# Design and Analysis of Algorithms

## Unit - IV

**Dr. R. Bhuvaneswari**
Assistant Professor
Department of Computer Science
Periyar Govt. Arts College, Cuddalore.

**Periyar Govt. Arts College**
**Cuddalore**

# Dynamic Programming

**General Method:**

- It is an algorithm design method that can be used when the solution to a problem can be viewed as a sequence of decisions.
- It obtains the solution using "**Principle of Optimality**".
- It states that " In an optimal sequence of decisions or choices, each subsequence must also be optimal", ie., whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence.
- The difference between the greedy method and dynamic programming is that in the greedy method only one decision sequence is ever generated.
- In dynamic programming, many decision sequences may be generated.
- Sequences containing suboptimal subsequences cannot be optimal and so will not be generated.

# Multistage Graphs

- A multistage graph $G = (V, E)$ is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets $V_i$, $1 \leq i \leq k$.
- If $\langle u, v \rangle$ is an edge in E, then $u \in V_i$ and $v \in V_{i+1}$.
- The sets $V_1$ and $V_k$ are such that $|V_1| = |V_k| = 1$.
- The vertex **s** is the source and the **t** the sink (destination).
- The multistage graph problem is to find a minimum cost path from s to t.
- The cost of s to t is the sum of the cost of the edges on the path.
- The multistage graph problem can be solved in 2 ways.
    - Forward method
    - Backward method

# Multistage Graphs

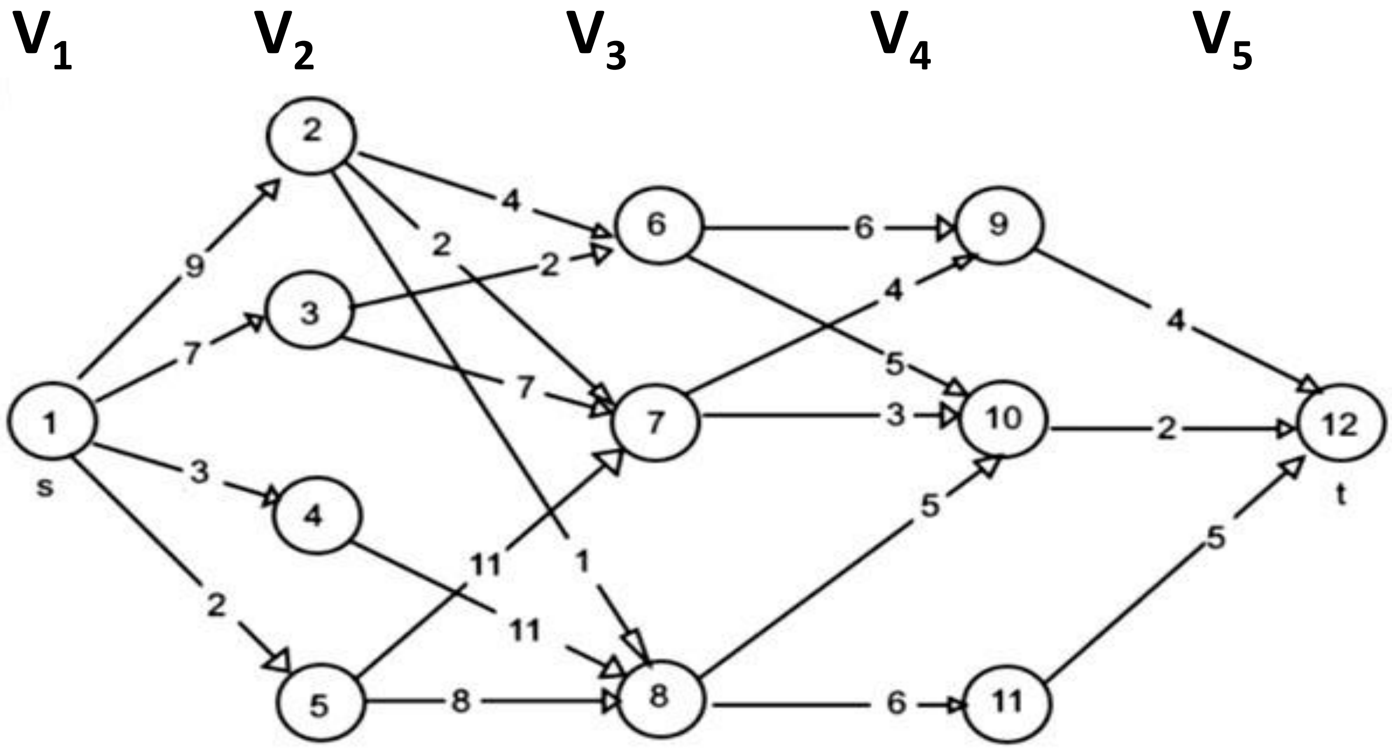
## Forward Approach

- In the forward approach, the cost of each and every node is found starting from the k stage to the $1^{st}$ stage.
- The minimum cost path from the source to destination is found ie., stage 1 to stage k.
- For forward approach,

$$\mathbf{Cost(i\ ,j) = \min_{\substack{l \in V_{i+1} \\ \langle j,\, l \rangle \in E}} \{c(j,\, l) + cost(i+1,\, l)\}}$$

where i is the level number.

# Multistage Graphs



# Multistage Graphs

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

| | | Min. Cost |
|---|---|---|
| cost(5,12) | 0 | 0 |
| cost(4,9) | min{c(9,12)+cost(5,12)} = {4 + 0} | 4 |
| cost(4,10) | min{**c(10,12)+cost(5,12)**} = {2 + 0} | 2 |
| cost(4,11) | min{c(11,12)+cost(5,12)} = {5+ 0} | 5 |
| cost(3,6) | min{c(6,9)+cost(4,9), **c(6,10)+cost(4,10)**}<br>= min{6+ 4, **5+2**} | 7 |
| cost(3,7) | min{c(7,9)+cost(4,9), **c(7,10)+cost(4,10)**}<br>= min{4+4, **3+2**} | 5 |
| cost(3,8) | min{c(8,10)+cost(4,10), c(8,11)+cost(4,11)}<br>= min{5+2, 6+5} | 7 |

# Multistage Graphs

| | | Min. Cost |
|---|---|---|
| cost(2,2) | min{c(2,6)+cost(3,6), **c(2,7)+cost(3,7)**, c(2,8)+cost(3,8)}<br>= min{4+7, 2+5, 1+7} | 7 |
| cost(2,3) | min{**c(3,6)+cost(3,6)**, c(3,7)+cost(3,7)}<br>= min{2+7, 7+5} | 9 |
| cost(2,4) | min{c(4,8)+cost(3,8)}<br>= min{11+7} | 18 |
| cost(2,5) | min{c(5,7)+cost(3,7), c(5,8)+cost(3,8)}<br>= min{11+5, 8+7} | 15 |
| cost(1,1) | min{**c(1,2)+cost(2,2)**, **c(1,3)+cost(2,3)**, c(1,4)+cost(2,4), c(1,5)+cost(2,5)}<br>= min{9+7, 7+9, 3+18, 2+15} | 16 |

1 ⇨ 2 ⇨ 7 ⇨ 10 ⇨ 12

1 ⇨ 3 ⇨ 6 ⇨ 10 ⇨ 12


Dr. R. Bhuvaneswari

Periyar Govt. Arts College
Cuddalore

# Multistage Graphs

```
Algorithm FGraph(G, k, n, p)
//p[1:k] is a minimum cost path
{
   cost[n] = 0.0;
   for j = n-1 to 1 step -1 do
  {
     Let r be a vertex such that ⟨j, r⟩ is an edge of G and c[j, r]+cost[r] is
     minimum;
    cost[j] = c[j, r] + cost[r];
    d[j] = r;
  }
  p[1] = 1; p[k] = n;
  for j = 2 to k-1 do
            p[j] = d[p[j-1]];
}
```

# Multistage Graphs

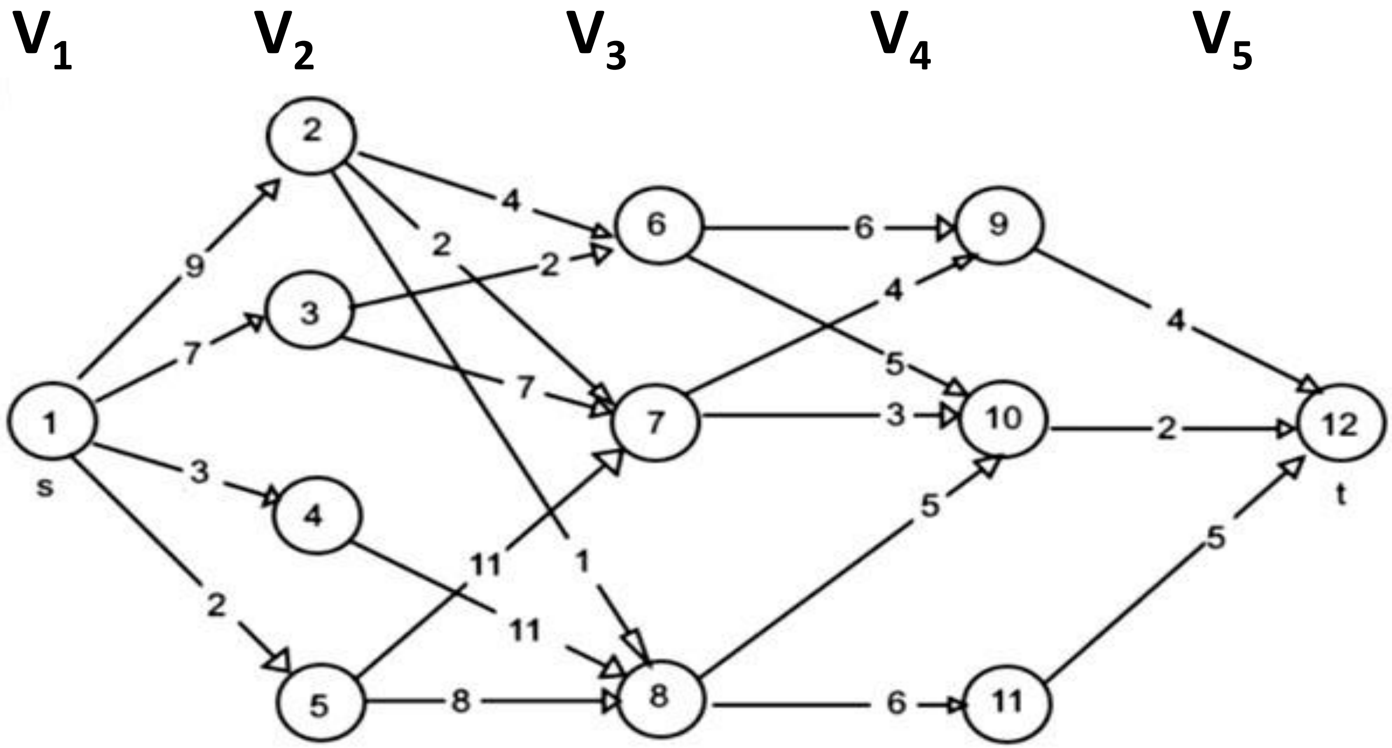
## Backward Approach

- In the backward approach, the cost of each and every node is found starting from the 1st stage to the kth stage.
- The minimum cost path from the source to destination is found ie., stage k to stage 1.
- For backward approach,

$$\mathrm{bcost}(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j\rangle \in E}} \{\mathrm{bcost}(i-1, l) + c(l, j)\}$$

where i is the level number.

# Multistage Graphs



# Multistage Graphs

$$\text{bcost}(i, j) = \min_{\substack{l \in V_{i-1} \\ \langle l, j\rangle \in E}} \{\text{bcost}(i-1, l) + c(l, j)\}$$

| | | Min. Cost |
|---|---|---|
| bcost(1,1) | 0 | 0 |
| bcost(2,2) | min{**bcost(1,1)+c(1,2)**} =min{0+9} | 9 |
| bcost(2,3) | min{**bcost(1,1)+c(1,3)**} =min{0+7} | 7 |
| bcost(2,4) | min{bcost(1,1)+c(1,4)} =min{0+3} | 3 |
| bcost(2,5) | min{bcost(1,1)+c(1,5)} =min{0+2} | 2 |
| bcost(3,6) | min{bcost(2,2)+c(2,6),**bcost(2,3)+c(3,6)**}<br>= min{9+4,7+2} | 9 |
| bcost(3,7) | min{**bcost(2,2)+c(2,7)**,bcost(2,3)+c(3,7),<br>bcost(2,5)+c(5,7)}<br>= min{9+2,7+7,2+11} | 11 |

# Multistage Graphs

| | | Min. Cost |
|---|---|---|
| bcost(3,8) | min{bcost(2,2)+c(2,8),bcost(2,4)+c(4,8), bcost(2,5)+c(5,8)}<br>= min{9+1,3+11,2+8} | 10 |
| bcost(4,9) | min{bcost(3,6)+c(6,9),Bcost(3,7)+c(7,9)}<br>= min{9+6,11+4} | 15 |
| bcost(4,10) | min{**bcost(3,6)+c(6,10)**,**bcost(3,7)+c(7,10),** bcost(3,8)+c(8,10)}<br>= min{9+5,11+3,10+5} | 14 |
| bcost(4,11) | min{bcost(3,8)+c(8,11)} = min{10+6} | 16 |
| Bcost(5,12) | min{bcost(4,9)+c(9,12),**bcost(4,10)+c(10,12)**, bcost(4,11)+c(11,12)}<br>= min{15+4,14+2,16+5} | 16 |

12 ⇨ 10 ⇨ 7 ⇨ 2 ⇨ 1 1⇨ 2 ⇨ 7 ⇨ 10 ⇨ 12

12 ⇨ 10 ⇨ 6 ⇨ 3 ⇨ 1 1⇨ 3 ⇨ 6 ⇨ 10 ⇨ 12

# Multistage Graphs

```
Algorithm BGraph(G, k, n, p)
{
  bcost[1] = 0.0;
  for j = 2 to n do
  {
    Let r be such that ⟨r, j⟩ is an edge of G and bcost[r] + c[r, j] is minimum;
    bcost[j] = bcost[r] + c[r, j];
    d[j] = r;
  }
  p[1] = 1; p[k] = n;
  for j = k-1 to 2 step -1 do
          p[j] = d[p[j+1]];
}
```

# All pair shortest paths

- All pairs shortest path problem is the determination of the shortest graph distances between every pair of vertices in a given directed graph G.
- That is, for every pair of vertices (i, j), we are to find a shortest path from i to j as well as from j to i. These two paths are the same when G is undirected.
- Let $G = (V, E)$ be a directed graph with n vertices.
- Let cost be a cost adjacency matrix for G such that $cost(i, i) = 0, 1 \le i \le n$.
- $Cost(i, j)$ is the length of edge $\langle i, j\rangle$ if $\langle i, j\rangle \in E(G)$ and $cost(i, j) = \infty$ if $i \neq j$ and $\langle i, j\rangle \notin E(G)$.
- All pair 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.
- Since each application of this procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time.

# All pair shortest paths

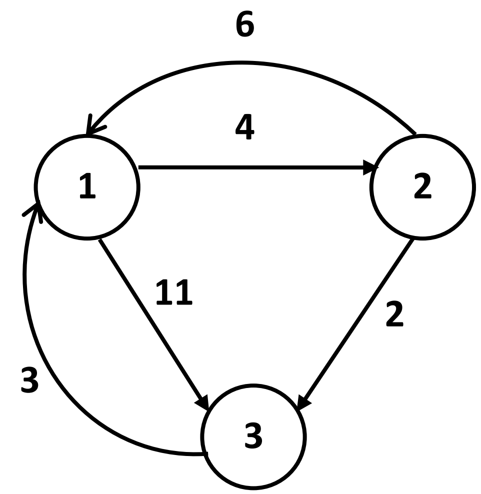
- The shortest i to j path in G, $i \neq j$ originates at vertex i and goes through some intermediate vertices and terminates at vertex j.
- If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively.
- Otherwise, the i to j path is not of minimum length.
- So, the principle of optimality holds.
- Let $A^k(i, j)$ represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain:

$$A^k(i,j) = \min_{1 \le k \le n} \{A^{k-1}(i,k) + A^{k-1}(k,j), cost(i,j)\}$$

- Time complexity of this algorithm is $O(n^3)$

# All pair shortest paths

```
Algorithm AllPaths(cost, A, n)
{
   for i =1 to n do
   {
     for j = 1 to n do
           A[i, j] = cost[i, j];
   }
   for k = 1 to n do
   {
     for j = 1 to n do
     {
       for j = 1 to n do
           A[i, j] = min{A[i, j], A[i, k]+A[k, j]};
     }
   }
}
```




1->3 = 11

1->2->3 = 6

2->1 = 6

2->3->1 = 5

# All pair shortest paths

Solve the problem for k = 1, 2, 3

Cost adjacency matrix

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

Solving the equation for, k = 1

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

Solving the equation for, k = 2

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

Solving the equation for, k = 3

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

# String Editing

- Given two strings $X = x_1, x_2, \ldots , x_n$ and $Y = y_1, y_2, \ldots, y_n$, where $x_i$, $1 \leq i \leq n$, and $y_j$, $1 \leq j \leq m$, are members of a finite set of symbols known as the alphabet.
- We want to transform X into Y using a sequence of edit operations on X.
- The permissible edit operations are insert, delete and change, and there is a cost associated with each operation.
- The cost of a sequence of operations is the sum of the costs of the individual operations in the sequence.
- The problem of string editing is to identify a minimum-cost sequence of edit operations that will transform X into Y.
- $D(x_i)$ – cost of deleting the symbol $x_i$ from X
- $I(y_j)$ – the cost of inserting the symbol $y_j$ into X
- $C(x_i, y_j)$ – cost of changing the symbol $x_i$ of X into $y_j$
- Cost of changing any symbol to any other symbol is 2.

# String Editing

- Cost associated with each insertion and deletion is 1.
- A dynamic programming solution to this problem can be obtained as follows.
- Define cost(i, j) be the minimum cost of any edit sequence for transforming $x_1, x_2, \ldots, x_i$ into $y_1, y_2, \ldots, y_j$
- Compute cost(i, j) for each i and j.
- Then cost(n, m) is the cost of an optimal edit sequence.
- For i = j = 0, cost(i, j) = 0, since the two sequences are identical and empty.
- If j = 0 and i > 0, we can transform X into Y by a sequence of deletes.

  $\text{Cost}(i, 0) = \text{cost}(i\text{-}1, 0) + D(x_i)$.
- If i = 0 and j > 0, we can transform X into Y by a sequence of insertions

  $\text{Cost}(0, j) = \text{cost}(0, j\text{-}1) + I(y_j)$

# String Editing

- If $i \neq 0$ and $j \neq 0$, $x_1, x_2, \ldots., x_i$ can transformed into $y_1, y_2, \ldots y_j$ in one of the following ways:
  - Transform $x_1, x_2, \ldots., x_{i-1}$ into $y_1, y_2, \ldots, y_j$ using a minimum-cost edit sequence and then delete $x_i$. The cost is cost(i-1, j) + $D(x_i)$
  - Transform $x_1, x_2, \ldots., x_{i-1}$ into $y_1, y_2, \ldots, y_{j-1}$ using a minimum-cost edit sequence and then change the symbol $x_i$ to $y_j$. The cost is cost(i-1, j-1) + $C(x_i, y_j)$
  - Transform $x_1, x_2, \ldots., x_i$ into $y_1, y_2, \ldots, y_{j-1}$ using a minimum-cost edit sequence and then insert $y_j$. The cost is cost(i, j-1) + $I(y_j)$
- The minimum cost of any edit sequence is the minimum of the above three costs, according to the principle of optimality.

# String Editing

- The recurrence equation for cost(i, j) is

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

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($y_j$)}

# String Editing

Given two strings, X and Y and edit operations (given below). Convert string X into Y with minimum number of operations.

Allowed Operations:

- Insertion – Insert a new character.
- Deletion – Delete a character.
- Replace – Replace one character by another.

**Example 1:**

String X = "horizon"

String Y = "horzon"

Output: {remove 'i' from string X}

**Example 2:**

String X = a, a, b, a, b

String Y = b, a, b, b

For the cases $i = 0, j > 1$, and $j = 0, i > 1$, cost(i, j) can be computed first and tabulated in the form of a table. The rest of the entries in the table can be computed in the row-major order.

# String Editing

Cost(1,1) = min{cost(0,1)+D(x1), **cost(0,0) + C(x1,y1)**, cost(1,0)+I(y1)}
= min{1+1, **0+2**, 1+1} = 2

Cost(1,2) = min{cost(0,2)+D(x1), **cost(0,1) + C(x1,y2)**, cost(1,1)+I(y2)}
= min{2+1, **1+0**, 2+1} = 1

Cost(1,3) = min{cost(0,3)+D(x1), cost(0,2) + C(x1,y3), cost(1,2)+I(y3)}
= min{3+1, 2+2, 1+1} = 2

Cost(1,4) = min{cost(0,4)+D(x1), cost(0,3) + C(x1,y4), cost(1,3)+I(y4)}
= min{4+1, 3+2, 2+1} = 3

Cost(2,1) = min{cost(1,1)+D(x2), cost(1,0) + C(x2,y1), cost(2,0)+I(y1)}
= min{2+1, 1+2, 2+1} = 3

Cost(2,2) = min{**cost(1,2)+D(x2)**, **cost(1,1) + C(x2,y2)**, cost(2,1)+I(y2)}
= min{**1+1**, **2+0**, 3+1} = 2

Cost(2,3) = min{cost(1,3)+D(x2), cost(1,2) + C(x2,y3), cost(2,2)+I(y3)}
= min{2+1, 1+2, 2+1} = 3

Cost(2,4) = min{cost(1,4)+D(x2), cost(1,3) + C(x2,y4), cost(2,3)+I(y4)}
= min{3+1, 2+2, 3+1} = 4

# String Editing

Cost(3,1) = min{cost(2,1)+D(x3), **cost(2,0) + C(x3,y1)**, cost(3,0)+I(y1)}
= min{3+1, **2+0**, 3+1} = 2

Cost(3,2) = min{cost(2,2)+D(x3), cost(2,1) + C(x3,y2), cost(3,1)+I(y2)}
= min{2+1, 3+2, 2+1} = 3

Cost(3,3) = min{cost(2,3)+D(x3), **cost(2,2) + C(x3,y3)**, cost(3,2)+I(y3)}
= min{3+1, **2+0**, 3+1} = 2

Cost(3,4) = min{cost(2,4)+D(x3), cost(2,3) + C(x3,y4), cost(3,3)+I(y4)}
= min{4+1, 3+0, 2+1} = 3

Cost(4,1) = min{cost(3,1)+D(x4), cost(3,0) + C(x4,y1), cost(4,0)+I(y1)}
= min{2+1, 3+2, 4+1} = 3

Cost(4,2) = min{cost(3,2)+D(x4), **cost(3,1) + C(x4,y2)**, cost(4,1)+I(y2)}
**=** min{3+1, **2+0**, 3+1} = 2

Cost(4,3) = min{**cost(3,3)+D(x4)**, cost(3,2) + C(x4,y3), **cost(4,2)+I(y3)**}
= min{**2+1**, 3+2, **2+1**} = 3

Cost(4,4) = min{cost(3,4)+D(x4), cost(3,3) + C(x4,y4), cost(4,3)+I(y4)}
= min{3+1, 2+2, 3+1} = 4

# String Editing

Cost(5,1) = min{cost(4,1)+D(x5), cost(4,0) + C(x5,y1), cost(5,0)+I(y1)}
= min{3+1, 4+0, 5+1} = 4

Cost(5,2) = min{cost(4,2)+D(x5), cost(4,1) + C(x5,y2), cost(5,1)+I(y2)}
= min{2+1, 3+2, 4+1} = 3

Cost(5,3) = min{cost(4,3)+D(x5), **cost(4,2) + C(x5,y3)**, cost(5,2)+I(y3)}
= min{3+1, **2+0**, 3+1} = 2

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

| 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 |

**Optimal operations are:**

- **Insert $y_1$, delete $x_2$ and $x_4$**
- **Change $x_1$ by $y_1$, delete $x_4$**
- **Delete $x_1$ and $x_2$, insert $y_3$**
- **Delete $x_1$ and $x_2$, insert $y_4$**

Time complexity: O(mn)

# 0/1 Knapsack Problem

- Given n objects and a knapsack or bag.
- $w_i \rightarrow$ weight of object i.
- $m \rightarrow$ knapsack capacity.
- As the name suggests, objects are indivisible in this method. No fractional objects can be taken. An object can either be taken completely or left completely.
- Objective is to fill the knapsack that maximizes the total profit earned.
- Problem can be stated as

$$\text{maximize} \sum_{1 \leq i \leq n} p_i \, x_i$$

$$\text{subject to} \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$x_i = 0 \text{ or } 1, 1 \leq i \leq n$$

# 0/1 Knapsack Problem

0/1 knapsack problem is solved using dynamic programming in the following steps-

- Draw a table say 'V' with (n+1) number of rows and (w+1) number of columns.
- Fill all the boxes of $0^{th}$ row and $0^{th}$ column with zeroes.
- Start filling the table row wise top to bottom from left to right.
- Use the following formula:

$$V[i, W] = \max\{V[i-1, W], V[i-1, W - w[i]] + p[i]\}$$

- value of the last box represents the maximum possible value that can be put into the knapsack.

# 0/1 Knapsack Problem

- To identify the items that must be put into the knapsack to obtain the maximum profit,
  - ➢ Consider the last column of the table.
  - ➢ Start scanning the entries from bottom to top.
  - ➢ On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
  - ➢ After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.
- O(nw) time is taken to solve 0/1 knapsack problem using dynamic programming.

# 0/1 Knapsack Problem

$P_i = \{1, 2, 5, 6\}$

$w_i = (2, 3, 4, 5\}$

$m = 8, n = 4$

| | W→ | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_i$ | $w_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 8 |

**V[i ,W] = max{V[i-1 ,W] , V[i-1, W – w[i]] + p[i]}**

V[1,1] = max{V[0,1], V[0,1-2]+1} = max{0, -} = 0

V[1,2] = max{V[0,2], V[0,2-2]+1} = max{0, 0+1} = 1

V[1,3] = max{V[0,3], V[0,3-2]+1} = max{0,0+1} = 1

V[1,4] = max{V[0,4], V[0,4-2]+1} = max{0, 0+1} = 1

V[1,5] = max{V[0,5], V[0,5-2]+1} = max{0, 0+1} = 1

V[1,6] = max{V[0,6], V[0,6-2]+1} = max{0, 0+1} = 1

V[1,7] = max{V[0,7], V[0,7-2]+1} = max{0, 0+1} = 1

V[1,8] = max{V[0,8], V[0,8-2]+1} = max{0, 0+1} = 1

# 0/1 Knapsack Problem

V[2,1] = max{V[1,1],V[1,1-3]+2} = max{0, -} = 0
V[2,2] = max{V[1,2],V[1,2-3]+2} = max{1, -} = 1
V[2,3] = max{V[1,3],V[1,3-3]+2} = max{1, 0+2} = 2
V[2,4] = max{V[1,4],V[1,4-3]+2} = max{1, 0+2} = 2
V[2,5] = max{V[1,5],V[1,5-3]+2} = max{1, 1+2} = 3
V[2,6] = max{V[1,6],V[1,6-3]+2} = max{1, 1+2} = 3
V[2,7] = max{V[1,7],V[1,7-3]+2} = max{1, 1+2} = 3
V[2,8] = max{V[1,8],V[1,8-3]+2} = max{1, 1+2} = 3

V[3,1] = max{V[2,1],V[2,1-4]+5} = max{0, -} = 0
V[3,2] = max{V[2,2],V[2,2-4]+5} = max{1, -} = 1
V[3,3] = max{V[2,3],V[2,3-4]+5} = max{2, -} = 2
V[3,4] = max{V[2,4],V[2,4-4]+5} = max{2, 0+5} = 5
V[3,5] = max{V[2,5],V[2,5-4]+5} = max{2, 0+5} = 5
V[3,6] = max{V[2,6],V[2,6-4]+5} = max{2, 1+5} = 6

# 0/1 Knapsack Problem

V[3,7] = max{V[2,7],V[2,7-4]+5} = max{2, 2+5} = 7
V[3,8] = max{V[2,8],V[2,8-4]+5} = max{2, 2+5} = 7

V[4,1] = max{V[3,1],V[3,1-5]+6} = max{0, -} = 0
V[4,2] = max{V[3,2],V[3,2-5]+6} = max{1, -} = 1
V[4,3] = max{V[3,3],V[3,3-5]+6} = max{2, -} = 2
V[4,4] = max{V[3,4],V[3,4-5]+6} = max{5, -} = 5
V[4,5] = max{V[3,5],V[3,5-5]+6} = max{5, 0+6} = 6
V[4,6] = max{V[3,6],V[3,6-5]+6} = max{6, 0+6} = 6
V[4,7] = max{V[3,7],V[3,7-5]+6} = max{7, 1+6} = 7
V[4,8] = max{V[3,8],V[3,8-5]+6} = max{7, 2+6} = **8**

$\mathbf{x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1}$

# 0/1 Knapsack Problem

```
for (i = 0; i ≤ n; i++)
{
   for(w = 0; w ≤ m; w++)
   {
      if(i==0 || w==0)
            k[i][w] = 0;
      else if(wt[i] ≤ w)
            k[i][w] = max(p[i]+k[i-1][w-wt[i], k[i-1][w]);
      else
            k[i][w] = k[i-1][w];
   }
}
```

# Traveling Salesperson Problem

- The traveling salesperson problem is to find a tour of minimum cost.
- Let G = (V, E) be a directed graph with edge cost $C_{ij} = \infty$ if $\langle i, j \rangle \notin E$
- Let |V| = n and assume |n| > 1
- A tour 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.
- 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.

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

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

# Traveling Salesperson Problem

$g(i,\phi) = C_{i1},\ 1 \le i \le n.$

**$|S| = \phi$**

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

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

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

Using equation 2, we obtain

**$|S| = 1$**

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

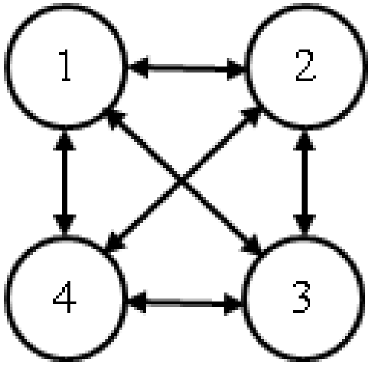
$g(2,\{4\}) = c_{24} + g(4,\phi) = 10+8 = 18$

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

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

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

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




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 9 | 10 |
| 3 | 6 | 13 | 0 | 12 |
| 4 | 8 | 8 | 9 | 0 |

# Traveling Salesperson Problem

**|S| = 2**

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

$= \min\{9+20, 10+15\} = 25$

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

$= \min\{13+18, 12+13\} = 25$

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

$= \min\{8+15, 9+18\} = 23$

Using equation 1, we obtain

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

$= \min\{10+25, 15+25, 20+23\} = 35$

**The optimal tour is**

**1->2->4->3->1**

$O(2^n n^2)$ time is taken to solve the traveling salesperson problem