Algorithm:

Floyd's_Warshall (W)

1. $n \leftarrow$ rows [W] $\quad D^{(0)} \leftarrow W$
2. for $k \leftarrow 1$ to n
3. do for $i \leftarrow 1$ to n
4. do for $j \leftarrow 1$ to n
5. do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{k-1} + d_{kj}^{k-1} \right)$.
6.
7. return $D^{(n)}$.

(distance) $d_{ij}^{k} = \begin{cases} w_{ij} & ; \text{if } k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \end{cases}$

k is intermediate vertex if $k \neq 0$ i.e. there is no intermediate vertex.

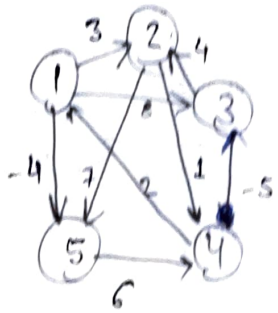
Construct the predecessor matrix Π from the D matrix (distance matrix).

Matrix Π print all pairs shortest path procedures can be used to print the vertices on the given shortest path.

(path) $\rightarrow \Pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i=j \text{ or } w_{ij}=\infty \\ i & \text{if } i \neq j \text{ or } w_{ij} < \infty \end{cases}$

$\Pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{k} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$

Apply floyd's Warshall to find the shortest path. (27)



NOTE: If no. of vertices is n, then we make n×n matrices.
If i = j then put 0 & if there path does not exist between i & j, put ∞.

$D^{(0)}$ =

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$\Pi^{(0)}$ =

NIL	1	1	–	1
–	–	–	2	2
–	3	–	3	3
4	–	4	–	–
–	–	–	5	–

$D^{(1)}$ =

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$\Pi^{(1)}$ =

–	1	1	–	1
–	–	–	2	2
–	3	–	–	–
4	1	4	–	1
–	–	–	5	–

$D^{(2)}$ =

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$\Pi^{(2)}$ =

–	1	1	2	1
–	–	–	2	2
–	3	–	2	2
4	1	4	–	1
–	1	–	5	–

$D^{(3)}$ =

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

$\Pi^{(3)}$ =

–	1	1	2	1
–	–	–	2	2
–	3	–	2	2
4	3	4	–	1
–	–	–	5	–

$D^{(4)}$ =

	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$\Pi^{(4)}$ =

–	1	4	2	1
4	–	4	2	1
4	3	–	2	1
4	3	4	–	1
4	3	4	5	–

$D^{(5)} =$

0	1	-3	2	-4	
3	0	-4	1	-1	
7	4	0	5	3	
2	-1	-5	0	-2	
6	5	1	6	0	

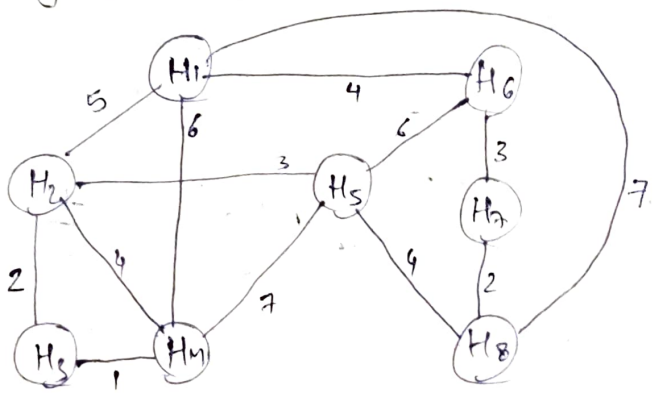
$\rightarrow$

-	~~5~~ 3	~~5~~ 4	5	1	
4	-	~~5~~ 4	2	4	1
4	3	-	2	4	1
4	3	4	-	1	
4	4	4	5	-	
	3				

Travelling Salesman Problem (TSP):-

In this problem, a salesman need to visit n-cities in such a manner that all the cities must be visited at least once. and in the end we returns to the city where we started the visit. with minimum cost.

(1) TSP using greedy Algorithm:-



	H1	H2	H3	H4	H5	H6	H7	H8
H1	0	5	0	6	0	(4)	0	7
H2	5	0	(2)	4	3	0	0	0
H3	0	2	0	(1)	0	0	0	0
H4	(6)	4	(13)	0	7	0	0	0
H5	0	(3)	0	7	0	6	0	4
H6	4	0	0	0	6	0	(3)	0
H7	0	0	0	0	0	3	0	(2)
H8	~~7~~	0	0	0	(4)	0	2	0

$H_1 \xrightarrow{4} H_6$

$H_6 \xrightarrow{3} H_7$

$H_7 \xrightarrow{2} H_8$

~~$H_8 \not\to H_1$~~

$H_8 \xrightarrow{4} H_5$

$H_5 \xrightarrow{3} H_2 \xrightarrow{2} H_3 \xrightarrow{1} H_4 \xrightarrow{6} H_1$

~~total cost = 25~~

TSP using Dynamic Programming:-

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

The general formula is:-

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

$$g(i,\phi) = c_{i1}$$

e.g.: cost 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

path: $1 \xrightarrow{10} 2 \xrightarrow{10} 4 \xrightarrow{9} 3 \xrightarrow{6} 1$

II: $g(4,\{2,3\}) = \min_{j \in S} \{ c_{42} + g(2,\{3\}), c_{43} + g(3,\{2\}) \}$

$8+6$ (23) $\quad 9+5$ (24)

minimum cost = 35 (Answer)

(43) (40) (35)

Let the starting vertex is 1.

$i=1 \quad S=\{2,3,4\}$

$$g(1,\{2,3,4\}) = \min_{j \in S} \{ c_{12} + g(2,\{3,4\}), c_{13} + g(3,\{2,4\}), c_{14} + g(4,\{2,3\}) \}$$

$10 + (25)$ (I) $\quad 15$ (25) (II) $\quad 20$ (23) (III)

I: $g(2,\{3,4\}) = \min_{j \in S} \{ c_{23} + g(3,\{4\}), c_{24} + g(4,\{3\}) \}$

$9 + 20$ (IV) $\quad 10$ (V)

IV: $g(3,\{4\}) = \min_{j \in S} \{ c_{34} + g(4,\phi) \}$

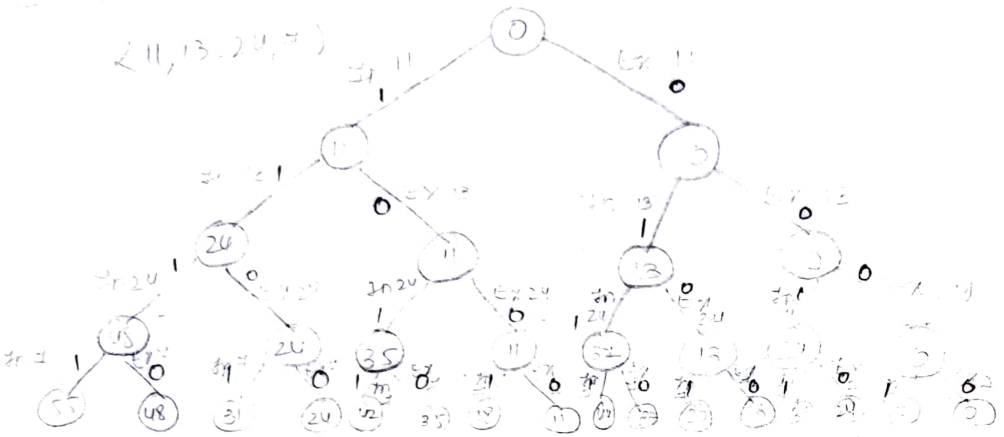
V: $g(4,\{3\}) = \min_{j \in S} \{ c_{43} + g(3,\phi) \}$

$g(4,1) = 8 + c_{41} = 20$

$g(3,1) = 6 + c_{31} = 15$

II: $g(3,\{2,4\}) = \min_{j \in S} \{ c_{32} + g(2,\{4\}), c_{34} + g(4,\{2\}) \}$

$13 + (18)$ $\quad 12 + 13$ (25)



Algorithm:

Assume that all the weight are in increasing order.

H = total sum of all elements in S

S = sum of desired element when we check (S = 0 initially) as we add element S, we subtract from H.

K = that element which is to be added.

```
sumofsubset(S,K,H)
1.  x[K] ← 1
2.  if (S + w[K] = W)
3.  then for 1 to K
4.          print x[i]
5.  else if (S + w[K] + w[K+1] ≤ W)
6.       then sumofsubset (S + w[K], K+1,
                           H - w[K])
7.  if ((S + H - w[K]) and (S + w[K+1]) ≤ W)
8.     then x[K] ← 0
9.  Sum of subsets (S, K+1, H - w[K])
```

The complexity of this Algorithm

NOTE: You have to draw space tree for both fixed size and variable size solutions in exam.

Qu. $S = \{5, 10, 12, 13, 15, 18\}$ and $W = 30$

(a) Fixed size solution

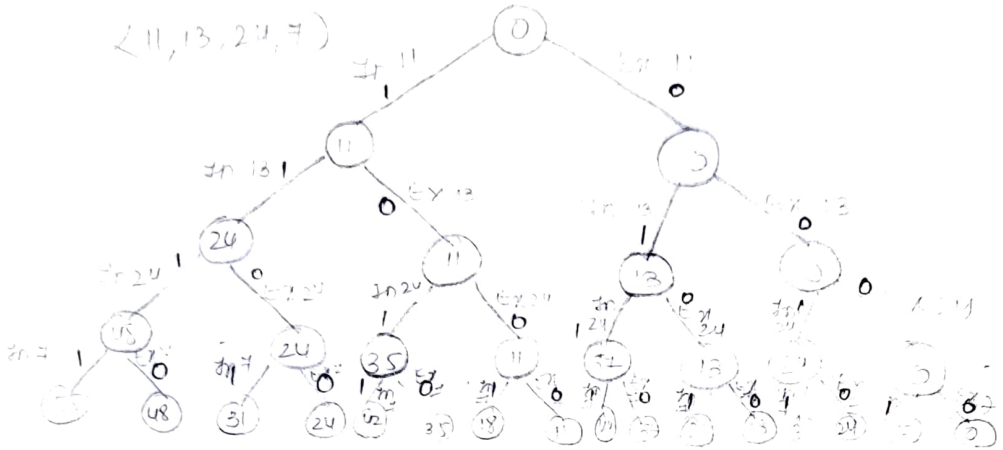
$S = \{1, 1, 0, 0, 1, 0\}$ and

$\{0, 0, 1, 0, 0, 1\}$ etc. and $\{1, 0, 1, 1, 0, 0\}$

(b) Variable size solution

$S = \{1, 2, 5\}$ and $\{3, 6\}$

and $\{1, 3, 4\}$.



Algorithm:

Assume that all the weight are in increasing order.

H = total sum of all elements in S

S = sum of desired element when we check (S = 0 initially) as we add element S, we subtract from H.

K = that element which is to be added.

```
sumofsubset(S,K,H)
1.  x[K] ← 1
2.  if (S + w[K] = W)
3.  then for 1 to K
4.          print x[i]
5.  else if (S + W[K] + W[K+1] ≤ W)
6.          then sumofsubset (S + w[K], K+1,
                              H - w[K])
7.  if ((S + H - w[K]) and (S + w[K+1]) ≤ W)
8.      then x[K] ← 0
9.  Sum of subsets (S, K+1, H - w[K])
```

The complexity of this Algorithm

NOTE: you have to draw space tree for both fixed size and variable size solutions in exam.

Ques. S = {5, 10, 12, 13, 15, 18} and W = 30

(a) Fixed size solution

S = {1, 1, 0, 0, 1, 0} and

{0, 0, 1, 0, 0, 1} etc. and {1, 0, 1, 0, 0, 1}

(b) Variable size solution

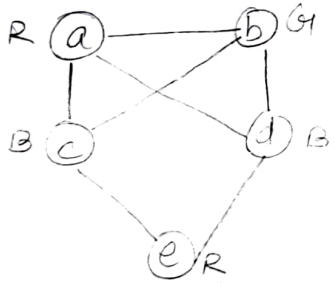
S = {1, 2, 5} and {3, 6}

and {1, 3, 4}.

16.Oct ⑤ **Graph coloring Problem**: Graph coloring problem can be solved by using back-tracking method. In this problem we are given a graph G(V, E) and the numbers of colors 'm', we want to discover whether the nodes of graph 'G' can be colored in such a way that no two adjacent nodes have the same color by using m colors. This is said to be 'm' - colorability decision problem.

This problem can be optimization problem if we determine the smallest integer 'm' for which the graph 'G' is coloured. This integer 'm' can be referred to as chromatic number of the graph.

For eg:



Consider the state space tree having the vertices n = 3 and maximum colors is to be used is 3.

Qu: Consider the following graph & the chromatic number is 3

Algorithm:

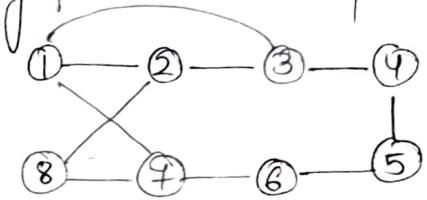
```
Graph-coloring (K)
1. While (1)
2.   do next value (K)
3.     if (x[K] = 0) then return
4.     if (k=n)
5.     then for i ← 1 to n
6.            print x[i]
7.     else Graph-coloring (K+1)

Next-value (K)
1. While (1)
2. do x[K] ← (x[K]+1) mod (m+1)
3.   if (x[K]=0) then return // All colors have been used
4.   For i ← 1 to n
5.   do if (G[K,i] ≠ 0) and x[K] = x[i]
6.   then break // if edge (K,i) is an edge and if adjacent vertices have the same color
7.   if (i=n+1)
8.   then return // new color found
```

4. Hamiltonian Circuit problem: Let G(V, E) be a connected graph with 'n' vertices. A hamiltonian cycle is a round trip that visits every vertex once and returns to its starting position. This problem used backtracking method.

1. Firewall →
2. Routers →

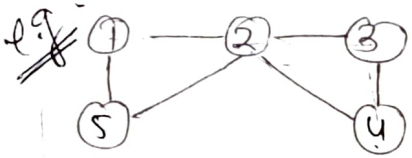
eg:



Hamiltonian circuits are:

1, 3, 4, 5, 6, 7, 8, 2, 1
1, 2, 8, 7, 6, 5, 4, 3, 1
2, 8, 7, 6, 5, 4, 3, 1, 2
3, 4, 5, 6, 7, 8, 2, 1, 3

eg:



No such hamiltonian circuit is possible.

Algorithm:

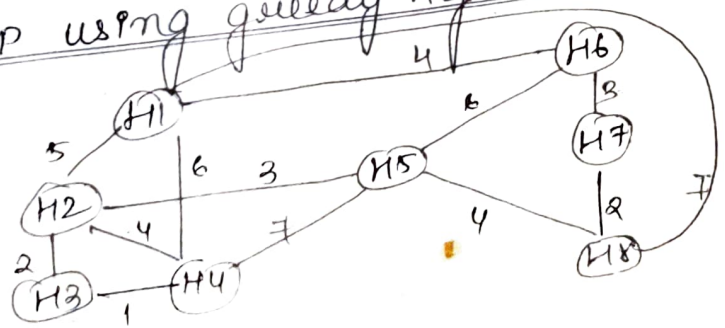
```
Hamiltonian (K)
1. While (1)
2.   do nextvalue(K)
3.   if (n[K]=0) then return
4.     if (k=n)
5.     then for i ← 1 to n
6.     do print x[i]
7.     else Hamiltonian(K+1)
```

```
Next value (k)
1. while (1)
2. do x[k] ← (x[k+1]) mod (n+1) // next vertex
3.    if (x[k] = 0) then return
4.    if ((G[x[k-1]), x[k]≠0)
5.    then for j ← 1 to k-1
6.         do if x[j] = x[k] // check for distinctness
                 break;
7.
8.    if (j = k) // if true then vertex is distinct
9.    then if ((k<n) or (k=n) and G[x[n], n(1)] ≠0)
10.   then return
```

TRAVELLING SALESMAN PROBLEEM: (TSP)

In this problem a salesman need to visit 'n' cities, in such a manner that all cities must be visited at least once and in the end he returns to the city where he started the visit with minimum cost.

(a) TSP using greedy Algorithm:



The cost of adjancy matrix graph is:

	H1	H2	H3	H4	H5	H6	H7	H8
H1	0	5	0	6	0	(4)	0	7
H2	5	0	(2)	4	3	0	0	0
H3	0	2	0	(1)	0	0	0	0
H4	(6)	4	1	0	7	0	0	0
H5	0	(3)	0	7	0	6	0	4
H6	4	0	0	0	6	0	(3)	0
H7	0	0	0	0	0	3	0	(2)
H8	7	0	0	0	(4)	0	2	0

Let H_1 be the starting vertex

$H_1 \xrightarrow{4} H_6 \xrightarrow{3} H_7 \xrightarrow{2} H_8 \xrightarrow{4} H_5$

$H_5 \xrightarrow{3} H_2 \xrightarrow{2} H_3 \xrightarrow{1} H_4 \xrightarrow{6} H_1$

Total cost = 25

(b) TSP using Dynamic Approach:

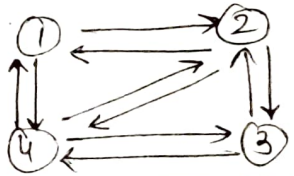
Let $g(i, S)$ be length of shortest path starting at vertex 'i' going through all vertex in S, and terminating at the vertex '1'.

Its general formula is:

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

$$g(i, \phi) = C_{i1}$$

eg:



$g(i, \{2,3,4\}) = \min_{j \in S} \{ C_{12} + g(\{3,4\}) \}$

The most serious drawback of this dynamic programming solution is the space needed. The space needed is $O(n2^n)$. This is too large.