# Advanced Data Structures and Algorithms

# Dynamic Programming

**Dr G.Kalyani**

Department of Information Technology

Velagapudi Ramakrishna Siddhartha Engineering College

# Topics

- General method
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# Dynamic programming

- 1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What's that equal to?“

# Dynamic programming

- 1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What's that equal to?"
- Counting “Fourteen!"
- Then 1+1+1+1+1+1+1+1+1+1+1+1+1+1 +1 =?
- "What about that?"

# Dynamic programming

- 1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What's that equal to?"
- Counting “Fourteen!"
- Then 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What about that?“
- It is “Fifteen”
- How'd you know it was fifteen so fast?

# Dynamic programming

- 1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What's that equal to?"
- Counting "Fourteen!"
- Then 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1 =?
- "What about that?"
- It is "Fifteen"
- How'd you know it was fifteen so fast?
- So you didn't need to recount because you remembered the previous result as fourteen!
- Dynamic Programming is just a fancy way to say remembering stuff to save time later!"

# Dynamic programming

- is repeating the things for which you already have the answer, a good thing ?
- No
- That's what Dynamic Programming is about.
- **Divide the problem as sub problems and**
- ***Always* remember answers to the sub-problems you've already solved.**
- **Use answers of the sub problems in solving the main problem**

# Dynamic programming

- **Dynamic Programming** is a general algorithm design technique for solving problems defined by or formulated as **recurrences with overlapping sub instances.**

- Invented by American mathematician **Richard Bellman to solve optimization problems** .

- **Main idea:**
  - set up a recurrence relating a solution to a larger instance with solutions of some smaller instances
  - solve smaller instances once
  - record solutions in a table
  - extract solution to the initial instance from that table

# Characteristics of Dynamic Programming

- DP is used to solve problems with the following characteristics:

- Simple sub problems
  - We should be able to break the original problem to **smaller sub problems that have the same structure**

- Optimal substructure of the problems
  - The **optimal solution to the problem contains optimal solutions to its sub problems.**

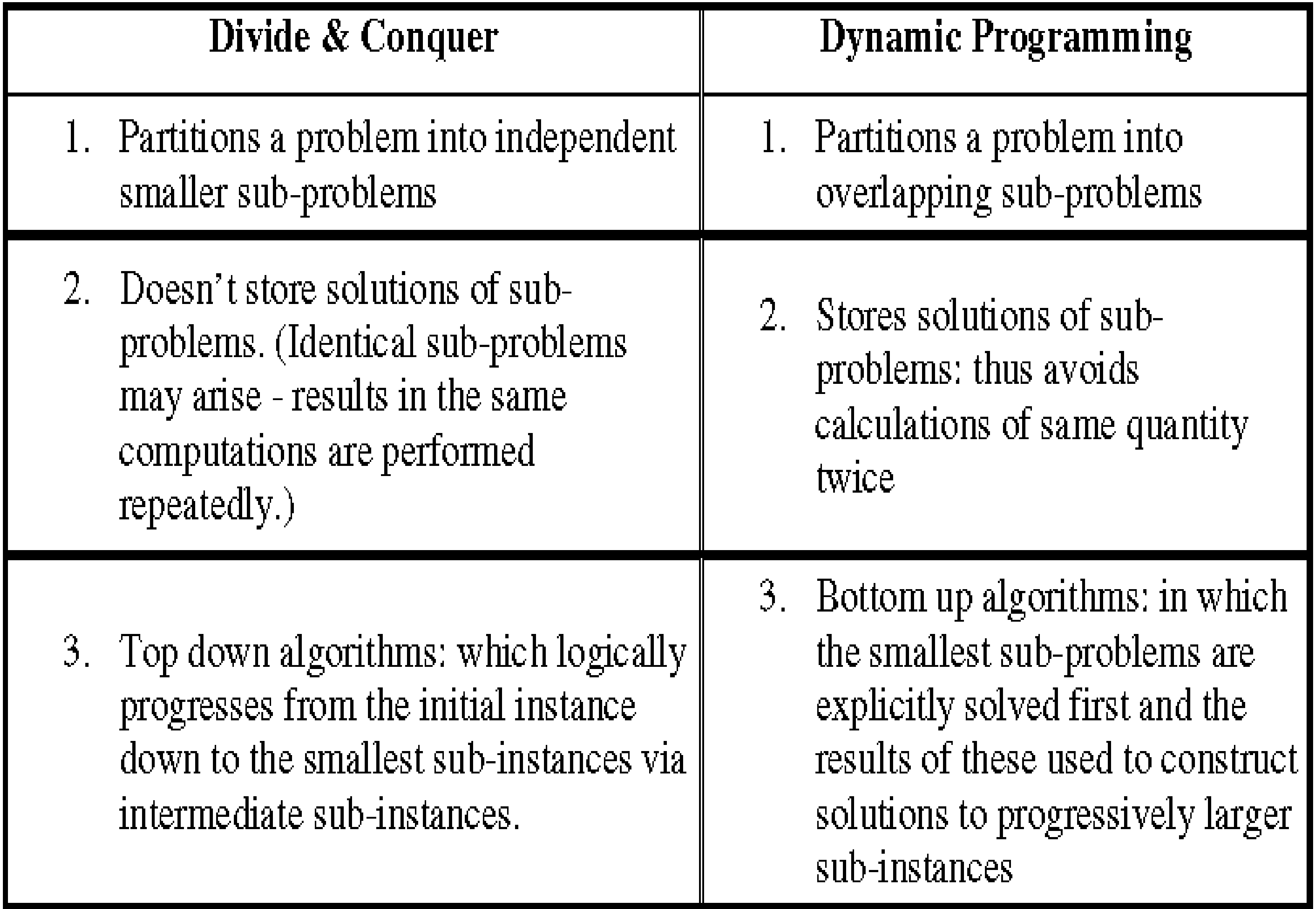
- Overlapping sub-problems
  - there exist some places where we solve the **same sub problem more** than **once.**

# Dynamic Programming: Top Down Vs Bottom Up

- **Bottom Up:**
  - Bottom up approach starts with small problems and go on to large problem.

- **Top Down:**
  - Top down approach will try to solve large first, if any small is required it will try to solve that and use it.

# Dynamic Programming Vs. Divide & Conquer



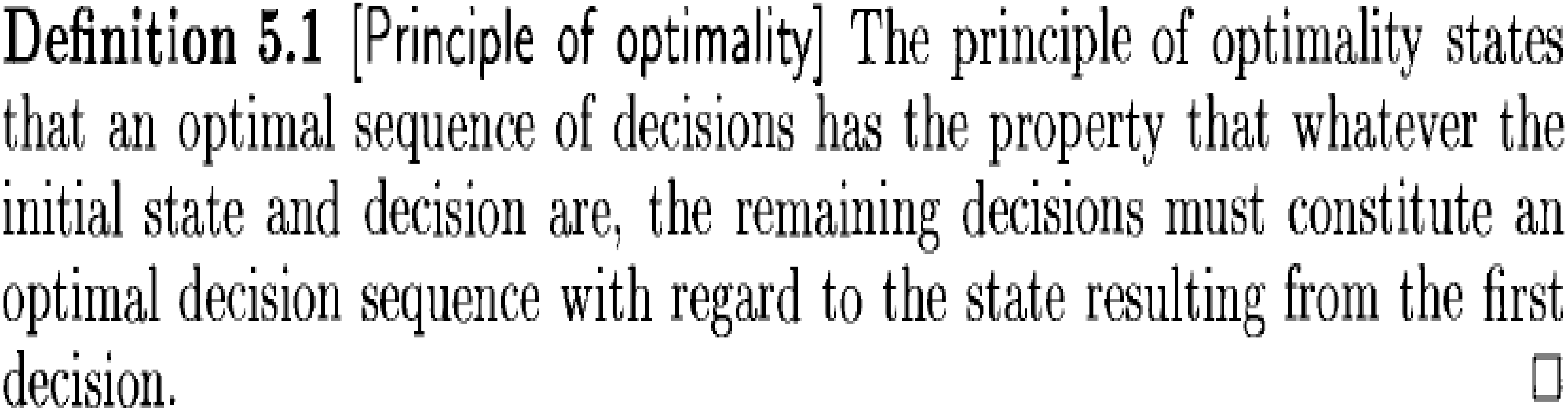
| Divide & Conquer | Dynamic Programming |
|---|---|
| 1. Partitions a problem into independent smaller sub-problems | 1. Partitions a problem into overlapping sub-problems |
| 2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise - results in the same computations are performed repeatedly.) | 2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice |
| 3. Top down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances. | 3. Bottom up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances |

# Dynamic Programming vs. Greedy Method

| Dynamic Programming | Greedy Method |
|---|---|
| 1. Dynamic Programming is used to solve optimization and combinatorial problems. | 1. Greedy Method is used solve optimization problems only. |
| 2. In Dynamic Programming, we choose at each step, but the choice may depend on the solution to sub-problems. | 2. In a greedy Algorithm, we make whatever choice seems best at the moment and then solve the sub-problems arising after the choice is made. |
| 3. It is guaranteed that Dynamic Programming will generate an optimal solution using Principle of Optimality. | 3. In Greedy Method, there is no such guarantee of getting Optimal Solution. |
| 4. Dynamic programming computes its solution bottom up or top down by synthesizing them from smaller optimal sub solutions. | 4. The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices. |
| 5. Example: 0/1 Knapsack | 5. Example: Fractional Knapsack |

# Principle of Optimality

- The dynamic programming works on a principle of optimality.



**Definition 5.1** [Principle of optimality] The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision. □

# Topics

- General method
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# The Travelling Salesperson Problem

- A **traveler** needs to **visit all the cities in 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**?

- Travelling salesman problem is the most **notorious computational problem**. We can use **brute-force approach** to evaluate **every possible tour** and **select the best one**. For **n number of vertices** in a graph, there are **(n - 1)! number of possibilities.**

- Instead of brute-force using **dynamic programming approach**, **the solution can be obtained in lesser time.**

# The Travelling Salesperson Problem

**Problem Definition:**

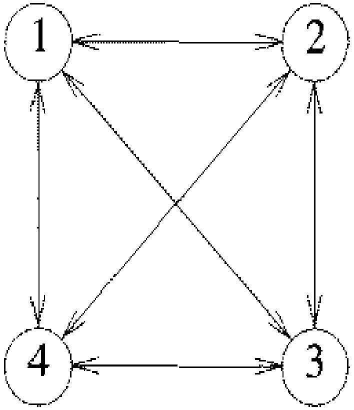
- Let **G (V, E)** be a directed graph with **edge cost $c_{i,j}$** is defined such **that $c_{i,j}$ >0 for all i and j and $c_{i,j}$ =∝ , if <i, j> ∉ E.**
- Let **V = n** and assume n>1.
- **The traveling salesman problem is to find a tour of minimum cost.**
- **A tour of Graph G is a directed cycle that include every vertex in V with starting and ending vertex as same.**
- **The cost of the tour is the sum of cost of the edges on the tour.**

# DP Solution to Travelling Salesperson Problem

The function **g(i, S)** is the **length of an optimal tour**.

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

# Example-TSP

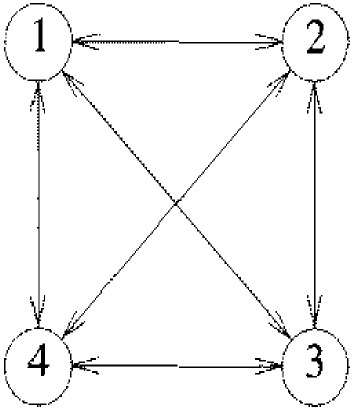


$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

**Find the shortest path tour for the travelling salesperson which starts at 1 and ends at 1**

**1 - __ - __ - __ - 1**

# Example-TSP



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

$| s | = 0.$

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

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

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

|S | = 1

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

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

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

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

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

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

$|S| = 2$

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

$\min\{ 9+20,\ 10+15\}$

$\min\{29, 25\} = 25$

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

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

$\min\{31,25\} = 25$

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

$\min\{8+15, 9+18\}$

$\min\{23, 27\} = 23$

$|S| = 3$

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

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

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

optimal cost is 35

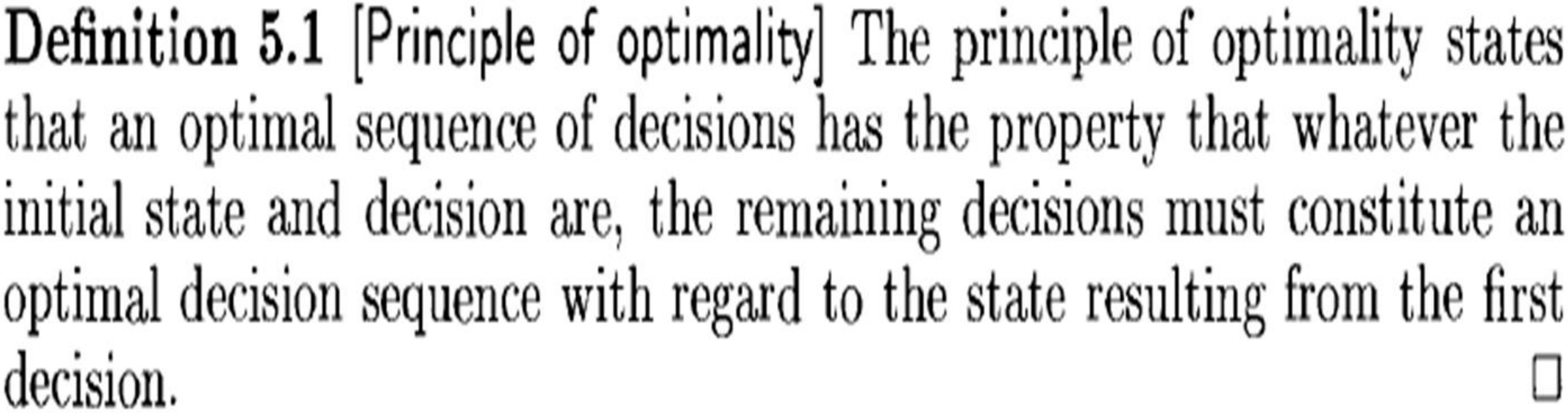
the shortest path is,

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

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

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

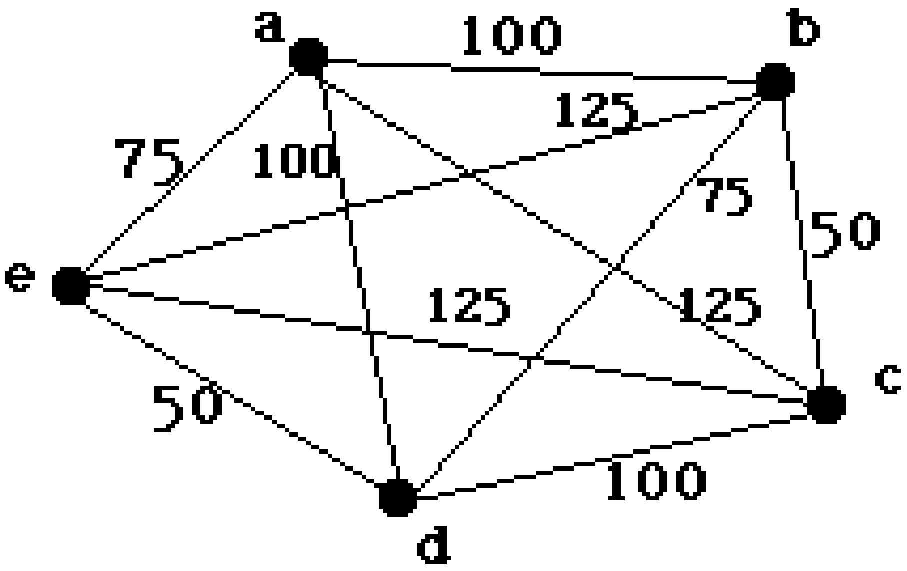
so the optimal tour is 1 → 2 → 4 →3 → 1



**Definition 5.1** [Principle of optimality] The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision. □

# Example 2

An Instance of the
Traveling Salesman Problem



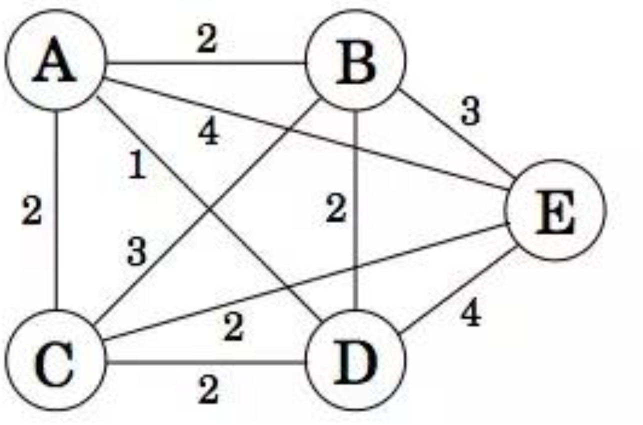
# Time Complexity Analysis

- An **algorithm** that proceeds to find an optimal tour will require **$\theta(n^2 2^n)$ time** as the computation of g(i,S) with |S|= k requires k= 1 comparisons when solving .

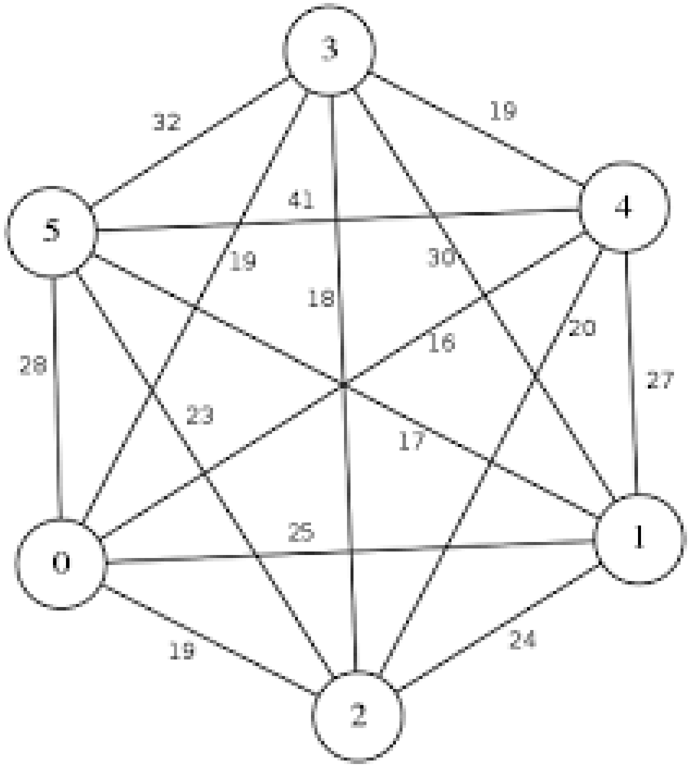
- This is better than enumerating all **n! different tours to find the best one**. The most serious drawback of this **dynamic programming solution is the space needed, 0($n2^n$).**

- This is too large even for modest values of n.

# Problems for practicing



Find the tour for TSP by considering “A” as the starting point



Find the tour for TSP by considering “3” as the starting point

# Topics

- General method
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# All Pairs Shortest Path Problem

- Let G = (V, E) be a directed graph with n vertices.

- Let cost be a adjacency matrix for G such that cost(i,i)=0,1< i < n.

- The cost(i,j) is the length (or cost) of edge(i,j)
  - cost(i, j)=x if $(i,j) \in E(G)$ and
  - cost(i,j)= $\infty$ if $i \neq j$ and $(i,j) \notin E(G)$.

- The all-pairs shortest-path problem is to determine a matrix A such that A(i,j) is the length of a shortest path from i to j.

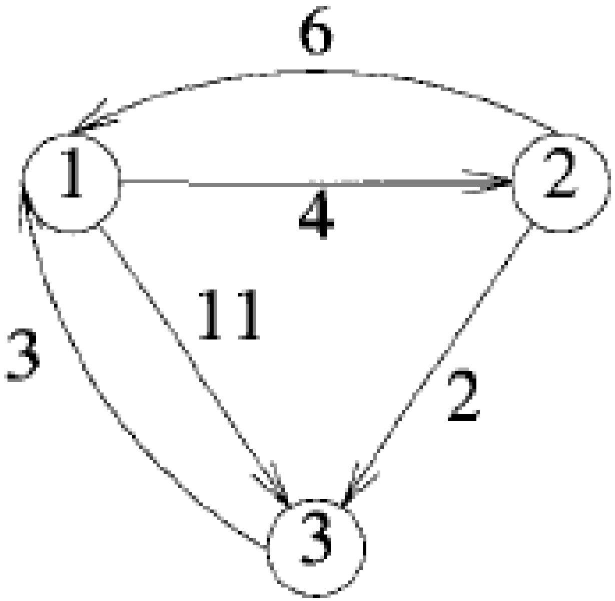
# Recurrence relating for APSP problem

- Using $A^k(i,j)$ to represent the length of a shortest path from i to j going through vertex less than k as intermediates.

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

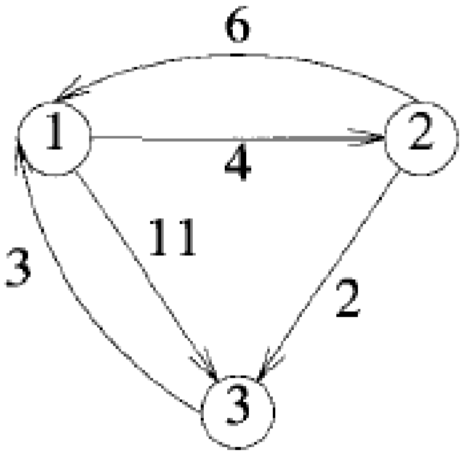
$$A^0(i,j) = cost(i,j),\ 1 \leq i \leq n,\ 1 \leq j \leq n.$$

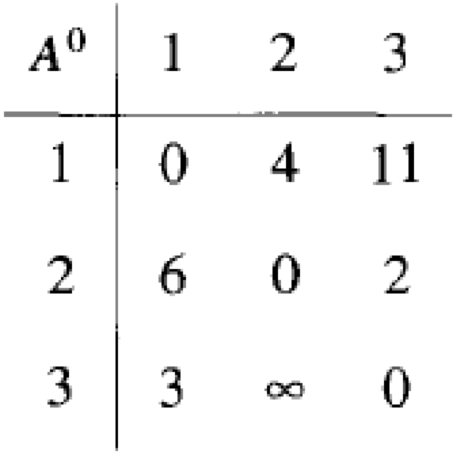
# Example



| $A^0$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | $\infty$ | 0 |

# Example





| $A^0$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | ∞ | 0 |

$A^1(1,2)= \min\{A^0(1,2), A^0(1,1)+A^0(1,2)\}=\min(4,0+4)=4$

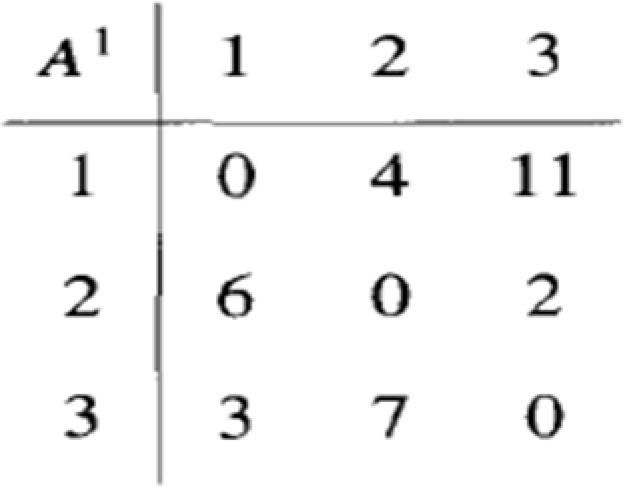
$A^1(1,3)= \min\{A^0(1,3), A^0(1,1)+A^0(1,3)\}=\min(11,0+11)=11$

$A^1(2,1)= \min\{A^0(2,1), A^0(2,1)+A^0(1,1)\}=\min(6,6+0)=6$

$A^1(2,3)= \min\{A^0(2,3), A^0(2,1)+A^0(1,3)\}=\min(2,6+11)=2$

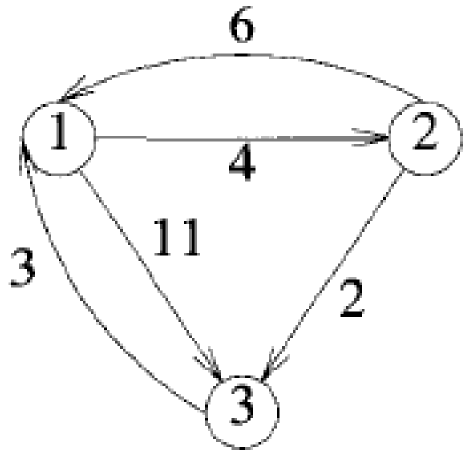
$A^1(3,1)= \min\{A^0(3,1), A^0(3,1)+A^0(1,1)\}=\min(3,3+0)=3$

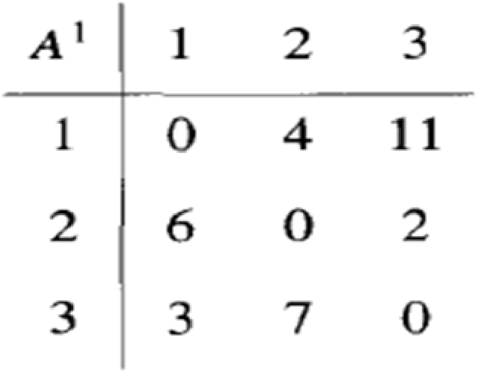
$A^1(3,2)= \min\{A^0(3,2), A^0(3,1)+A^0(1,2)\}=\min(\infty,3+4)=7$



| $A^1$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

# Example





| $A^1$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

$A^2(1,2)$= min{$A^1(1,2)$, $A^1(1,2)+A^1(2,2)$}=min(4,4+0)=4

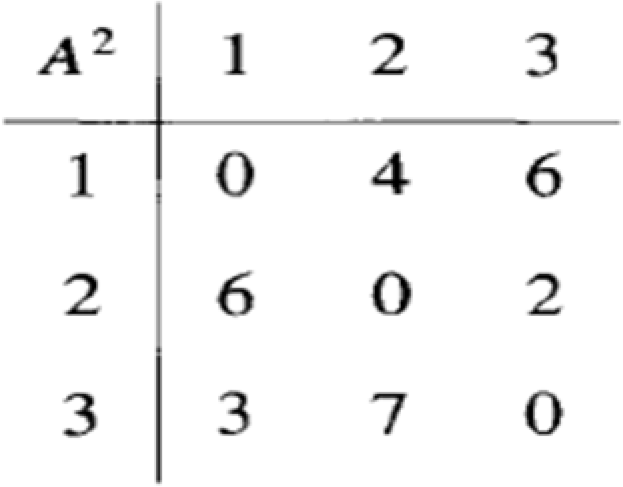
$A^2(1,3)$= min{$A^1(1,3)$, $A^1(1,2)+A^1(2,3)$}=min(11,4+2)=6

$A^2(2,1)$= min{$A^1(2,1)$, $A^1(2,2)+A^1(2,1)$}=min(6,0+6)=6

$A^2(2,3)$= min{$A^1(2,3)$, $A^1(2,2)+A^1(2,3)$}=min(2,0+2)=2

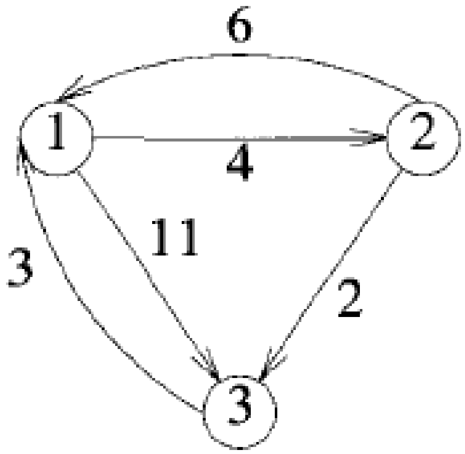
$A^2(3,1)$= min{$A^1(3,1)$, $A^1(3,2)+A^1(2,1)$}=min(3,7+6)=3

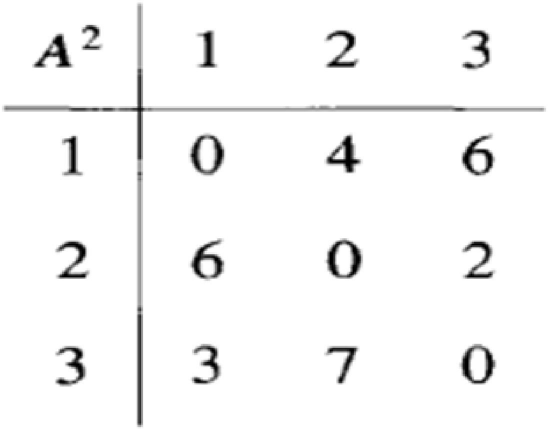
$A^2(3,2)$= min{$A^1(3,2)$, $A^1(3,2)+A^1(2,2)$}=min(7,7+0)=7



| $A^2$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

# Example





| $A^2$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

$A^3(1,2)= \min\{A^2(1,2), A^2(1,3)+A^2(3,2)\}=\min(4,6+7)=4$

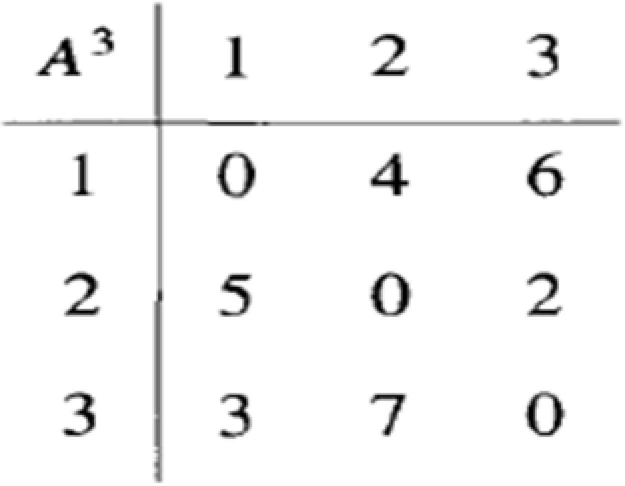
$A^3(1,3)= \min\{A^2(1,3), A^2(1,3)+A^2(3,3)\}=\min(6,6+0)=6$

$A^3(2,1)= \min\{A^2(2,1), A^2(2,3)+A^2(3,1)\}=\min(6,2+3)=5$

$A^3(2,3)= \min\{A^2(2,3), A^2(2,3)+A^2(3,3)\}=\min(2,2+0)=2$

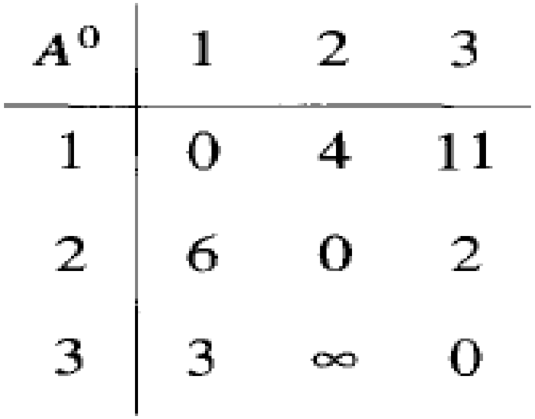
$A^3(3,1)= \min\{A^2(3,1), A^2(3,3)+A^2(3,1)\}=\min(3,0+3)=3$

$A^3(3,2)= \min\{A^2(3,2), A^2(3,3)+A^2(3,2)\}=\min(7,0+7)=7$



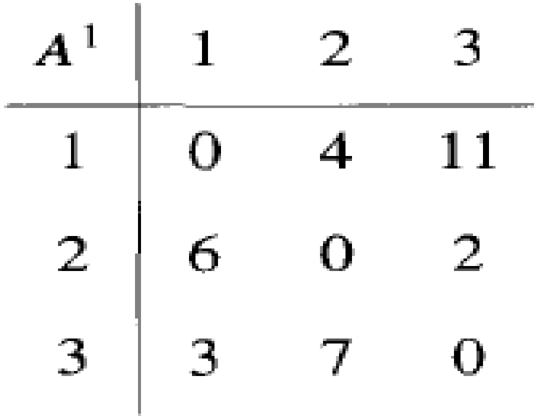
| $A^3$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 5 | 0 | 2 |
| 3 | 3 | 7 | 0 |

# Example



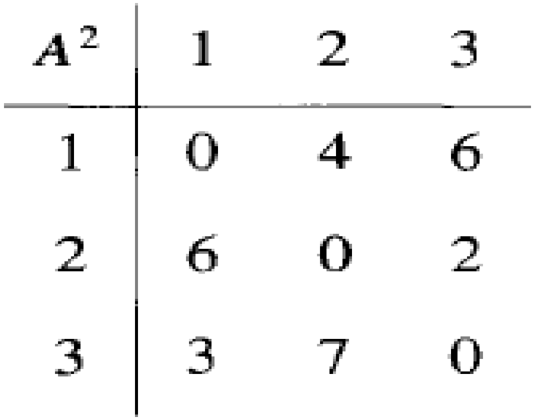
| $A^0$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | $\infty$ | 0 |

(b) $A^0$



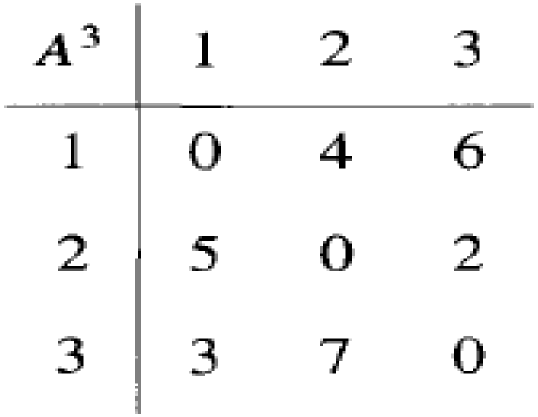
| $A^1$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

(c) $A^1$



| $A^2$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 6 | 0 | 2 |
| 3 | 3 | 7 | 0 |

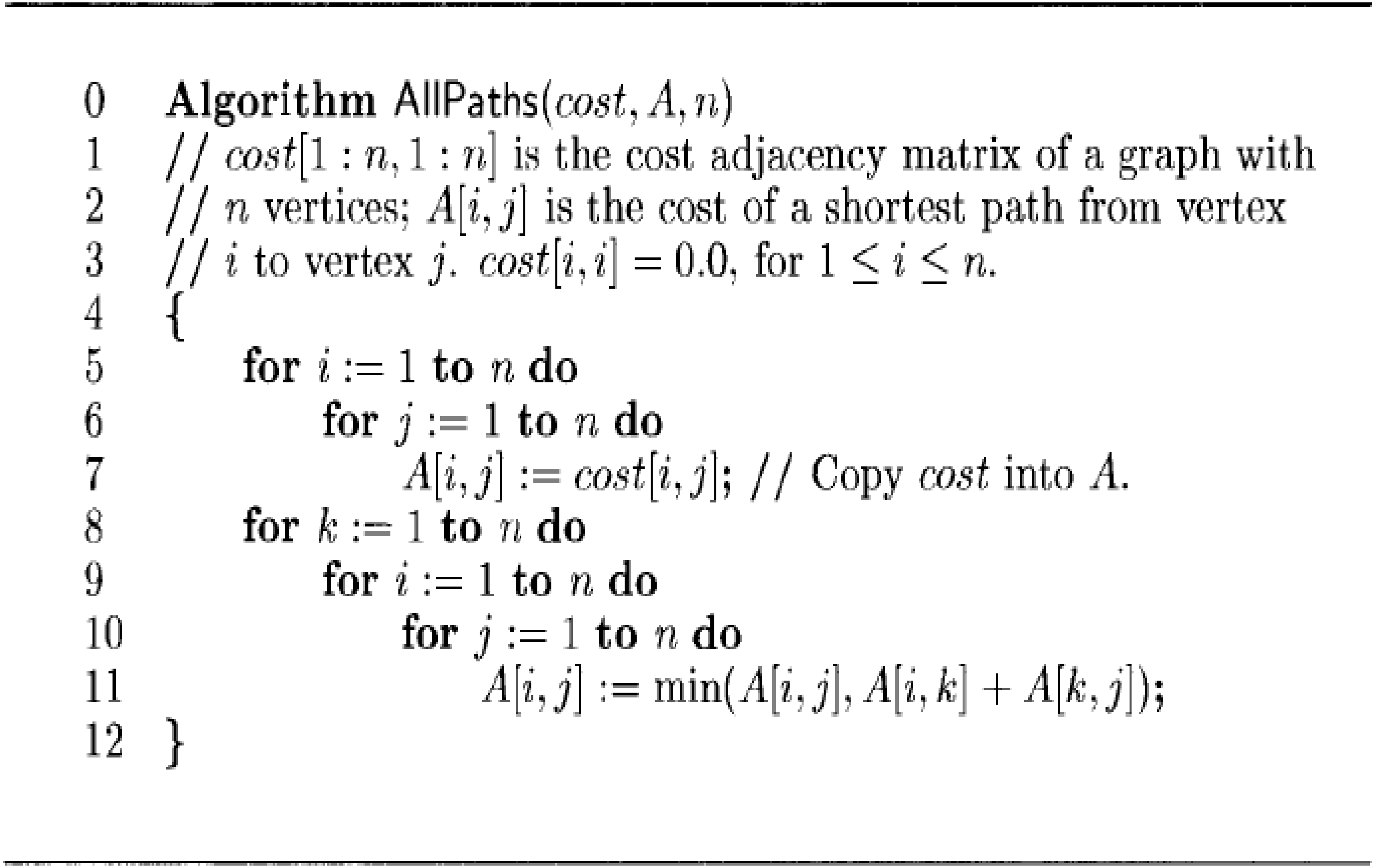
(d) $A^2$



| $A^3$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 6 |
| 2 | 5 | 0 | 2 |
| 3 | 3 | 7 | 0 |

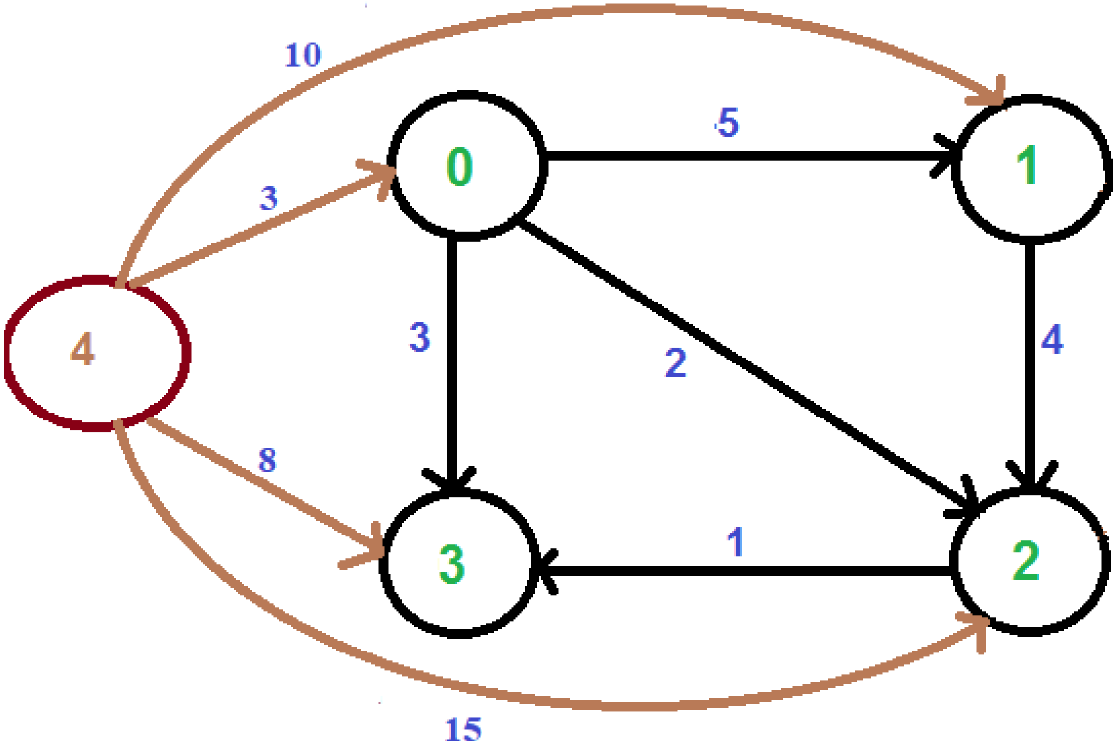
(e) $A^3$

# Algorithm for
# All Pairs Shortest Path Problem



```
Algorithm AllPaths($cost, A, n$)
// $cost[1:n, 1:n]$ is the cost adjacency matrix of a graph with
// $n$ vertices; $A[i, j]$ is the cost of a shortest path from vertex
// $i$ to vertex $j$. $cost[i, i] = 0.0$, for $1 \le i \le n$.
{
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $A[i, j] := cost[i, j]$; // Copy $cost$ into $A$.
    for $k := 1$ to $n$ do
        for $i := 1$ to $n$ do
            for $j := 1$ to $n$ do
                $A[i, j] := \min(A[i, j], A[i, k] + A[k, j])$;
}
```

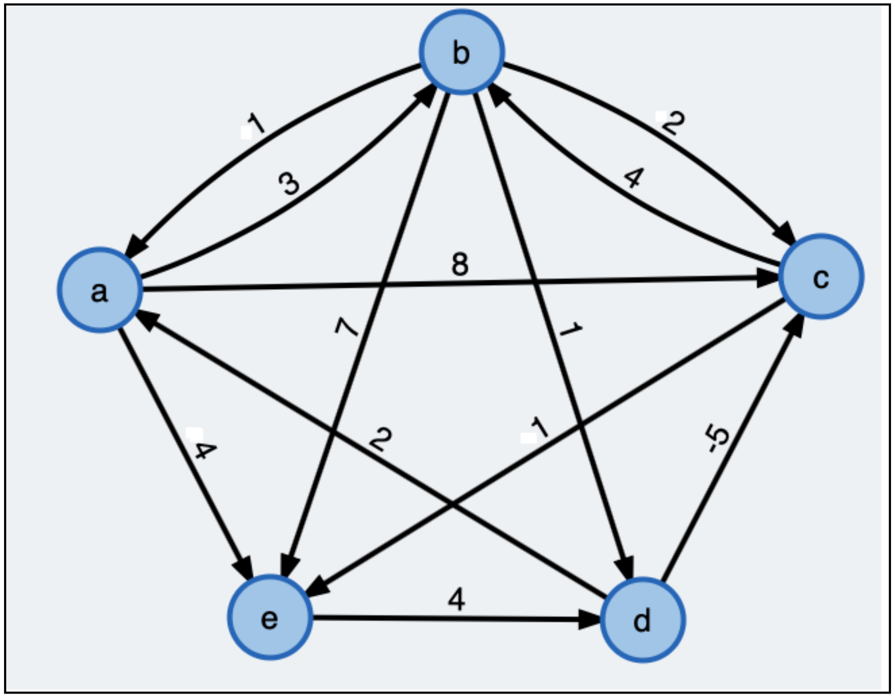
# Example 2

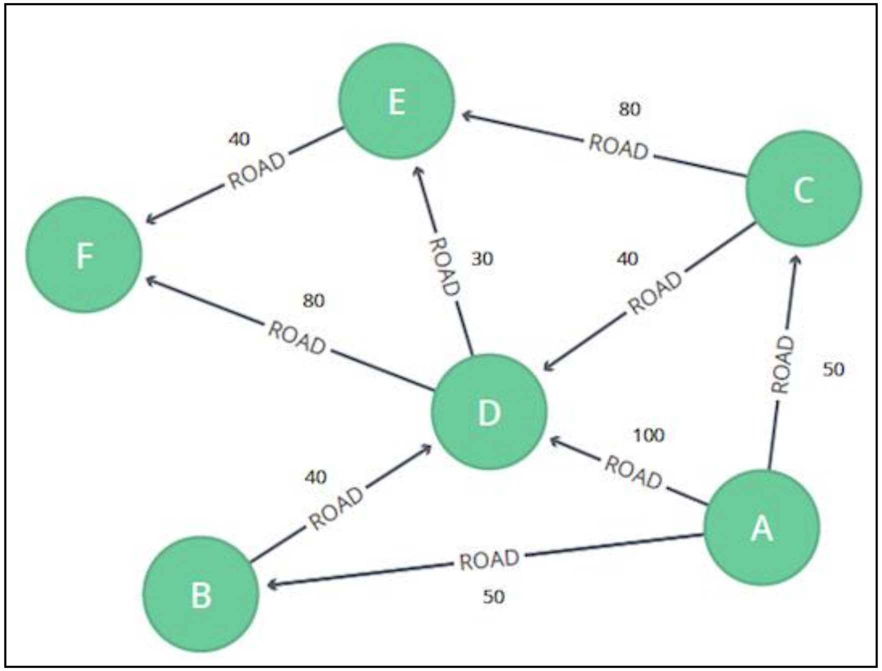


# Time Complexity

- All pair shortest path algorithm also known as Floyd-Warshall Algorithm.
- The time needed by All Paths is especially easy to determine because the looping is independent of the data in the matrix A.
- Line 11 is iterated $n^3$ times, and so the time for AllPaths is $O(n^3)$.

# Problems for Practice





# Topics

- General method
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# 0/1 or 0-1 Knapsack Problem

- Earlier we have discussed Fractional Knapsack problem using Greedy approach.

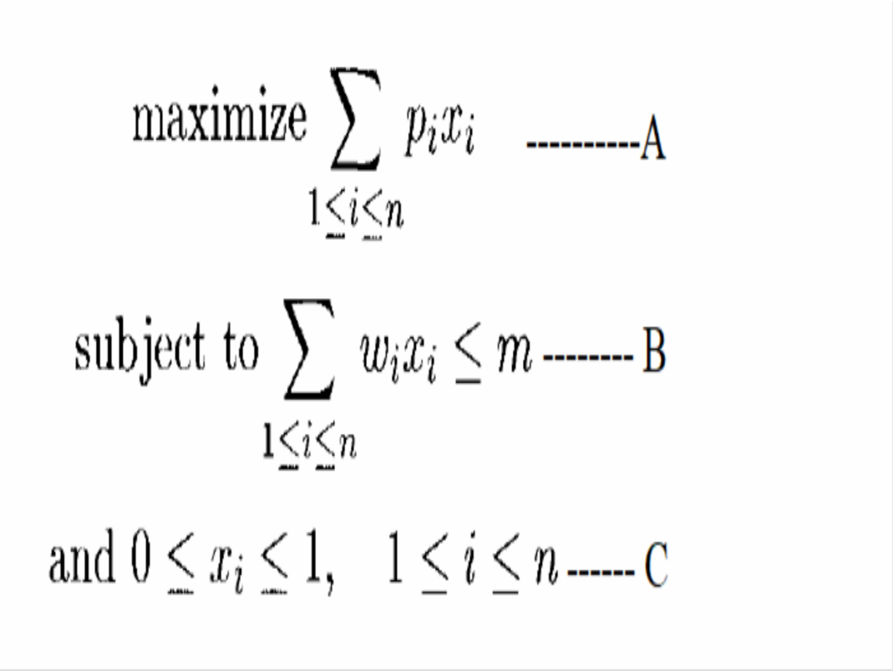
- We have shown that Greedy approach gives an optimal solution for Fractional Knapsack problem.

- In 0/1 or 0-1 Knapsack, items cannot be broken which means the we should take the item as a whole or should leave it.

- Hence, in case of 0-1 Knapsack, the value of $x_i$ can be either ***0*** or ***1***, where other constraints remain the same.

# Fractional Vs 0/1 Knapsack Problem

## Fractional Knapsack Problem

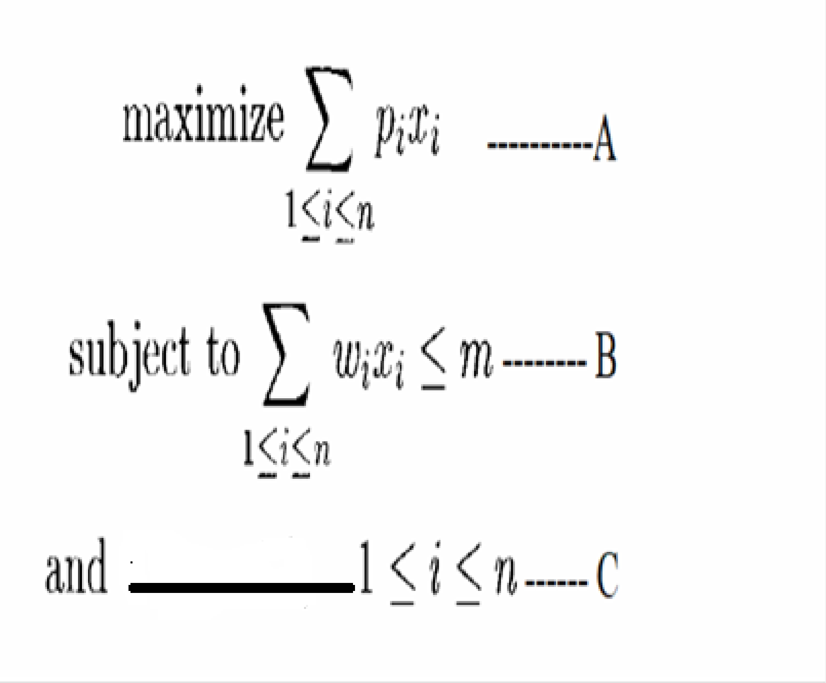


$$\text{maximize} \sum_{1 \le i \le n} p_i x_i \quad \text{----------A}$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le m \text{ --------B}$$

$$\text{and } 0 \le x_i \le 1, \quad 1 \le i \le n \text{ ------C}$$

## 0/1 Knapsack Problem



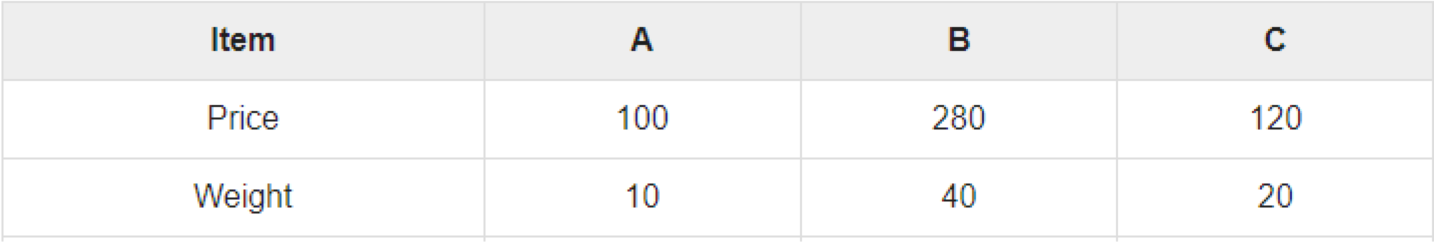
$$\text{maximize} \sum_{1 \le i \le n} p_i x_i \quad \text{----------A}$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le m \text{ --------B}$$

$$\text{and } \underline{\hspace{3cm}}\, 1 \le i \le n \text{ -----C}$$

# 0/1 Knapsack Problem

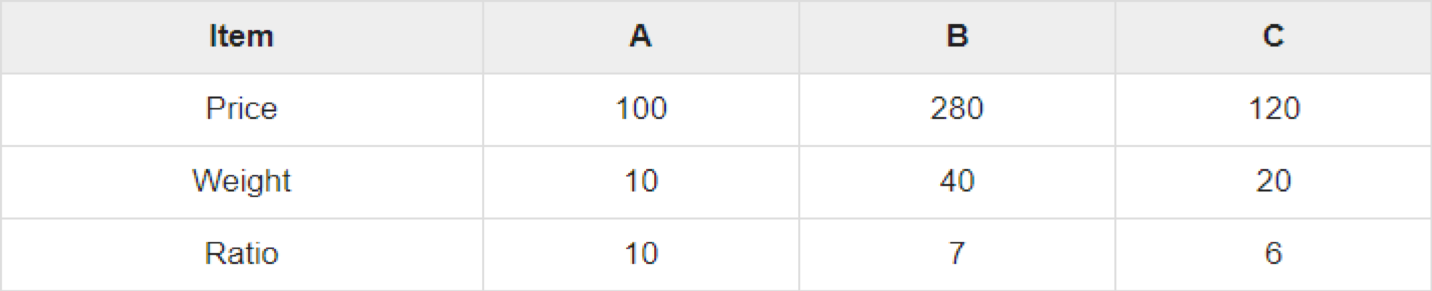
- 0/1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution.
- Ex: Consider the following data with knapsack capacity m=60



| Item | A | B | C |
|---|---|---|---|
| Price | 100 | 280 | 120 |
| Weight | 10 | 40 | 20 |

# 0/1 Knapsack Problem

- 0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution.
- Ex: Consider the following data with knapsack capacity m=60



| Item | A | B | C |
|---|---|---|---|
| Price | 100 | 280 | 120 |
| Weight | 10 | 40 | 20 |
| Ratio | 10 | 7 | 6 |

- XA=1 → m=60-10=50
- WB<=m → XB=1 → m=50-40 = 10
- WC<=m failed.so stop the process
- Hence the optimal solution as per greedy method is (1,1,0) with total profit of 100+280=380.
- But this is not an optimal solution for the problem. The optimal solution for the problem is _____________ with total profit of ___________.

# 0/1 knapsack problem with Dynamic Programming

- $\boldsymbol{s^0 = (0,0)}$

- **compute $\boldsymbol{S^1}$ to $\boldsymbol{S^n}$**

- $\boldsymbol{S^i = S^{i-1} \cup S_1^{i-1}}$

- **$\boldsymbol{S^i}$ is the set of all possibilities up to $\boldsymbol{i}$th object**

- $\boldsymbol{S_1{}^{i-1} = \left\{(P + p_i,, W + w_i),\ \forall (P, W) \in S^{i-1}\right\}}$

# Example for 0/1 Knapsack Problem

**Ex:** n=3; m=6; w[]={2,3,4}; p[]={1,2,5}

- Start with $S^0 = (0,0)$ [pair is (P,W)]
- Compute $S^1$ to $S^n$

$$S^i = S^{i-1} \cup S_1^{i-1}$$

- $S^1 = S^0 \cup S_1^0$
  - $S_1^0 = (1,2) \rightarrow S^1 = \{(0,0),(1,2)\}$
- $S^2 = S^1 \cup S_1^1$
  - $S_1^1 = \{(2,3),(3,5)\} \rightarrow S^2 = \{(0,0),(1,2),(2,3),(3,5)\}$

# Example for 0/1 Knapsack Problem

- $S^3 = S^2 \cup S_1^{\,2}$ $S^2 = \{(0,0),(1,2),(2,3),(3,5)\}$
- $S_1^{\,2} = \{(5,4),(6,6),(7,7),(8,9)\}$

$$\mathbf{S}^{i} = \mathbf{S}^{i-1} \cup \mathbf{S}_1^{\,i-1}$$

- $S^3 = \{(0,0),(1,2),(2,3),(3,5),(5,4),(6,6),(7,7),(8,9)\}$
- $S^3 = \{(0,0),(1,2),(2,3),$~~(3,5)~~$,(5,4),(6,6),(7,7),(8,9)\}$

# Example continuation

- Hence finally $S^3$ = ={(0,0),(1,2),(2,3),(5,4),(6,6)}
- Because of highest profit pair is (6,6), it means you will highest profit of 6 with a total weight in the knapsack 6.
- But what is the solution for this highest profit?
- The selected pair is (6,6) from $S^3$ .
- Since $S^3 = S^2 \cup S_1^2$ -> Check whether (6,6) is in $S^2$ or not.
- (6,6) $\notin S^2$ ➔ x3=1
- (6,6)- (5,4)= (1,2) Check whether in $S^1$ or not
- (1,2) $\in S^1$ ➔x2=0
- (1,2) Check whether in $S^0$ or not
- (1,2) $\notin S^0$ ➔ x1=1
- Hence the solution is (1,0,1) with a total profit of 6.

# Example 2

- **Ex:** n=6; m=165;
- w[]= P[]= {100,50,20,10,7,3};

profit of 6

eg:- $n=6$ $(P_1, P_2, P_3, P_4, P_5, P_6) = (w_1, w_2, w_3, w_4, w_5, w_6)$

$= (100, 50, 20, 10, 7, 3)$

$M = 165$

$S^0 = \{0\}$ $S_1^0 = 100$

$S^1 = \{0, 100\}$ $S_1^1 = \{50, 150\}$

$S^2 = \{0, 50, 100, 150\}$ $S_1^2 = \{20, 70, 120, 170\}$

$S^3 = \{0, 20, 50, 70, 100, 120, 150, 170\}$

$S_1^3 = \{10, 30, 60, 80, 110, 130, 160\}$

$S^4 = \{0, 10, 20, 30, 50, 60, 70, 80, 100, 110, 120, 130, 150, 160\}$

$S_1^4 = \{7, 17, 27, 37, 57, 67, 77, 87, 107, 117, 127, 137, 157, 167\}$

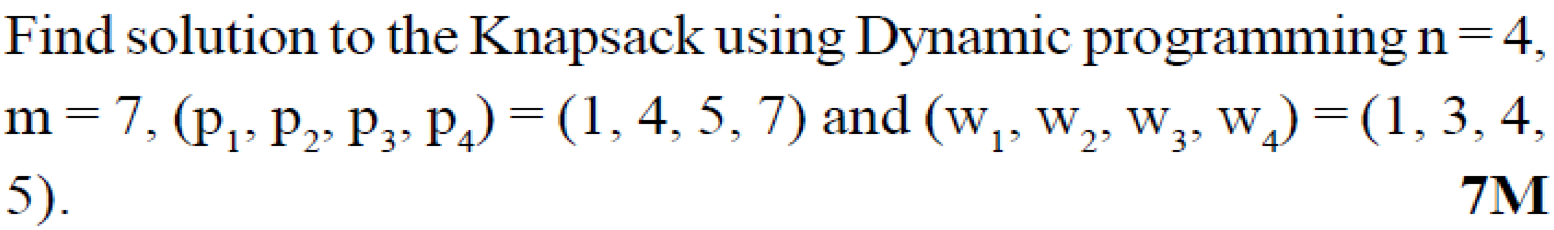
$S^5 = \{0, 7, 10, 17, 20, 27, 30, 37, 50, 57, 60, 67, 70, 77, 80, 87, 100, 107, 110, 117,$
$120, 127, 130, 137, 150, 157, 160, 167\}$

$S_1^5 = \{3, 10, 13, 20, 23, 30, 33, 40, 53, 60, 63, 70, 73, 80, 83, 90, 10$
$110, 113, 120, 123, 130, 133, 140, 153, 160, 163\}$

- The selected pair is (163,163) from $S^6$
- Check whether (163,163) is in $S^5$ or not.
- (163,163) $\notin S^5$ ➔ x6=1
- (163,163)-(3,3)= (160,160) Check whether in $S^4$ or not
- (160,160) $\in S^4$ ➔x5=0
- (160,160) Check whether in $S^3$ or not
- (160,160) $\notin S^3$ ➔ x4=1
- (160,160)-(10,10)= (150,150) Check whether in $S^2$ or not
- (150,150) $\in S^2$ ➔x3=0
- (150,150) Check whether in $S^1$ or not
- (150,150) $\notin S^1$ ➔ x2=1
- (150,150)-(50,50)= (100,100) Check whether in $S^0$ or not
- (100,100) $\notin S^0$ ➔ x1=1
- Hence the solution is (1,1,0,1,0,1) with a total profit of 163.

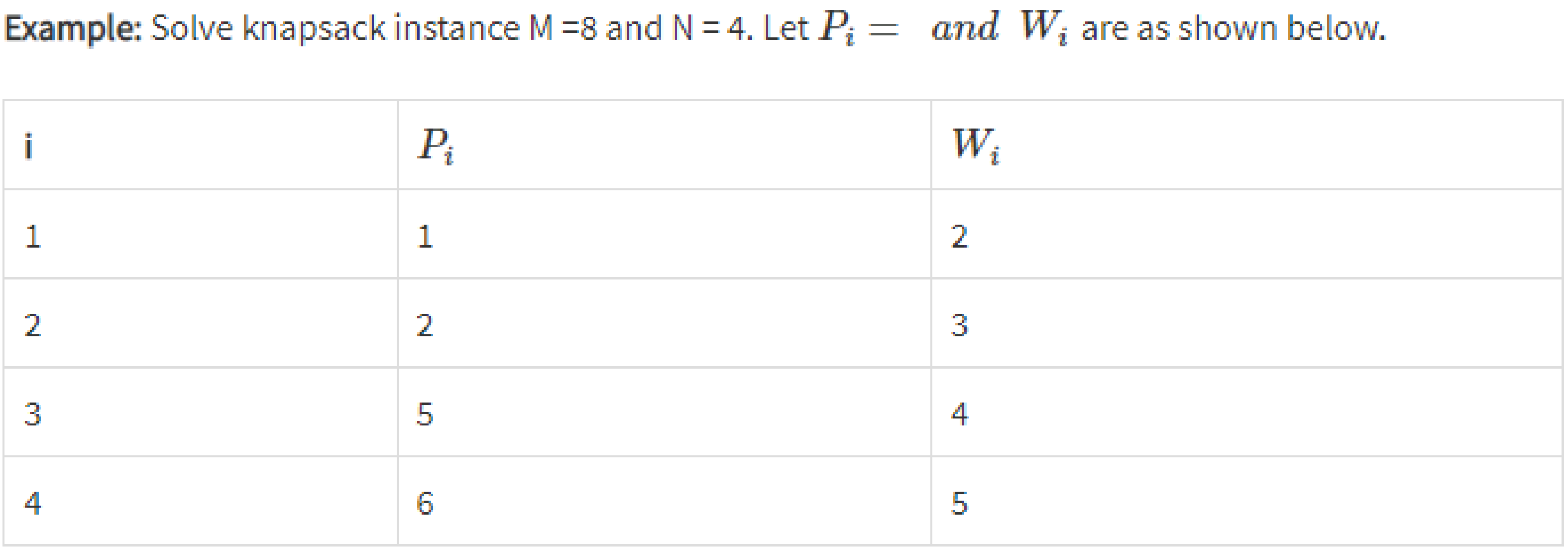
# Problems for Practice

## Problem-1:



Find solution to the Knapsack using Dynamic programming n = 4, m = 7, $(p_1, p_2, p_3, p_4) = (1, 4, 5, 7)$ and $(w_1, w_2, w_3, w_4) = (1, 3, 4, 5)$. **7M**

## Problem-2:



**Example:** Solve knapsack instance M =8 and N = 4. Let $P_i = \quad and \;\; W_i$ are as shown below.

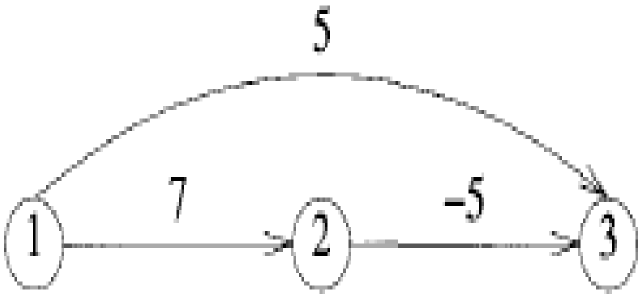
| i | $P_i$ | $W_i$ |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 5 | 4 |
| 4 | 6 | 5 |

# Topics

- General metho
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# Single-source shortest paths: general weights

- The single-source shortest path problem discussed in Greedy method.
- When some or all of the edges of the directed graph G have negative length, Shortest Path Algorithm does not necessarily give the correct results on such graphs



| V | Dist [v} |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 5 |

| V | Dist [v} |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 5 |

For 2=min(7, 5+∞)= 7

But the correct answer is
dist[2]=7 and
dist[3]=2

# Single-source shortest paths: general weights

- 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 consist of a finite number of edges.
- When there are no cycles of negative length, there is a **shortest path between any two vertices of an n-vertex graph that has at most n-1edges on it.**
- Let **$dist^{L}[u]$** be the length of the shortest path from source vertex v to vertex u under the constrain that the shortest path contains at most L edges. Then, **$dist^{1}[u]$ = cost[v,u],** $1< u < n$.
- Our goal is to **compute $dist^{n-1}[u]$ for all u** using the dynamic programming methodology.

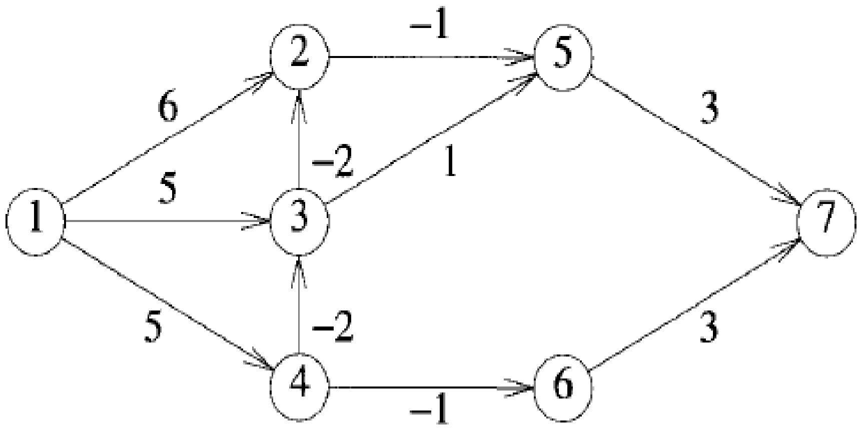
# Single-source shortest paths: general weights

- The recurrence equation to solve dist[u]

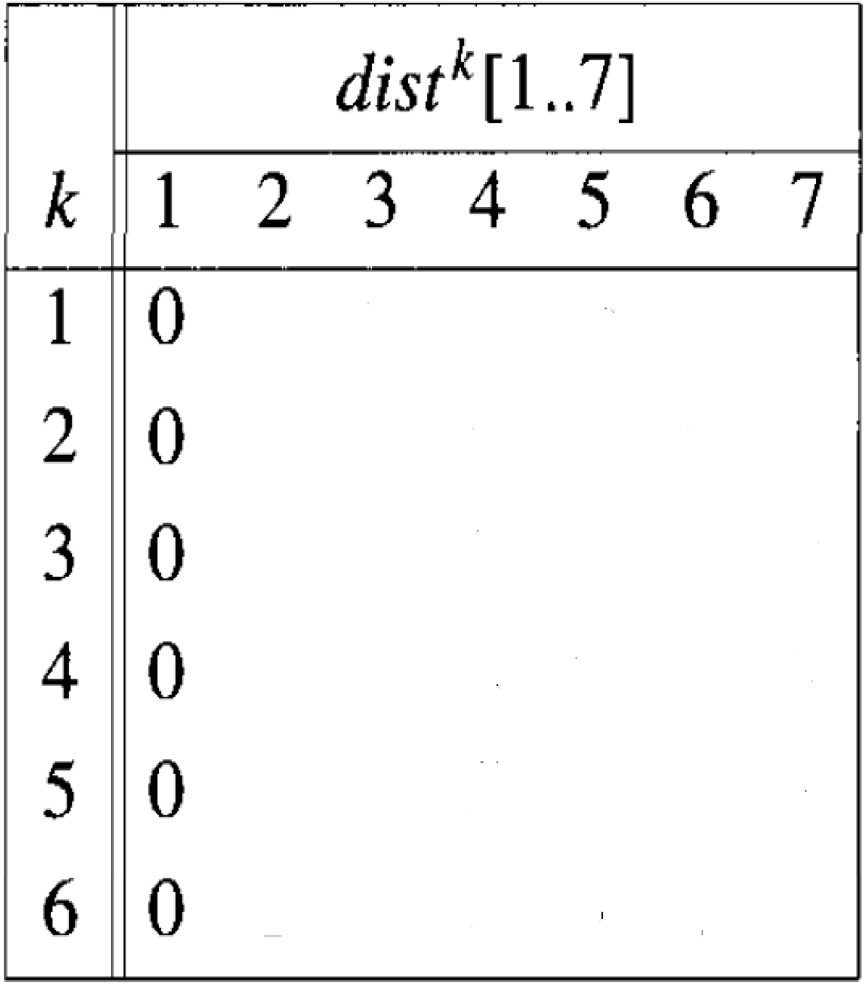
$$dist^{k}[u] = \min\{dist^{k-1}[u], \min_{i}\{dist^{k-1}[i] + cost[i,u]\}\}$$

- Calculate dist$^{1}$[u] to dist$^{n-1}$[u], $\forall\ 1 \leq U \leq n$

# Single-source shortest paths: general weights

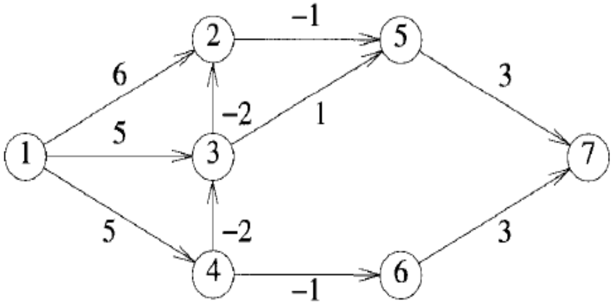


(a) A directed graph



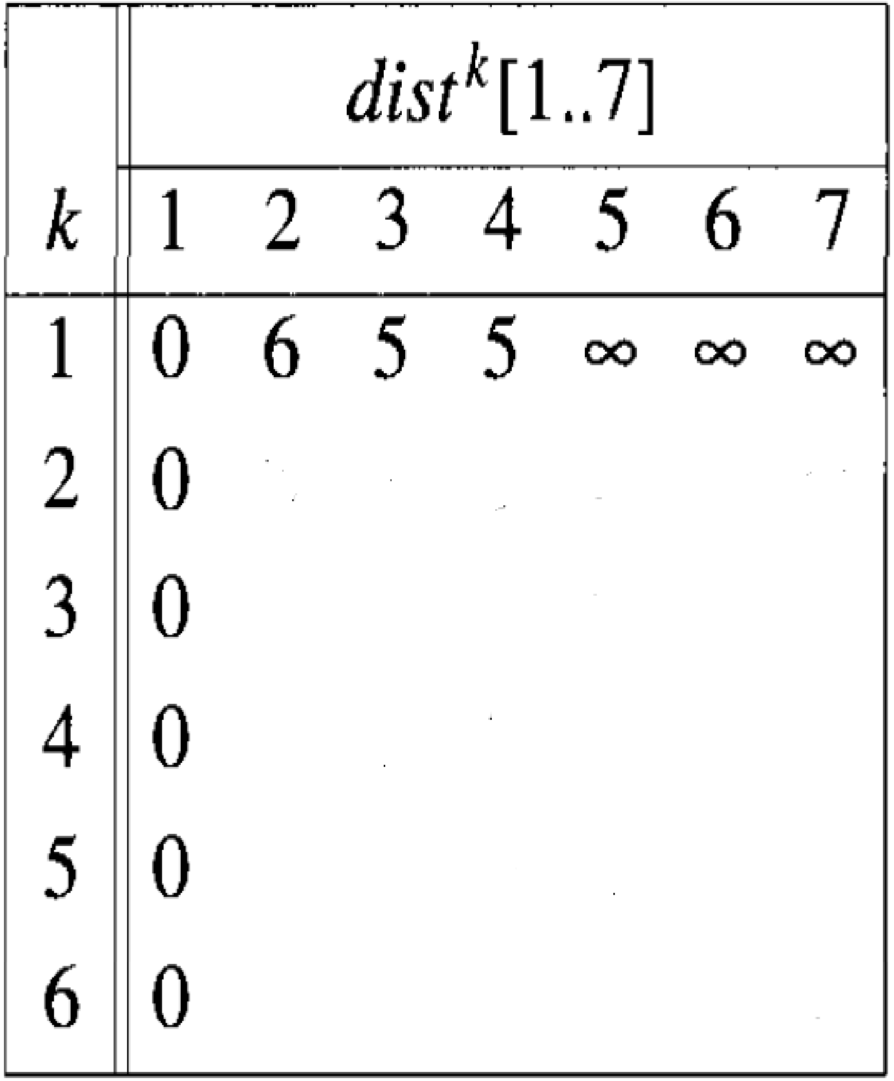
| $k$ | $dist^k[1..7]$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | | | | | | |
| 2 | 0 | | | | | | |
| 3 | 0 | | | | | | |
| 4 | 0 | | | | | | |
| 5 | 0 | | | | | | |
| 6 | 0 | | | | | | |

# Single-source shortest paths: general weights



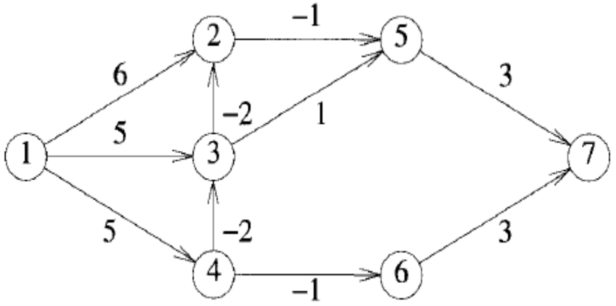
$$\mathrm{dist}^k[u] = \min\{\mathrm{dists}^{k-1}[u], \min_i\{\mathrm{dists}^{k-1}[i] + \mathrm{cost}[i,u]\}\}$$

$$\mathrm{dist}^1[v] = cost(1.v)$$



| | $dist^k[1..7]$ | | | | | | |
|---|---|---|---|---|---|---|---|
| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 0 | 6 | 5 | 5 | $\infty$ | $\infty$ | $\infty$ |
| 2 | 0 | | | | | | |
| 3 | 0 | | | | | | |
| 4 | 0 | | | | | | |
| 5 | 0 | | | | | | |
| 6 | 0 | | | | | | |

# Single-source shortest paths: general weights



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

$$\mathrm{dist}^2[2] = \min\left\{\mathrm{dists}^1[2], \min_i\left\{\begin{matrix} i = 1\text{: } \mathrm{dists}^1[1] + \mathrm{cost}[1,2] \\ \vdots \\ i = 7\text{: } \mathrm{dists}^1[7] + \mathrm{cost}[7,2] \end{matrix}\right\}\right\}$$

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

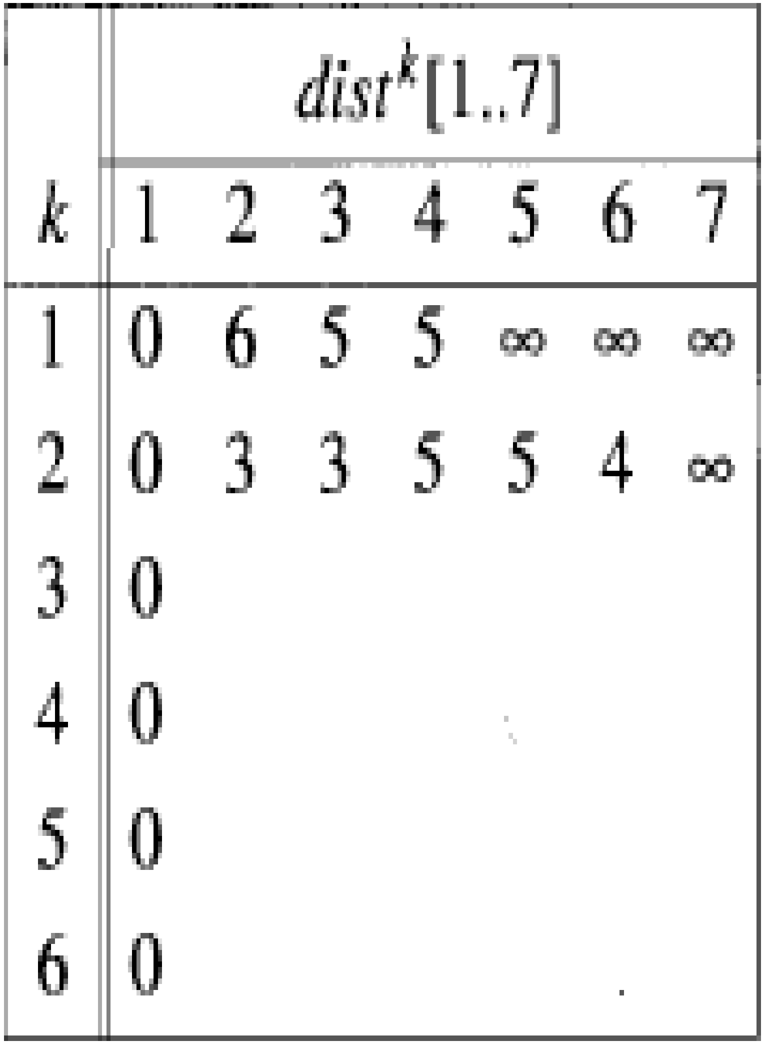
$$\begin{aligned}\mathrm{dist}^2[3] &= \min\{5, \min_i\{0 + 5, 6 + \infty, 5 - 2, \infty + \infty, \infty + \infty, \infty + \infty\}\} \\ &= \min\{5, \min_i\{5, 6, 3, \infty, \infty, \infty\}\} = 3\end{aligned}$$

$$\begin{aligned}\mathrm{dist}^2[4] &= \min\{5, \min_i\{0 + 5, 6 + \infty, 5 + \infty, \infty + \infty, \infty + \infty, \infty + \infty\}\} \\ &= \min\{5, \min_i\{5, 6, 5, \infty, \infty, \infty\}\} = 5\end{aligned}$$

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

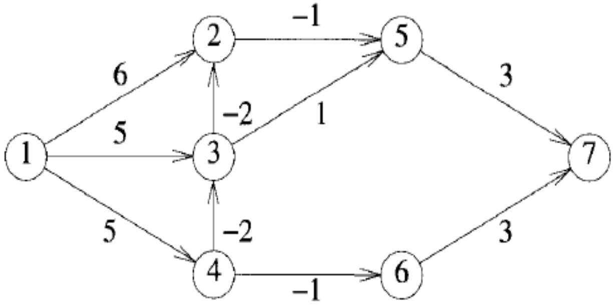
$$\begin{aligned}\mathrm{dist}^2[6] &= \min\{\infty, \min_i\{0 + \infty, 6 + \infty, 5 + \infty, 5 - 1, \infty + \infty, \infty + \infty\}\} \\ &= \min\{\infty, \min_i\{\infty, \infty, \infty, 4, \infty, \infty\}\} = 4\end{aligned}$$

$$\begin{aligned}\mathrm{dist}^2[7] &= \min\{\infty, \min_i\{0 + \infty, 6 + \infty, 5 + \infty, 5 + \infty, \infty + 3, \infty + 3\}\} \\ &= \min\{\infty, \min_i\{\infty, \infty, \infty, \infty, \infty, \infty\}\} = \infty\end{aligned}$$



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

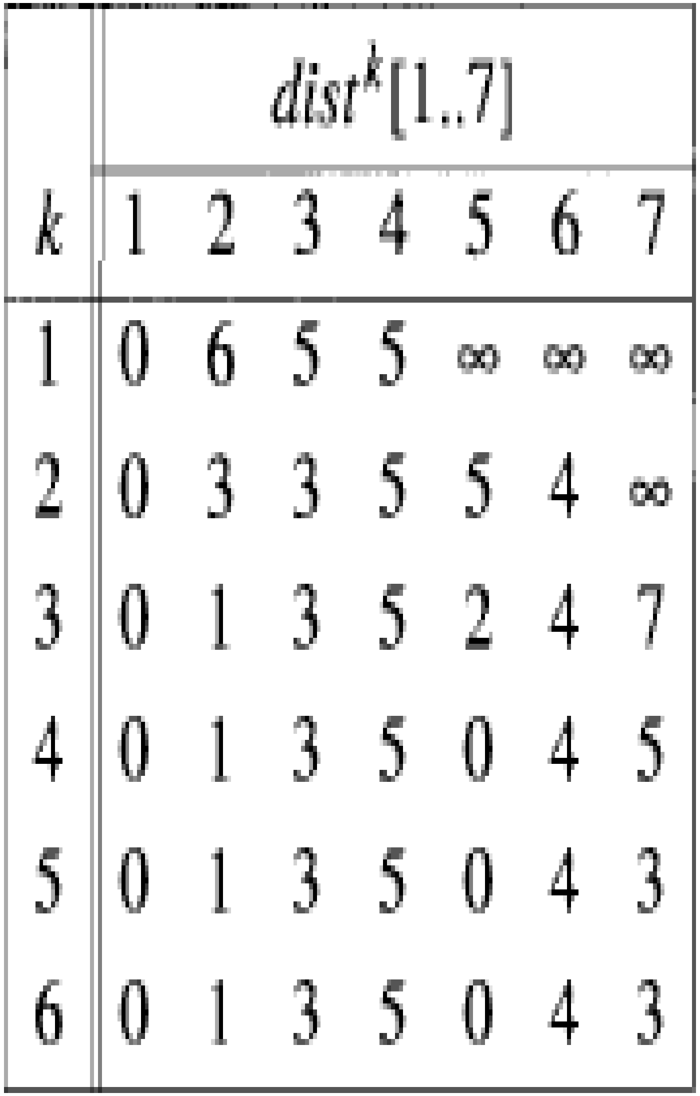
# Single-source shortest paths: general weights



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

**Repeat the process up to k=n-1**

$$\mathrm{dist}^6[2] = \min\left\{\mathrm{dists}^5[2], \min_i \begin{Bmatrix} i = 1\!: \ \mathrm{dists}^5[1] + \mathrm{cost}[1,2] \\ \vdots \\ i = 7\!: \mathrm{dists}^5[7] + \mathrm{cost}[7,2] \end{Bmatrix}\right\}$$



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

# Topics

- General method
- Travelling sales person problem
- All pairs shortest Path problem
- 0/1 knapsack problem
- Single-source shortest paths: general weights
- String Editing

# String Editing

- **Given two strings X= x1,x2,…,xn and Y=y1,y2,…ym, where** 1<=xi<=n and 1<=yj<=m are members of a set of finite set of symbols known as *alphabet*.
- The problem is to **transform X to Y using a sequence of edit operations on X**. The permissible edit operations are insert, delete and change.
- The **cost of transforming X to Y is the sum of cost of the individual edit operations** in the sequence of operations.
- The **problem of string editing is to identify the minimum cost sequence of edit operations that will transform X to Y**.
- Let **D(xi) be the cost of deleting a symbol xi** from X, **I(yj) be the cost of inserting a symbol yj** into X, and **C(xi,yj) be the cost of changing the symbol xi of X to yj of Y.**

# String Editing

$$cost(i,j) = \begin{cases} 0 & i = j = 0 \\ cost(i-1,0) + D(x_i) & j = 0,\ i > 0 \\ cost(0,j-1) + I(y_j) & i = 0,\ j > 0 \\ cost'(i,j) & i > 0,\ j > 0 \end{cases}$$

$$\text{where } cost'(i,j) = \min \{ \begin{array}{l} cost(i-1,j) + D(x_i), \\ cost(i-1,j-1) + C(x_i,y_j), \\ cost(i,j-1) + I(y_j)\ \} \end{array}$$

**Compute Cost(i,j) for all 0<=i<=n and 0<=j<=m.**

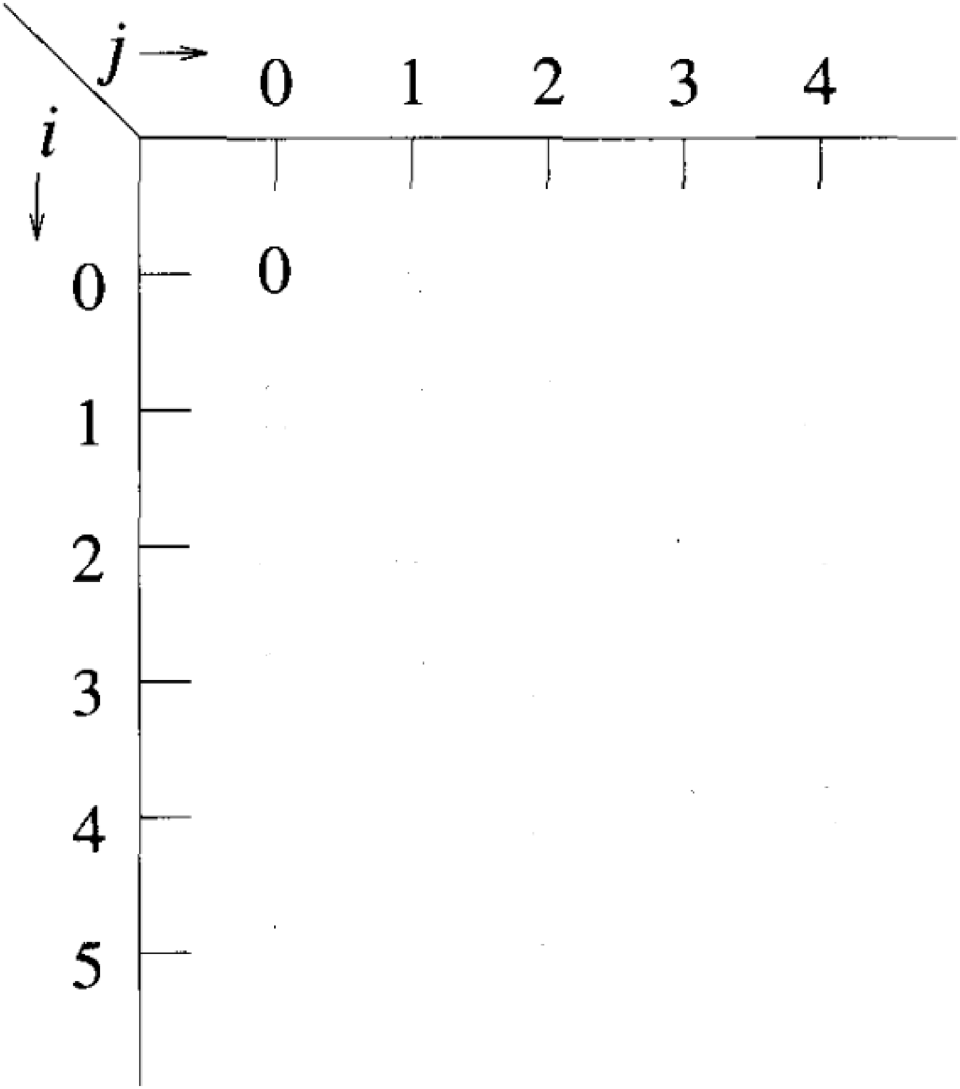
# Example for String Editing

- Consider the string editing problem as

X= a, a, b, a, b   and   Y= b, a, b, b.

with each insertion and deletion costs 1 unit and change costs 2 units. Find the minimum cost edit sequence of operations to convert X to Y.

# Example for String Editing



| j →<br>i ↓ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | | | | |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

Cost(0, j )→ requires insertion operation
Cost (i, 0)→ requires delete operation

Cost(0,1)=cost(0,0)+I(y1) = 0+1 = 1
Cost(0,2)=cost(0,1)+I(y2) = 1+1 = 2
Cost(0,3)=cost(0,2)+I(y3) = 2+1 = 3
Cost(0,4)=cost(0,3)+I(y4) = 3+1 = 4

Similarly,

Cost(1,0)=cost(0,0)+D(x1) = 0+1 = 1
Cost(2,0)=cost(1,0)+D(x2) = 1+1 = 2
Cost(3,0)=cost(2,0)+D(x3) = 2+1 = 3
Cost(4,0)=cost(3,0)+D(x4) = 3+1 = 4
Cost(5,0)=cost(4,0)+D(x5) = 4+1 = 5

# Example for String Editing

| i ↓ \ j → | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 |  |  |  |  |
| 2 | 2 |  |  |  |  |
| 3 | 3 |  |  |  |  |
| 4 | 4 |  |  |  |  |
| 5 | 5 |  |  |  |  |

$$cost(i,j) = \begin{cases} 0 & i = j = 0 \\ cost(i-1,0) + D(x_i) & j = 0,\ i > 0 \\ cost(0,j-1) + I(y_j) & i = 0,\ j > 0 \\ cost'(i,j) & i > 0,\ j > 0 \end{cases}$$

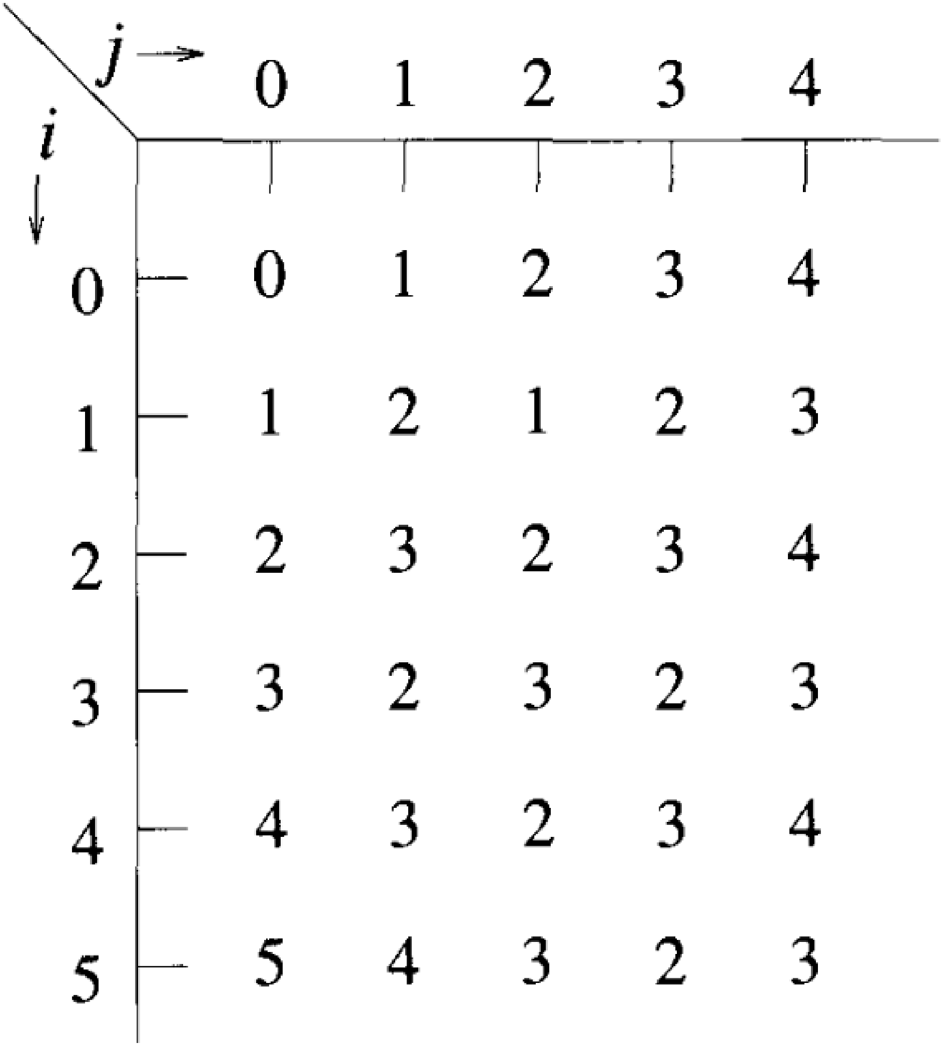
where $cost'(i,j) = \min\{\ cost(i-1,j) + D(x_i),\ cost(i-1,j-1) + C(x_i,y_j),\ cost(i,j-1) + I(y_j)\ \}$

$$\begin{aligned} cost(1,1) &= \min\{cost(0,1) + D(x_1), cost(0,0) + C(x_1,y_1), cost(1,0) + I(y_1)\} \\ &= \min\{2,2,2\} = 2 \end{aligned}$$

Next is computed $cost(1,2)$.

$$\begin{aligned} cost(1,2) &= \min\{cost(0,2) + D(x_1), cost(0,1) + C(x_1,y_2), cost(1,1) + I(y_2)\} \\ &= \min\{3,1,3\} = 1 \end{aligned}$$

# Example for String Editing



| i \ j | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 1 | 2 | 3 |
| 2 | 2 | 3 | 2 | 3 | 4 |
| 3 | 3 | 2 | 3 | 2 | 3 |
| 4 | 4 | 3 | 2 | 3 | 4 |
| 5 | 5 | 4 | 3 | 2 | 3 |

- Finally, cost(5,4) = 3 indicates the meaning that the minimum cost of 3 units is required to convert the given X=a,a,b,a,b into Y=b,a,b,b.

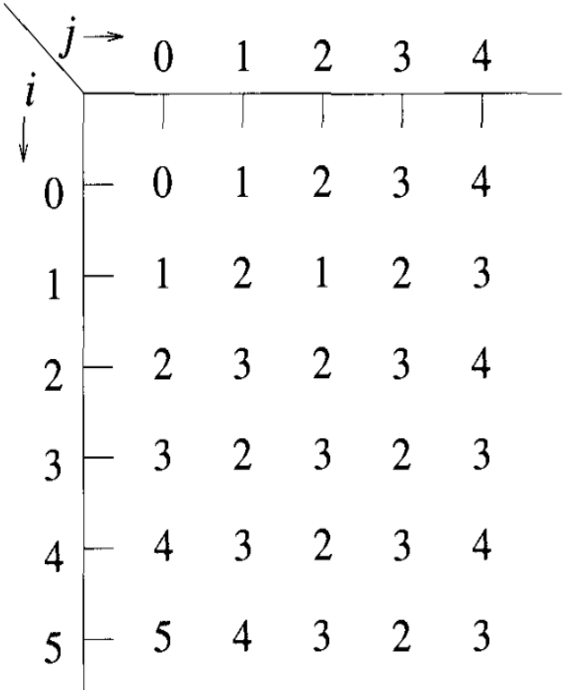
- Check the formula Cost(5,4)
- Cost(5,4) = min(

$$\begin{array}{l} cost(4,4) + D(x5) \\ cost(5,3) + I(y4) \\ cost(4,3) + c(x5,y4) \end{array} \quad )$$

=min( 4+1, 2+1, 3+0) =3

- Since two are the same minimum element, the problems 2 solutions with same minimum cost 3.

# Example for String Editing



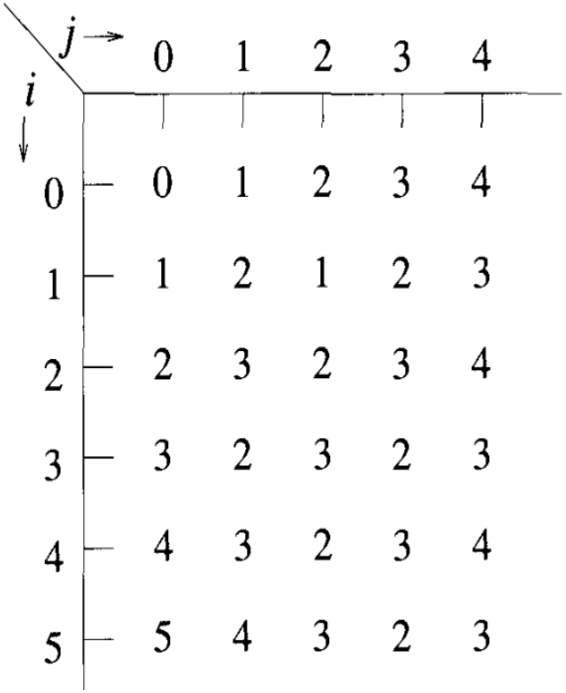
| i \ j | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 1 | 2 | 3 |
| 2 | 2 | 3 | 2 | 3 | 4 |
| 3 | 3 | 2 | 3 | 2 | 3 |
| 4 | 4 | 3 | 2 | 3 | 4 |
| 5 | 5 | 4 | 3 | 2 | 3 |

- Cost(5,4) = min(
  $cost(4,4) + D(x5)$
  $cost(5,3) + I(y4)$ )
  $cost(4,3) + c(x5, y4)$
  =min( 4+1, **2+1,** 3+0) =3
- .

- Cost(5,3) = min(
  $cost(4,3) + D(x5)$
  $cost(5,2) + I(y3)$ )
  $cost(4,2) + c(x5, y3)$
  =min( 3+1, 3+1, **2+0**) =2
- Cost(4,2)= min(
  $cost(3,2) + D(x4)$
  $cost(4,1) + I(y2)$
  $cost(3,1) + c(x4, y2)$
  =min(3+1,3+1,2+0) = 2
- Cost(3,1)= min(
  $cost(2,1) + D(x3)$
  $cost(3,0) + I(y1)$
  $cost(2,0) + c(x3, y1)$
  =min(3+1,3+1,2+0) = 2

- Cost(2,0) = cost(1,0)+D(x2)
  =1**+1** =2
- Cost(1,0)= cost(0,0)+D(x1)
  =0+1 = 1
- Hence the sequence of operations are D(x1), D(x2) and I(y4).
- **X= a, a, b, a, b   and**
- **Y= b, a, b, b.**
- After applying the operations
  **X= ~~a, a~~, b, a, b, b   and**
  **Y= b, a, b, b.**

# Example for String Editing



| i ↓ \ j → | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 1 | 2 | 1 | 2 | 3 |
| 2 | 2 | 3 | 2 | 3 | 4 |
| 3 | 3 | 2 | 3 | 2 | 3 |
| 4 | 4 | 3 | 2 | 3 | 4 |
| 5 | 5 | 4 | 3 | 2 | 3 |

- Cost(5,4) = min(
  $cost(4{,}4) + D(x5)$
  $cost(5{,}3) + I(y4)$ )
  $cost(4{,}3) + c(x5, y4)$
  =min( 4+1, 2+1, 3+0) =3

- Cost(4,3) = min(
  $cost(3{,}3) + D(x4)$
  $cost(4{,}2) + I(y3)$ )
  $cost(3{,}2) + c(x4, y3)$
  =min( 2+1, 2+1, 3+2) =3

- Cost(3,3)= min(
  $cost(2{,}3) + D(x3)$
  $cost(3{,}2) + I(y3)$
  $cost(2{,}2) + c(x3, y3)$
  =min(3+1,3+1,2+0) = 2

- Cost(2,2)= min(
  $cost(1{,}2) + D(x2)$
  $cost(2{,}1) + I(y2)$
  $cost(1{,}1) + c(x2, y2)$
  =min(1+1,3+1,2+0) = 2

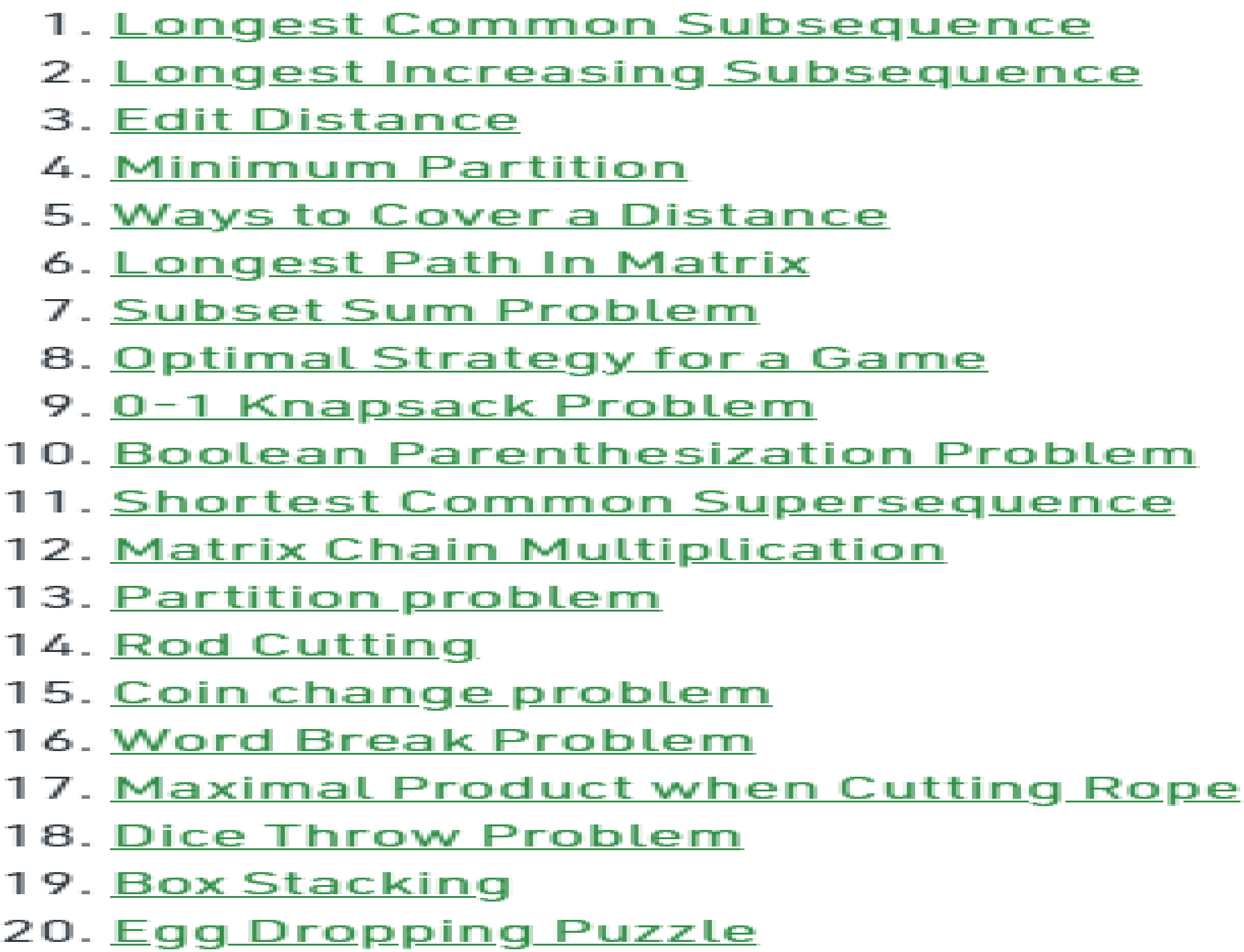
- Cost(1,1) = min(
  cost(0,1)+D(x1)
  cost(1,0)+I(y1)
  cost(0,0)+c(x1,y1)
  =min(1+1,1+1,0+2)
  =2

- Hence the sequence of operations are C(x1,y1) ,D(x4).
- **X= a, a, b, a, b and**
- **Y= b, a, b, b.**
- After applying the operations

  **X= ~~a~~ b, a, b, ~~a~~, b and**
  **Y= b, a, b, b.**

- .

# Frequently asked interview questions on Dynamic programming



1. Longest Common Subsequence
2. Longest Increasing Subsequence
3. Edit Distance
4. Minimum Partition
5. Ways to Cover a Distance
6. Longest Path In Matrix
7. Subset Sum Problem
8. Optimal Strategy for a Game
9. 0-1 Knapsack Problem
10. Boolean Parenthesization Problem
11. Shortest Common Supersequence
12. Matrix Chain Multiplication
13. Partition problem
14. Rod Cutting
15. Coin change problem
16. Word Break Problem
17. Maximal Product when Cutting Rope
18. Dice Throw Problem
19. Box Stacking
20. Egg Dropping Puzzle

# Topics

- General Method
- Travelling Sales Person Problem
- All Pairs Shortest Path Problem
- 0/1 Knapsack Problem
- Reliability Design Problem