Dynamic Programming

Travelling Sales person problem:-

Consider $G=(V,E)$

Assume source vertex = 1.

A tour is defined as a simple path that starts and ends at source vertex 1.

* The tour may start from any of the edge $(1,K)$ where $K \in V-\{1\}$
  - $V-\{1\}$: all nodes except source vertex 1
  - K: source vertex.
* There is a path from vertex K to 1. It goes thro' each vertex in $V-\{1,K\}$
  - $V-\{1,K\}$: nodes in b/w K to 1.
* Let $g(i,S)$ be the cost of a shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1.
* The fn. $g(1,V-\{1\})$ is the least cost of an optimal sales person tour.

From the principle of optimality,

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

The above eqn gives min cost of tour from source vertex 1.

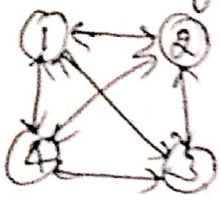
* It can be generalized as

$$g(i,S) = \min_{j \in S} \{ c_{ij} + g(j, S-\{j\}) \}$$ where $|S| < n-1$, $i \neq 1$, $i \neq S$, $1 \notin S$

* The base case is given by

$$g(i,\phi) = c_{i1} \text{ for } 1 \le i \le n \text{ and } i \neq 1 \text{ when } |S| = 0$$

Ex1 Solve the following TSP which is represented as digraph & whose edge lengths are given by cost adjacency matrix C



C

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

Given: No of nodes = 4.
Cost adjacency matrix C

The TSP can be solved using 3 rules as shown;

Case 1: $S = \{\phi\}$, so $|S| = 0$

$\therefore g(i, \phi) = C_{i1}$ where $1 \le i \le n$ and $i \neq 1$

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

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

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

Case 2: $|S| = 1$ In this case $i \neq 1$, $1 \notin S$, $i \notin S$ and solve the following relation

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

When $i = 2$

$g(2, \{3\}) = \min_{j \in \{3\}} \{C_{23} + g(3, \phi)\} = \min(9 + 6) = 15$

$g(2, \{4\}) = \min_{j \in \{4\}} \{C_{24} + g(4, \phi)\} = \min(10 + 8) = 18$

When $i = 3$

$g(3, \{2\}) = \min_{j \in \{2\}} \{C_{32} + g(2, \phi)\} = \min(13 + 5) = 18$

$g(3, \{4\}) = \min_{j \in \{4\}} \{C_{34} + g(4, \phi)\} = \min(12 + 8) = 20$

When $i = 4$

$g(4, \{2\}) = \min_{j \in \{2\}} \{C_{42} + g(2, \phi)\} = \min(8 + 5) = 13$

$g(4, \{3\}) = \min_{j \in \{3\}} \{C_{43} + g(3, \phi)\} = \min(9 + 6) = 15$

Dynamic Programming

It is a technique for solving pbms with overlapping subpbms. Typically, these subpbms arise from a recurrence relating a soln. to a given pbm with soln. to its smaller subpbms of the same type.

Computing a Binomial Co-efficient

It is a std ex. of applying Dy. Pg. to a nonoptimization pbm. B.C. is denoted $C(n,k)$ or $\binom{n}{k}$ is the no. of combinations (subsets) of k elements from an n-element set $(0 \le k \le n)$. The name bin-coeff comes from the participation of these nos in the bin formula

$$(a+b)^n = C(n,0)a^n + \dots + C(n,k)a^{n-k}b^k + \dots + C(n,n)b^n$$

We concentrate on a property of Bin-coeff

$$C(n,k) = C(n-1,k-1) + C(n-1,k) \quad \text{for } n>k>0 \quad \text{--- (1)}$$

&

$$C(n,0) = C(n,n) = 1 \quad \text{--- (2)}$$

We record the values of B.C. in a table of n+1 rows & k+1 cols numbered 0 to n & from 0 to k, as shown

	0	1	2	...	k-1	k
0	1					
1	1	1				
2	1	2	1			
.	1	–	–	1		
.						
k	1	.				1
.						
n-1	1			$C(n-1,k-1)$	$C(n-1,k)$	
n	1				$C(n,k)$	

ALGORITHM Binomial(n, k)
// computes C(n,k) by the dynamic programming
// algorithm
// I/p: A pair of nonnegative integers $n \geq k \geq 0$
// O/p: The value of C(n,k)

```
for i ← 0 to n do
    for j ← 0 to min(i,k) do
        if j = 0 or j = i
            C[i,j] ← 1
        else C[i,j] ← C[i-1, j-1] + C[i-1, j]
return C[n,k].
```

Time efficiency:

Basic-op: Addition

Let A(n,k) be total no of additions made by the algo. in computing C(n,k)

Since the first k+1 rows of the table form a Δ^k while the remaining n-k rows form a rectangle, we have to split the sum expressing A(n,k) into 2 parts:

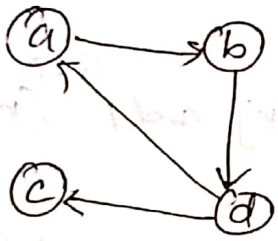
$$A(n,k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1 \qquad \sum_{i=k+1}^{n} \sum_{j=1}^{i} 1$$

$$= \sum_{i=1}^{k} (i-1) + \sum_{i=k+1}^{n} k$$

$$= \frac{(k-1)k}{2} + k(n-k) \in \Theta(nk)$$

Warshall's Algorithm

The transitive closure of a directed graph with n vertices can be defined as the n-by-n boolean matrix $T = \{t_{ij}\}$ in which the element in the i^{th} row $(1 \le i \le n)$ and the j^{th} column $(1 \le j \le n)$ is 1 if there exists a nontrivial directed path (i.e. a directed path of a positive length) from the i^{th} vertex to the j^{th} vertex, otherwise t_{ij} is 0.



Digraph

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Its adjacency matrix

$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Its transitive closure.

Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n-by-n boolean matrices

$$R^{(0)} \ldots R^{(k-1)} \rightarrow R^{(k)} \ldots R^{(n)} \quad \ldots (A)$$

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \text{ or } \left(r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)} \right)$$

Algorithm Warshall (A[1..n, 1..n])

// Implements Warshall's algorithm for computing
// the transitive closure.

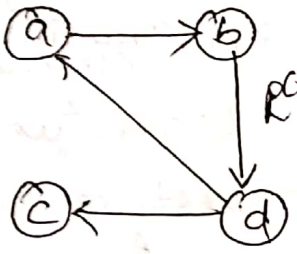
// I/p: The adjacency matrix A of a digraph
with n vertices

// O/p: The transitive closure of a digraph.

$R^{(0)} \leftarrow A$

for k ← 1 to n do
  for i ← 1 to n do
    for j ← 1 to n do
      $R^{(k)}[i,j] \leftarrow R^{(k-1)}[i,j]$ or
      ($R^{(k-1)}[i,k]$ and $R^{(k-1)}[k,j]$)

returns $R^{(n)}$.



$R^{(0)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

Ones reflect the existence of paths with no intermediate vertices.
$R^{(0)}$ is just the adjacency matrix.

$R^{(1)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

Ones reflect the existence of paths with intermediate vertices numbered not higher than 1 i.e just vertex a.

$R^{(2)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

not higher than 2 i.e a and b.

$R^{(3)}$ =

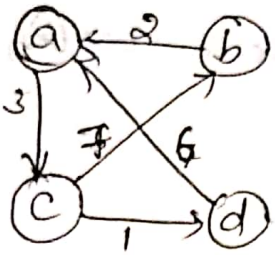
	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

--3 a, b & c

$R^{(4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ --4 a, b, c & d

Floyd's Algorithm for The All-Pairs shortest path problem

Given a weighted connected graph (undirected or directed), the all-pairs shortest-path problem asks to find the distances (the lengths of the shortest paths) from each vertex to all other vertices.

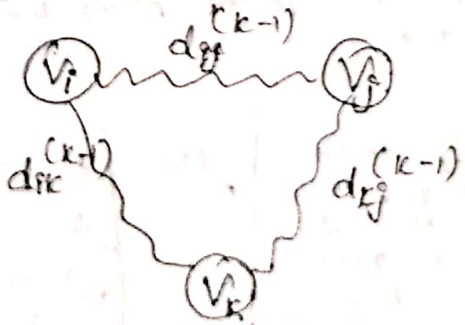


$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array} \qquad D = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$

The lengths of shortest paths are recorded in an n-by-n matrix D called distance matrix: the element d_{ij} in the i^{th} row and the j^{th} col. of this matrix indicates the length of the shortest path from the i^{th} vertex to the j^{th} vertex ($1 \le i, j \le n$)

Floyd's algo computes the distance matrix of a weighted graph with n-vertices through a series of n-by-n matrices:

$$D^{(0)}, \ldots D^{(k-1)} \; D^{(k)} \ldots D^{(n)}$$



Each path is made up of a path from v_i to v_k with each intermediate vertex numbered not higher than $k-1$ and a path from v_k to v_j with each intermediate vertex numbered not higher than $k-1$.

Recurrence:

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \text{ for } k \ge 1, \quad d_{ij}^{(0)} = w_{ij}$$

Algorithm Floyd(W[1..n, 1..n])

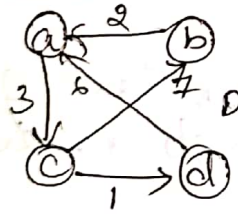
// Implements Floyd's algorithm for the all-pairs-
// shortest paths problem
// I/p: The weight matrix W of a digraph with no
// negative length cycle
// O/p: The distance matrix of the shortest paths
lengths, ~~D ← W~~

D ← W // is not necessary if W can be overwritten.

for k ← 1 to n do
  for i ← 1 to n do
    for j ← 1 to n do
      D[i,j] ← min(D[i,j], D[i,k] + D[k,j])
return D.

Time efficiency: cubic

Richard Bellman called it principle of optimality



$D^{(0)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

lengths of the shortest paths with no intermediate vertices.

$D^{(1)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

lengths of the shortest paths with intermediate vertices numbered not higher than 1 i.e (just a from two new paths from b to c and from d to c.

$D^{(2)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

lengths of shortest paths with I.V. numbered not higher than 2. i.e a and b (new path from c to a)

$D^{(3)}$ =

0	10	3	4
2	0	5	6
9	7	0	1
6	16	9	0

$D^{(4)}$ =

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

- - no not higher than 4.
i.e a, b, c, and d (make a new shortest path from c to a.

Dynamic Programming

Single source shortest paths: General weights

Dijkstra's Algo in Greedy method does not support negative weights.

Bellman and Ford Recurrence Relation

$d[i] = cost(v, i)$ for $i = 1$ to n

$d^k[u] = \min(d^{k-1}[u], \min(d^{k-1}[i] + cost[i, u]))$ for $k = 2$ to $n-1$

for each u such that $u \neq v$ and for each $[i, u]$ where $i \neq u$

ALGORITHM BellmanFord (v, cost, d, n)

// purpose: To compute shortest distance from a given // node to all other nodes of the graph.

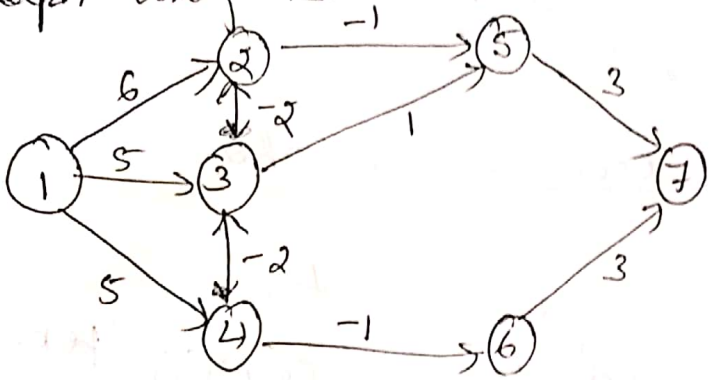
// I/p: v is the source node,
cost " " Cost adjacency matrix
n is the no of nodes in the graph

// o/p: d is the distance matrix that gives the shortest distance from source node v to all other vertices of the graph.

Step 1: [Initialization to find the shortest distance]
for $i \leftarrow 1$ to n do
$d[i] \leftarrow cost[v, i]$
end for

Step 2: [Compute the shortest distance]
for $k \leftarrow 2$ to $n-1$ do
for each u such that $u \neq v$ and u has atleast one incoming edge
for each (i, u) in the graph
if $(d[i] + cost[i, u] < d[u])$
$d[u] = d[i] + cost[i, u]$
end if
end for
end for
end for

Ex: Find the shortest paths from node 1 to every other node in the graph following ~~graph~~ using Bellman and Ford algorithm.



Soln: The shortest path can be computed using Bellman & Ford algorithm with the help of following recurrence relⁿ

$d[i] = cost[v,i]$ for $i = 1$ to n

$d^k[u] = \min(d^{k-1}[u], \min(d^{k-1}[i] + cost[i,u]))$ } for $k = 2$ to $n-1$ for each u such that $u \neq v$ & for each (i,u) in the graph

Given source = 1.

The cost adj matrix is

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

since source = 1

1	2	3	4	5	6	7
0	6	5	5	∞	∞	∞

Initial distance of

The value of d^k: $d^k[u] = \min(d^{k-1}[u], \min\{d^{k-1}[i] + cost[i,u]\})$ for $k = 2, 3, 4, 5$, and 6 can be computed as shown below:

When $k = 2$

d	i	$d^2[2]$	$d^2[3]$	$d^2[4]$	$d^2[5]$	$d^2[6]$	$d^2[7]$
0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
6	→ 2	6+	6+	6+	6+	6+	6+
5	→ 3	5+	5+	5+	5+	5+	5+
5	→ 4	5+	5+	5+	5+	5+	5+
∞	→ 5	∞+	∞+	∞+	∞+	∞+	∞+
∞	→ 6	∞+	∞+	∞+	∞+	∞+	∞+
∞	→ 7	∞+	∞+	∞+	∞+	∞+	∞+
	min:	3	3	5	5	4	∞

When $k=3$,

	d	i	$d^3[2]$	$d^3[3]$	$d^3[4]$	$d^3[5]$	$d^3[6]$	$d^3[7]$
			3	3	5	5	4	∞
1	0	1	0+6	0+5	0+5	0+∞	0+∞	0+6
2	3	2	3+0	3+∞	3+6	3+1	3+∞	3+∞
3	3	3	3+2	3+0	3+∞	3+1	3+∞	3+∞
4	5	4	5+∞	5+2	5+0	5+∞	5+1	5+∞
5	5	5	5+∞	5+∞	5+∞	5+0	5+∞	5+3
6	4	6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
7	∞	7	∞+∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+0
		min	1	3	5	2	4	7

When $k=4$

	d	i	$d^4[2]$	$d^4[3]$	$d^4[4]$	$d^4[5]$	$d^4[6]$	$d^4[7]$
			1	3	5	2	4	7
	0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
	1	→ 2	1+0	1+∞	1+∞	1+1	1+∞	1+∞
	3	→ 3	3+2	3+0	3+∞	3+1	3+∞	3+∞
	5	→ 4	5+∞	5+2	5+0	5+∞	5+1	5+∞
	2	→ 5	2+∞	2+∞	2+∞	2+0	2+∞	2+3
	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
	7	→ 7	7+∞	7+∞	7+∞	7+∞	7+∞	7+0
		min	1	3	5	0	4	5

When $k=5$

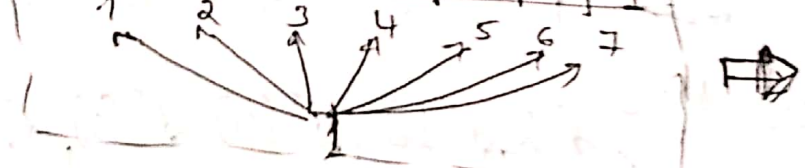
	d	i	$d^5[2]$	$d^5[3]$	$d^5[4]$	$d^5[5]$	$d^5[6]$	$d^5[7]$
			1	3	5	0	4	5
	0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
	1	→ 2	1+0	1+∞	1+∞	1+1	1+∞	1+∞
	3	→ 3	3+2	3+0	3+∞	3+1	3+∞	3+∞
	5	→ 4	5+∞	5+2	5+0	5+∞	5+1	5+∞
	0	→ 5	0+∞	0+∞	0+∞	0+0	0+∞	0+3
	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
	5	→ 7	5+∞	5+∞	5+∞	5+∞	5+∞	5+0
		min	1	3	5	0	4	3

When $k=6$

d	x	$d^6[2]$	$d^6[3]$	$d^6[4]$	$d^6[5]$	$d^6[6]$	$d^6[7]$
		1	3	5	0	4	3
0	1	0+6	0+5	0+5	0+∞	0+∞	0+∞
1	2	1+0	1+∞	1+∞	1−1	1+∞	1+∞
3	3	3−2	3+0	3+∞	3+1	3+∞	3+∞
5	4	5+∞	5−2	5+0	5+∞	5−1	5+∞
0	5	0+∞	0+∞	0+∞	0+0	0+∞	0+3
4	6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
3	7	3+∞	3+∞	3+∞	3+∞	3+∞	3+0
min =		1	3	5	0	4	3

So the final distance matrix obtained for each $k = 2, 3, 4, 5, 6$ is shown below.

	1	2	3	4	5	6	7
Initial distance d	0	6	5	5	∞	∞	∞
for k=2, distance d	0	3	3	5	5	4	∞
" k=3, " d	0	1	3	5	2	4	7
" k=4 " d	0	1	3	5	0	4	5
" k=5 " d	0	1	3	5	0	4	3
" k=6, " d	0	1	3	5	0	4	3



Shortest distance from 1 to 1 = 0

" " 1 to 2 = 1

" " 1 to 3 = 3

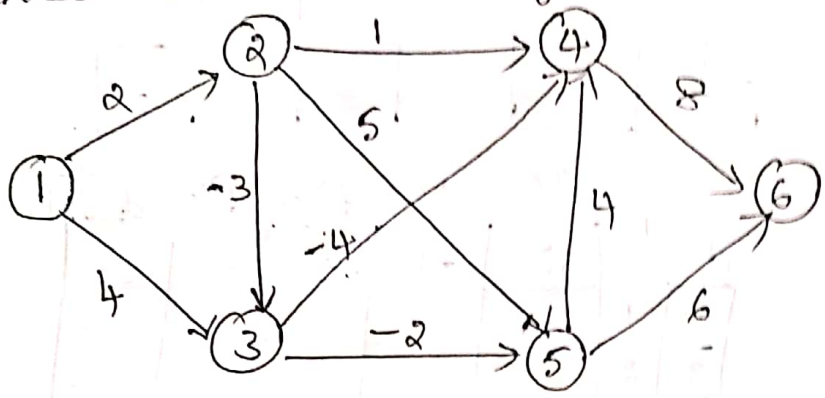
" " 1 to 4 = 5

" " 1 to 5 = 0

" " 1 to 6 = 4

1 to 7 = 3

Ex 2 Find the shortest paths from node 1 to every other node in the graph using Bellman and Ford algorithm.



Soln: Recurrence Rely

	1	2	3	4	5	6
1	0	2	4	∞	∞	∞
2	∞	0	-3	1	∞	∞
3	∞	∞	0	-4	-2	∞
4	∞	∞	∞	∞	∞	8
5	∞	∞	∞	4	0	6
6	∞	∞	∞	∞		0

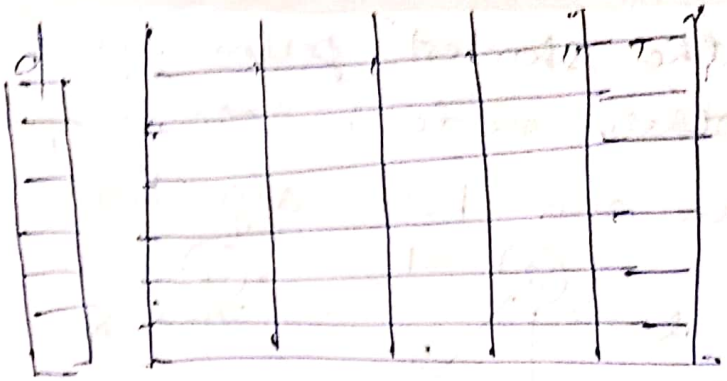
Simple source = 1

1	2	3	4	5	6
0	2	4	∞	∞	∞

When $x = 2$

d	$d^2[2]$	$d^2[3]$	$d^2[4]$	$d^2[5]$	$d^2[6]$
0	2	4	∞	∞	∞
2					
4					
∞					
∞					
∞					
min	2	-1	0	2	∞

When $k=3$



When $k=4$

d	i	$d^3(2)$	$d^3(3)$	$d^3(4)$	$d^3(5)$	$d^3(6)$
0	1	2	-1	-5	-3	8
2	2					
-1	3					
-5	4					
-3	5					
8	6					
min		2	-1	-5	-3	3

When $k=5$

When $k=6$

So the final distance matrix for each $k=2,3,4,5,6$,

0	2	4	∞	∞	∞
0	2	-1	0	2	∞
0	2	-1	-5	-3	8
0	2	-1	-5	-3	3
0	2	-1	-5	-3	3
1	2	3	4	5	6

$\Rightarrow$

Shortest distance from

1 to 1 = 0
1 to 2 = 2
1 to 3 = -1
1 to 4 = -5
1 to 5 = -3
1 to 6 = 3

The 0/1 knapsack problem

Defn: Given a knapsack (bag or container) with:

- M - capacity of the knapsack
- n - no. of objects
- w - an array consisting of weights $w_1, w_2, \ldots w_n$
- p - an array consisting of profits $p_1, p_2 \ldots p_n$
- x - an array consisting of either 0 or 1. A 0 in x_i represent that i^{th} object has not been selected & a 1 in x_i, i^{th} object has been selected.

Problem stmt: Maximize $\sum_{i=1}^{n} p_i x_i$

subject to the constraint $\sum_{i=1}^{n} w_i x_i \leq M$

Design methodology:-

Let i represent the i^{th} object, to be not to be selected to get the maximum profit with weight w_i and profit p_i with remaining capacity j.

$V[i,j]$ is the profit obtained by considering all i objects with capacity j.

There are 2 cases to be considered:

case 1: If there are no objects (i.e. $i=0$) or capacity of knapsack itself is 0 (i.e $j=0$) then profit will be 0. The total profit obtained is

$V[i,j] = 0$ if $i=0$ or $j=0$ ---- (1)

Case 2: Suppose weight of i^{th} object w_i is greater than remaining capacity j, i^{th} object cannot be selected. So profit obtained is

$$V[i,j] = V[i-1,j] \text{ if } w_i > j \quad \text{---} (2)$$

Case 3: Suppose w_i is less than remaining capacity j, there are 2 alternatives:

- i^{th} object rejected. Then profit obtained is
$$V[i,j] = V[i-1,j]$$
- i^{th} object selected. Then profit obtained is
$$V[i,j] = V[i-1, j-w_i] + P_i$$

Out of the two options, the one which yields maximum profit is considered.

So, profit obtained if $w_i <= j$ is given by

$$V[i,j] = \max\{ V[i-1,j], V[i-1, j-w_i] + P_i \} \quad \text{if } w_i <= j \quad \text{---} (3)$$

The final recurrence rel^n for knapsack problem is

$$V[i,j] = \begin{cases} 0 & \text{if } i = j = 0 \\ V[i-1,j] & \text{if } w_i > j \\ \max(V[i-1,j], V[i-1, j-w_i] + P_i) & \text{if } w_i <= j \end{cases}$$

Note: $V[n,m]$ gives the optimal profit

```
ALGORITHM Knapsack(n, m, w, p, v)
// To find the optimal soln. for the knapsack
// plom using dynamic programming.
// I/p: n - no of objects to be selected
//      m - capacity of the knapsack
//      w - weights of all objects
//      p - profits of all objects
// O/p: v - the optimal soln. for the no of objects selected
//      with specified remaining capacity

for i ← 0 to n do
    for j ← 0 to m do
        if (i=0 or j=0)
            v[i,j] = 0
        else if (w[i] > j)
            v[i,j] = v[i-1, j]
        else
            v[i,j] = max(v[i-1, j], v[i-1, j-w[i]] + p[i])
        endif
    endfor
endfor
return
```

Analysis: Parameters: n & m

$$f(n,m) = \sum_{i=0}^{n} \sum_{j=0}^{m} 1 = \sum_{i=0}^{n} m-0+1 = \sum_{i=0}^{n} m+1$$

$$= (m+1) \sum_{i=0}^{n} 1 = (m+1)(n-0+1)$$

$$= (m+1)(n+1)$$

$$= mn + m + n + 1$$

$\boxed{f(n,m) = mn}$ for very large values of m & n.

Q5a Apply bottom-up dynamic programming technique to the following instance of the knapsack problem with capacity M = 5.

Item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

Soln: No of objects, n = 4

capacity of knapsack m = 5

$w_1 = 2$, $w_2 = 1$, $w_3 = 3$, $w_4 = 2$

$P_1 = 12$ $P_2 = 10$, $P_3 = 20$, $P_4 = 15$

Step 1: Since n = 4, m = 5, the soln. matrix consists of n+1 rows & m+1 cols.

Step 2: i = 0 or j = 0. Using ①,

$V[i,j] = 0$, if $(i = 0$ or $j = 0)$

So, when i = 0, (when no objects) irrespective of remaining capacity j = 0, 1, 2, 3, 4, 5, the profit $V[i,j] = 0$,

IIIrly when j = 0 (no knapsack) the profit $V[i,j] = 0$.

The resulting table is: → M

i ↓ N=4 \ j → m	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Profit table

Ex Step 3: When $i=1$, $w_i=2$, $P_i=12$.
Compute results for $j=1,2,3,4,5$; using

$V[i,j] = V[i-1,j]$ if $w_i > j$ — (1)

$V[i,j] = \max(V[i-1,j], V[i-1,j-w_i] + P_i)$ if $w_i \le j$ — (2)

w_i	j	
2	1	$V[1,1] = V[0,1] = 0$
2	2	$V[1,2] = \max(V[0,2], V[0,0]) + 12 = \max(0, 0+12) = 12$
2	3	$V[1,3] = \max(V[0,3], V[0,1] + 12 = \max(0, 0+12) = 12$
2	4	$V[1,4] = \max(V[0,4], V[0,2] + 12 = \max(0, 0+12) = 12$
2	5	$V[1,5] = \max(V[0,5], V[0,3] + 12 = \max(0, 0+12) = 12$

When $i=2$, $w_i=1$ $P_i=10$

w_i	j	
1	1	$V[2,1] = V[1,1] = 0$
1	2	$V[2,2] = \max(V[1,2], V[1,1]+10) = \max(12, 0+10) = 12$
1	3	$V[2,3] = \max(V[1,3], V[1,2]+10 = \max(12, 12+10) = 22$
1	4	$V[2,4] = \max$
1	5	$V[2,5] =$

When $i=3$, $w_i=3$ $P_i=20$

w_i	j	
3	1	$V[3,1]$
3	2	$V[3,2]$
3	3	$V[3,3] =$
3	4	$V[3,4]$
3	5	$V[3,5] =$

When $i=4$, $w_i=2$, $P_i=15$

w_i	j	
2	1	$V[4,1] =$
2	2	$V[4,2] =$
2	3	$V[4,3] =$
2	4	$V[4,4] =$
2	5	$V[4,5] =$

$V[n,m] = V[4,5] = \underline{37}$ is the optimal profit.