

## UNIT-3

### DYNAMIC PROGRAMMING

General method-multistage graphs-all pair shortest path algorithm-0/1 knapsack and traveling salesman problem-chained matrix multiplication-approaches using recursion-memory functions

### BASIC SEARCH AND TRAVERSAL TECHNIQUES

The techniques-and/or graphs-bi\_connected components-depth first search-topological sorting-breadth first search.

### DYNAMIC PROGRAMMING

- The idea of dynamic programming is thus quit simple: avoid calculating the same thing twice, usually by keeping a table of known result that fills up a sub instances are solved.
- Divide and conquer is a top-down method.
- When a problem is solved by divide and conquer, we immediately attack the complete instance, which we then divide into smaller and smaller sub-instances as the algorithm progresses.
- Dynamic programming on the other hand is a bottom-up technique.
- We usually start with the smallest and hence the simplest sub- instances.
- By combining their solutions, we obtain the answers to sub-instances of increasing size, until finally we arrive at the solution of the original instances.
- The essential difference between the greedy method and dynamic programming is that the greedy method only one decision sequence is ever generated.
- In dynamic programming, many decision sequences may be generated. However, sequences containing sub-optimal sub-sequences can not be optimal and so will not be generated.

### ALL PAIR SHORTEST PATH

- ❖ Let  $G=\langle N,A \rangle$  be a directed graph 'N' is a set of nodes and 'A' is the set of edges.
- ❖ Each edge has an associated non-negative length.
- ❖ We want to calculate the length of the shortest path between each pair of nodes.

- ❖ Suppose the nodes of G are numbered from 1 to n, so  $N=\{1,2,...N\}$ , and suppose G matrix L gives the length of each edge, with  $L(i,j)=0$  for  $i=1,2,...n, L(i,j)>0$  for all i & j, and  $L(i,j)=\infty$ , if the edge (i,j) does not exist.
- ❖ The principle of optimality applies: if k is the node on the shortest path from i to j then the part of the path from i to k and the part from k to j must also be optimal, that is shorter.
- ❖ First, create a cost adjacency matrix for the given graph.
- ❖ Copy the above matrix-to-matrix D, which will give the direct distance between nodes.
- ❖ We have to perform N iteration after iteration k. the matrix D will give you the distance between nodes with only (1,2,...,k) as intermediate nodes.
- ❖ At the iteration k, we have to check for each pair of nodes (i,j) whether or not there exists a path from i to j passing through node k.

#### COST ADJACENCY MATRIX:

$$D_0 = L = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 7 & 5 & \infty & \infty \\ 2 & 7 & \infty & \infty & 2 \\ 3 & \infty & 3 & \infty & \infty \\ 4 & 4 & \infty & 1 & \infty \end{vmatrix} \quad \begin{vmatrix} 11 & 12 & - & - \\ 21 & - & - & 24 \\ - & 32 & - & - \\ 41 & - & 43 & - \end{vmatrix}$$

vertex 1:

$$\begin{vmatrix} 7 & 5 & \infty & \infty \\ 7 & \mathbf{12} & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \mathbf{9} & 1 & \infty \end{vmatrix} \quad \begin{vmatrix} 11 & 12 & - & - \\ 21 & \mathbf{212} & - & 24 \\ - & 32 & - & - \\ 41 & \mathbf{412} & 43 & - \end{vmatrix}$$

vertex 2:

$$\begin{vmatrix} 7 & 5 & \infty & \mathbf{7} \\ 7 & 12 & \infty & 2 \\ \mathbf{10} & 3 & \infty & \mathbf{5} \\ 4 & 9 & 1 & \mathbf{11} \end{vmatrix} \quad \begin{vmatrix} 11 & 12 & - & \mathbf{124} \\ 21 & 212 & - & 24 \\ \mathbf{321} & 32 & - & \mathbf{324} \\ 41 & 412 & 43 & \mathbf{4124} \end{vmatrix}$$

vertex 3:

7	5	$\infty$	7	11	12	-	124
7	12	$\infty$	2	21	212	-	24
10	3	$\infty$	5	321	32	-	324
4	<b>4</b>	1	<b>6</b>	41	<b>432</b>	43	<b>4324</b>

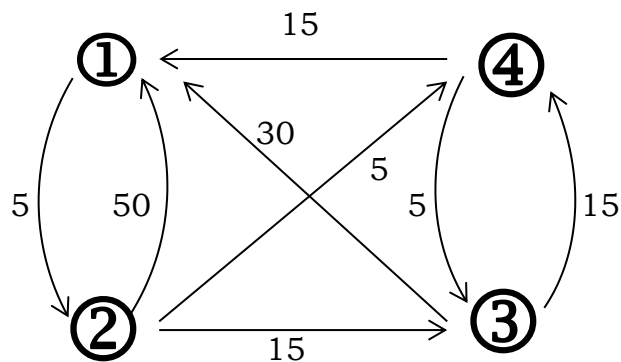
vertex 4:

7	5	<b>8</b>	7	11	12	<b>1243</b>	124
<b>6</b>	<b>6</b>	<b>3</b>	2	<b>241</b>	<b>2432</b>	<b>243</b>	24
<b>9</b>	3	<b>6</b>	5	<b>3241</b>	32	<b>3243</b>	324
4	4	1	6	41	432	43	4324

❖ At 0<sup>th</sup> iteration it nil give you the direct distances between any 2 nodes

$$D0 = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{vmatrix}$$

❖ At 1<sup>st</sup> iteration we have to check the each pair(i,j) whether there is a path through node 1.if so we have to check whether it is minimum than the previous value and if I is so than the distance through 1 is the value of d1(i,j).at the same time we have to solve the intermediate node in the matrix position p(i,j).

$$D1 = \begin{vmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \mathbf{35} & 0 & 15 \\ 15 & \mathbf{20} & 5 & 0 \end{vmatrix} \quad \begin{matrix} p[3,2]= 1 \\ p[4,2]= 1 \end{matrix}$$


**Fig: floyd's algorithm and work**

- ❖ likewise we have to find the value for N iteration (ie) for N nodes.

$$D2 = \begin{vmatrix} 0 & 5 & \mathbf{20} & \mathbf{10} \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix} \quad \begin{array}{l} P[1,3] = 2 \\ P[1,4] = 2 \end{array}$$

$$D3 = \begin{vmatrix} 0 & 5 & 20 & 10 \\ \mathbf{45} & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix} \quad P[2,1]=3$$

$$D4 = \begin{vmatrix} 0 & 5 & \mathbf{15} & 10 \\ 20 & 0 & \mathbf{10} & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{vmatrix} \quad \begin{array}{l} P[1,3]=4 \\ P[2,3]=4 \end{array}$$

- ❖ D4 will give the shortest distance between any pair of nodes.
- ❖ If you want the exact path then we have to refer the matrix p. The matrix will be,

$$P = \begin{vmatrix} 0 & 0 & 4 & 2 \\ 3 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix} \quad 0 \longrightarrow \text{direct path}$$

- ❖ Since,  $p[1,3]=4$ , the shortest path from 1 to 3 passes through 4.
- ❖ Looking now at  $p[1,4]$  &  $p[4,3]$  we discover that between 1 & 4, we have to go to node 2 but that from 4 to 3 we proceed directly.
- ❖ Finally we see the trips from 1 to 2, & from 2 to 4, are also direct.
- ❖ The shortest path from 1 to 3 is 1,2,4,3.

**ALGORITHM :**

Function Floyd (L[1..r,1..r]):array[1..n,1..n]  
array D[1..n,1..n]

```
D = L
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 ]
Return D
```

### ANALYSIS:

This algorithm takes a time of  $\theta(n^3)$

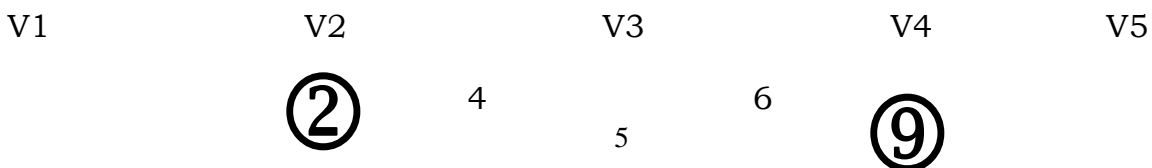
## MULTISTAGE GRAPH

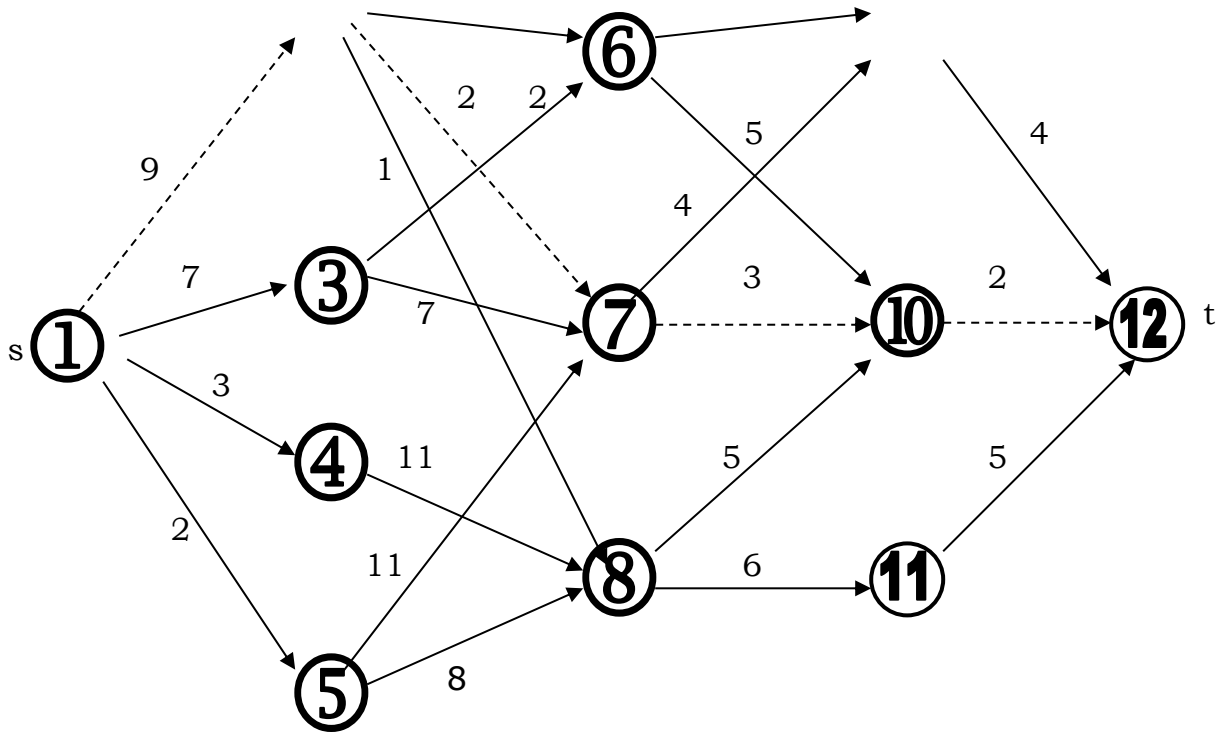
1. A multistage graph  $G = (V, E)$  is a directed graph in which the vertices are partitioned into  $K \geq 2$  disjoint sets  $V_i, 1 \leq i \leq k$ .
2. In addition, if  $\langle u, v \rangle$  is an edge in  $E$ , then  $u \in V_i$  and  $v \in V_{i+1}$  for some  $i, 1 \leq i < k$ .
3. If there will be only one vertex, then the sets  $V_i$  and  $V_k$  are such that  $|V_i| = |V_k| = 1$ .
4. Let 's' and 't' be the source and destination respectively.
5. The cost of a path from source (s) to destination (t) is the sum of the costs of the edges on the path.
6. The *MULTISTAGE GRAPH* problem is to find a minimum cost path from 's' to 't'.
7. Each set  $V_i$  defines a stage in the graph. Every path from 's' to 't' starts in stage-1, goes to stage-2 then to stage-3, then to stage-4, and so on, and terminates in stage-k.
8. This *MULTISTAGE GRAPH* problem can be solved in 2 ways.
  - a) Forward Method.
  - b) Backward Method.

### FORWARD METHOD

1. Assume that there are 'k' stages in a graph.
2. In this *FORWARD* approach, we will find out the cost of each and every node starting from the ' $k^{th}$ ' stage to the  $1^{st}$  stage.
3. We will find out the path (i.e.) minimum cost path from source to the destination (ie) [ Stage-1 to Stage-k ].

### PROCEDURE:





- ❖ Maintain a cost matrix cost (n) which stores the distance from any vertex to the destination.
- ❖ If a vertex is having more than one path, then we have to choose the minimum distance path and the intermediate vertex, which gives the minimum distance path, will be stored in the distance array 'D'.
- ❖ In this way we will find out the minimum cost path from each and every vertex.
- ❖ Finally cost(1) will give the shortest distance from source to destination.
- ❖ For finding the path, start from vertex-1 then the distance array D(1) will give the minimum cost neighbour vertex which in turn give the next nearest vertex and proceed in this way till we reach the Destination.
- ❖ For a 'k' stage graph, there will be 'k' vertex in the path.
- ❖ In the above graph V1...V5 represent the stages. This 5 stage graph can be solved by using forward approach as follows,

**STEPS: -**

**DESTINATION, D**

Cost (12)=0	→	D (12)=0
Cost (11)=5	→	D (11)=12
Cost (10)=2	→	D (10)=12
Cost (9)=4	→	D (9)=12

- ❖ For forward approach,

$$\text{Cost (i,j)} = \min \{ C (j,l) + \text{Cost (i+1,l)} \}$$

$l \in V_{i+1}$ $(j,l) \in E$
-------------------------------

$$\begin{aligned} \text{Cost}(8) &= \min \{ C(8,10) + \text{Cost}(10), C(8,11) + \text{Cost}(11) \} \\ &= \min(5 + 2, 6 + 5) \\ &= \min(7, 11) \\ &= 7 \end{aligned}$$

$$\text{cost}(8) = 7 \Rightarrow D(8) = 10$$

$$\begin{aligned} \text{cost}(7) &= \min(c(7,9) + \text{cost}(9), c(7,10) + \text{cost}(10)) \\ &\quad (4+4, 3+2) \\ &= \min(8, 5) \\ &= 5 \end{aligned}$$

$$\text{cost}(7) = 5 \Rightarrow D(7) = 10$$

$$\begin{aligned} \text{cost}(6) &= \min(c(6,9) + \text{cost}(9), c(6,10) + \text{cost}(10)) \\ &= \min(6+4, 5+2) \\ &= \min(10, 7) \\ &= 7 \end{aligned}$$

$$\text{cost}(6) = 7 \Rightarrow D(6) = 10$$

$$\begin{aligned} \text{cost}(5) &= \min(c(5,7) + \text{cost}(7), c(5,8) + \text{cost}(8)) \\ &= \min(11+5, 8+7) \\ &= \min(16, 15) \\ &= 15 \end{aligned}$$

$$\text{cost}(5) = 15 \Rightarrow D(5) = 18$$

$$\begin{aligned} \text{cost}(4) &= \min(c(4,8) + \text{cost}(8)) \\ &= \min(11+7) \\ &= 18 \end{aligned}$$

$$\text{cost}(4) = 18 \Rightarrow D(4) = 8$$

$$\begin{aligned} \text{cost}(3) &= \min(c(3,6) + \text{cost}(6), c(3,7) + \text{cost}(7)) \\ &= \min(2+7, 7+5) \\ &= \min(9, 12) \\ &= 9 \end{aligned}$$

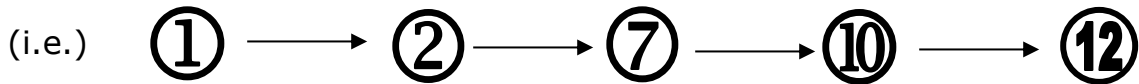
$$\text{cost}(3) = 9 \Rightarrow D(3) = 6$$

$$\begin{aligned} \text{cost}(2) &= \min(c(2,6) + \text{cost}(6), c(2,7) + \text{cost}(7), c(2,8) + \text{cost}(8)) \\ &= \min(4+7, 2+5, 1+7) \\ &= \min(11, 7, 8) \\ &= 7 \end{aligned}$$

$$\text{cost}(2) = 7 \Rightarrow D(2) = 7$$

$$\begin{aligned}
 \text{cost}(1) &= \min (c(1,2)+\text{cost}(2), c(1,3)+\text{cost}(3), c(1,4)+\text{cost}(4), c(1,5)+\text{cost}(5)) \\
 &= \min(9+7, 7+9, 3+18, 2+15) \\
 &= \min(16, 16, 21, 17) \\
 &= 16 \\
 \text{cost}(1) &= 16 \Rightarrow D(1) = 2
 \end{aligned}$$

.....→ The path through which you have to find the shortest distance.



Start from vertex - 2

$$\begin{aligned}
 D(1) &= 2 \\
 D(2) &= 7 \\
 D(7) &= 10 \\
 D(10) &= 12
 \end{aligned}$$

So, the minimum –cost path is,



∴ The cost is  $9+2+3+2=16$

### ALGORITHM: FORWARD METHOD

#### Algorithm FGraph (G,k,n,p)

```

// The I/p is a k-stage graph G=(V,E) with 'n' vertex.
// Indexed in order of stages E is a set of edges.
// and c[i,j] is the cost of <i,j>, p[1:k] is a minimum cost path.
{
    cost[n]=0.0;
    for j=n-1 to 1 step-1 do
    {
        //compute cost[j],
        // let 'r' be the vertex such that <j,r> is an edge of 'G' &
        // c[j,r]+cost[r] is minimum.

        cost[j] = c[j+r] + cost[r];
        d[j] =r;
    }
    // find a minimum cost path.

```



```

P[1]=1;
P[k]=n;
For j=2 to k-1 do
P[j]=d[p[j-1]];
}

```

#### ANALYSIS:

The time complexity of this forward method is  $O(|V| + |E|)$

#### BACKWARD METHOD

- if there one 'K' stages in a graph using back ward approach. we will find out the cost of each & every vertex starting from 1<sup>st</sup> stage to the k<sup>th</sup> stage.
- We will find out the minimum cost path from destination to source (ie)[from stage k to stage 1]

#### PROCEDURE:

1. It is similar to forward approach, but differs only in two or three ways.
2. Maintain a cost matrix to store the cost of every vertices and a distance matrix to store the minimum distance vertex.
3. Find out the cost of each and every vertex starting from vertex 1 up to vertex k.
4. To find out the path star from vertex 'k', then the distance array D (k) will give the minimum cost neighbor vertex which in turn gives the next nearest neighbor vertex and proceed till we reach the destination.

#### STEP:

$$\text{Cost}(1) = 0 \Rightarrow D(1)=0$$

$$\text{Cost}(2) = 9 \Rightarrow D(2)=1$$

$$\text{Cost}(3) = 7 \Rightarrow D(3)=1$$

$$\text{Cost}(4) = 3 \Rightarrow D(4)=1$$

$$\text{Cost}(5) = 2 \Rightarrow D(5)=1$$

$$\begin{aligned} \text{Cost}(6) &= \min(c(2,6) + \text{cost}(2), c(3,6) + \text{cost}(3)) \\ &= \min(13, 9) \end{aligned}$$

$$\text{cost}(6) = 9 \Rightarrow D(6)=3$$

$$\begin{aligned} \text{Cost}(7) &= \min(c(3,7) + \text{cost}(3), c(5,7) + \text{cost}(5), c(2,7) + \text{cost}(2)) \\ &= \min(14, 13, 11) \end{aligned}$$

$$\text{cost}(7) = 11 \Rightarrow D(7)=2$$

$$\text{Cost}(8) = \min(c(2,8) + \text{cost}(2), c(4,8) + \text{cost}(4), c(5,8) + \text{cost}(5)) \\ = \min(10, 14, 10)$$

$$\text{cost}(8) = 10 \Rightarrow D(8) = 2$$

$$\text{Cost}(9) = \min(c(6,9) + \text{cost}(6), c(7,9) + \text{cost}(7)) \\ = \min(15, 15)$$

$$\text{cost}(9) = 15 \Rightarrow D(9) = 6$$

$$\text{Cost}(10) = \min(c(6,10) + \text{cost}(6), c(7,10) + \text{cost}(7), c(8,10) + \text{cost}(8)) = \min(14, 14, 15) \\ \text{cost}(10) = 14 \Rightarrow D(10) = 6$$

$$\text{Cost}(11) = \min(c(8,11) + \text{cost}(8))$$

$$\text{cost}(11) = 16 \Rightarrow D(11) = 8$$

$$\text{cost}(12) = \min(c(9,12) + \text{cost}(9), c(10,12) + \text{cost}(10), c(11,12) + \text{cost}(11)) \\ = \min(19, 16, 21)$$

$$\text{cost}(12) = 16 \Rightarrow D(12) = 10$$

#### **PATH:**

Start from vertex-12

$$D(12) = 10$$

$$D(10) = 6$$

$$D(6) = 3$$

$$D(3) = 1$$

So the minimum cost path is,

$$1 \xrightarrow{7} 3 \xrightarrow{2} 6 \xrightarrow{5} 10 \xrightarrow{2} 12$$

The cost is 16.

#### **ALGORITHM : BACKWARD METHOD**

##### **Algorithm BGraph (G,k,n,p)**

// The I/p is a k-stage graph  $G=(V,E)$  with 'n' vertex.

// Indexed in order of stages E is a set of edges.

// and  $c[i,j]$  is the cost of  $\langle i,j \rangle$ ,  $p[1:k]$  is a minimum cost path.

```
{
    bcost[1]=0.0;
    for j=2 to n do
```

```

{
  //compute bcost[j],
  // let 'r' be the vertex such that <r,j> is an edge of 'G' &
  // bcost[r]+c[r,j] is minimum.

  bcost[j] = bcost[r] + c[r,j];
  d[j] =r;
}
// find a minimum cost path.

  P[1]=1;
  P[k]=n;
  For j= k-1 to 2 do
    P[j]=d[p[j+1]];
}

```

## TRAVELLING SALESMAN PROBLEM

- Let  $G(V,E)$  be a directed graph with edge cost  $c_{ij}$  is defined such that  $c_{ij} > 0$  for all  $i$  and  $j$  and  $c_{ij} = \infty$ , if  $\langle i,j \rangle \notin E$ .  
Let  $|V| = n$  and assume  $n > 1$ .
- The traveling salesman problem is to find a tour of minimum cost.
- A tour of  $G$  is a directed cycle that include every vertex in  $V$ .
- The cost of the tour is the sum of cost of the edges on the tour.
- The tour is the shortest path that starts and ends at the same vertex (ie) 1.

### APPLICATION :

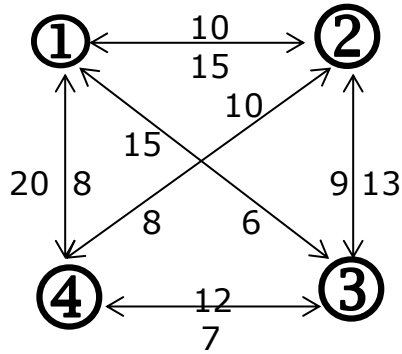
1. Suppose we have to route a postal van to pick up mail from the mail boxes located at 'n' different sites.
2. An  $n+1$  vertex graph can be used to represent the situation.
3. One vertex represent the post office from which the postal van starts and return.
4. Edge  $\langle i,j \rangle$  is assigned a cost equal to the distance from site 'i' to site 'j'.
5. the route taken by the postal van is a tour and we are finding a tour of minimum length.
6. every tour consists of an edge  $\langle 1,k \rangle$  for some  $k \in V - \{1\}$  and a path from vertex  $k$  to vertex 1.
7. the path from vertex  $k$  to vertex 1 goes through each vertex in  $V - \{1,k\}$  exactly once.
8. the function which is used to find the path is

$$g(1, V - \{1\}) = \min \{ c_{ij} + g(j, S - \{j\}) \}$$

9.  $g(i,s)$  be the length of a shortest path starting at vertex  $i$ , going through all vertices in  $S$ , and terminating at vertex 1.
10. the function  $g(1, V - \{1\})$  is the length of an optimal tour.

### STEPS TO FIND THE PATH:

1. Find  $g(i, \Phi) = c_{i1}$ ,  $1 \leq i \leq n$ , hence we can use equation(2) to obtain  $g(i, s)$  for all  $s$  to size 1.
2. That we have to start with  $s=1$ , (ie) there will be only one vertex in set 's'.
3. Then  $s=2$ , and we have to proceed until  $|s| < n-1$ .
4. for example consider the graph.



### Cost matrix

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

$g(i, s) \rightarrow$  set of nodes/vertex have to visited.



starting position

$$g(i, s) = \min\{c_{ij} + g(j, s - \{j\})\}$$

### STEP 1:

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

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

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

$$= 35$$

### STEP 2:

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

$$\begin{aligned} & \min\{9+20, 10+15\} \\ & \min\{29, 25\} \\ & =25 \end{aligned}$$

$$g(3, \{2, 4\}) = \min\{c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\})\}$$

$$\begin{aligned} & \min\{13+18, 12+13\} \\ & \min\{31, 25\} \\ & =25 \end{aligned}$$

$$g(4, \{2, 3\}) = \min\{c_{42} + g(2, \{3\}), c_{43} + g(3, \{2\})\}$$

$$\begin{aligned} & \min\{8+15, 9+18\} \\ & \min\{23, 27\} \\ & =23 \end{aligned}$$

**STEP 3:**

$$\begin{aligned} 1. \quad g(3, \{4\}) &= \min\{c_{34} + g(4, \Phi)\} \\ & 12+8 = 20 \end{aligned}$$

$$\begin{aligned} 2. \quad g(4, \{3\}) &= \min\{c_{43} + g(3, \Phi)\} \\ & 9+6 = 15 \end{aligned}$$

$$\begin{aligned} 3. \quad g(2, \{4\}) &= \min\{c_{24} + g(4, \Phi)\} \\ & 10+8 = 18 \end{aligned}$$

$$\begin{aligned} 4. \quad g(4, \{2\}) &= \min\{c_{42} + g(2, \Phi)\} \\ & 8+5 = 13 \end{aligned}$$

$$\begin{aligned} 5. \quad g(2, \{3\}) &= \min\{c_{23} + g(3, \Phi)\} \\ & 9+6 = 15 \end{aligned}$$

$$\begin{aligned} 6. \quad g(3, \{2\}) &= \min\{c_{32} + g(2, \Phi)\} \\ & 13+5 = 18 \end{aligned}$$

**STEP 4:**

$$g\{4, \Phi\} = c_{41} = 8$$

$$g\{3, \Phi\} = c_{31} = 6$$

$$g\{2, \Phi\} = c_{21} = 5$$

||

$$s = 0.$$

$$i = 1 \text{ to } n.$$

$$g(1, \Phi) = c_{11} \Rightarrow 0$$

$$g(2, \Phi) = c_{21} \Rightarrow 5$$

$$g(3, \Phi) = c_{31} \Rightarrow 6$$

$$g(4, \Phi) = c_{41} \Rightarrow 8$$

$$\left| s \right| = 1$$

$$i = 2 \text{ to } 4$$

$$\begin{aligned} g(2, \{3\}) &= c_{23} + g(3, \Phi) \\ &= 9 + 6 = 15 \end{aligned}$$

$$\begin{aligned} g(2, \{4\}) &= c_{24} + g(4, \Phi) \\ &= 10 + 8 = 18 \end{aligned}$$

$$\begin{aligned} g(3, \{2\}) &= c_{32} + g(2, \Phi) \\ &= 13 + 5 = 18 \end{aligned}$$

$$\begin{aligned} g(3, \{4\}) &= c_{34} + g(4, \Phi) \\ &= 12 + 8 = 20 \end{aligned}$$

$$\begin{aligned} g(4, \{2\}) &= c_{42} + g(2, \Phi) \\ &= 8 + 5 = 13 \end{aligned}$$

$$\begin{aligned} g(4, \{3\}) &= c_{43} + g(3, \Phi) \\ &= 9 + 6 = 15 \end{aligned}$$

$$\left| s \right| = 2$$

$$i \neq 1, 1 \in s \text{ and } i \in s.$$

$$\begin{aligned} g(2, \{3, 4\}) &= \min\{c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\})\} \\ &\quad \min\{9 + 20, 10 + 15\} \\ &\quad \min\{29, 25\} \\ &= 25 \end{aligned}$$

$$\begin{aligned} g(3, \{2, 4\}) &= \min\{c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\})\} \\ &\quad \min\{13 + 18, 12 + 13\} \end{aligned}$$

$$\begin{aligned} & \min\{31,25\} \\ & =25 \end{aligned}$$

$$\begin{aligned} g(4,\{2,3\}) &= \min\{c_{42}+g(2\{3\}), c_{43}+g(3,\{2\})\} \\ & \min\{8+15, 9+18\} \\ & \min\{23,27\} \\ & =23 \end{aligned}$$

$$|s| = 3$$

$$\begin{aligned} g(1,\{2,3,4\}) &= \min\{c_{12}+g(2\{3,4\}), c_{13}+g(3,\{2,4\}), c_{14}+g(4,\{2,3\})\} \\ & \min\{10+25, 15+25, 20+23\} \\ & \min\{35,35,43\} \\ & =35 \end{aligned}$$

optimal cost is 35

the shortest path is,

$$g(1,\{2,3,4\}) = c_{12} + g(2,\{3,4\}) \Rightarrow 1 \rightarrow 2$$

$$g(2,\{3,4\}) = c_{24} + g(4,\{3\}) \Rightarrow 1 \rightarrow 2 \rightarrow 4$$

$$g(4,\{3\}) = c_{43} + g(3\{\Phi\}) \Rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$$

so the optimal tour is **1 → 2 → 4 → 3 → 1**

## CHAINED MATRIX MULTIPLICATION:

If we have a matrix A of size  $p \times q$  and B matrix of size  $q \times r$ . the product of these two matrix C is given by,

$$c_{ij} = \sum a_{ik} b_{kj}, \quad 1 \leq i \leq p, \quad 1 \leq j \leq r, \quad 1 \leq k \leq q.$$

it requires a total of  $pqr$  scalar multiplication.

Matrix multiplication is associative, so if we want to calculate the product of more than 2 matrixes  $m = m_1 m_2 \dots m_n$ .

For example,

$$A = 13 \times 5$$

$$B = 5 \times 89$$

$$C = 89 \times 3$$

$$D = 3 \times 34$$

Now, we will see some of the sequence,

$$\begin{aligned} M &= (((A.B).C).D) \\ &\quad \quad \quad A.B \quad \quad \quad C \\ &= (13 * 5 * 89) * (89 * 3) \\ &\quad \quad \quad A.B.C. \quad \quad \quad D \\ &= (13 * 89 * 3) * (3 * 34) \\ &\quad \quad \quad A.B.C.D \\ &= 13 * 3 * 34 \end{aligned}$$

$$\begin{aligned} (ic) &= 13 * 5 * 89 + 13 * 89 * 3 + 13 * 3 * 34 \\ &= 10,582 \text{ no. of multiplications one required for that sequence.} \end{aligned}$$

2<sup>nd</sup> Sequence,

$$\begin{aligned} M &= (A * B) * (C * D) \\ &= 13 * 5 * 89 + 89 * 3 * 34 + 13 * 89 * 34 \\ &= 54201 \text{ no. of Multiplication} \end{aligned}$$

3<sup>rd</sup> Sequence,

$$\begin{aligned} M &= (A.(BC)) . D \\ &= 5 * 89 * 3 + 13 * 5 * 3 + 13 * 3 * 34 \\ &= 2856 \end{aligned}$$

For comparing all these sequence, (A(BC)).D sequences less no. of multiplication.

For finding the no. of multiplication directly, we are going to the Dynamic programming method.

### **STRAIGHT FORWARD METHOD:**

Our aim is to find the total no. of scalar multiplication required to compute the matrix product.

Let (M1,M2,.....Mi) and Mi+1,Mi+2,.....Mn be the chain of matrix to be calculated using the dynamic programming method.

In dynamic programming, we always start with the smallest instances and continue till we reach the required size.

We maintain a table mij,  $1 \leq i \leq j \leq n$ ,  
Where mij gives the optimal solution.



Sizes of all the matrixes are stored in the array  $d[0..n]$

We build the table diagonal by diagonal; diagonal  $s$  contains the elements  $m_{ij}$  such that  $j-i = s$ .

**RULES TO FILL THE TABLE  $M_{ij}$ :**

$S = 0, 1, \dots, n-1$

If  $s=0 \Rightarrow m(i,i) = 0, i = 1, 2, \dots, n$

If  $s=1 \Rightarrow m(i,i+1) = d(i-1) * d_i * d(i+1)$

$i = 1, 2, \dots, n-1$ .

If  $1 < s < n \Rightarrow m_{i,i+s} = \min(m_{ik} + m_{k+1,i+s} + d_{i-1}d_kd_{i+s})$

$i \leq k \leq i+s \quad i = 1, 2, \dots, n-s$

apply this to the example,

$A \Rightarrow 13 \times 5$

$B \Rightarrow 5 \times 89$

$C \Rightarrow 89 \times 3$

$D \Rightarrow 3 \times 34$

Single dimension array is used to store the sizes.

$d[0] = 13$

$d[1] = 5$

$d[2] = 89$

$d[3] = 3$

$d[4] = 34$

if  $s=0$ ,

$m(1,1) = 0$

$m(2,2) = 0$

$m(3,3) = 0$

$m(4,4) = 0$

if  $s=1$ ,

$m_{i,i+1} = d(i-1) * d_i * d(i+1)$

$m_{12} = d_0 * d_1 * d_2 = 13 * 5 * 89 = 5785$

$m_{23} = d_1 * d_2 * d_3 = 5 * 89 * 3 = 1335$

$m_{34} = d_2 * d_3 * d_4 = 89 * 3 * 34 = 9078$

if  $s=2$ ,

$$m_{i,i+s} = \min(m_{ik} + m_{k+1,i+s} + d_{i-1}d_kd_{i+s})$$

$$\begin{aligned} m_{13} &= \min(m_{11} + m_{23} + d_0d_1d_3, m_{12} + m_{33} + d_0d_2d_3) \\ &= \min(0 + 1335 + (13 \cdot 5 \cdot 3), 5785 + 0 + (13 \cdot 89 \cdot 3)) \\ &= \min(1530, 9256) \\ &= 1530 \end{aligned}$$

$$\begin{aligned} m_{13} &= \min(m_{22} + m_{34} + d_1d_2d_4, m_{23} + m_{44} + d_1d_3d_4) \\ &= \min(0 + 9078 + (5 \cdot 89 \cdot 34), 1335 + 0 + (5 \cdot 3 \cdot 34)) \\ &= \min(24208, 1845) \\ &= 1845 \end{aligned}$$

$$\begin{aligned} m_{14} &= \min(m_{11} + m_{24} + d_0d_1d_4, m_{12} + m_{34} + d_0d_2d_4, m_{13} + m_{44} + d_0d_3d_4) \\ &= \min(4055, 54201, 2856) \\ &= 2856 \end{aligned}$$

The matrix table  $m_{ij}$  will look like,

	1	2	3	4	
1	0	5785	1530	2856	$s=3$
2		0	1335	1845	$s=2$
3			0	9078	$s=1$
4				0	$s=0$

No need to fill the lower diagonal

### ALGORITHM:

#### Procedure cmatrix( $n, d[0..n]$ )

```

For  $s=0$  to  $n-1$  do
{
    if( $s==0$ )
        for  $I=1$  to  $n$  do
             $m(i,i) = 0$ 
    if( $s==1$ )

```

```

for i=1 to n-1 do
  m(i,i+1) =d(i-1)*d(i)*d(i+1)
else
{
  m=∞
  for i=1 to n-s do
    for k=i to i+s do
    {
      if (min>[m(i,k) +m(k+1,i+s)+d(i-1)*d(k)*d(i+s)])
        min=[m(i,k) +m(k+1,i+s)+d(i-1)*d(k)*d(i+s)]
    }
    m(i,i+s) =min
  }
}

```

### 0/1 KNAPSACK PROBLEM:

- This problem is similar to ordinary knapsack problem but we may not take a fraction of an object.
- We are given ‘N’ object with weight  $W_i$  and profits  $P_i$  where  $i$  varies from 1 to  $N$  and also a knapsack with capacity ‘M’.
- The problem is, we have to fill the bag with the help of ‘N’ objects and the resulting profit has to be maximum.

- Formally, the problem can be stated as, maximize  $\sum_{i=1}^n X_i P_i$

$$\text{subject to } \sum_{i=1}^n X_i W_i \leq M$$

- Where  $X_i$  are constraints on the solution  $X_i \in \{0,1\}$ . (u)  $X_i$  is required to be 0 or 1. if the object is selected then the unit is 1. if the object is rejected then the unit is 0. That is why it is called as 0/1, knapsack problem.
- To solve the problem by dynamic programming we use a table  $T[1 \dots N, 0 \dots M]$  (ic) the size is  $N$ . where ‘N’ is the no. of objects and column starts with ‘0’ to capacity (ic) ‘M’.
- In the table  $T[i,j]$  will be the maximum value of the objects  $i$  varies from 1 to  $n$  and  $j$  varies from 0 to  $M$ .

### RULES TO FILL THE TABLE:-

- If  $i=1$  and  $j < w(i)$  then  $T(i,j) = 0$ , (ic) 0 pre is filled in the table.

- If  $i=1$  and  $j \geq w(i)$  then  $T(i,j) = p(i)$ , the cell is filled with the profit  $p[i]$ , since only one object can be selected to the maximum.
- If  $i>1$  and  $j < w(i)$  then  $T(i,j) = T(i-1,j)$  the cell is filled the profit of previous object since it is not possible with the current object.
- If  $i>1$  and  $j \geq w(i)$  then  $T(i,j) = \{f(i) + T(i-1,j-w(i))\}$ , since only '1' unit can be selected to the maximum. If is the current profit + profit of the previous object to fill the remaining capacity of the bag.
- After the table is generated, it will give details the profit.

### ES TO GET THE COMBINATION OF OBJECT:

- Start with the last position of  $i$  and  $j$ ,  $T[i,j]$ , if  $T[i,j] = T[i-1,j]$  then no object of 'i' is required so move up to  $T[i-1,j]$ .
- After moved, we have to check if,  $T[i,j]=T[i-1,j-w(i)] + p[i]$ , if it is equal then one unit of object 'i' is selected and move up to the position  $T[i-1,j-w(i)]$
- Repeat the same process until we reach  $T[i,0]$ , then there will be nothing to fill the bag stop the process.
- Time is  $O(nw)$  is necessary to construct the table  $T$ .
- Consider a Example,

$$\begin{aligned} M &= 6, \\ N &= 3 \\ W_1 &= 2, W_2 = 3, W_3 = 4 \\ P_1 &= 1, P_2 = 2, P_3 = 5 \end{aligned}$$

$$\begin{aligned} i &\longrightarrow 1 \text{ to } N \\ j &\longrightarrow 0 \text{ to } 6 \end{aligned}$$

$$i=1, j=0 \text{ (ic) } i=1 \text{ \& } j < w(i)$$

$$0 < 2 \longrightarrow T_{1,0} = 0$$

$$i=1, j=1 \text{ (ic) } i=1 \text{ \& } j < w(i)$$

$$1 < 2 \longrightarrow T_{1,1} = 0 \text{ (Here } j \text{ is equal to } w(i) \longrightarrow P(i)$$

$$\begin{aligned} i=1, j=2 \\ 2 \text{ o,} = T_{1,2} = 1. \end{aligned}$$

$$i=1, j=3 \\ 3 > 2, = T_{1,3} = 1.$$

$$i=1, j=4 \\ 4 > 2, = T_{1,4} = 1.$$

$$i=1, j=5 \\ 5 > 2, = T_{1,5} = 1.$$

$$i=1, j=6 \\ 6 > 2, = T_{1,6} = 1.$$

$$\Rightarrow \quad i=2, j=0 \text{ (ic) } i > 1, j < w(i) \\ 0 < 3 = T(2,0) = T(i-1,j) = T(2) \\ T_{2,0} = 0$$

$$i=2, j=1 \\ 1 < 3 = T(2,1) = T(i-1) \\ T_{2,1} = 0$$

## BASIC SEARCH AND TRAVERSAL TECHNIQUE:

### GRAPH

#### DEFINING GRAPH:

A graph  $g$  consists of a set  $V$  of vertices (nodes) and a set  $E$  of edges (arcs). We write  $G=(V,E)$ .  $V$  is a finite and non-empty set of vertices.  $E$  is a set of pair of vertices; these pairs are called as edges. Therefore,

$V(G)$ , read as  $V$  of  $G$ , is a set of vertices and  $E(G)$ , read as  $E$  of  $G$  is a set of edges.

An edge  $e=(v, w)$  is a pair of vertices  $v$  and  $w$ , and to be incident with  $v$  and  $w$ .

A graph can be pictorially represented as follows,

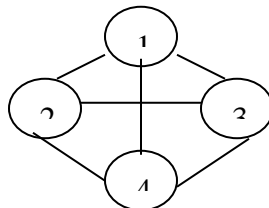


FIG: Graph G

We have numbered the graph as 1,2,3,4. Therefore,  $V(G)=(1,2,3,4)$  and

$$E(G) = \{(1,2),(1,3),(1,4),(2,3),(2,4)\}.$$

### **BASIC TERMINOLOGIES OF GRAPH:**

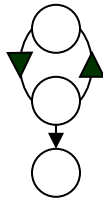
#### **UNDIRECTED GRAPH:**

An undirected graph is that in which, the pair of vertices representing the edges is unordered.

#### **DIRECTED GRAPH:**

An directed graph is that in which, each edge is an ordered pair of vertices, (i.e.) each edge is represented by a directed pair. It is also referred to as digraph.

#### **DIRECTED GRAPH**



#### **COMPLETE GRAPH:**

An  $n$  vertex undirected graph with exactly  $n(n-1)/2$  edges is said to be complete graph. The graph  $G$  is said to be complete graph .

### **TECHNIQUES FOR GRAPHS:**

- The fundamental problem concerning graphs is the reach-ability problem.
- In it simplest form it requires us to determine whether there exist a path in the given graph,  $G = (V, E)$  such that this path starts at vertex 'v' and ends at vertex 'u'.
- A more general form is to determine for a given starting vertex  $v \in V$  all vertex 'u' such that there is a path from v to u.
- This problem can be solved by starting at vertex 'v' and systematically searching the graph 'G' for vertex that can be reached from 'v'.
- We describe 2 search methods for this.
  - i. Breadth first Search and Traversal.
  - ii. Depth first Search and Traversal.

### **BREADTH FIRST SEARCH AND TRAVERSAL:**

#### **Breadth first search:**

In Breadth first search we start at vertex  $v$  and mark it as having been reached. The vertex  $v$  at this time is said to be unexplored. A vertex is said to have been explored by an algorithm when the algorithm has visited all vertices adjacent from it. All unvisited vertices adjacent from  $v$  are visited next. There are new unexplored vertices. Vertex  $v$  has now been explored. The newly visited vertices have not been explored and are put onto the end of the list of unexplored

vertices. The first vertex on this list is the next to be explored. Exploration continues until no unexplored vertex is left. The list of unexplored vertices acts as a queue and can be represented using any of the standard queue representations.

- In Breadth First Search we start at a vertex 'v' and mark it as having been reached (visited).
- The vertex 'v' is at this time said to be unexplored.
- A vertex is said to have been explored by an algorithm when the algorithm has visited all vertices adjacent from it.
- All unvisited vertices adjacent from 'v' are visited next. These are new unexplored vertices.
- Vertex 'v' has now been explored. The newly visit vertices have not been explored and are put on the end of a list of unexplored vertices.
- The first vertex on this list in the next to be explored. Exploration continues until no unexplored vertex is left.
- The list of unexplored vertices operates as a queue and can be represented using any of the start queue representation.

### ALGORITHM:

Algorithm BPS (v)

```
// A breadth first search of 'G' is carried out.
// beginning at vertex-v; For any node i, visit.
// if 'i' has already been visited. The graph 'v'
// and array visited [] are global; visited []
// initialized to zero.
{ y=v; // q is a queue of unexplored 1visited (v)= 1
repeat
{ for all vertices 'w' adjacent from u do
  { if (visited[w]=0) then
    { Add w to q;
      visited[w]=1
    }
  }
}
if q is empty then return;// No delete u from q;
} until (false)
}
```

algorithim : breadth first traversal  
algorithm BFT(G,n)

```
{
  for i= 1 to n do
    visited[i] =0;
  for i =1 to n do
```

```

    if (visited[i]=0)then BFS(i)
}

```

here the time and space required by BFT on an  $n$ -vertex  $e$ -edge graph one  $O(n+e)$  and  $O(n)$  resp if adjacency list is used. if adjacency matrix is used then the bounds are  $O(n^2)$  and  $O(n)$  resp

## DEPTH FIRST SEARCH

A depth first search of a graph differs from a breadth first search in that the exploration of a vertex  $v$  is suspended as soon as a new vertex is reached. At this time the exploration of the new vertex  $u$  begins. When this new vertex has been explored, the exploration of  $u$  continues. The search terminates when all reached vertices have been fully explored. This search process is best-described recursively.

```

Algorithm DFS(v)
{
visited[v]=1
for each vertex w adjacent from v do
{
If (visited[w]=0)then
DFS(w);
}
}

```

## TOPOLOGICAL SORT

A topological sort of a DAG  $G$  is an ordering of the vertices of  $G$  such that for every edge  $(e_i, e_j)$  of  $G$  we have  $i < j$ . That is, a topological sort is a linear ordering of all its vertices such that if DAG  $G$  contains an edge  $(e_i, e_j)$ , then  $e_i$  appears before  $e_j$  in the ordering. DAG is cyclic then no linear ordering is possible.

In simple words, a topological ordering is an ordering such that any directed path in DAG  $G$  traverses vertices in increasing order.

It is important to note that if the graph is not acyclic, then no linear ordering is possible. That is, we must not have circularities in the directed graph. For example, in order to get a job you need to have work experience, but in order to get work experience you need to have a job.

**Theorem:** *A directed graph has a topological ordering if and only if it is acyclic.*

*Proof:*

Part 1.  $G$  has a topological ordering if  $G$  is acyclic.



Let  $G$  is topological order.

Let  $G$  has a cycle (Contradiction).

Because we have topological ordering. We must have  $i_0, < i_1, < \dots < i_{k-1} < i_0$ , which is clearly impossible.

Therefore,  $G$  must be acyclic.

Part 2.  $G$  is acyclic if has a topological ordering.

Let is  $G$  acyclic.

Since is  $G$  acyclic, must have a vertex with no incoming edges. Let  $v_1$  be such a vertex. If we remove  $v_1$  from graph, together with its outgoing edges, the resulting digraph is still acyclic.

Hence resulting digraph also has a vertex \*

### ALGORITHM: TOPOLOGICAL\_SORT( $G$ )

1. For each vertex find the finish time by calling DFS( $G$ ).
2. Insert each finished vertex into the front of a linked list.
3. Return the linked list.

Example:

1. given graph  $G$ ; start node  $u$

Diagram

with no incoming edges, and we let  $v_2$  be such a vertex. By repeating this process until digraph  $G$  becomes empty, we obtain an ordering  $v_1 < v_2 < \dots, v_n$  of vertices of digraph  $G$ . Because of the construction, if  $(v_i, v_j)$  is an edge of digraph  $G$ , then  $v_i$  must be detected before  $v_j$  can be deleted, and thus  $i < j$ . Thus,  $v_1, \dots, v_n$  is a topological sorting.

Total running time of topological sort is  $\theta(V+E)$ . Since DFS( $G$ ) search takes  $\theta(V+E)$  time and it takes  $O(1)$  time to insert each of the  $|V|$  vertices onto the front of the linked list.