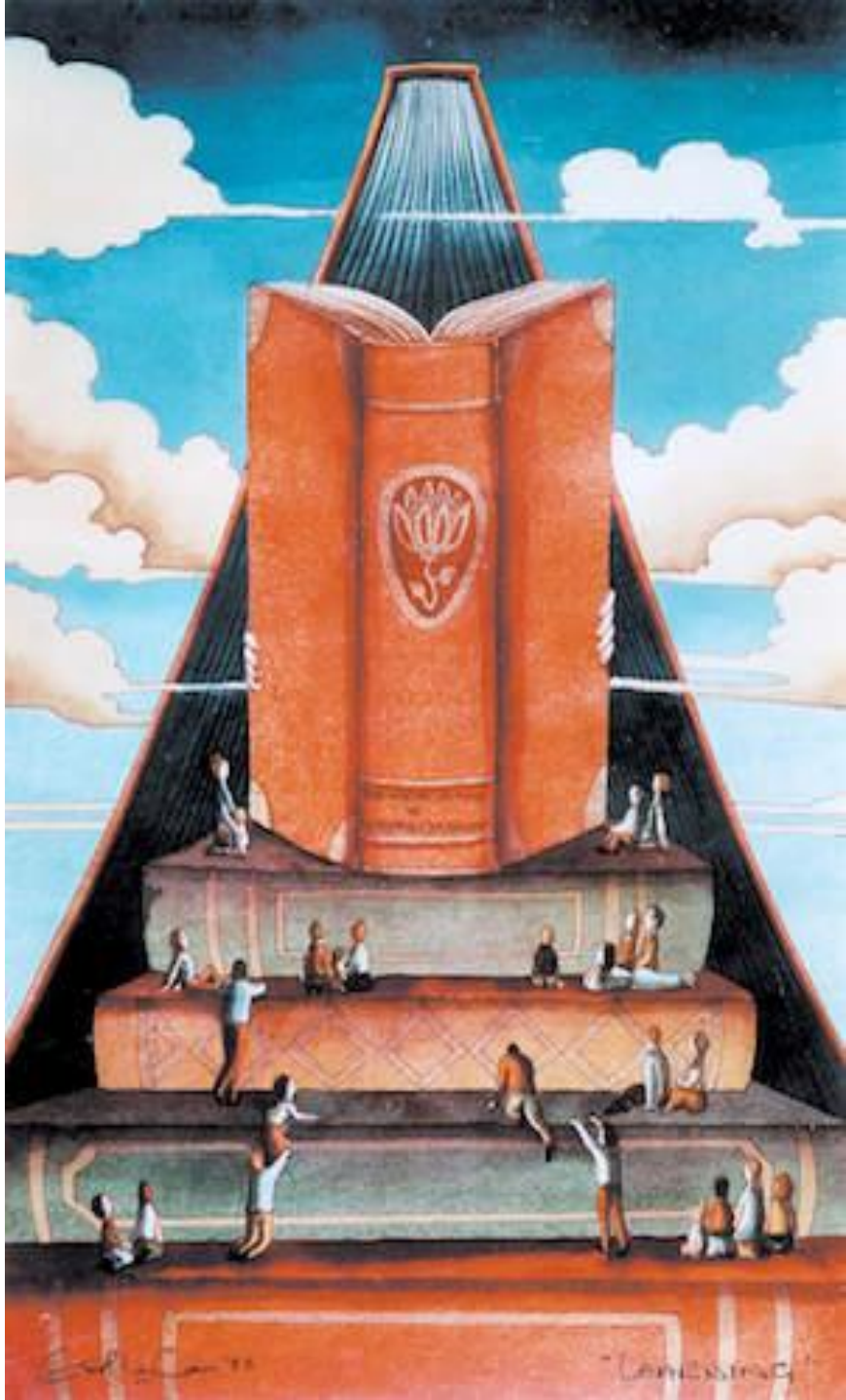
DYNAMIC PROGRAMMING



Outlines

- Introduction
- Multistage Graphs
- All-Pairs Shortest Paths
- Single-Source Shortest Paths: General Weights
- Optimal Binary Search Trees
- Traveling Salesperson Problem

Introduction

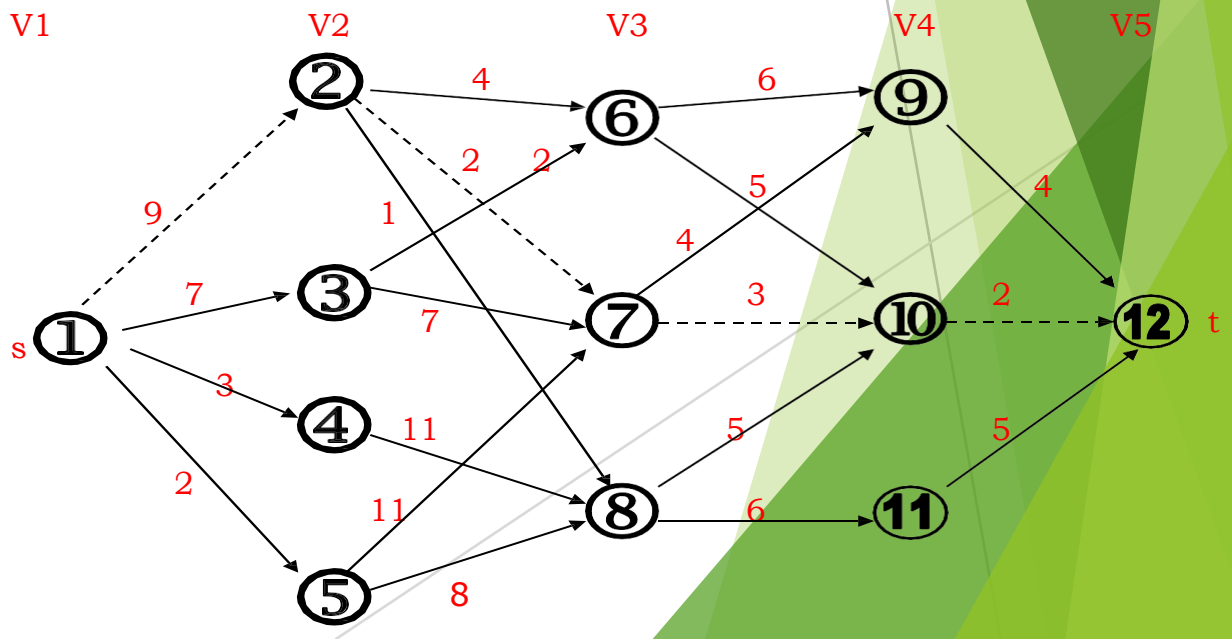
- Dynamic programming is an algorithm design method that can be used when the solution of a problem can be viewed as the result of a sequence of decisions
- Dynamic programming is typically applied to optimization problems.
- Problems that can be solved by dynamic programming satisfy the principle of optimality
- Important feature of the dynamic programming approach is that optimal solutions to subproblems are retained so as to avoid recomputing their values
- Dynamic programming is a bottom-up technique

Multistage Graphs

- A multistage graph G = (V,E) is a directed graph in which the vertices are portioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$.
- In addition, if <u,v> is an edge in E, then $u \in V_i$ and $v \in V_{i+1}$ for some i, $1 \leq i < k$.
- If there will be only one vertex, then the sets V_1 and V_k are such that $|V_1| = |V_k| = 1$.
- Let ‘s’ and ‘t’ be the source and destination respectively
- The cost of a path from source (s) to destination (t) is the sum of the costs of the edges on the path.
- The MULTISTAGE GRAPH problem is to find a minimum cost path from ‘s’ to ‘t’.

Multistage Graphs (Cont..)

- 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.
- This MULISTAGE GRAPH problem can be solved in 2 ways.
  - Forward Method
  - Backward Method

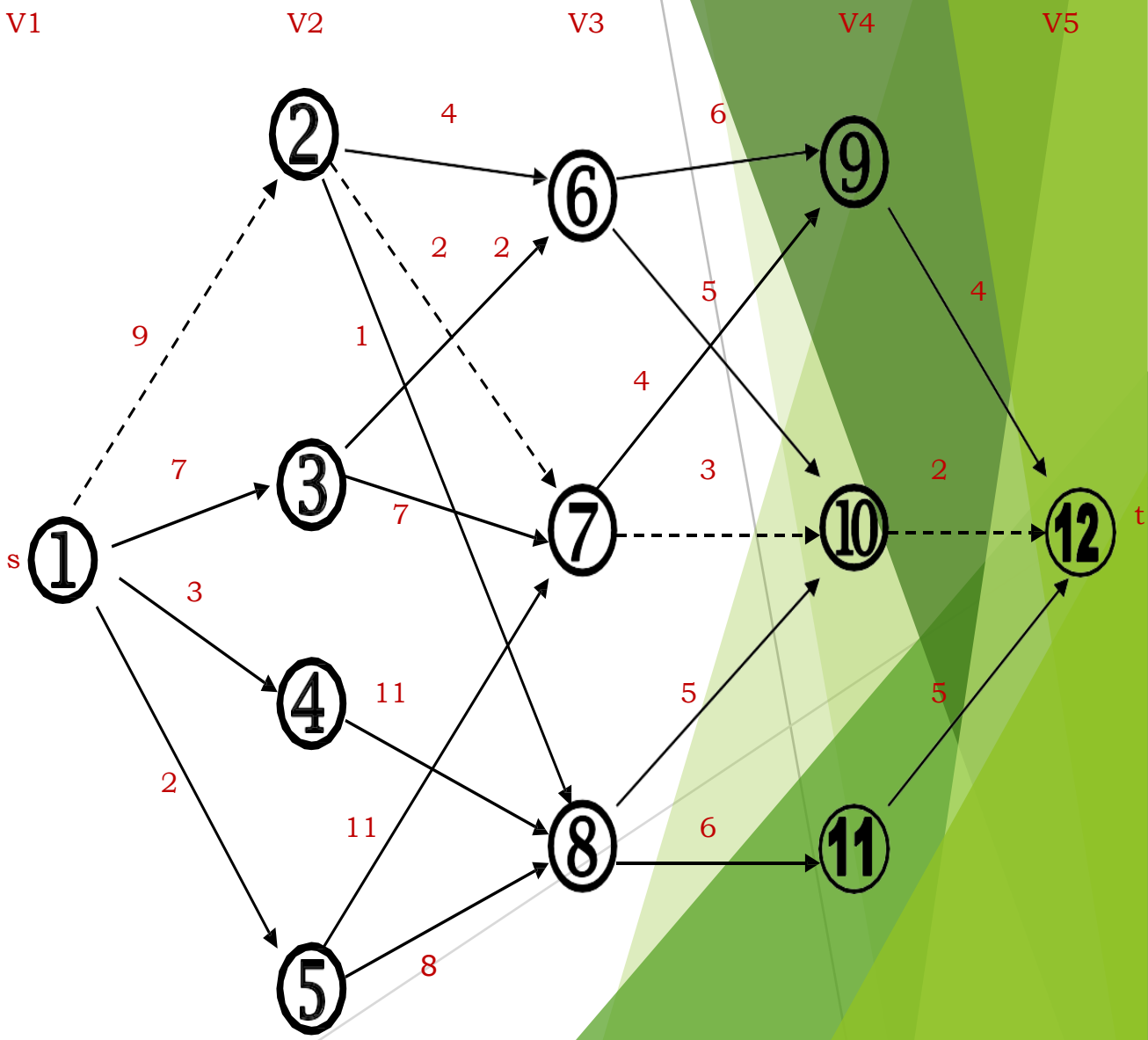


Multistage Graphs (Cont..)

- **Forward Method**
  - Assume that there are ‘k’ stages in a graph.
  - In this FORWARD approach, we will find out the cost of each and every node starting from the ‘k’ th stage to the 1st stage.
  - We will find out the path (i.e.) minimum cost path from source to the destination (i.e,) [Stage-1 to Stage-k].

Multistage Graphs (Cont..)

- 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.



Multistage Graphs (Cont..)

- For finding the path, start from vertex-1 then the distance array D(1) will give the minimum cost neighbor 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.

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

$$\text{Cost}(i,j) = \min_{\substack{l \in V_{i+1} \\ (j,l) \in E}} \{C(j,l) + \text{Cost}(i+1,l)\}$$

Multistage Graphs (Cont..)

Cost(8) = min {C (8,10) + Cost (10), C (8,11) + Cost (11) }
= min (5 + 2, 6 + 5)
= min (7,11)
= 7

cost(8) =7 =>D(8)=10

cost(7) = min(c (7,9)+ cost(9),c (7,10)+ cost(10))
(4+4,3+2)
= min(8,5)
= 5

cost(7) = 5 =>D(7) = 10

cost(6) = min (c (6,9) + cost(9),c (6,10) +cost(10))
= min(6+4 , 5 +2)
= min(10,7)
= 7

cost(6) = 7 =>D(6) = 10

Multistage Graphs (Cont..)

$$\text{cost}(5) = \min(c(5,7) + \text{cost}(7), c(5,8) + \text{cost}(8))$$

$$= \min(11+5 , 8 +7)$$
$$= \min(16,15)$$
$$= 15$$

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

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

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

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

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

Multistage Graphs (Cont..)

cost(2) = min (c (2,6) + cost(6),c (2,7) +cost(7) ,c (2,8) +cost(8))
= min(4+7 , 2+5 , 1+7)
= min(11,7,8)
= 7

cost(2) = 7 =>D(2) = 7

cost(1) = min (c (1,2)+cost(2) ,c (1,3)+cost(3) ,c (1,4)+cost(4) ,c(1,5)+cost(5))
= min(9+7 , 7 +9 , 3+18 , 2+15)
= min(16,16,21,17)
= 16
cost(1) = 16 =>D(1) = 2

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

(i.e.) ① ⟶ ② ⟶ ⑦ ⟶ ⑩ ⟶ ⑫

Multistage Graphs (Cont..)

D (1) = 2
D (2) = 7
D (7) = 10
D (10) = 12

So, the minimum –cost path is,

∴ The cost is 9+2+3+2+=16

Multistage Graphs (Cont..)

```
1   Algorithm FGraph(G, k, n, p)
2   // The input is a k-stage graph G = (V, E) with n vertices
3   // indexed in order of stages. E is a set of edges and c[i, j]
4   // is the cost of ⟨i, j⟩. p[1 : k] is a minimum-cost path.
5   {
6       cost[n] := 0.0;
7       for j := n − 1 to 1 step −1 do
8       { // Compute cost[j].
9           Let r be a vertex such that ⟨j, r⟩ is an edge
10          of G and c[j, r] + cost[r] is minimum;
11          cost[j] := c[j, r] + cost[r];
12          d[j] := r;
13      }
14      // Find a minimum-cost path.
15      p[1] := 1; p[k] := n;
16      for j := 2 to k − 1 do p[j] := d[p[j − 1]];
17  }
```

Multistage Graphs (Cont..)

- **Backward Method**
  - If there are ‘k’ stages in a graph, using back ward approach we will find out the cost of each & every vertex starting from 1^{st} stage to the k^{th} stage.
  - We will find out the minimum cost path from destination to source (i.e.,) from stage k to stage 1.
- Process:
  - It is similar to forward approach, but differs only in two or three ways.
  - Maintain a cost matrix to store the cost of every vertices and a distance matrix to store the minimum distance vertex.
  - Find out the cost of each and every vertex starting from vertex 1 up to vertex k.

Multistage Graphs (Cont..)

- Process:
  - 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.

$$\text{Cost}(1) = 0 \Rightarrow D(1)=0$$
$$\text{Cost}(2) = 9 \Rightarrow D(2)=1$$
$$\text{Cost}(3) = 7 \Rightarrow D(3)=1$$
$$\text{Cost}(4) = 3 \Rightarrow D(4)=1$$
$$\text{Cost}(5) = 2 \Rightarrow D(5)=1$$

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

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

Multistage Graphs (Cont..)

Cost(7) =min(c (3,7) + cost(3),c (5,7) + cost(5) ,c (2,7) + cost(2))
=min(14,13,11)

cost(7) = 11 =>D(7)=2

Cost(8) =min(c (2,8) + cost(2),c (4,8) + cost(4) ,c (5,8) +cost(5))
=min(10,14,10)

cost(8) = 10 =>D(8)=2

Cost(9) =min(c (6,9) + cost(6),c (7,9) + cost(7))
=min(15,15)

cost(9) = 15 =>D(9)=6

Multistage Graphs (Cont..)

Cost(10)=min(c(6,10)+cost(6),c(7,10)+cost(7)),c (8,10)+cost(8)) =min(14,14,15)
cost(10)= 14 =>D(10)=6

Cost(11) =min(c (8,11) + cost(8))

cost(11) = 16 =>D(11)=8

cost(12)=min(c(9,12)+cost(9),c(10,12)+cost(10),c(11,12)+cost(11))
=min(19,16,21)

cost(12) = 16 =>D(12)=10

Multistage Graphs (Cont..)

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.

Multistage Graphs (Cont..)

```
1   Algorithm BGraph(G, k, n, p)
2   // Same function as FGraph
3   {
4       bcost[1] := 0.0;
5       for j := 2 to n do
6       { // Compute bcost[j].
7           Let r be such that ⟨r, j⟩ is an edge of
8           G and bcost[r] + c[r, j] is minimum;
9           bcost[j] := bcost[r] + c[r, j];
10          d[j] := r;
11      }
12      // Find a minimum-cost path.
13      p[1] := 1; p[k] := n;
14      for j := k − 1 to 2 do p[j] := d[p[j + 1]];
15  }
```

All-Pairs Shortest Paths

- Let G=<V,E> be a directed graph 'V' is a set of nodes and 'E' 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 V={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)>=for all i & j, and L(i,j)=infinity, 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.

All-Pairs Shortest Paths (Cont..)

- 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.

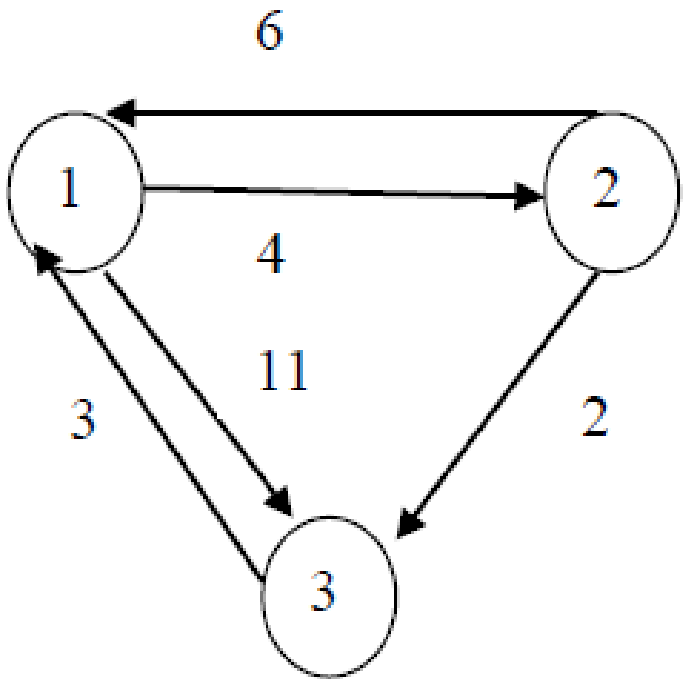
$$A^k(i, j) = min\{^{k-}(i, j), A^{k-}(i, k) + A^{k-}(k - 1, j)\}\ k \geq 1$$

$$A^k(i, j) - Length\ of\ the\ shortest\ path\ from\ i\ to\ j.$$

- Let G=(V,E) is a directed graph with 'N' Vertices. Let cost be the cost of the adjacency matrix for G such that cost(i,i)=0 ($1 \leq i \leq n$) then cost(i,j) is the length or cost of the edge <i,j>. If <i,j> exists the edge belongs to E(G) and cost(i,j)=α, if $i \neq j$ and <i,j> $\notin$ E(G)

All-Pairs Shortest Paths (Cont..)

Example:



Matrix Representation:

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \alpha & 0 \end{bmatrix}$$

All-Pairs Shortest Paths (Cont..)

$A^0 =$

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \alpha & 0 \end{bmatrix}$$

The path of the vertices through vertex-1 is given as

$A^1 =$

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

All-Pairs Shortest Paths (Cont..)

The path of the vertices through vertex-2 is given as

$A^2 =$

$$\begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

The path of the vertices through vertex-3 is given as

$A^3 =$

$$\begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

All-Pairs Shortest Paths (Cont..)

- At 1st 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).

All-Pairs Shortest Paths (Cont..)

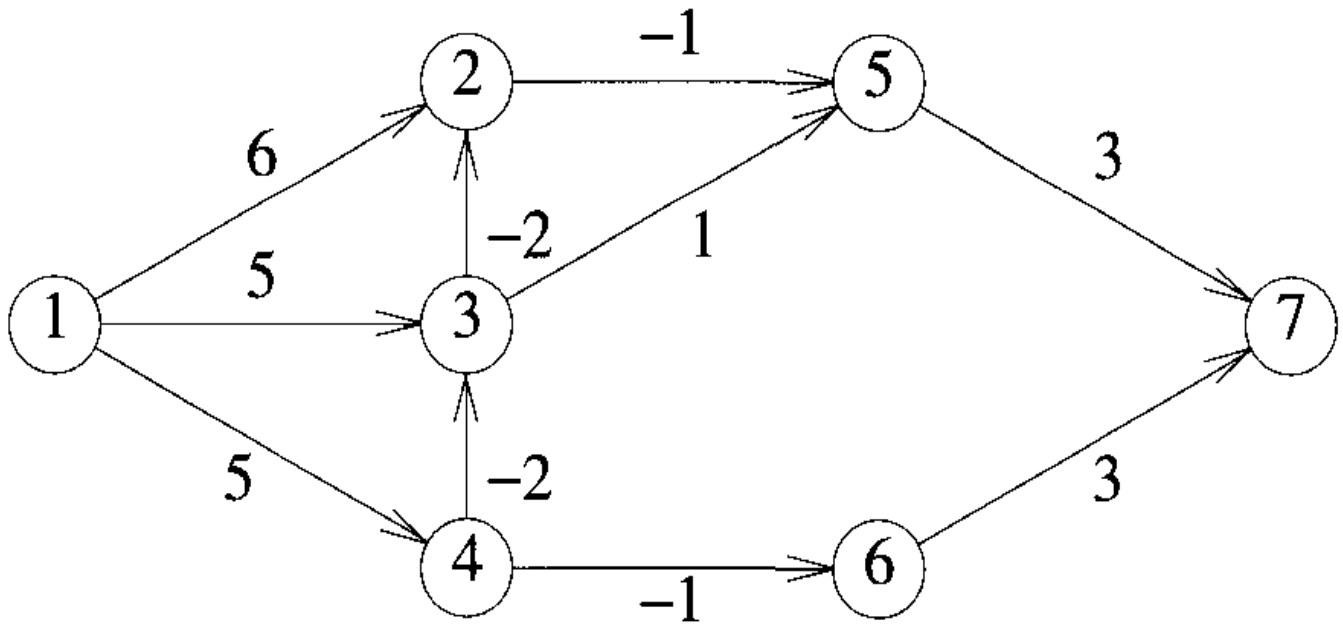
```
0   Algorithm AllPaths(cost, A, n)
1   // cost[1 : n, 1 : n] is the cost adjacency matrix of a graph with
2   // n vertices; A[i, j] is the cost of a shortest path from vertex
3   // i to vertex j. cost[i, i] = 0.0, for 1 ≤ i ≤ n.
4   {
5       for i := 1 to n do
6           for j := 1 to n do
7               A[i, j] := cost[i, j]; // Copy cost into A.
8       for k := 1 to n do
9           for i := 1 to n do
10              for j := 1 to n do
11                  A[i, j] := min(A[i, j], A[i, k] + A[k, j]);
12  }
```

Single-Source Shortest Paths: General Weights

- The recurrence relation for distance is as follows:

$$dist^k[u] = \min\{dist^{k-1}[u], \min_i\{dist^{k-1}[i] + cost[i,u]\}\}$$

- This recurrence can be used to compute $dist^k$ from $dist^{k-1}$, for k=2,3,…,n-1



Single-Source Shortest Paths: General Weights

- $dist^k[1]=0$ for all k as this is source node
- $dist^1[2]=6$, $dist^1[3]=5$, $dist^1[4]=5$
- The distance $dist^1[5/6/7]$ is inf as there are no edges to these from 1

	$dist^k[1..7]$						
k	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

$$dist^2[2] = \min\{dist^1[2], \min_i dist^1[i] + cost[i,2]\}$$
$$= \min\{6, 0+6, 5-2, 5+\infty, \infty+\infty, \infty+\infty, \infty+\infty\} = 3$$

Single-Source Shortest Paths: General Weights

```
1   Algorithm BellmanFord(v, cost, dist, n)
2   // Single-source/all-destinations shortest
3   // paths with negative edge costs
4   {
5       for i := 1 to n do // Initialize dist.
6           dist[i] := cost[v, i];
7       for k := 2 to n − 1 do
8           for each u such that u ≠ v and u has
9                       at least one incoming edge do
10                  for each ⟨i, u⟩ in the graph do
11                      if dist[u] > dist[i] + cost[i, u] then
12                          dist[u] := dist[i] + cost[i, u];
13  }
```

Travelling Salesperson Problem

- Given a complete, weighted graph on n nodes, find the least weight Hamiltonian cycle, a cycle that visits every node once.
- Though this problem is easy enough to explain, it is very difficult to solve.
- Finding all the Hamiltonian cycles of a graph takes exponential time. Therefore, TSP is in the class NP.

Travelling Salesperson Problem (Cont..)

- The TSP has many practical applications
  - Manufacturing
  - Plane routing
  - Telephone routing
  - Networks
  - Traveling salespeople
  - Structure of crystals

Travelling Salesperson Problem (Cont..)

- Let G=(V,E) be a directed graph with edge cost c_{ij}
- The variable c_{ij} is defined such that $c_{ij}>0$ for all i and j and $c_{ij}=\infty$ if $(i,j) \notin E$
- A tour of G is a directed simple cycle that includes every vertex in V
- The cost of a tour is the sum of the cost of the edges on the tour
- The traveling salesperson problem is to find a tour of minimum cost

Travelling Salesperson Problem (Cont..)

- We shall regard a tour to be a simple path that starts and ends at vertex 1
- Every tour consists of an edge (1,k) for some $k \in V - \{1\}$ and a path from vertex k to vertex 1
- The path from vertex k to vertex 1 goes through each vertex in $V - \{1, k\}$ exactly once
- It is easy to see that if the tour is optimal, then the path from k to 1 must be a shortest k to 1 path going through all vertices in $V - \{1, k\}$
- Hence, the principle of optimality holds

Travelling Salesperson Problem (Cont..)

- Let 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
- The function g(1, V-{1}) is the length of an optimal salesperson tour
- From the principal of optimality it follows that

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

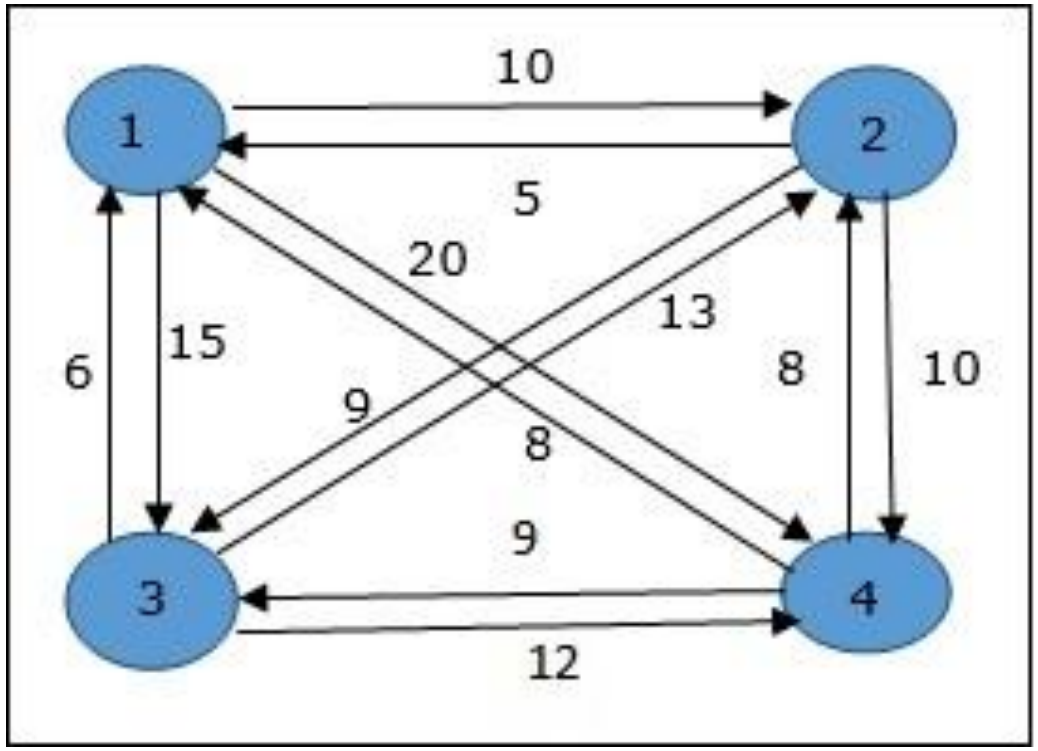
- Generalizing the above, we obtain (for $i \notin S$)

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

Travelling Salesperson Problem (Cont..)

- First equation can be solved for g(1,V-{1}) if we know g(k,V-{1,k}) for all choices of k
- The g values can be obtained by using the second equation
- Clearly, $g(i, \emptyset) = c_{i1}$
- Hence, we can obtain g(i,S) for all S of size 1
- Then we can obtain g(i,S) for S with |S|=2 and so on..
- When |S|<n-1, the values of i and S for which g(i,S) is needed are such that $i \neq 1, 1 \notin S, and\ i \notin S$

TSP Example



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Matrix C

TSP Example (Cont..)

- Thus,

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

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

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

- Now,

$$g(2,\{3\}) = c_{23} + g(3,\phi) = 15 \qquad g(2,\{4\}) = 18$$
$$g(3,\{2\}) = 18 \qquad g(3,\{4\}) = 20$$
$$g(4,\{2\}) = 13 \qquad g(4,\{3\}) = 15$$

TSP Example (Cont..)

Next, we compute $g(i, S)$ with $|S| = 2$, $i \neq 1$, $1 \notin S$ and $i \notin S$.

$$
\begin{aligned}
g(2, \{3,4\}) &= \min \{c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\})\} = 25 \\
g(3, \{2,4\}) &= \min \{c_{32} + g(2, \{4\}), c_{34} + g(4, \{2\})\} = 25 \\
g(4, \{2,3\}) &= \min \{c_{42} + g(2, \{3\}), c_{43} + g(3, \{2\})\} = 23
\end{aligned}
$$

- Finally,

$$
\begin{aligned}
g(1, \{2,3,4\}) &= \min \{c_{12} + g(2, \{3,4\}), c_{13} + g(3, \{2,4\}), c_{14} + g(4, \{2,3\})\} \\
&= \min \{35, 40, 43\} \\
&= 35
\end{aligned}
$$

Thanks for your Attention