

Dynamic Programming

- **Dynamic Programming** is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions.
- One way to solve problems for which it is not possible to make a sequence of stepwise decisions leading to an optimal decision sequence is to try all possible decision sequences.
- Enumerate all decision sequences and then pick out the best. But the time and space requirements may be prohibitive.
- Dynamic programming often drastically reduces the amount of enumeration by avoiding the enumeration of some decision sequences that cannot possibly be optimal.
- In dynamic programming an optimal sequence of decisions is obtained by making explicit appeal to the **principle of optimality**



Richard Ernest Bellman- 1950

- A **dynamic programming algorithm** proceeds by solving small problems, then combining them to find the solution to larger problems.
- Dynamic programming relies on working “**from the bottom up**” and saving the results of solving simpler problems
 - **These solutions to simpler problems are then used to compute the solution to more complex problems**
- **Dynamic programming solutions can often be quite complex and tricky**
- Dynamic programming is a technique for finding an *optimal* solution
- **Only problems that satisfy the principle of optimality are suitable for dynamic programming solutions.**

Principle of optimality:

The principle of optimality states that, an optimal sequence of decisions has the property that, whatever the initial state and decisions are, the remaining decisions must constitute an optional decision sequence ,with regard to the state resulting from the first decision.

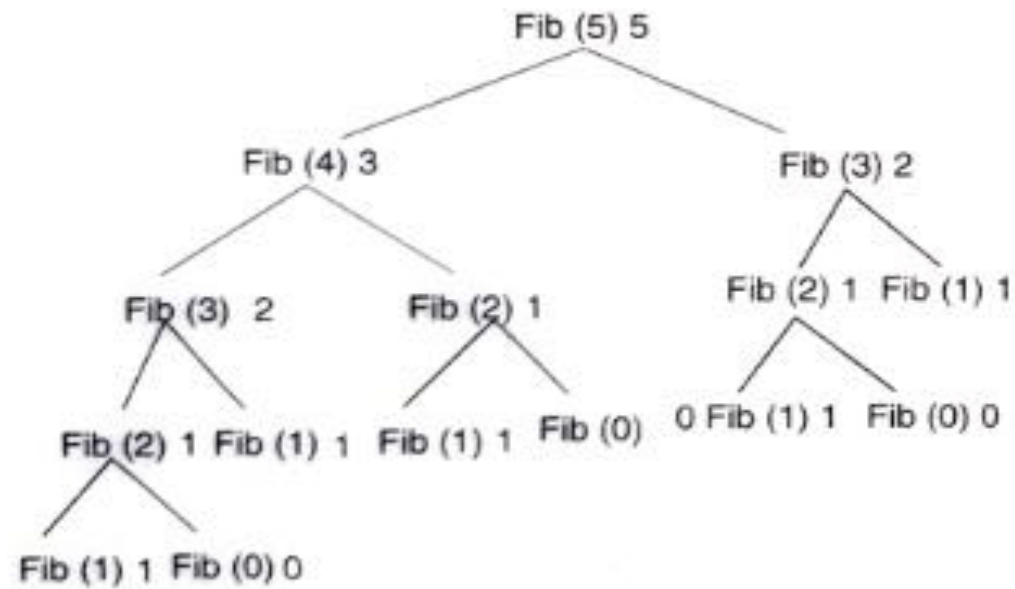
- The principle of optimality holds if every optimal solution to a problem contains optimal solutions to all sub problems
- The principle of optimality does *not* say if you have optimal solutions to all sub problems then you can combine them to get an optimal solution
- In the greedy method only one decision sequence is generated.
- In dynamic programming many decision sequences may be generated.
- Dynamic programming algorithms often have a polynomial complexity

- In dynamic programming technique, the same sub problems are not solved again and again but they are solved once and the results are stored in a table.
- Example:
- The recursive formula for calculating the nth element of the Fibonacci series is

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

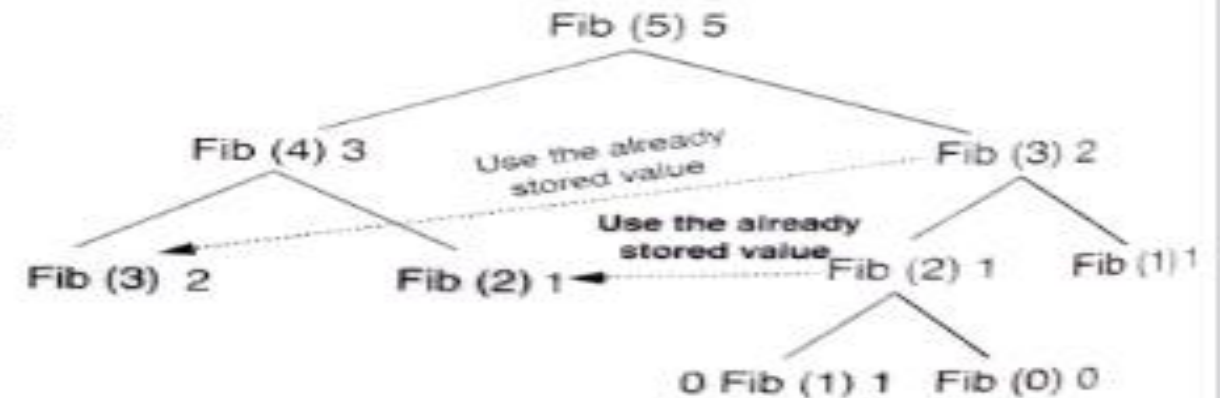
- Where $f(0)=0$ and $f(1)=1$ are base values when $n=0$ or $n=1$
- $(n-1)$ and $(n-2)$ needed to compute nth number of the series

Divide-and-Conquer Technique



(a)

Dynamic Programming



(b)

Characteristics of Dynamic Programming

To apply dynamic programming, an optimization problem must have:

- 1. Optimal substructure:** The problem can be decomposed into several sub problems. First, optimal solutions to sub problems are found and the solution of the original problem is expressed in terms of solutions for smaller problems. If an optimal solution contains optimal sub solutions, then a problem exhibits optimal substructure.
- 2. Overlapping sub problems:** When the recursive algorithm revisits same sub problems over and over, the problems is said to have overlapping sub problems.

Components of Dynamic Programming

1. Characterize the optimal substructure
2. Use the recursive solution
3. Compute the optimal value
4. Construct the optimal solution

Dynamic Vs Greedy Algorithm

Dynamic programming :

- Make a choice at each step
- Choice depends on knowing optimal solutions to sub problems.
- Solve sub problems first.
- Solve bottom-up.

Greedy :

- Make a choice at each step.
- Make the choice before solving the sub problems.
- Solve top-down.

Dynamic Vs Divide & Conquer

Dynamic programming

- partition a problem into overlapping sub problems and independent ones
- store solutions to sub problems

divide-and-conquer

- Partition a problem into sub problems and solve them individually
- Sub problems generally need not overlap
- Combine the sub problem solutions
- no need to store solutions to sub problems

All Pairs Shortest Path

- The all-pairs shortest path problem can be considered the mother of all routing problems.
- It aims to compute the shortest path from each vertex v to every other u .
- The problem:
find the shortest path between every pair of vertices of a graph
- The graph:
may contain negative edges but no negative cycles
- A representation:
a weight matrix where
$$W(i,j)=0 \text{ if } i=j.$$
$$W(i,j)=\infty \text{ if there is no edge between } i \text{ and } j.$$
$$W(i,j)=\text{“weight of edge”}$$



Matrix becomes

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{bmatrix} \end{matrix}$$

Problem definition

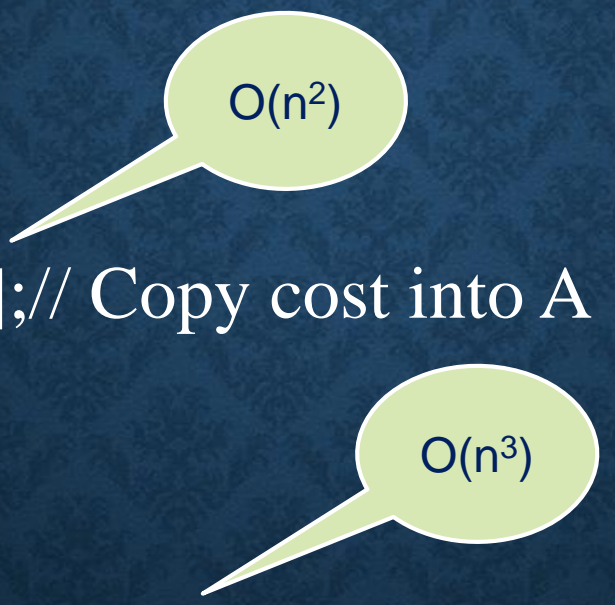
Determine a matrix A such that $A(i,j)$ is the length of a shortest path from i to j

Solution is

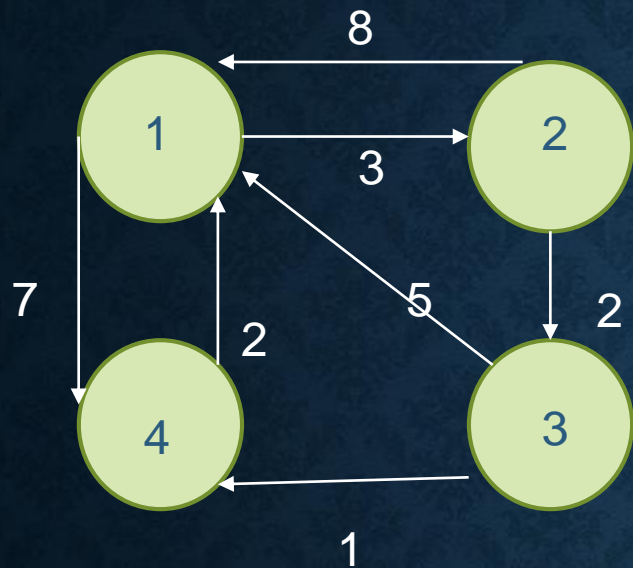
$$A^k(i, j) = \min\{A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j)\}, \quad k \geq 1$$

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 i to vertex j
3. Cost[i,i]=0.0, for  $1 \leq i \leq n$ .
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. }
```



The diagram shows two green speech bubbles with pointers indicating complexity. The first bubble, labeled $O(n^2)$, points to the nested loops for i and j in lines 5-7. The second bubble, labeled $O(n^3)$, points to the triple nested loops for k , i , and j in lines 8-11.



$A^0 = A^2$

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

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

0	3	∞	7
8	0	2	15
5	8	0	1
2	5	∞	0

A^1

Consider intermediate vertex as vertex 1

$$\begin{aligned} \text{Min}\{A^0[2,3] &= A^0[2,1] + A^0[1,3] \} \\ 2 &< 8 + \infty \\ \text{Min}\{A^0[2,4] &= A^0[2,1] + A^0[1,4] \} \\ \infty &> 8 + 7 = 15 \\ \text{Min}\{A^0[3,2] &= A^0[3,1] + A^0[1,2] \} \\ \infty &> 5 + 3 = 8 \\ \text{Min}\{A^0[3,4] & \\ \text{Min}\{A^0[4,2] & \\ \text{Min}\{A^0[4,3] & \end{aligned}$$

A₂

0	3	5	7
8	0	2	15
5	8	0	1
2	5	7	0

Consider intermediate vertex as vertex 2

A² [1,3]

$$\text{Min}\{A^1[1,3] \quad \infty \quad > \quad A^1[1,2] + A^1[2,3] \}$$
$$3 + 2 = 5$$

A² [1,4]

$$\text{Min}\{A^1[1,4] \quad 7 \quad < \quad A^1[1,2] + A^1[2,4] \}$$
$$3 + 15 = 18$$

A² [3,1]

A² [3,4]

A² [4,1]

A² [1,4]

A² [4,3]

A³

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

Consider intermediate vertex as vertex 3

A³ [1,2]

Min{A²[1,2]

3 >

A²[1,3] + A¹[3,2] }

5+8=13

A³ [1,4]

A³ [2,1]

A³ [2,4]

A³ [4,1]

A³ [4,2]

A⁴

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

Consider intermediate vertex as vertex 4

A⁴ [1,2]

Min{A³[1,2]

3 >

A⁴ [1,3]

A⁴ [2,1]

A⁴ [2,3]

A⁴ [3,1]

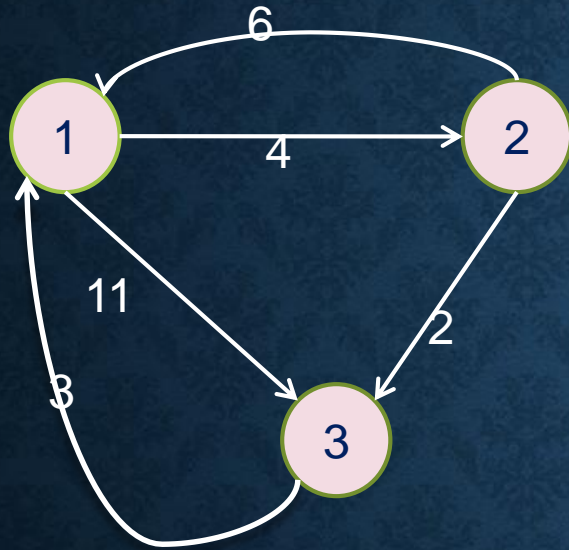
A⁴ [3,2]

A³[1,4] + A¹[4,2] }
6+5=11

Complexity Analysis of All Pair Shortest Path

- Line 7 ---- $O(n^2)$
- Line 11---- $O(n^3)$
- The total time is $O(n^3)$

Example Digraph



A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

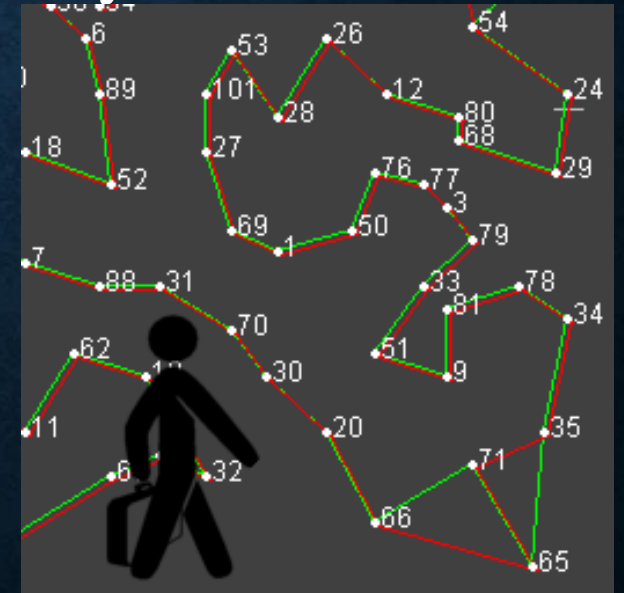
A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

Travelling Salesman Problem

Problem Statement

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

The goal of the problem is to find the shortest path for the salesperson to travel visiting each city and then returning to the starting city.



- Let $G=(V, E)$ be a directed graph with edge cost c_{ij}
- $C_{ij} = \infty$ if $(i,j) \notin E$ and let $|V| = n$.
- A tour of G is a directed simple cycle that includes every vertex in V
- Every tour starts & ends at vertex 1.
- Cost of the tour is the sum of cost of edges on the tour
- The travelling salesman problem is to find a tour of minimum cost

Applications

- Solving optimization problems in the areas of bioinformatics, operating system(time sharing and scheduling),control theory(flight control, cruise control, Robotics),information theory, operations research, artificial intelligence, inventory management.
- Fibonacci Sequence.
- Multistage graph
- Matrix-chain multiplication
- Longest common sequence.
- 0/1 knapsack problem
- All pair shortest path problem
- Travelling salesperson problem.

Solution

- A tour to be a simple path that starts and ends at vertex 1.
- Since the tour is the shortest path goes to all vertex exactly once .
- So the principal of optimality holds.

- Let $g(i,S)$ be the length of a shortest path starting at vertex 1, going through all vertex in S , and terminating at vertex 1
- Let The function $g(1,V-\{1\})$ is the length of an optimal salesperson tour
- From the optimality it follows

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

$$C_{1k} = 10 + 25 = 35 //$$

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

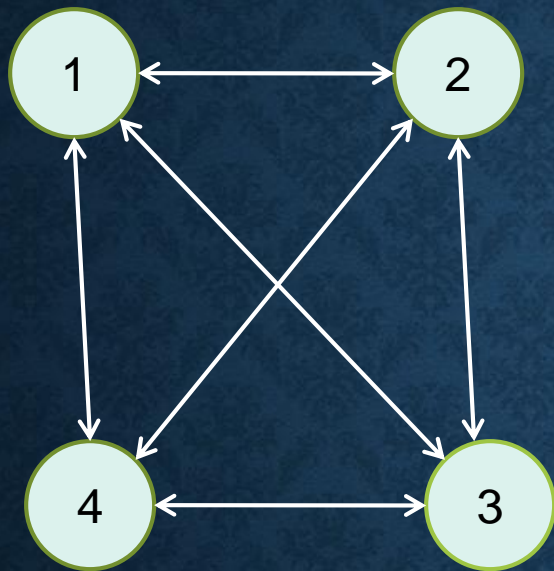
- Generalizing (1) we obtain

$O(n^2 2^n)$

$$g(i, S) = \min_{j \in S} \{c_{ij} + g(j, S - \{j\})\} \dots \dots \dots (2)$$

- So

$$g(i, \phi) = c_{i1} \quad 1 \leq i \leq n \quad \dots \dots \dots (3)$$



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

Solution

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

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

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

From (2) we get

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

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

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

$$g(3,\{4\})=C_{34} + g(4,\Phi)=12+8=20$$

$$g(4,\{2\})=C_{42} + g(2,\Phi)=8+5=13$$

$$g(4,\{3\})=C_{43} + g(3,\Phi)=9+6=15$$

$$g(2,\{3,4\})=\min\{C_{23} + g(3,\{4\}), C_{24} + g(4,\{3\})\}=\min(9+20,10+15)=25$$

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

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

$$g(1,\{2,3,4\})=\min\{C_{12} + g(2,\{3,4\}), C_{13} + g(3,\{2,4\}), C_{14} + g(4,\{2,3\})\} = \min(10+25, \quad 15+25, \quad 20+23)$$

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

$$=35$$

From (2)

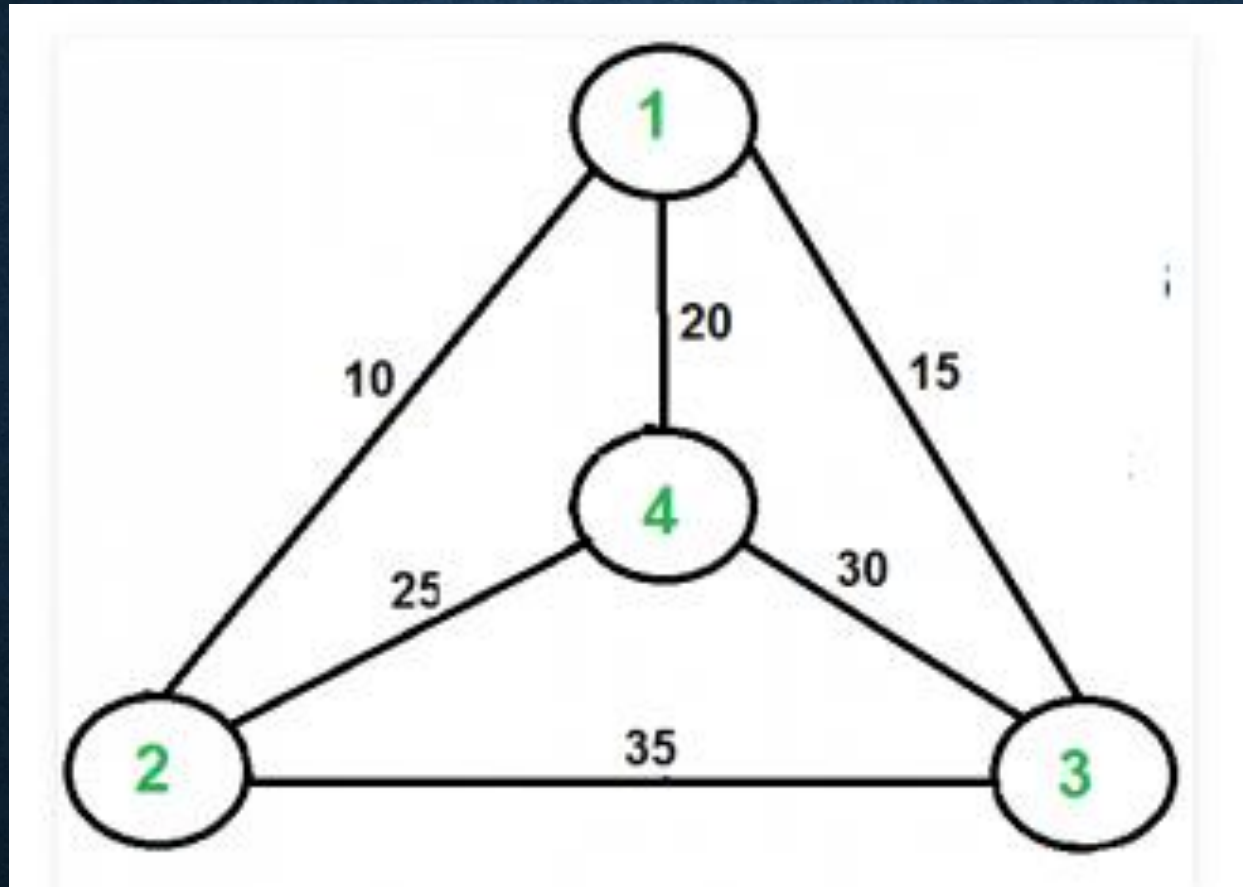
$g(1, \{2, 3, 4\}) = 35$ is min so $j=2$

Now

From 2 node min value=23 so $j=4$

Next is 3

So the path is 1,2,4,3,1



Cost 80

Path : 1-2-4-3-1

Single source shortest paths :
General Weights /Bellman
and Ford algorithm

- Some or all the edges of directed graph G may have negative length. When negative edge lengths are permitted, we require that the graph have no cycles of negative length.
- This is necessary to ensure that shortest paths consists of a finite number of edges.
- Let $\text{dist}^l[u]$ be the length of a shortest path from the source vertex v to vertex u under the constraint that the shortest path contains at most l edges.
- Then $\text{dist}^1[u] = \text{cost}[v, u]$, $1 \leq u \leq n$. $\text{dist}^{n-1}[u]$ is the length of an unrestricted shortest path from v to u .
- This problem is to compute $\text{dist}^{n-1}[u]$ for all u . this can be done using the dynamic programming methodology.

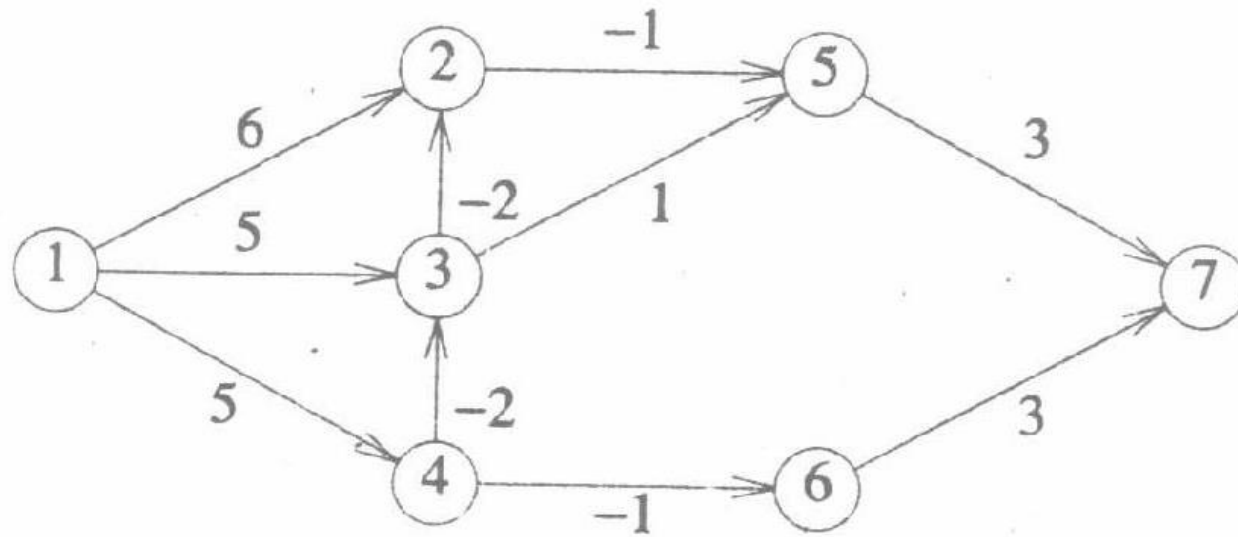
- Following recurrence can be used for dist:
- $\text{dist}^k[u] = \min \{ \text{dist}^{k-1}[u], \min (\text{dist}^{k-1}[i] + \text{cost}[i, u]) \}$
- This recurrence can be used to compute $\text{dist}^k[u]$ from dist^{k-1} for $k= 2, 3, \dots, n-1$

- Example

1 to these nodes. The distance $dist^1[]$ is ∞ for the nodes 5, 6, and 7 since there are no edges to these from 1.

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

Here the terms $0 + 6, 5 - 2, 5 + \infty, \infty + \infty, \infty + \infty$, and $\infty + \infty$ correspond to a choice of $i = 1, 3, 4, 5, 6$, and 7 , respectively. The rest of the entries are computed in an analogous manner. \square



(a) A directed graph

k	$dist^k[1..7]$						
	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

(b) $dist^k$

Figure 5.10 Shortest paths with negative edge lengths

- The algorithm to find shortest path is called Bellman and Ford algorithm. This is given below:

```
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  $\langle i, u \rangle$  in the graph do
11                 if dist[u] > dist[i] + cost[i, u] then
12                     dist[u] := dist[i] + cost[i, u];
13 }
```

Algorithm 5.4 Bellman and Ford algorithm to compute shortest paths

- For loop take $O(n^2)$ time if the adjacency matrices are used and $O(e)$ time if adjacency lists are used.
- Here e is the number of edges in the graph.
- The overall complexity is $O(n^3)$ if adjacency matrices are used and $O(ne)$ is the time if adjacency lists are used.

Thank You