => (1, 3) = 1

(1, 3) = 1 & (3, 4) = 1 => (1, 4) = 1

(2, 3) = 1 & (3, 1) = 1 => (2,1)=1

(2, 3) = 1 & (3, 2) = 1 => (2,2)=1

(2, 3) = 1 & (3, 3) = 1 => (2,3)=1

(2, 3) = 1 & (3, 4) = 1 => (2,4)=1

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

Step 4: All the entries in 4th row is 0 which means there is no path through 4.

Finally path Matrix is

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

Warshall's Algorithm.

A: Adjacency Matrix
P: path Matrix

```
Algorithm Warshall (A[1...n][1...n], x)
{   // Initialisation
    for i ← 1 to n do
        for j ← 1 to n do
            P[i,j] = A[i,j];

    // Checking paths
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                if (P[i,j]==0 && (P[i,k]==1 && P[k,j]==1))
                    P[i,j] = 1;
            End for j
        End for i
    End for k
    return P;
}
```

$i \xrightarrow{0} j \Rightarrow i \xrightarrow{1} j$ (via $i \to k \to j$)

for every k check (i,k) corresponds for i

Note

→ Initially the path matrix will be same as the adjacency matrix.

→ for every intermidiate vertex 'k' ranging from 1 ... n we need to check the condition of transitive closure for every pair (i,j) both ranging from 1 ... n. Hence, for loop w.r.t to 'k' is the outermost for loop, followed with i and j for loop.

Analysis

→ I/p size : n (no of vertices)

→ Basic operation : condition within 'if'

P[i,j]==0 && (P[i,k]==1 && P[k,j]==1)

→ No of times B.O executed

$$T(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} 1$$

$$= \sum_{k=1}^{n} \sum_{i=1}^{n} (n-1+1)$$

$$= \sum_{k=1}^{n} n \sum_{i=1}^{n} 1$$

$$= \sum_{k=1}^{n} n^2$$

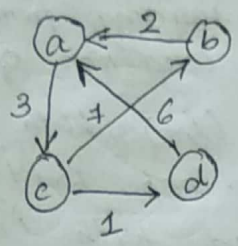
$$= n^2 \sum_{k=1}^{n} 1$$

$$= n^3$$

$\therefore T(n) \in \Theta(n^3)$

Floyd's Algorithm [All pair shortest path problem] (APSP)

- Edge, starts & ends at one vertex : 0 (self loop)
- No edge : ∞



→ Every vertex to all other vertices (self loop = 0)

Adjacency Matrix : (4×4)

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

→ Floyd's Algorithm is used to find shortest path from every vertex to all other vertices in the directed and weighted graph.

→ To represent a weighted graph we use 'cost adjacency matrix' to the following conditions.

(i) cost(i, i) = 0 // for self loops.

(ii) cost(i, j) = W i.e., assign the weight if there is a edge from i to j

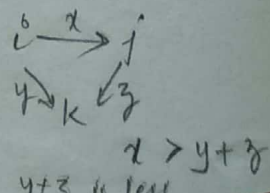
(iii) cost(i, j) = ∞ // No edge between i & j

Shortest path Matrix : (4×4)

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

C: Adjacency Matrix

D: Shortest path Matrix



x > y + z

y + z is less

Floyd's Algorithm

Algorithm Floyds (C[1...n][1...n], n)

{ for i ← 1 to n do
    for j ← 1 to n do
      D[i,j] = C[i,j];

  for k ← 1 to n do
    for i ← 1 to n do
      for j ← 1 to n do
        D[i,j] = min (D[i,j], D[i,k] + D[k,j]);
      end for j
    end for i
  end for k
  return D;

}

Analysis

→ i/p size : n (no of vertices)

→ B.O : D[i,j] = min (D[i,j], D[i,k] + D[k,j])

→ No of times B.O executed.

$$T(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} 1.$$

$$= \sum_{k=1}^{n} \sum_{i=1}^{n} (n-1+1)$$

$$= \sum_{k=1}^{n} n \sum_{i=1}^{n} 1.$$

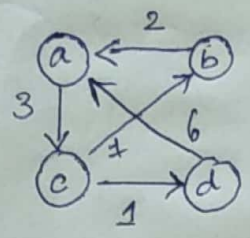
$$= n^2 \sum_{k=1}^{n} 1$$

$$= n^3$$

$$T(n) \in \Theta(n^3)$$

Problem

1. Apply Floyd's Algorithm to solve 'All Pair shortest path problem' for the given digraph



The cost adjacency matrix is as below.

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Step 1 : Consider the shortest path distance through the vertex 'a'. Hence consider a^{th} row & a^{th} column from the above cost adjacency matrix and the entries which are non zero and non infinity.

Consider column a, 2 corresponds to (b,a)

Combine with a^{th} row, 3 corresponds to (a,c)

So, $(b,a) = 2$ & $(a,c) = 3$

Hence $(b,c) = \min\{(b,c), (b,a)+(a,c)\}$

$= \min\{\infty, 2+3\}$

$\therefore \boxed{(b,c) = 5}$

Consider column club with its corresponding row

$(d,a) = 6$ & $(a,c) = 3$

Hence $(d,c) = \min\{(d,c), (d,a)+(a,c)\}$

$= \min\{\infty, 6+3\}$

$\therefore \boxed{(d,c) = 9}$

Hence the resultant matrix is.

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

Step 2 : Consider the shortest path distance through vertex b consider bth row & bth column from the matrix obtained in step 1.

So, $(c, b) = 7$ & $(b, a) = 2$

$(c, a) = \min \{ (c, a), (c, b) + (b, a) \}$

$= \min \{ \infty, 9 \}$

$\boxed{c, a = 9}$

$(c, b) = 7$ & $(b, c) = 5$

$(c, c) = \min \{ (c, 5), (c, b) + (b, c) \}$

$= \min \{ 0, 12 \}$

$\boxed{(c, c) = 0}$

Hence the resultant matrix is

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

Step 3 : Consider the shortest path ~~matri~~ distance through vertex c consider cth row & cth column from matrix obtained in step 2

So, $(a, c) = 3$ & $(c, a) = 9$

$(a, a) = \min \{ (a, a), (a, c) + (c, a) \}$

$= \min \{ 0, 12 \}$

$\boxed{(a, a) = 0}$

$(a,c) = 3$ & $(c,b) = 7$

$(a,b) = \min\{(a,b), (a,c) + (c,b)\}$

$= \min\{\infty, 10\}$

$\boxed{(a,b) = 10}$

$(a,c) = 3$ & $(c,d) = 1$

$(a,d) = \min\{(a,d), (a,c) + (c,d)\}$

$= \min\{\infty, 4\}$

$\boxed{(a,d) = 4}$

$(b,c) = 5$ & $(c,a) = 9$

$(b,a) = \min\{(b,a), (b,c) + (c,a)\}$

$= \min\{2, 14\}$

$\boxed{(b,a) = 2}$

$(b,c) = 5$ & $(c,b) = 7$

$(b,b) = \min\{(b,b), (b,c) + (c,b)\}$

$= \min\{0, 12\}$

$\boxed{(b,b) = 0}$

$(b,c) = 5$ & $(c,d) = 1$

$(b,d) = \min\{(b,d), (b,c) + (c,d)\}$

$= \min\{\infty, 6\}$

$\boxed{(b,d) = 6}$

$(d,c) = 9$ & $(c,a) = 9$

$(d,a) = \min\{(d,a), (d,c) + (c,a)\}$

$= \min\{6, 18\}$

$\boxed{(d,a) = 6}$

$(d,c) = 9$ & $(c,b) = 7$

$(d,b) = \min\{(d,b), (d,c) + (c,b)\}$

$= \min\{\infty, 16\}$

$(d,b) = 16$

$(d,c) = 9$ & $(c,d) = 1$

$(d,d) = \min\{(d,d), (d,c) + (c,d)\}$

$= \min\{0, 10\}$

$(d,d) = 0$

Hence the resultant matrix is

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

Steps: consider vertex 'd', consider dth row & dth column

~~$(d,a) = 4$ & d~~

$(a,d) = 4$ & $(d,a) = 6$

$(a,a) = \min\{0, 10\}$

$(a,a) = 0$

$(a,d) = 4$ & $(d,b) = 16$

$(a,b) = \min\{10, 20\}$

$(a,b) = 10$

$(a,d) = 4$ & $(d,c) = 9$

$(a,c) = \min\{3, 13\}$

$(a,c) = 3$

$(b,d) = 6$ & $(d,a) = 6$

$(b,a) = \min\{2, 12\}$ $(b,a) = 2$

$(b,d) = 6$ & $(d,b) = 16$

$(b,b) = \min\{0, 12\}$ $(b,b) = 0$

$(b,d) = 6$ & $(d,c) = 1$

$(b,c) = \min\{5, 7\}$ $(b,c) = 5$

$(c,d) = 1 \quad + \quad (d,a) = 6$

$(c,a) = \min\{9, 7\}$ $\boxed{(c,a) = 7}$

$(c,d) = 1 \quad + \quad (d,b) = 16$

$(c,b) = \min\{7, 17\}$ $\boxed{(c,b) = 7}$

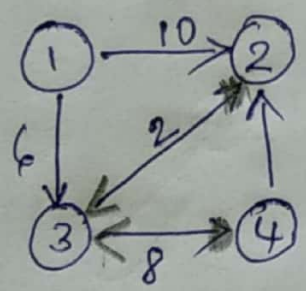
$(c,d) = 1 \quad + \quad (d,c) = 9$

$(c,c) = \min\{0, 10\}$ $\boxed{(c,c) = 0}$

Resultant matrix is

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

2. Solve the All pair shortest path problem for the given digraph



Multi-stage Graph Problem

$G=(V,E)$

→ Multi stage Graph is a directed graph where the vertices are partioned into 'k' stages where $k \geq 2$

(vertices are related)

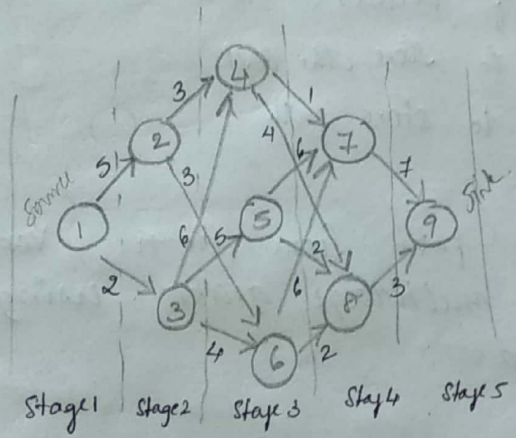
→ Every stage contains disjoint sets V_i, $1 \leq i \leq k$ (No common vertices)

→ At every stage number of vertices should not exceed 'k'

→ $|V_1| = |V_k| = 1$

(No of vertices in 1st stage) = (No of vertices in k^{th} stage)

Example



Multi stage problem is to find the shortest path from the source vertex 's' to the sink vertex 't'

To find the shortest path we have two approaches

(i) Forward approach

(ii) Backward approach.

1. FORWARD APPROACH

General procedure

n: no of vertices

```
c[n] = 0;
p[n] = 0;        'p' holds shortest path.
```

for j=n-1 down to 1 start from vertex j-1

Let r ∈ V, for each edge <j,r> ∈ E, find r: vertex ∈ V there will be a edge

minimum of a[j,r]+c[r] a[j,r] : respective weight.

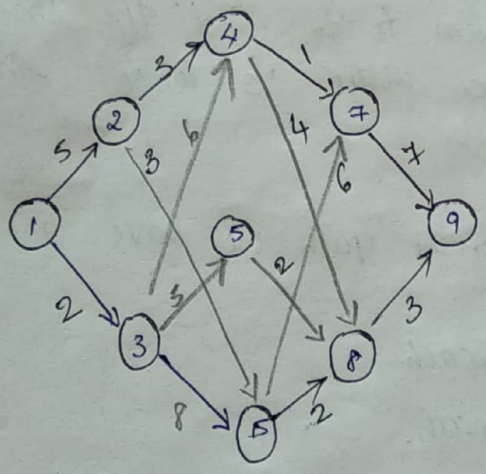
c[j] = a[j,r] + c[r];

p[j] = r

End for

→ When n: no of vertices
→ C array is used to store the cost
→ P array is used to store path

1. Obtain the shortest path from ~~1 to n~~, vertex 1 to 9 in the following multistage graph. using multistage method



Initialization : n=9, s=1, t=9, c[9]=0, p[9]=9

1) Vertex 8 : $c[8] = \min\{a[8,9] + c[9]\}$

$= \min\{3+0\}$

$\boxed{c[8]=3}$ & $\boxed{p[8]=9}$

2. Vertex 7: $C[7] = \min\{a[7,9] + C[9]\}$

$= \min\{7 + 0\}$

$C[7] = 7$ $\quad$ $P[7] = 9$

3. Vertex 6: $C[6] = \min\{a[6,7] + C[7]$ (edge from 6 to 7), $a[6,8] + C[8]\}$

$C[6] = \min\{6+7, 2+3\}$

$= \min\{13, 5\}$

$C[6] = 5$ $\quad$ $P[6] = 8$ (because it gives minimum data)

4. Vertex 5: $C[5] = \min\{a[5,7] + C[7], a[5,8] + C[8]\}$

$C[5] = \min\{6+7, 2+3\}$

$C[5] = \min\{13, 5\}$

$C[5] = 5$ $\quad$ $P[5] = 8$

5. Vertex 4: $C[4] = \min\{a[4,7] + C[7], a[4,8] + C[8]\}$

$C[4] = \min\{11 + 7, 4+3\}$

$= \min\{8, 7\}$

$C[4] = 7$ $\quad$ $P[4] = 8$

6. Vertex 3: $C[3] = \min\{a[3,4] + C[4], a[3,5] + C[5], a[3,6] + C[6]\}$

$C[3] = \min\{6+7, 5+5, 8+5\}$

$= \min\{13, 10, 13\}$

$C[3] = 10$ $\quad$ $P[3] = 5$

7. Vertex 2 : $c[2] = \min\{ a[2,4] + c[4],\ a[2,6] + c[6] \}$

$c[2] = \min\{ 3+7,\ 3+5 \}$

$= \min\{ 10, 8 \}$

$\boxed{c[2] = 8}$ $\boxed{p[2] = 6}$

8. Vertex 1 : $c[1] = \min\{ a[1,2] + c[2],\ a[1,3] + c[3] \}$

$c[1] = \min\{ 5+8,\ 2+10 \}$

$\boxed{c[1] = 12}$ $\boxed{p[1] = 3}$

∴ Note : The cost obtained w.r.t the source vertex i.e., $c[s]$ is the shortest path from source to destination.

Therefore in the above problem shortest path distance from 1 to 9 is $c[1] = 12$]

To obtain the shortest path which leads to the cost 12 use the array p as follows

	1	2	3	4	5	6	7	8	9
p	3	6	5	8	8	8	9	9	9

(i) To print the path : print the source first

(ii) print $p[1] = 3$ $1 \to 3 \to 5 \to 8 \to 9$

(iii) print $p[3] = 5$

(iv) print $p[5] = 8$

(v) print $p[8] = 9$

Stop as '9' is sink

20/04/18

Multistage graph Problem

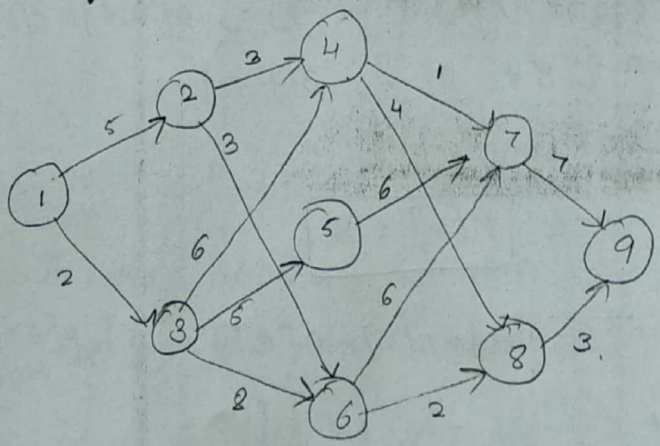
is used to found the shortest path from a node in stage one to a node in stage k. Though source & sink nodes are not specified do consider the vertex at stage 1 as source & at stage -k as sink.

Backward approach

The approach used the following method.

```
c[1] = 0
P[1] = 1;
for j ← 2 to x do.
    for r ∈ V, for each edge <r,j> ∈ E, find
    minimum of a[r,j] + c[r] then
    c[j] ← a[r,j] + c[r];
    P[j] ← r;
end for
```

Apply multistage graph problem to the following graph using backword approach.



→ In the above graph $n = 9$, $s = 1$, $d = 9$ &

$c[1] = 0$, $P[1] = 1$

Vertex 2: $C[2] = \min \{ a[1,2] + c[1] \}$

$= \min \{ 5 + 0 \}$

$\boxed{C[2] = 5}$ & $\boxed{P[2] = 1}$

Vertex 3: $C[3] = \min \{ a[1,3] + c[1] \}$

$= \min \{ 2 + 0 \}$

$\boxed{C[3] = 2}$ & $\boxed{P[3] = 1}$

Vertex 4: $C[4] = \min \{ a[2,4] + c[2], a[3,4] + c[3] \}$

$= \min \{ 3 + 5, 6 + 2 \}$

$\boxed{C[4] = 8}$ & $\boxed{P[4] = 2 \text{ or } P[4] = 3}$

Vertex 5: $C[5] = \min \{ a[3,5] + c[3] \}$

$= \min \{ 5 + 2 \}$

$\boxed{C[5] = 7}$ & $\boxed{P[5] = 3}$

Vertex 6: $C[6] = \min \{ a[3,6] + c[3], a[2,6] + c[2] \}$

$= \min \{ 8 + 2, 3 + 5 \}$

$= \min \{ 10, 8 \}$

$\boxed{C[6] = 8}$ & $\boxed{P[6] = 2}$

Vertex 7: $C[7] = \min \{ a[4,7] + c[4], a[5,7] + c[5], a[6,7] + c[6] \}$

$= \min \{ 1 + 8, 6 + 7, 6 + 8 \}$

$= \min \{ 9, 13, 14 \}$

$\boxed{C[7] = 9}$ & $\boxed{P[7] = 4}$

vertex 8: $c[8] = \min\{a[4,8] + c[4],\ a[5,8] + c[5],\ a[6,8] + c[6]\}$

$= \min\{4+8,\ 2+7,\ 2+8\}$.

$= \min\{12,\ 9,\ 10\}$

$\boxed{c[8] = 9}$ & $\boxed{p[8] = 5}$

vertex 9: $c[9] = \min\{a[7,9] + c[7],\ a[8,9] + c[8]\}$

$= \min\{7+9,\ 8+9\}$

$= \min\{16,\ 12\}$

$\boxed{c[9] = 12}$ & $\boxed{p[9] = 8}$

→ The shortest path distance from source to sink is $c[9] = 12$

→ To print the path use the array 'p'.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
p	1	1	1	2/3	3	2	4	5	8

First, Print the sink node: 9

Then print p[9]: 8

p[8]: 5

p[5]: 3

p[3]: 1 (stop bcz 1 is the source node)

∴ path is

$9 \leftarrow 8 \leftarrow 5 \leftarrow 3 \leftarrow 1$

Note: In the above example p[4] has 2 entries (2 & 3) this may lead to a different paths if & only iff p[4] was used while printing the path.

→ If P[4] was used to print the path then we need to consider 2 paths with the same shortest distance.

algorithm for both the approaches.

Algorithm Forward-graph (a, n, s, d)
{
  c[n] = 0
  p[n] = n;
  for j ← n-1 down to 1 do
    for r ∈ V, for each edge <j, r> ∈ E, find minimum of a[j, r] + c[r] then
      c[j] ← a[j, r] + c[r];
      p[j] ← r;
    end for
    Print (shortest path distance is c[1]);
    j = 1;
    while (j != d)
      print (j, '→');
      j ← p[j];
    end while
    Print (j);
}

Backward approach:

Algorithm Backward-graph (G, n, a, s, d)
{
  c[s] = 0
  p[s] = s;

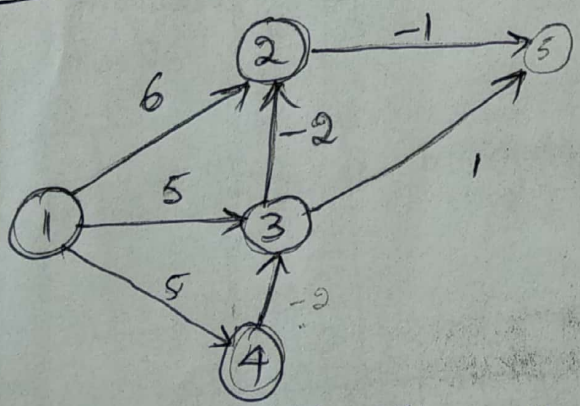
```
for j ← 2 to n
    for r ∈ V, for each edge e < r, j > ∈ E, find
    minimum of a [r,j] + c[r] then
    c[j] ← a [r,j] + c[r];
        p[j] ← r;
end for
Print (shortest path distance is c[n]);
j = d;
    while (j != s)
        print (j, '←');
            j ← p[j];
    end while
print (j);
```

y.

Bellman Ford Algorithm

source node = 1

ex:



Shortcut Path distance

1 → 1 = 0
1 → 2 = 1
1 → 3 = 3
1 → 4 = 5
1 → 5 = 0

Bellman Ford algorithm is to find shortest path from a source vertex to all other vertices in a given directed weighted graph which has negative edges.

algorithm (use the following design)

* for ← 1 to n do
  {
    d[i] = a[s, i]; // distance array
    P[i] = s; // path array
  }

* for i ← 1 to n-1 do // iterations.
  for each edge ⟨u, v⟩ in E where u != v
  (source)
  if (d[u] + a[u, v] < d[v]) source → vertex
    d[v] = d[u] + a[u v];
    P[v] = u;
  end if
  end for;

Apply Bellman ~~code~~ ford algorithm to the following graph

→ initializing the distance matrix d & path array P.

d

[1]	[2]	[3]	[4]	[5]
0	6	5	5	∞

according to graph given as ex:

P

[1]	[2]	[3]	[4]	[5]
1	1	1	1	1

∴ P[i] = x (x = 1)

Since there are 5 vertices, 4 iterations are required. we can stop the iteration if we get same values for d & P arrays as that of a previous iteration.

Iteration 1:

Edge	$d[v] = \min\{d[v], d[u] + c[u,v]\}$	$P[v] = u$
$\langle 1,2 \rangle$	$d[2] = \min\{6, 0+6\} = 6$	—
$\langle 1,3 \rangle$	$d[3] = \min\{5, 0+5\} = 5$	—
$\langle 1,4 \rangle$	$d[4] = \min\{5, 0+5\} = 5$	—
$\langle 2,5 \rangle$	$d[5] = \min\{\infty, 6+(-1)\} = 5$	$P[5] = 2$
$\langle 3,2 \rangle$	$d[2] = \min\{6, 5+(-2)\} = 3$	$P[2] = 3$
$\langle 3,5 \rangle$	$d[5] = \min\{5, 5+(1)\} = 5$	—
$\langle 4,3 \rangle$	$d[3] = \min\{5, 5+(-2)\} = 3$	$P[3] = 4$

update d

Iteration 1	[1]	[2]	[3]	[4]	[5]
d	0	6	5	5	∞
	0	3	3	5	5

P	[1]	[2]	[3]	[4]	[5]
	1	1	1	1	1
	1	3	4	1	2

Repeat the same for Iteration 2 to 4, After the iterations, we get the d & P arrays as below.

	[1]	[2]	[3]	[4]	[5]
d	0	6	5	5	∞
Iteration-1	0	3	3	5	5
Iteration-2	0	1	3	5	2
Iteration-3	0	1	3	5	0
Iteration-4	0	1	3	5	0

P	[1]	[2]	[3]	[4]	[5]
	1	1	1	1	1
I-1	1	3	4	1	2
I-2	1	3	4	1	2
I-3	1	3	4	1	2
I-4	1	3	4	1	2

Now, we can get the shortest path distance and final path using the arrays d & P as shown below.

d	[1]	[2]	[3]	[4]	[5]
	0	1	3	5	0

P	[1]	[2]	[3]	[4]	[5]
	1	3	4	1	2

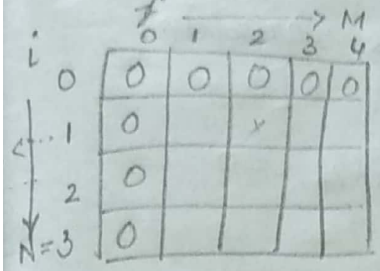
From	TO	Distance	Path
1	1	0	1→1
1	2	1	1→4→3→2
1	3	3	1→4→3
1	4	5	1→4
1	5	0	1→4→3→2→5

Module - IV

Dynamic programming

0/1 knapsack Problem

'n' objects, p_i (profit) & w_i (weight)
'm' knapsack capacity such that $\sum p_i x_i$ is maximum & $\sum w_i x_i \leq m$, $x_i \rightarrow 0$ or 1.



Profit table [V]

Entries in table denote profit
N+1 : rows (N : objects)
M+1 : column (M : knapsack capacity)
V[i][j] : profit

if i=0 ; no objects
j=0 ; knap sack capacity is 0

step 1 : V[i][j] = 0 if i=0 (or) j=0

step 2 : ith object weight is greater than current j
If $w_i > j$ of ith object, then ith object must be rejected.
$V[i, j] = V[i-1, j]$
previous row same column

step 3 : ith object not selected.
If $w_i \leq j$,
$V[i, j] = \max \{ V[i-1, j], V[i-1, j-w_i] + p_i \}$
ith object is selected.
profit (earned by selecting ith object) by making sufficient space

$V[i-1, j]$: profit by not selecting ith object.

$V[i-1, j-w_i] + p_i$: profit by selecting ith object.
(creat space)

Step 4 : After computing all the entries in profit table V, the last entry $V[n, m]$ gives the max. profit

Obtained for the problem.

Note:

Construct the n+1 rows & m+1 columns where the index starts from 0 to n and 0 to m respectively. Every entry in the table denotes the profit w.r.t ith object and knapsack capacity j.

Solve the following 0/1 knapsack problem using dynamic programming.

Item	Weight	Profit
1	2	12
2	1	10
3	3	20

m=4

Step 1 : Construct the profit table of N+1 rows and M+1 colums i.e., 4 rows, 5 columns (n=3, m=4)

i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	0	12	12	12
2	0	10	12	22	22
3	0	10	12	22	30

→ Every cell is sub problem.

Step 2 : Initialize V[i, j] = 0 When i = 0 (or) j = 0

i.e., V[i,j] = 0 for i=0 and j = 0 to 4

V[i,j] = 0 for j=0 and i = 0 to 3

Updating this in the above table.

Step 3 : Starting from i = 1,

here i=1, W[i] = 2 , P[i] = 12

Using the below relation compute the entries.

$$V[i,j] = \begin{cases} V[i-1, j] & , \quad w_i > j \\ \max\{V[i-1, j], V[i-1, j-w_i] + P_i\} & , \quad w_i \le j \end{cases}$$

Computation is shown below.

w_i	j	$i=1$ $V[i,j]$
2	1	$V[1,1] = V[0,1] = 0$
2	2	$V[1,2] = \max\{V[0,2], V[0,0]+12\} = \max\{0,12\} = 12$
2	3	$V[1,3] = \max\{V[0,3], V[0,1]+12\} = \max\{0,12\} = 12$
2	4	$V[1,4] = \max\{V[0,4], V[0,2]+12\} = \max\{0,12\} = 12$

Update the above entries in the profit table.

Steps: considering $i=2$, then $w[i]=1$, $P[i]=10$ & j using the above recurrance relation, computation is as follows.

$i=2$

w_i	j	$V[i,j]$
1	1	$V[2,1] = \max\{V[1,1], V[1,0]+10\} = \max\{0,10\} = 10$
1	2	$V[2,2] = \max\{V[1,2], V[1,1]+10\} = \max\{12,12\} = 12$
1	3	$V[2,3] = \max\{V[1,3], V[1,2]+10\} = \max\{12,22\} = 22$
1	4	$V[2,4] = \max\{V[1,4], V[1,3]+10\} = \max\{12,22\} = 22$

Update the above entries in the profit table

Steps: considering

$i = 3$, $w[i] = 3$, $p[i] = 20$

wi	j	V[i, j]
3	1	$V[3,1] = V[2,1] = 10$
3	2	$V[3,2] = V[2,2] = 12$
3	3	$V[3,3] = \max\{V[2,3], V[2,0]+20\}$ $\max\{22, 20\} = 22$
3	4	$V[3,4] = \max\{V[2,4], V[2,1]+20\}$ $\max\{22, 30\} = 30$

Update the above entries in the profit table. Hence, the maximum profit obtained for the problem is $V[n, m]$ i.e., $V[3,4]$

$\therefore$ profit = 30

The procedure for finding which objects are selected is as follows.

→ $V[3,4] \neq V[2,4]$

$x[3] = 1$ object selected $\quad \& \quad i = 2, \; j = 4 - 3 = 1$

→ $V[2,1] \neq V[1,1]$

$x[2] = 1$ object selected. $\quad \& \quad i = 1 \; \& \; j = 1 - 1 = 0$

```
for i←1 to n do
    x[i]←0;
i=n
j=m
While (i!=0 AA j!=0)
    if V[i,j] != V[i-1][j] then
        x[i]← 1 ;  // ith object selected
        j← j-w[i]; // remaining knapsack
                      capacity.
    end if
    i←i-1; // Move to the previous ith object.
end while

for i←1 to n do
    if (x[i]==1)
        write " object selected is", i
    end if
end for.
```

Algorithm of 0/1 knapsack problem & Analysis.

```
Algorithm knapsack (n, m, w, f)
// To solve 0/1 knapsack problem using dynamic
   programming.
// 'n' no of objects, 'm' knapsack capacity,
   'w[1:n]' weight of objects, 'p'[1:n] profit of objects
// O/P :  To return the profit table V
          V[0:n , 0:m]
   Where, V[n,m] denotes maximum profit.
for i:0 to n do(≥)
    for j←0 to m do
        if i==0 || j==0 then
            V[i,j] ← 0
```

```
    else if w[i] > j then
        V[i, j] ← V[i-1, j]
    else
        V[i, j] ← max{ V[i-1, j], V[i-1, j-w[i]] + P[i] }
    end if
  end for j
end for i
return V
}
```

Analysis:

- I/P size : No of objects and knapsack capacity i.e., n and m respectively.
- Basic operation : The basic operation here is to compute the values of $V[i, j]$

The statement $V[i, j] = \max\{ V[i-1, j], V[i-1, j-w[i]] + P[i] \}$ is executed maximum times, therefore this is the B.O.

- Setting up the summation formula that indicates how many times the B.O is executed.

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{m} 1$$

$$= \sum_{i=0}^{n} (m - 0 + 1)$$

$$= \sum_{i=0}^{n} m + 1$$

$$= (m+1) \sum_{i=0}^{n} 1.$$

$$= (m+1)(n+1)$$

$T(n) = mn + m + n + 1$

$T(n) \approx mn$ (for very large value of $m+n$)

$T(n) \in \theta(mn)$

Assignment

Problem: Solve the problem using dynamic programming where $n = 4$ and $m = 5$

Item	Weight	Profit
1	2	12
2	1	10
3	3	20
4	2	15

Travelling sales man Problem (TSMP)

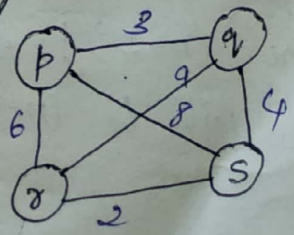
Problem statement: Given n cities the travelling sales man has to start from one city (source) and visit all (n-1) cities exactly once. And return to the city from where he has started. The objective of this problem is to find a route through the cities that minimizes the cost there by maximizing profit

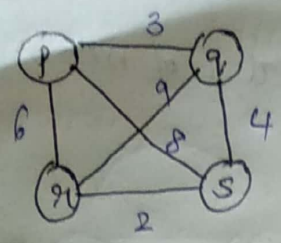
Design

→ consider a graph (undirected),
Where vertices → 'n' cities
weight of edge → 'cost' of reaching/travelling from one city to another.

→ 2 ways to solve.
* Brute force approach. (Trial & Error with ur knowledge)
* Dynamic Programming.

→ Example





Cities: p, q, r, s

source: p

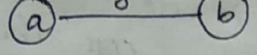
By Brute force.

The paths are.

* $p \xrightarrow{3} q \xrightarrow{9} r \xrightarrow{2} s \xrightarrow{8} p$ (cost = 22)
* $p \xrightarrow{3} q \xrightarrow{4} s \xrightarrow{2} r \xrightarrow{6} p$ (cost = 15)
* $p \xrightarrow{6} r \xrightarrow{2} s \xrightarrow{4} q \xrightarrow{3} p$ (cost = 15)
* $p \xrightarrow{8} s \xrightarrow{4} q \xrightarrow{9} r \xrightarrow{6} p$ (cost = 27)
* $p \xrightarrow{6} r \xrightarrow{9} q \xrightarrow{4} s \xrightarrow{8} p$ (cost = 27)
* $p \xrightarrow{8} s \xrightarrow{2} r \xrightarrow{9} q \xrightarrow{3} p$ (cost = 22)

Note: In Brute force we take different permutations of paths and select the one, which has the least cost

- In general, if we have 2 cities

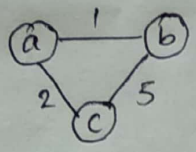
Eg: 

, the paths are $a \xrightarrow{8} b$, $b \xrightarrow{8} a$

$a \xrightarrow{8} b \xrightarrow{8} a$

∴ No of paths = 1.

- Consider 3 cities.

Eg: 

$a \xrightarrow{1} b \xrightarrow{5} c \xrightarrow{2} a$

$a \rightarrow c \rightarrow b \rightarrow a$

∴ No of paths = 2

- Consider 4 cities. ∴ No of paths = 6

∴ In general if we have n cities, then we have (n-1)! paths

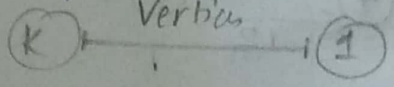
Thus an algorithm that uses Brute force approach need to determine (n-1)! paths. Which means, time complexity will be (n-1)!

TSP using dynamic programming

- The various points to be remembered.

→ The tours may start from any of the Edge $(1, k)$ where $k \in V - \{1\}$ (known source vertex) ↑ Source city

→ There is a path from k to 1.
The path from vertex k to 1 goes through each vertex in $V - \{1, k\}$



→ Let $g(i, S)$ – denotes the shortest path from the vertex i moving through the vertices in S and then terminating at vertex 1.

→ $g(1, V-\{1\}) = \min_{2 \le k \le n} \{ C_{1k} + g(k, V-\{1,k\}) \}$

Shortest path from 1 moving through vertices $V-\{1\}$ and terminating at vertex 1.
k = apart from 1.
Minimum cost ~~vertex~~ from source vertex 1.

→ In General

$$g(i, S) = \min_{j \in S} \{ C_{ij} + g(j, S-\{j\}) \}$$

$|S| \le n-1$ (set of vertices), $i \ne 1$

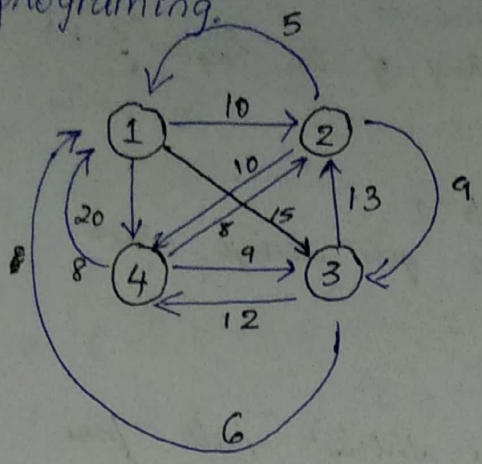
$i \notin S$, $1 \notin S$

→ Base Case

$g(i, \phi) = C_{i1}$ for $2 \le i \le n$ & $|S| = 0$

i to 1 vertex set

Solve the following problem using dynamic programing.



Cost adjacency matrix:

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

In the above graph we have '4' cities where source city is '1' and the problem is solved using the relations as shown below.

ϕ: No intermidiate nodes.

case(i): Here $S = \phi$ (NULL set) i.e., $|S| = 0$

in this case we solve the following relation

$$g(i, \phi) = C_{i1} \quad \text{for} \quad 2 \le i \le n \;\&\; |S| = 0$$

$$g(2, \phi) = C_{2,1} = 5$$

$$g(3, \phi) = C_{3,1} = 6$$

$$g(4, \phi) = C_{4,1} = 8$$

case(ii): Consider $|S| = 1$ in this case S can have 1 vertices.

$i \neq 1$, $i \notin S$, $1 \notin S$. using this solve the following relation

$$g(i, S) = \min_{j \in S} \{ C_{ij} + g(j, S - \{j\}) \}$$

When $i = 2$ we have $S = \{3\}$ & $S = \{4\}$

$g(2,\{3\}) = \min_{j=3}\{ C_{2,3} + g(3,\phi)\}$

$= \min\{9+6\} = 15$

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

When $i=3$, we have $S=\{2\}$ & $S=\{4\}$

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

$g(3,\{4\}) = \min_{j=4}\{ C_{3,4} + g(4,\phi)\} = \min\{12+8\} = 20$

When $i=4$, we have $S=\{2\}$ & $S=\{3\}$

shortest path from 4 & travel the graph come back to source.

$g(4,\{2\}) = \min\{C_{4,2} + g(2,\{2\}-\{2\})\} = \min\{C_{4,2} + g(2,\phi)\}$

$= \min\{8+5\}$

$g(4,\{2\}) = 13$

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

$g(4,\{3\}) = 15$

case (iii): Consider $|s| = 2$ (don't include 1 (source)) $\therefore g(i=2)$

When $i=2$, we have $S=\{3,4\}$, $i \notin S$

S: subset {1,2,3,4} S doesn't include source & i

i(size) = 2 (means & should not include 1, 2)

$i=2$, $S=\{3,4\}$.

$g(2,\{3,4\}) = \min_{j=3,4}\{ C_{23} + g(3,\{4\}),$ remove 3

consider edge(2,3) $C_{24} + g(4,\{3\})$

$= \min\{9+20, 10+15\} = \min\{29,25\}$

$g(2,\{3,4\}) = 25$

case(iv): When $i=3$, $S=\{2,4\}$

edge path

$g(3,\{2,4\}) = \min_{i=2,4}\{ C_{3,2} + g(2,\{4\}),\ C_{3,4} + g(4,\{2\})\}$

$= \min\{13+18, 12+13\} = 25$

case (iv) : When $i = 4 = \{2, 3\}$

$$g(4, \{2,3\}) = \min_{j=2,3} \{ C_{4,2} + g\{2, \{3\}\},\ C_{4,3} + g\{3, \{2\}\} \}$$

$$= \min\{ 8 + 15,\ 9 + 18 \} = \min\{23, 27\}$$

$g(4, \{3\}) = 23$

We should not consider as $|S| = 3$ because $|S| < n-1$ i.e., $|S| < 4-1$, $|S| < 3$ therefore use the final relation to get the optimal tour.

V: set of vertices

$$g\{1, V - \{1\}\} = \min_{2 \le k \le n} \{ C_{1k} + g(k, V - \{1, k\}) \}$$

$$g\{1, V - \{1\}\} = \min\{ C_{1,2} + g(2, V - \{1,2\}),$$

$$C_{1,3} + g(3, V - \{1,3\}),\ C_{1,4} + g(4, V - \{1,4\}) \}$$

$V = \{1, 2, 3, 4\}$

$$g\{1, \{1,2,3,4\} - \{1\}\} = \min\{ C_{1,2} + g(2, \{3,4\}),$$

$$C_{1,3} + g(3, \{2,4\}),$$

$$C_{1,4} + g(4, \{2,3\}) \}$$

$$g\{1, \{2,3,4\}\} = \min\{ 10 + 25,$$

$$15 + 25,$$

$$20 + 23 \}$$

Starting from 1 moving to all vertices {2, 3, 4} and back to source

$$= \min\{35, 40, 43\}$$

$$g\{1, \{2,3,4\}\} = 35$$

∴ The shortest path distance for the above TSP problem is 35.

$1 \to 2 \to 4 \to 3 \to 1$

To get the path just retain each value of $g(i, S)$ that gives the minimum value on RHS

$g(1, \{2,3,4\}) = C_{12} + g(2, \{3,4\})$,

$C_{24} + g(4, \{3\})$

$C_{43} + g(3, \phi)$

C_{31}

Shortest path : $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1 = 35$

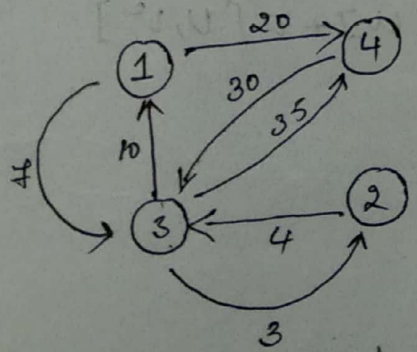
Djikstra's Algorithm (Greedy Method)

This algorithm is used to find the shortest path from a source vertex to all other vertices in a given graph $G = \{V, E\}$ this is also referred as "Single source shortest path problem"

Note : Djikstra works only for positive edge weights whereas Bell man ford works for both +ve and -ve edge weights.

Floyd's : All pair shortest path

Djikstra's : Single source "



Source vertex be 1

$1 \rightarrow 1 = 0$

$1 \rightarrow 2 = 10$

$1 \rightarrow 3 = 7$

$1 \rightarrow 4 = 20$

Cost Adjacency Matrix :

	1	2	3	4
1	0	∞	~~∞~~ 7	20
2	∞	0	4	∞
3	10	3	0	35
4	∞	∞	30	0

Algorithm Dijkstra(n, a, source, d)

// Purpose: To solve single source shortest path problem

// Input: No of vertices, a: cost adjacency matrix
source: source vertex

// output: The matrix d[1:n] which gives shortest distance from source to all other vertices.

for i ← 1 to n do (source row wrt all colu

d[i] ← a[source][i]
p[i] ← source;
s[i] ← 0;

vertices which are not visited: 0
// tells vertices which are visited: 1

edge path
Source → Vertex → All vertices

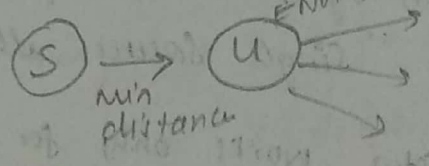
end for

s[source] ← 1 ; // add source to s
source vertex is visited.

for i ← 1 to n-1 do // To find n-1 paths.

Find u & d[u] such that d[u] is minimum & u ∈ V-S (u is not visited)

u: index ; Find a vertex from source.
d[u]: entry



add u to s i.e, s[u] ← 1
visited.

vertices which are not visited.

for every v ∈ V-S do

if (d[u] + a[u,v] < d[v])

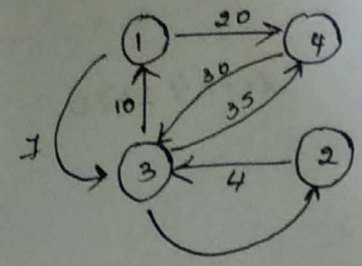
d[v] ← d[u] + a[u,v]

p[v] ← u

end if

end for.

Apply Dijkstra's Algorithm



S: Vertices which are visited.

V-S : Verties which are not visited.

	1	2	3	4
d	0	∞	7	20
(A)	0	(10)	7	20
(B)	0	10	7	20
(C)	0	10	7	20

	1	2	3	4
P	1	1	1	1
A	1	3	1	1
B	1	3	1	1
C	1	3	1	1

	1	2	3	4
S	∅	∅	∅	∅
	1	1	1	1

S	V=V-S	d[v]=min{d[v], d[u]+a[u,v]}	P[v]=u	u∈V-S & d[u] d[u]=min.
{1}	{2,3,4}	–	-	3, 7
now i am reaching 3				1 → 7 → 3
{1,3} (A)	{2,4}	d[2]=min{∞, 7+3} = 10	P[2]=3 reaching 2 via 3	2, 20
		d[4]=min{20, 7+35} = 20 only when d[u]+a[u,v]=min update P[v]=u	–	
{1,3,2} (B)	{4}	d[4]=min{20, 10+∞} = 20	–	4, 20
{1,3,24}	–	–	–	–

Final entries in the d array gives the shortest distance from soura to all other vertices

d

1	2	3	4
0	10	7	20

Source → 1 = 0 Source → 2 = 10 Src → 3 = 7 Src → 4 = 20

1 → 1 = 10

1 → 2 = 10

1 → 3 = 7

1 → 4 = 20

To find the paths use ϕ array

P

[1]	[2]	[3]	[4]
1	3	1	1

To find the path from 1 → 2,
[first print 2, then P[2] i.e.]

first print 2 : 2

P[2] : 3

print P[3] : 1 (stop at source)

~~2 → 3 → 1~~ = 10 1 → 3 → 2 = 10

To print 1 → 3,

print 3 : 3

P[3] = 1 (stop)

~~3 → 1~~ 1 → 3 = 7

To print 1 → 4,

print 4 : 4

P[4] : 1 (stop)

1 → 4 = 20