1. Which asymptotic notation specifies the upper bound?

Answer: We use big-O notation for asymptotic upper bounds, since it bounds the growth of the running time from above for large enough input sizes

2. In which sorting technique at every step each element is paced in its proper position?

Answer : Bubble Sort

3. Write the best time complexity of binary search algorithm.

Answer: The best time complexity of binary search algorithm is O(1)

4. What is the time complexity of Strassen's Matrix Multiplication.

Answer: The time complexity of Strassen's Matrix Multiplication is $O(n^{2.81})$

5. Does Greedy Algorithm always guarantee an optimal solution to 0/1 Knapsack Problem?

No, It does not always lead to an optimal solution. Sometimes the non-optimal solutions are close to optimal ones. Since the Greedy approach doesn't always ensure the optimality of the solution we have to analyze for each particular problem if the algorithm leads to an optimal solution

6. Write the worst case time complexity of Quick sort algorithm.

Answer: The worst case time complexity of Quick sort algorithm is $O(n^2)$.

DESIGN & ANALYSIS OF ALGORITHMS

UNIT - IV

4.1. Introduction: Dynamic Programming

- **Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.**
- **Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used.**
- **Mostly, these algorithms are used for optimization.**
- **Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems.**
- **The solutions of sub-problems are combined in order to achieve the best solution.**

Dynamic Programming involves in 4 Major Steps

1.Characterize the structure of an optimal solution that means develop a Mathematical notation that can express any solution and sub solution for the given problem

2.Define value of optimal solution recursively.

3.By using Bottom-Up technique compute value of optimal solution. For that we have to develop a recurrence relation that relates a solution to its sub solutions, using the Mathematical notation of step 1.

4.Compute an optimal solution from computed information.

The Dynamic programming Technique was developed by a U S Mathematician "Richard Bellman" in 1950 based up on his principle "Principle of Optimality".

Principle of Optimality: States that "in an optimal sequence of decisions or choices, each sub-sequences must be optimal".

When it is not possible to apply the principle of Optimality it is almost impossible to obtain the solution using Dynamic Programming approach.

General Characteristics of the Dynamic Programming:

1. The problem can be divided into stages with a policy decision required at each stage.

2. Each stage has number of states associated with it.

3. Given the current stage an optimal policy for the remaining stagers is independent of the policy adopted.

4. The solution procedure begins by finding the optimal policy for each state of the last stage.

5. A recursive relation is available which identifies the optimal policy for each stage with n stages remaining given the optimal policy for each stage with n-1 stages remaining.

Dynamic Programming	Greedy Approach
1. It is a method in which the solution to a problem can be viewed as the result of the sequence of the decisions	1. It is the most forward technique for constructing solution to an optimum problem through a sequence of steps
2. In this method, more than one decision is made at a time	2. In this method, only one decision is made at a time
3. It considers all possible sequences in order to obtain the optimum solution	3. This considers an optimum solution with out revising the previous solutions
4. Principle of Optimality holds and thus the solution is obtained is guaranteed.	4. Solution obtained is not guaranteed.
5. It solves the sub problems Bottom-Up. The problem can not be solved until we find all solutions of sub problems	5. This method solves the sub problems from Top-Down. We first need to find the Greedy choice for a problem, then reduce it into a smaller one
6. This is more expansive than Greedy, because we have to try every possibility before solving a problem	6. Greedy method is comparatively less expansive than Dynamic Programming.
7. In this method we can solve any problem	7. There are some problems that Greedy can not solve while Dynamic Programming can. Therefore, first we try with Greedy, if it fails then we try with Dynamic Programming.

Dynamic Programming	Divide – and - Conquer
1. It is a method in which the solution to a problem can be viewed as the result of the sequence of the decisions	1. It is a method, in which solutions to a problem can be obtained by dividing it in to several sub problems, till it can't be divided further and solving recursively.
2. It solves every sub problem just once and saves the result in a table, from this solution to the problem is obtained.	2. In this every sub problem is solved and the results are combined to get original solution.
3. It works best when all sub problems are dependent, we don't know where to partition the problem.	3. It works best when all sub problems are independent.
4. It is best suited for solving problems with overlapping sub problems	4. It is best suited for the case when no overlapping sub problems or independent.
5. It is splits its input at every possible splits and after trying all possible splits, it determine which split point is optimal.	5. It splits its input at a pre specified deterministic point(Example, always at middle)

4.2. Dynamic Programming Applications

4.2.1. Matrix chain multiplication

4.2.2.Optimal binary search trees

4.2.3. 0/1 knapsack problem

4.2.4. All pairs shortest path problem

4.2.5. Travelling sales person problem

4.2.6. Reliability design

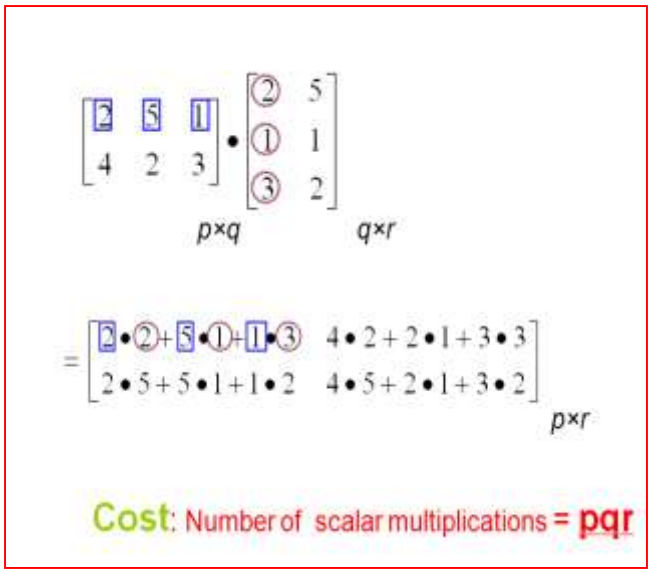
1.Matrix Chain Multiplication

A dynamic Programming approaches gives the optimal solution for this multiplication of n matrices, a series of operations are to be performed.

– Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for i=1, 2, …, n, matrix A_i has dimension $p_{i-1}Xp_i$

– Parenthesize the product $A_1A_2\ldots A_n$ such that the total number of scalar multiplications is minimized.

– Brute Force method of exhaustive search takes time exponential in n (2^n).



Example

Matrix	Dimensions
A_1	13 x 5
A_2	5 X 89
A_3	89 X 3
A_4	3 X 34

Parenthesization	Scalar multiplications
1 $((A_1 A_2) A_3) A_4$	10,582
2 $(A_1 A_2)(A_3 A_4)$	54,201
3 $(A_1 (A_2 A_3)) A_4$	2, 856
4 $A_1 ((A_2 A_3) A_4)$	4, 055
5 $A_1 (A_2 (A_3 A_4))$	26,418

1. 13 x 5 x 89 *scalar multiplications* to get (A B) 13 x 89 result
   13 x 89 x 3 *scalar multiplications* to get ((AB)C) 13 x 3 result
   13 x 3 x 34 *scalar multiplications* to get (((AB)C)D) 13 x 34

Dynamic Programming Approach

Let us use the notation $M_{i,j}$ for the cost of multiplying $A_i A_{i+1} \dots A_j$. Where cost is measured in the number of scalar multiplications.

- **Here $M_{i,i} = 0$ for all i. and $M_{1,n}$ is required solution.**
- **An optimal parenthesis of the product $A_1A_2\dots A_n$ splits the product between A_k and A_{k+1} for some integer k where $1 \le k < n$**
- **First compute matrices $A_{1..k}$ and $A_{k+1..n}$ then multiply them to get the final matrix $A_{1..n}$**
- **Key observation: parenthesis of the sub chains $A_1A_2\dots A_k$ and $A_{k+1}A_{k+2}\dots A_n$ must also be optimal if the parenthesis of the chain $A_1A_2\dots A_n$ is optimal. That is, the optimal solution to the problem contains within it the optimal solution to subproblems.**

Recursive Definition of the value of an optimal solution:

–Let m[i, j] be the minimum number of scalar multiplications necessary to compute $A_{i..j}$

–Minimum cost to compute $A_{1..n}$ is m[1, n]. Suppose the optimal parenthesis of $A_{i..j}$ splits the product between A_k and A_{k+1} for some integer k where $i \le k < j$

- $A_{i..j} = (A_i A_{i+1} \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_j) = A_{i..k} \cdot A_{k+1..j}$
- Cost of computing $A_{i..j}$ = cost of computing $A_{i..k}$ + cost of computing $A_{k+1..j}$ + cost of multiplying $A_{i..k}$ and $A_{k+1..j}$
- Cost of multiplying $A_{i..k}$ and $A_{k+1..j}$ is $p_{i-1} \cdot p_k \cdot p_j$
- $M[i, j] = M[i, k] + M[k+1, j] + P_{i-1} \cdot P_k \cdot P_j$ for $i \leq k < j$
- But… optimal parenthesis occurs at one value of k among all possible $i \leq k < j$

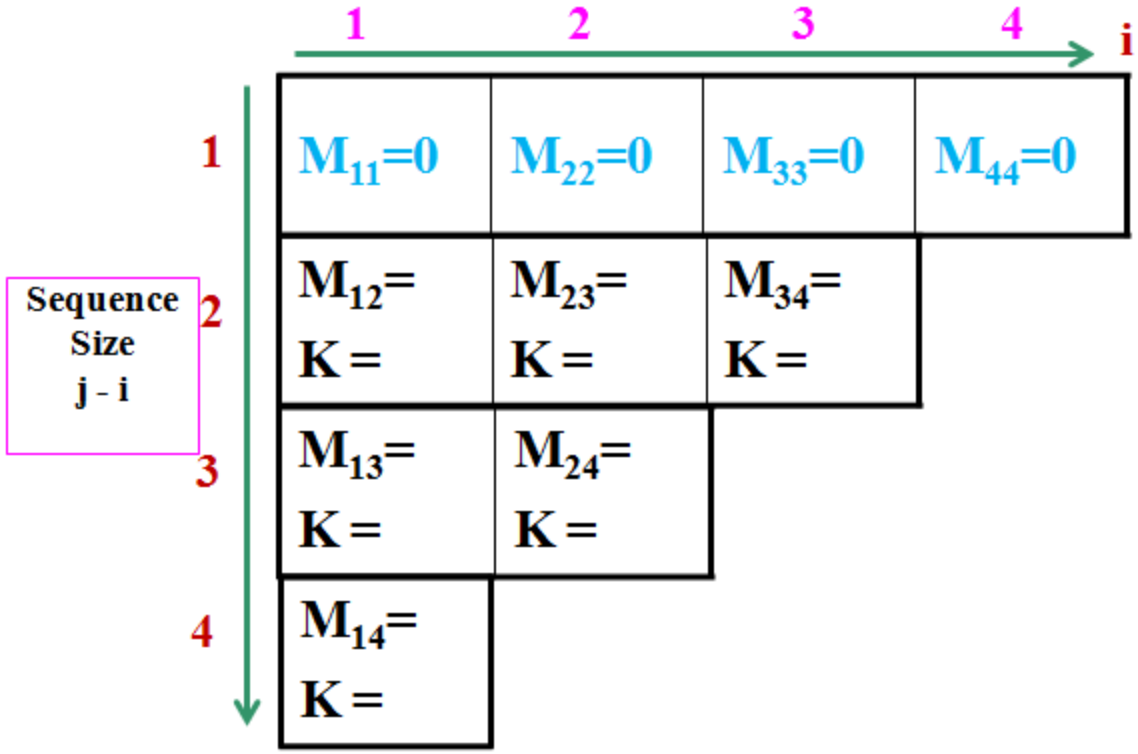
$$M[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \leq k < j} \{ M[i, k] + M[k+1, j] + P_{i-1} . P_k . P_j \} & \text{if } i < j \end{cases}$$

- Check all these and select the best one.
- To keep track of how to construct an optimal solution, we use a table S
- S[i, j] = value of k at which $A_i A_{i+1} \ldots A_j$ is split for optimal parenthesis.

Example : Consider $[A_1]_{5\times4}$ $[A_2]_{4\times6}$ $[A_3]_{6\times2}$ $[A_4]_{2\times7}$ and $P_0=5$, $p_1=4$, $p_2=6$, $p_3=2$, $p_4=7$

Since $M_{i,i} = 0$ for all i.

$M_{11}= 0, M_{22}= 0, M_{33}= 0, M_{44}= 0$



The computations are

$P_0=5, p_1=4, p_2=6, p_3=2, p_4=7$

Now we will compute $M_{i\,j}$

$M_{i\,j} = M[i, k] + M[k+1, j] + P_{i-1} \cdot P_k \cdot P_j$ for $i \le k < j$

Let i=1, j=2 and k=1

$M_{12} = M_{11} + M_{22} + P_0 \cdot P_1 \cdot P_2$

$M_{12} = 0 + 0 + 5 \cdot 4 \cdot 6$

$M_{12} = 120$

Let i=2, j=3 and k=2

$M_{23} = M_{22} + M_{33} + P_1 \cdot P_2 \cdot P_3$

$M_{23} = 0 + 0 + 4 \cdot 6 \cdot 2$

$M_{23} = 48$

Let i=3, j=4 and k=3

$M_{34} = M_{33} + M_{44} + P_2 \cdot P_3 \cdot P_4$

$M_{34} = 0 + 0 + 6 \cdot 2 \cdot 7$

$M_{34} = 84$

Sequence Size j - i	1	2	3	4
1	$M_{11}=0$	$M_{22}=0$	$M_{33}=0$	$M_{44}=0$
2	$M_{12}=120$ K = 1	$M_{23}=48$ K = 2	$M_{34}=84$ K = 3	
3	$M_{13}=$ K =	$M_{24}=$ K =		
4	$M_{14}=$ K =			

i

Now we will compute M_{ij}

$P_0=5, p_1=4, p_2=6, p_3=2, p_4=7$

$M_{ij} = M[i, k] + M[k+1, j] + P_{i-1} . P_k . P_j$ for $i \leq k < j$

Let i=1, j=4 and k=1 or 2 or 3

Let i=1, j=4 and k=1

$M_{14} = M_{11} + M_{24} + P_0 . P_1 . P_4$

$M_{14} = 0 + 104 + 5 . 4 . 7$

$M_{14} = 244$

Let i=1, j=4 and k=2

$M_{14} = M_{12} + M_{34} + P_0 . P_2 . P_4$

$M_{14} = 120 + 84 + 5 . 6 . 7$

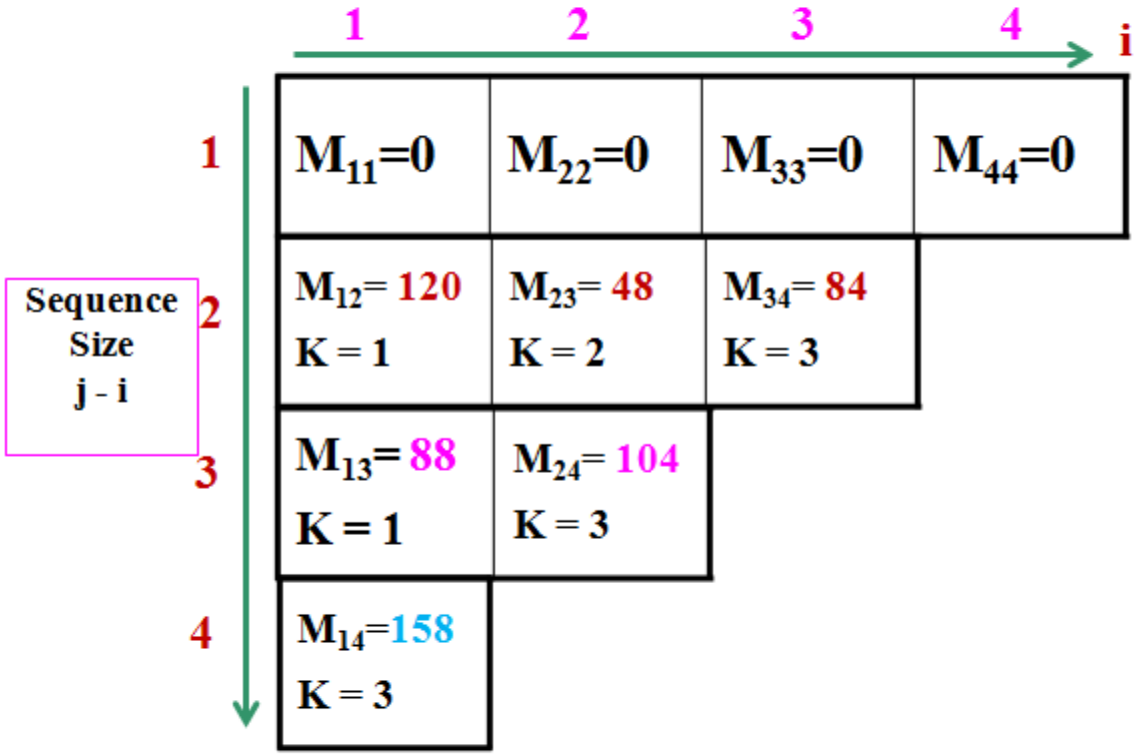
$M_{14} = 414$

Let i=1, j=4 and k=3

$M_{14} = M_{13} + M_{44} + P_0 . P_3 . P_4$

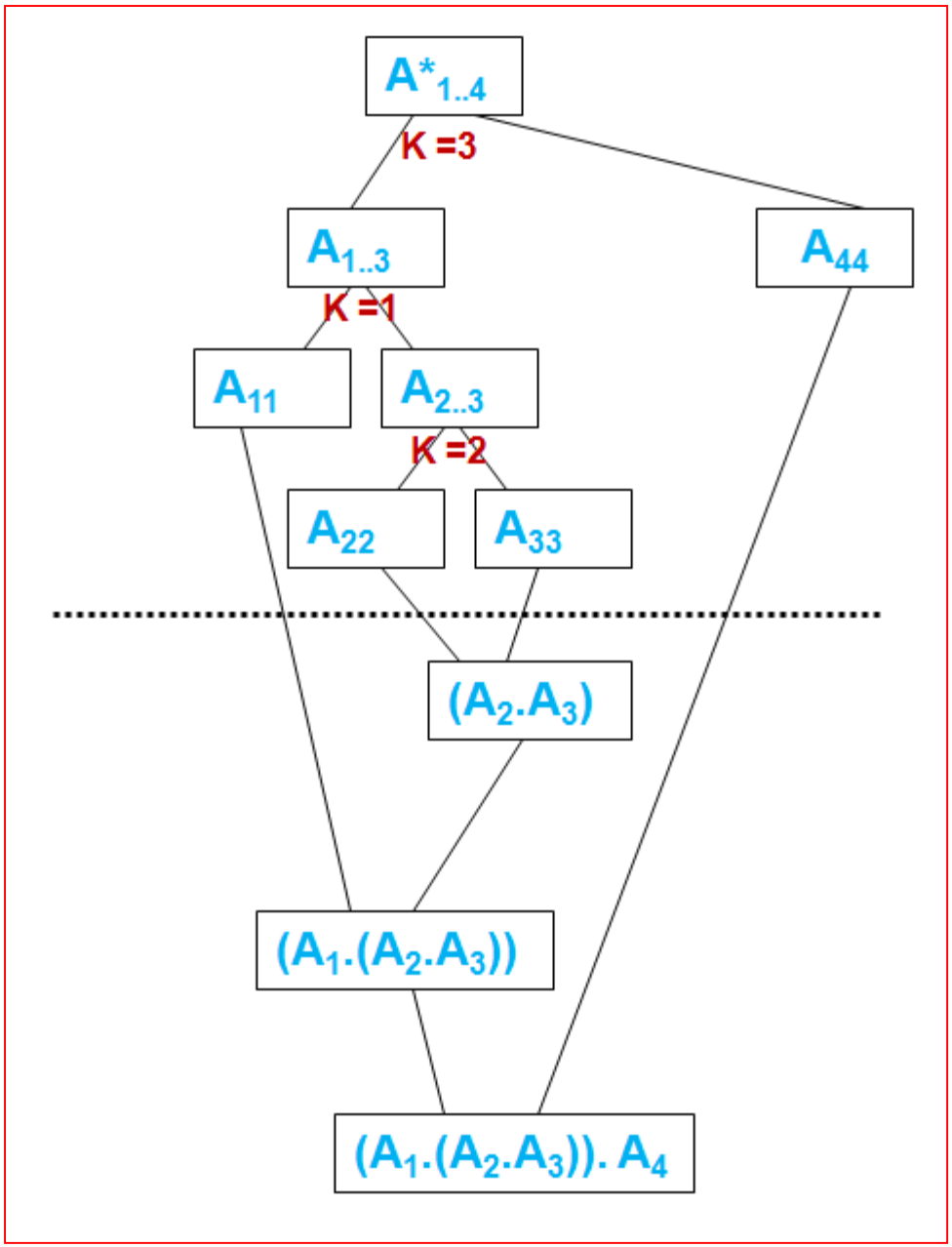
$M_{14} = 88 + 0 + 5 . 2 . 7$

$M_{14} = 158$



As we get minimum of M_{14} when k=3.

We will set k=3 and M_{24} = 158.



There fore the sequence is

((A_1 (A_2A_3))A_4)

Input: Array P[1…n] containing matrix dimensions.

Result: Minimum-cost table M

```
Algorithm  Matrix_Chain_Mul(P[1,n])
{
   for i:= 1 to n  do
      M[i, i]:= 0 ;
    for len:= 2 to n  do        // for lengths   2,3 and so on
    {
        for i:= 1 to ( n-len+1 ) do
        {
                    j:= i+len-1;
                    M[i, j]:= ∞ ;
                    for k:=i to j-1  do
                    {
                     q:= M[i, k] + M[k+1, j] + P[i-1] P[k] P[j];
                     if  (q < M[i, j]) then
                     {
                         M[i, j]:= q;
                          S[i, j]:= k;
                     }
                }
            }
        }
    return M[1,n];
}
```

Time complexity of Matrix Chain Multiplication algorithm is $O(n^3)$

4.2.2. Optimal Binary Search Trees (OBST)

A Binary Search Tree T is a binary tree, either it is empty or each node in the tree contains an identifier and,

- **All identifiers in the left subtree of T are less than the identifier in the root node T.**
- **All identifiers in the right subtree are greater than the identifier in the root node T.**
- **The left and right subtree of T are also binary search trees.**

If we use c(i,j) to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1},\ldots\ldots,a_j$ and $E_i,\ldots,E_j$, then

cost(l)=c(0,k-1), and cost(r)=c(k,n).

For the tree to be optimal, we must choose k such that p(k) + c(0,k-1) + c(k,n) + w(0,k-1) + w(k,n) is minimum.

Hence, for c(0,n) we obtain

$$c(0,n) = \min_{0<k<n} \{ c(0,k-1) + c(k,n) + p(k) + w(0,k-1) + w(k,n) \}$$

Example 1: using algorithm OBST compute w(i, j),r(i, j) and c(i, j), $0 \leq i \leq 4$ for the identifier set (a_1,a_2,a_3,a_4)=(and, goto, print, stop). Let p(1:4) = (3,3,1,1) and q(0:4)=(2,3,1,1,1). Using r(i, j) construct the optimal binary search tree.

Solution: Given that,

the identifier set (a_1,a_2,a_3,a_4)=(and, goto, print, stop)

$p_1=3$, $p_2=3$, $p_3=1$, $p_4=1$

$q_0=2$, $q_1=3$, $q_2=1$, $q_3=1$, $q_4=1$

Initially

c(i, i) = 0.

r(i, i) = 0.

w(i, i) = q(i).

$c(0,0) = 0$

$c(1,1) = 0$

$c(2,2) = 0$

$c(3,3) = 0$

$c(4,4) = 0$

$r(0,0) = 0$

$r(1,1) = 0$

$r(2,2) = 0$

$r(3,3) = 0$

$r(4,4) = 0$

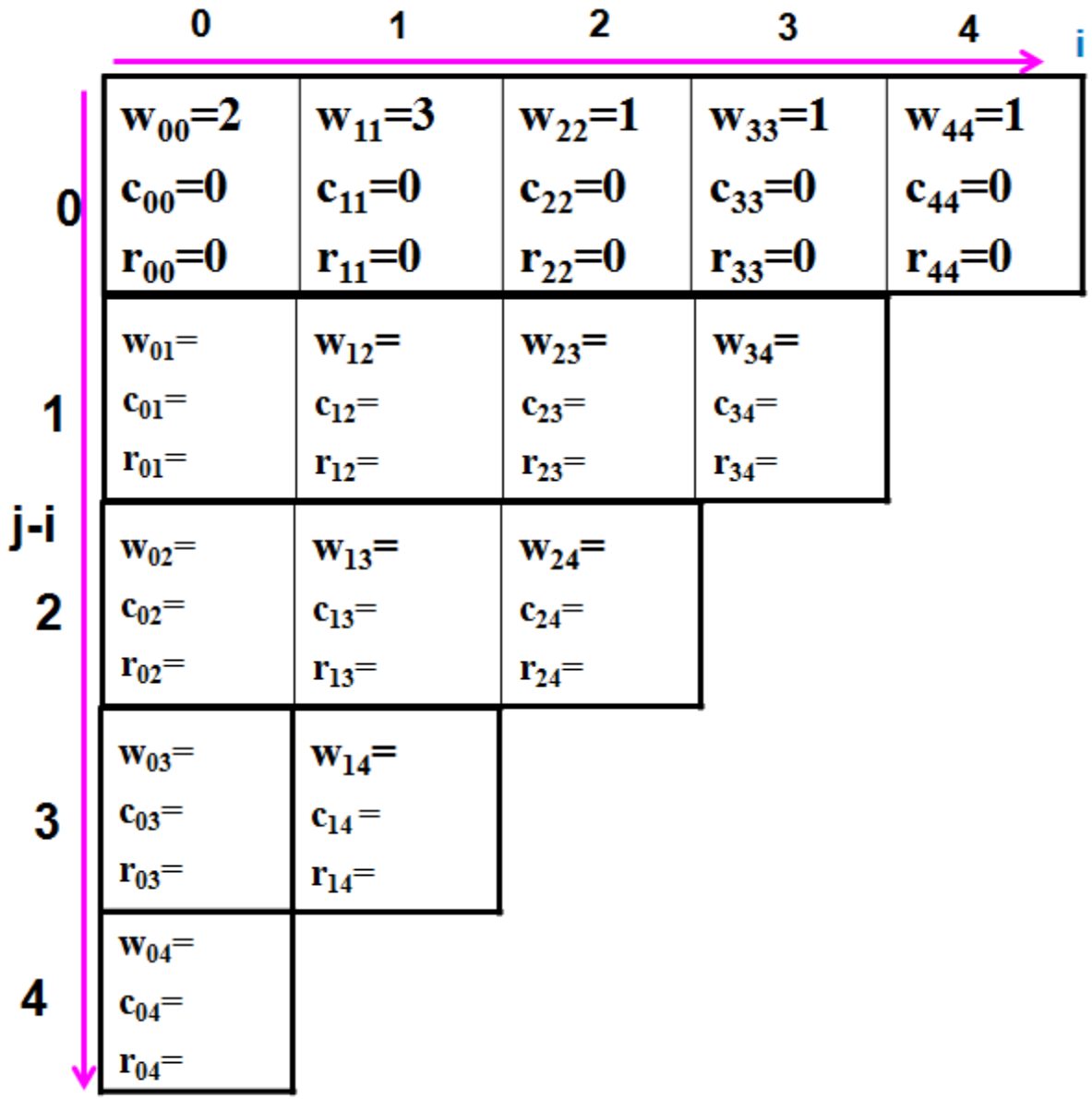
$w(0,0) = q(0) = 2$

$w(1,1) = q(1) = 3$

$w(2,2) = q(2) = 1$

$w(3,3) = q(3) = 1$

$w(4,4) = q(4) = 1$



Now,

$c(i, j) = \min_{i < k \le j} \{ c(i, k-1) + c(k, j) + w(i, j) \}$

$p_1=3, p_2=3, p_3=1, p_4=1$

$q_0=2, q_1=3, q_2=1, q_3=1, q_4=1$

$w(i, j) = p(j) + q(j) + w(i, j-1)$

$r(i, j) = k,\ i < k \le j$ (k chosen such that where cost is minimum)

For j – i = 1:

$w(0,1) = p(1) + q(1) + w(0, 0)$

$= 3 + 3 + 2$

$= 8$

$w_{01} = 8$

$c_{01} = 8$

$r_{01} = 1$

$c(0, 1) = \min_{0 < k \le 1} \{ c(0, 0) + c(1, 1) + w(0, 1) \}$ (Since k=1)

$= c(0, 0) + c(1, 1) + w(0, 1)$

$= 0 + 0 + 8$

$= 8$

$r(0, 1) = k,\ i < k \le j$ (k chosen such that where cost is minimum)

$= 1$

Now,

$c(i, j) = \min_{i < k \le j} \{ c(i, k-1) + c(k, j) \} + w(i, j)$

$p_1=3, p_2=3, p_3=1, p_4=1$

$q_0=2, q_1=3, q_2=1, q_3=1, q_4=1$

$w(i, j) = p(j) + q(j) + w(i, j-1)$

$r(i, j) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

For j – i = 1:

$w(1, 2) = p(2) + q(2) + w(1, 1)$

$= 3 + 1 + 3$

$= 7$

$w_{12} = 7$

$c_{12} = 7$

$r_{12} = 2$

$c(1, 2) = \min_{1 < k \le 2} \{ c(1, 1) + c(2, 2) \} + w(1, 2)$ (Since k = 2)

$= c(1, 1) + c(2, 2) + w(1, 2)$

$= 0 + 0 + 7$

$= 7$

$r(1, 2) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

$= 2$

Now,

$c(i, j) = \min_{i < k \le j} \{ c(i, k-1) + c(k, j) \} + w(i, j)$

$p_1=3, p_2=3, p_3=1, p_4=1$

$q_0=2, q_1=3, q_2=1, q_3=1, q_4=1$

$w(i, j) = p(j) + q(j) + w(i, j-1)$

$r(i, j) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

For j – i = 1:

$w(2, 3) = p(3) + q(3) + w(2, 2)$

$= 1 + 1 + 1$

$= 3$

$w_{23} = 3$

$c_{23} = 3$

$r_{23} = 3$

$c(2, 3) = \min_{2 < k \le 3} \{ c(2, 2) + c(3, 3) \} + w(2, 3)$ (Since k = 3)

$= c(2, 2) + c(3, 3) + w(2, 3)$

$= 0 + 0 + 3$

$= 3$

$r(2, 3) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

$= 3$

Now,

$c(i, j) = \min_{i < k \le j} \{ c(i, k-1) + c(k, j) + w(i, j) \}$

$p_1=3, p_2=3, p_3=1, p_4=1$

$q_0=2, q_1=3, q_2=1, q_3=1, q_4=1$

$w(i, j) = p(j) + q(j) + w(i, j-1)$

$r(i, j) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

For j – i = 1:

$w(3, 4) = p(4) + q(4) + w(3, 3)$

$= 1 + 1 + 1$

$= 3$

$w_{34} = 3$

$c_{34} = 3$

$r_{34} = 4$

$c(3, 4) = \min_{3 < k \le 4} \{ c(3, 3) + c(4, 4) \} + w(3, 4)$ (Since k = 4)

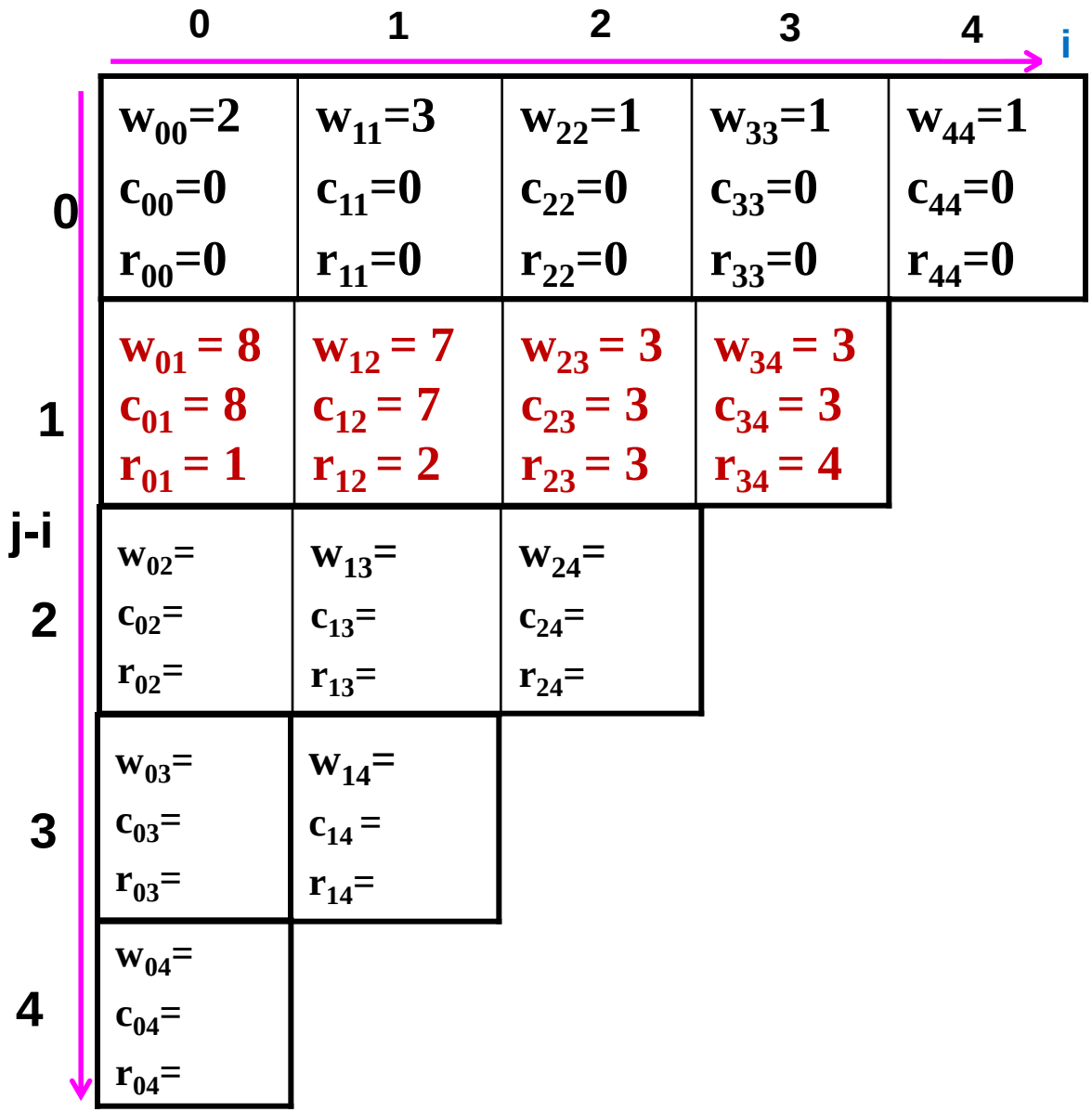
$= c(3, 3) + c(4, 4) + w(3, 4)$

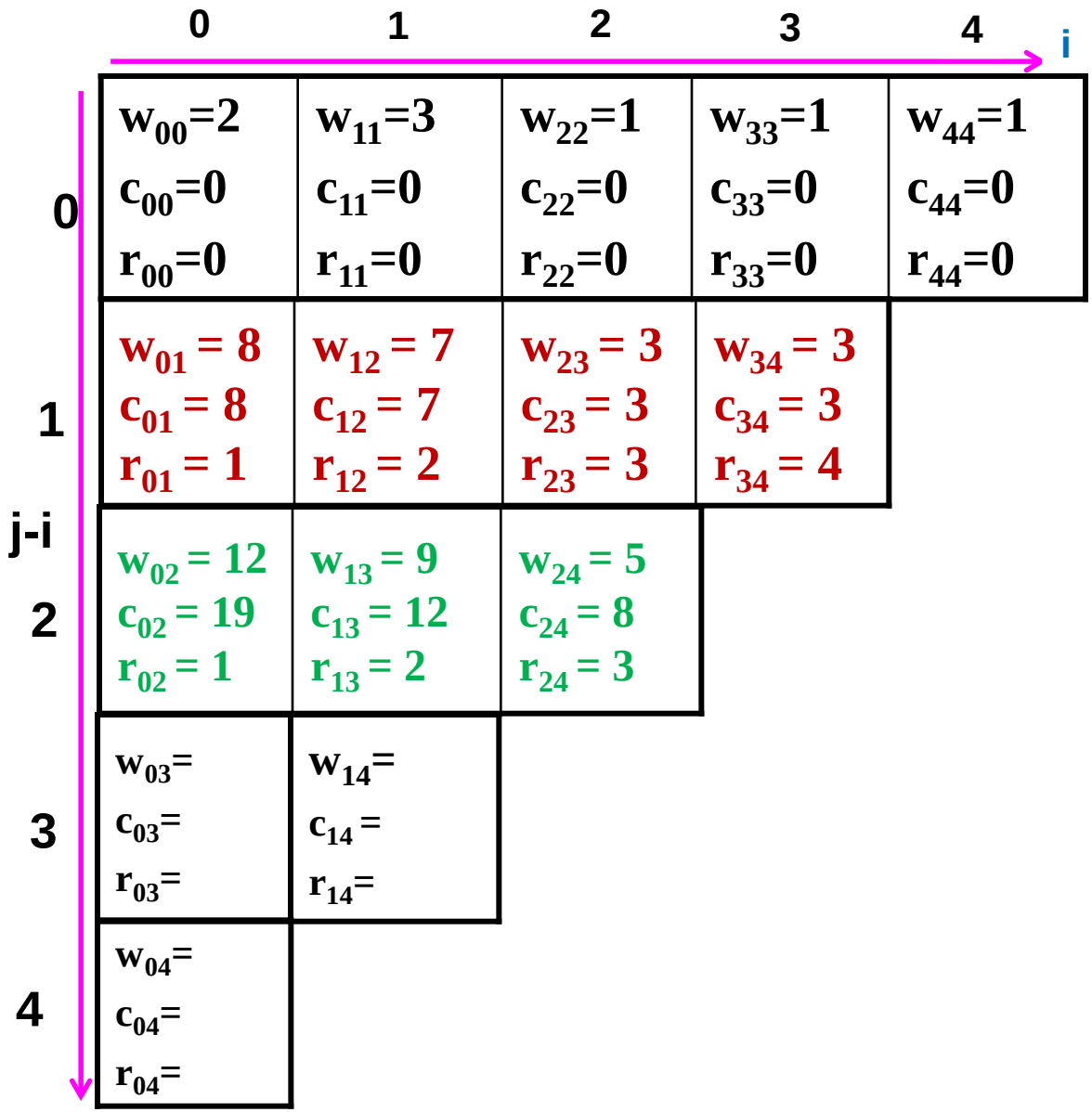
$= 0 + 0 + 3$

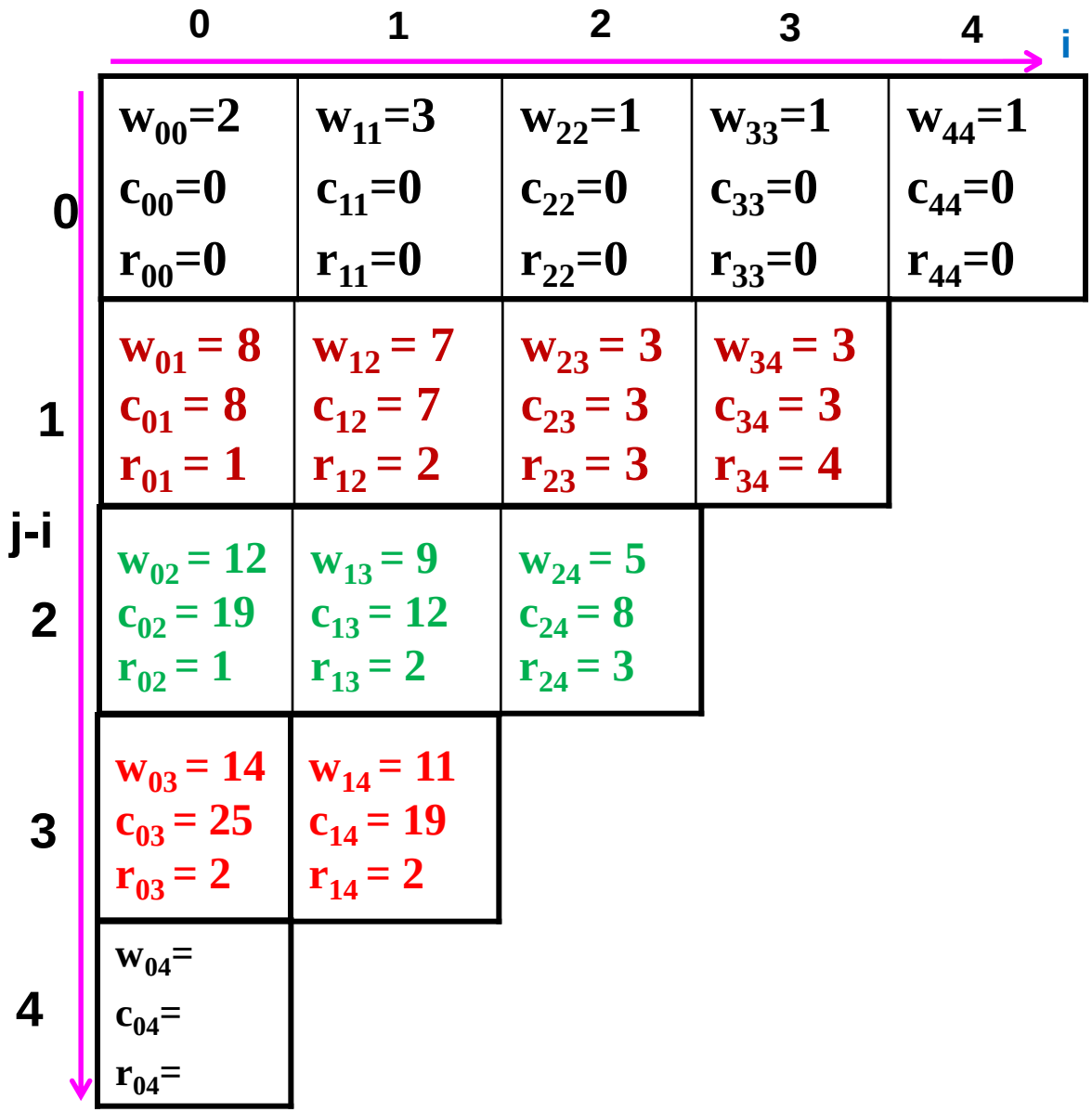
$= 3$

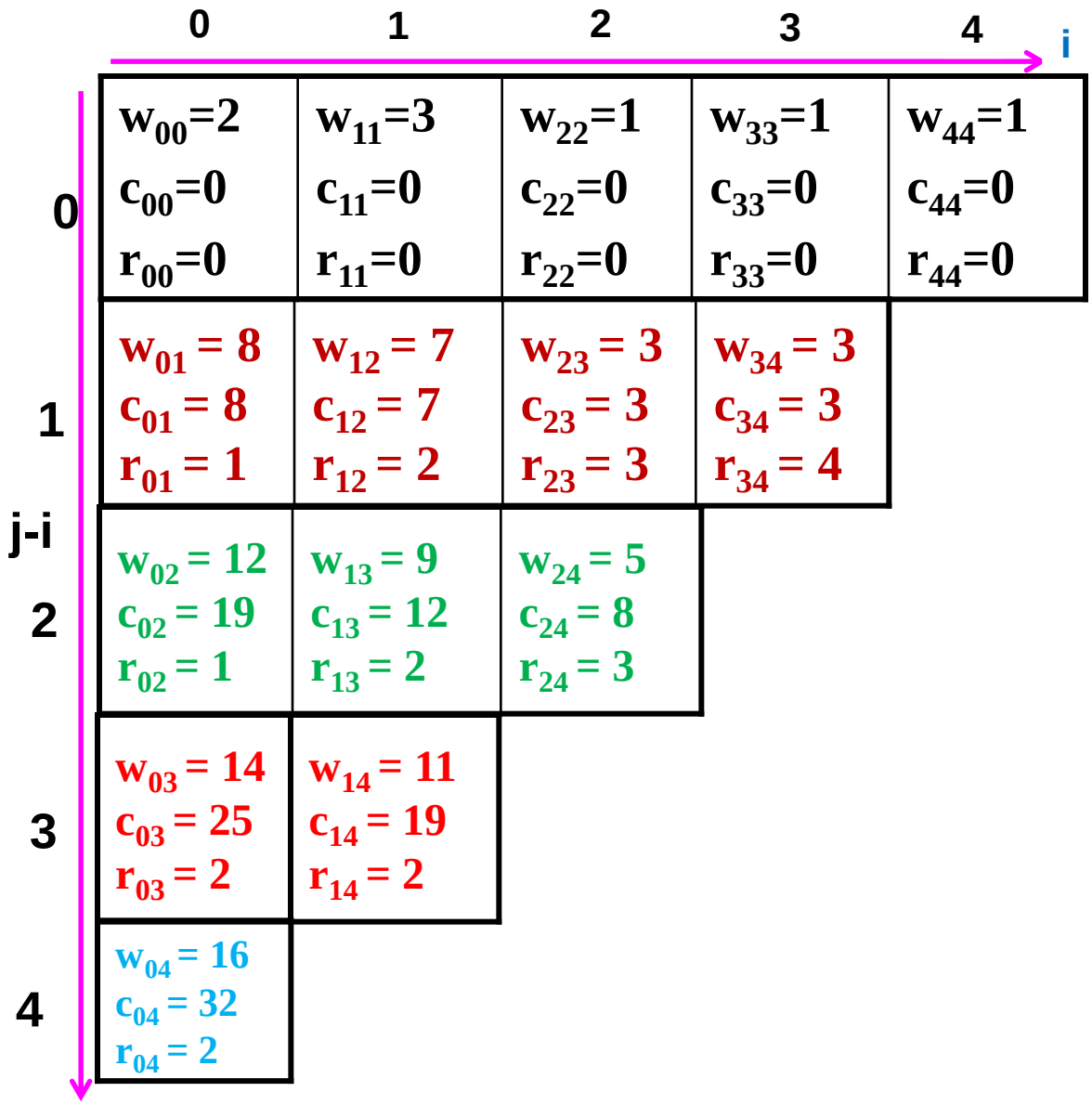
$r(3, 4) = k$, $i < k \le j$ (k chosen such that where cost is minimum)

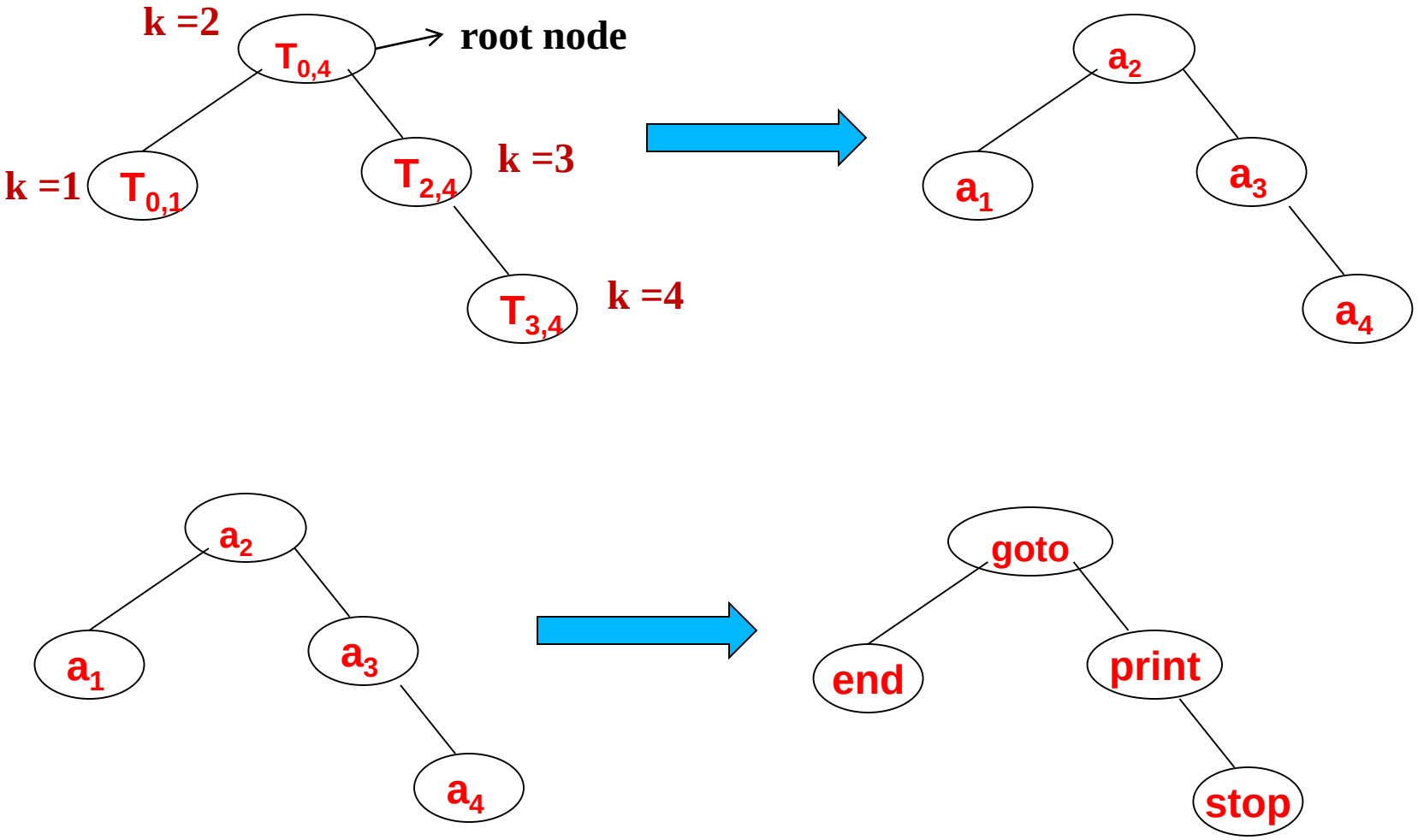
$= 4$


0
1
2
3
4
i
j-i
0
1
2
3
4
$w_{00}=2$ $c_{00}=0$ $r_{00}=0$
$w_{11}=3$ $c_{11}=0$ $r_{11}=0$
$w_{22}=1$ $c_{22}=0$ $r_{22}=0$
$w_{33}=1$ $c_{33}=0$ $r_{33}=0$
$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
$w_{01}=8$ $c_{01}=8$ $r_{01}=1$
$w_{12}=7$ $c_{12}=7$ $r_{12}=2$
$w_{23}=3$ $c_{23}=3$ $r_{23}=3$
$w_{34}=3$ $c_{34}=3$ $r_{34}=4$
$w_{02}=$ $c_{02}=$ $r_{02}=$
$w_{13}=$ $c_{13}=$ $r_{13}=$
$w_{24}=$ $c_{24}=$ $r_{24}=$
$w_{03}=$ $c_{03}=$ $r_{03}=$
$w_{14}=$ $c_{14}=$ $r_{14}=$
$w_{04}=$ $c_{04}=$ $r_{04}=$


0
1
2
3
4
i
j-i
0
1
2
3
4
$w_{00}=2$ $c_{00}=0$ $r_{00}=0$
$w_{11}=3$ $c_{11}=0$ $r_{11}=0$
$w_{22}=1$ $c_{22}=0$ $r_{22}=0$
$w_{33}=1$ $c_{33}=0$ $r_{33}=0$
$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
$w_{01}=8$ $c_{01}=8$ $r_{01}=1$
$w_{12}=7$ $c_{12}=7$ $r_{12}=2$
$w_{23}=3$ $c_{23}=3$ $r_{23}=3$
$w_{34}=3$ $c_{34}=3$ $r_{34}=4$
$w_{02}=12$ $c_{02}=19$ $r_{02}=1$
$w_{13}=9$ $c_{13}=12$ $r_{13}=2$
$w_{24}=5$ $c_{24}=8$ $r_{24}=3$
$w_{03}=$ $c_{03}=$ $r_{03}=$
$w_{14}=$ $c_{14}=$ $r_{14}=$
$w_{04}=$ $c_{04}=$ $r_{04}=$


0
1
2
3
4
i
j-i
0
1
2
3
4
$w_{00}=2$ $c_{00}=0$ $r_{00}=0$
$w_{11}=3$ $c_{11}=0$ $r_{11}=0$
$w_{22}=1$ $c_{22}=0$ $r_{22}=0$
$w_{33}=1$ $c_{33}=0$ $r_{33}=0$
$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
$w_{01}=8$ $c_{01}=8$ $r_{01}=1$
$w_{12}=7$ $c_{12}=7$ $r_{12}=2$
$w_{23}=3$ $c_{23}=3$ $r_{23}=3$
$w_{34}=3$ $c_{34}=3$ $r_{34}=4$
$w_{02}=12$ $c_{02}=19$ $r_{02}=1$
$w_{13}=9$ $c_{13}=12$ $r_{13}=2$
$w_{24}=5$ $c_{24}=8$ $r_{24}=3$
$w_{03}=14$ $c_{03}=25$ $r_{03}=2$
$w_{14}=11$ $c_{14}=19$ $r_{14}=2$
$w_{04}=$ $c_{04}=$ $r_{04}=$


0
1
2
3
4
i
j-i
0
1
2
3
4
$w_{00}=2$ $c_{00}=0$ $r_{00}=0$
$w_{11}=3$ $c_{11}=0$ $r_{11}=0$
$w_{22}=1$ $c_{22}=0$ $r_{22}=0$
$w_{33}=1$ $c_{33}=0$ $r_{33}=0$
$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
$w_{01}=8$ $c_{01}=8$ $r_{01}=1$
$w_{12}=7$ $c_{12}=7$ $r_{12}=2$
$w_{23}=3$ $c_{23}=3$ $r_{23}=3$
$w_{34}=3$ $c_{34}=3$ $r_{34}=4$
$w_{02}=12$ $c_{02}=19$ $r_{02}=1$
$w_{13}=9$ $c_{13}=12$ $r_{13}=2$
$w_{24}=5$ $c_{24}=8$ $r_{24}=3$
$w_{03}=14$ $c_{03}=25$ $r_{03}=2$
$w_{14}=11$ $c_{14}=19$ $r_{14}=2$
$w_{04}=16$ $c_{04}=32$ $r_{04}=2$



Thus the cost of OBST is 32 and the root is a_2.

```
Algorithm OBST(p,q,n)
{
    for i:= 0 to n-1  do
     {          //  initialize.
        w[i, i] :=q[i];
         r[i, i] :=0;
         c[i, i]=0;     //  Optimal trees  with one node.
        w[i, i+1]:= p[i+1] + q[i+1] + q[i] ;
         c[i, i+1]:= p[i+1] + q[i+1] + q[i] ;
         r[i, i+1]:= i + 1;
     }
     w[n, n] :=q[n];
     r[n, n] :=0;
     c[n, n]=0; // Find optimal trees with m nodes.
```

```
for m:= 2 to n do
    {
        for i := 0 to n – m do
         {
            j:= i + m ;
            //Solve using Knuth's result
            w[i,j]:=p[j]+q[j]+w[i,j-1];
            x := Find( c, r, i, j );
            c[i, j] := w[i, j] + c[i, x-1] + c[x, j];
            r[ i, j ] :=x;
           }
     }
```

```
Algorithm  Find( c, r, i, j )
{
   for k := r[i,j-1] to r[i+1, j] do
    {
        min :=∞;
        if ( c[i,k-1]+c[k,j]<min) then
       {
            min := c[i, k-1] + c[k, j];
            y:= k;
       }
   }
   return y;
}
```

0/1 Knapsack Problem

In this, the thief can't place fraction of ornament powders in the bag. That is either he can place the ornament powders completely in the bag or he can’t place ornament powder. So, $x_i = 1$ when item i is selected or $x_i = 0$ when item i is not selected.

This problem contains either 0 or 1, that's why this problem is called as 0/1 Knapsack Problem.

Sequence Of Decisions are

- **Decide the x_i values in the order $x_1, x_2, x_3, \ldots, x_n$.**

OR

- **Decide the x_i values in the order $x_n, x_{n-1}, x_{n-2}, \ldots, x_1$.**

All profits and weights are positive.

Example: Solve 0/1 Knapsack instance n=3, $(w_1,w_2,w_3) = (2,3,4)$, $(p_1,p_2,p_3)=(1,2,5)$ and M=6.

Solution:

Let us build the sequence of decisions S^0,S^1,S^2,S^3.

Initially, $s^0 = \{(0, 0)\}$

$$S_1^{\,i} = S^{i-1} + \{p_i,w_i\}$$

$S_1^{\,1} = S^0 + \{p_1, w_1\}$ **Addition**

$= \{(0,0)\} + (1,2)$

$= \{(0 + 1, 0 + 2)\}$

$=\{(1,2)\}$

$$S^i = S^{i-1} + S_1^{\,i}$$

$S^1 = S^0 + S_1^{\,1}$ **Merging**

$= \{(0,0)\} + \{(1,2)\}$

$= \{(0,0), (1,2)\}$

$S_1^2 = S^1 + \{ p_2, w_2 \}$ (Since here + means Addition)

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

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

$= \{(2, 3), (3, 5)\}$

$S^2 = S^1 + S_1^2$ (Since here + means Merging)

$= \{(0, 0), (1, 2)\} + \{(2, 3), (3, 5)\}$

$= \{(0,0), (1,2),(2,3), (3,5)\}$

$S_1^3 = S^2 + \{p_3,w_3\}$ (Since here + means Addition)

$= \{(0,0), (1,2),(2,3), (3,5)\} + \{(5,4)\}$

$= \{(0 + 5, 0 + 4), (1 + 5, 2 + 4), (2 + 5, 3 + 4), (3 + 5, 5 + 4)\}$

$= \{(5,4), (6,6), (7,7),(8,9)\}$

$S^3 = S^2 + S_1^3$ (Since here + means Merging)

$= \{(0,0), (1,2),(2,3), (3,5)\} + \{(5,4), (6,6), (7,7),(8,9)\}$

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

Purging Rule or Dominance Rule:

If one of S^{i-1} and S_1^i has a pair (p_j,w_j) and other has pair (p_k,w_k) and $p_j \le p_k$ while $w_j \ge w_k$ then the pair (p_j,w_j) is discarded.

Now Apply the Purging Rule or Dominance Rule on S^3

There fore S^3 = {(0,0), (1,2),(2,3), (3,5), (5,4), (6,6), (7,7),(8,9)}

In this problem $(3,5) \in S^2$ and $(5,4) \in S_1^3$

According to purging rule,

$p_j = 3$, $w_j = 5$, $p_k = 5$, $w_k = 4$

$3 \le 5$ ($p_j \le p_k$) and $5 \ge 4$ ($w_j \ge w_k$) then

The pair (p_j,w_j) (that is (3,5)) is discarded.

There fore S^3 = {(0,0), (1,2),(2,3), (5,4), (6,6), (7,7),(8,9)}

The bag size M=6. So, discard (7,7) and (8,9)

There fore S^3 = {(0,0), (1,2),(2,3), (5,4), (6,6)}

After applying the Purging Rule, we will check the following conditions in order to find solution.

If $((p_j,w_j) \in S^n$ and $(p_j,w_j) \notin S^{n-1})$then

$x_n = 1$

else

$x_n = 0$

As M=6,

If $((6,6) \in S^3$ and $(6,6) \notin S^2)$ then condition becomes true

$x_3 = 1$,

that is 3rd item is placed entirely in the bag.

To get next item to be placed in the bag, subtract (p_3,w_3) from (6,6).

Now $(6 - p_3, 6 - w_3) = (6 - 5, 6 - 4)$

$= (1, 2)$

If (The pair (1,2) $\in S^2$ and (1,2) $\notin S^1$)then condition becomes fail

$x_2 = 0$,

that is 2nd item is not placed in the bag.

If (The pair (1,2) $\in S^1$ and (1,2) $\notin S^0$) then condition becomes true

$x_1 = 1$,

that is 1st item is completely placed in the bag.

There fore $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$.

Maximum Profit is $\Sigma\ p_i . x_i$

$$= p_1.x_1 + p_2.x_2 + p_3.x_3$$

$$= 1 \times 1 + 2 \times 0 + 5 \times 1 = 1 + 0 + 5 = 6$$

There fore, the optimal solution is $(x_1, x_2, x_3) = (1, 0, 1)$.

There fore, the maximum profit is $\Sigma\ p_i . x_i = 6$

Time complexity of Knapsack Problem is O(n).

Time complexity of Knapsack Problem is $O(2^{n/2})$

Algorithm DKP(p, w, n, m)

{

S^0 := {(0,0)};

for i: = 1 to n-1 do

{

S^{i-1}:= {(p, w)/(p - p_i, w-w_i) ∈ S^{i-1} and w ≤ m};

S^i := MergePurge(S^{i-1}, S_1^{i-1});

}

(p_x, w_x):= last pair in S^{n-1};

(p_y, w_y):= ($p^|$ + p_n, $w^|$ + w_n) where w^1 is the largest weight w in any pair in s^{n-1} such that w+w_n≤m;

// Trace back for xn,xn-1,……

if(p_x>p_y) then

x_n=0;

else

x_n=1;

TracebackFor(x_{n-1},x_{n-2},…….x_1);

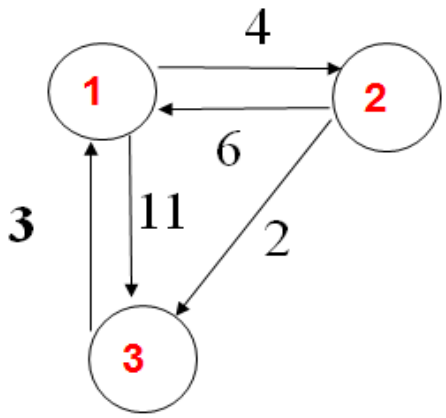
}

4.2.4. All pairs shortest path problem (Floyd's Algorithm)

- Let G=(V, E) be a directed graph consisting of n vertices and each edge is associated with weight.
- Let Cost be a cost adjacency matrix for G such that cost(i,i)=0 1<i<n. Then cost of (i,j) isthe length (or cost) of edge<i,j>. If <i,j>E(G) and cost(i,j) = if i=j and <i,j> E(G).
- The All Pairs Shortest Path problem is to determine a matrix A such that A(i,j) is the length of the shortest path from vertex i to vertex j.
- The problem of finding the shortest path between all pairs of vertices in a graph is called "**All Pairs Shortest Path Problem**". This problem can be solved by using Dynamic Programming technique.
- Assume that this path contains no cycles. If k is an intermediate vertex on this path, then the sub path from i to k and from k to j are the shortest paths from i to k and k to j respectively other wise the path is not shortest path.

$$A^k(i,j) = \begin{cases} w(i,j) & \text{if } k = 0 \quad \text{(no intermediate vertices at all)} \\ \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\} & \text{if } k \geq 1 \end{cases}$$

Example: The Graph is



Solution: The cost Adjacency Matrix is $A^0(i, j) = w(i, j) =$

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$$A^k(i, j) = \begin{cases} w(i, j) & \text{if } k = 0 \text{ (no intermediate vertices at all)} \\ \min \{ A^{k-1}(i, j),\ A^{k-1}(i, k) + A^{k-1}(k, j) \} & \text{if } k \geq 1 \end{cases}$$

For k = 1, that is , going from i to j through intermediate vertex '1'.

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

Means that i = 1, j = 1 or 2 or 3 and k = 1

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

$= \min\{0, 0 + 0\}$

$= 0$

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

$= \min\{4, 0 + 4\}$

$= 4$

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

$= \min\{11, 0 + 11\}$

$= 11.$

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

For k = 1, that is , going from i to j through intermediate vertex '1'.

i = 2:

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

Means that i = 2, j = 1 or 2 or 3 and k = 1

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

$= \min\{6, 6 + 0\}$

$= 6$

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

$= \min\{0, 6 + 4\}$

$= 0$

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

$= \min\{2, 6 + 11\}$

$= 2.$

	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

For k = 1, that is , going from i to j through intermediate vertex '1'.

i = 3: $\min\{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\}$ if $k \geq 1$

Means that i = 3, j = 1 or 2 or 3 and k = 1

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

$= \min\{3, 3 + 0\}$

$= 3$

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

$= \min\{\infty, 3 + 4\}$

$= 7$

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

$= \min\{0, 3 + 11\}$

$= 0.$

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

$\therefore$ $A^1(i,j) =$

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

For k = 3, that is , going from i to j through intermediate vertex '3'.

i = 3: $\min\{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\}$ if $k \geq 1$

Means that i = 3, j = 1 or 2 or 3 and k = 3

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

$= \min\{3,\ 0 + 3\}$

$= 3$

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

$= \min\{7,\ 0 + 7\}$

$= 7$

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

$= \min\{0,\ 0 + 0\}$

$= 0$

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

$\therefore$ $A^3(i,j) =$

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

This gives the All Pairs Shortest Path Solution

```
Algorithm All_Pairs_Shortest_ Path( w, A, n )
// w[1:n,1:n] weighted array matrix, n is the number of vertices, A[i,j] is the cost of shortest path from i to j
{
    for i := 1 to n do
      for j := 1 to n do
          A[i, j] := w[i, j] ;     // copy w 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]);
}
```

Time Complexity for All Pairs Shortest Path Problem is $O(n^3)$

4.2.5. Travelling Sales Person Problem (TSPP)

Let G = (V, E) be a directed graph with edge costs $C_{i,j}$.

The variable $C_{i,j}$ is defined such that $C_{i,j} > 0$ for all i and j $C_{i,j} = \infty$ if $\langle i,j \rangle \notin E$. let |V| = n and assume n>1.

The cost of a tour is the sum of the costs of the edges on the tour.

The Travelling Sales Person Problem is to find a tour of minimum cost.

Suppose we have route a postal van to pick up mail from, mail boxes located at n different sites.

An n+1 vertex graph can be used to represent the situation.

One vertex represents the post office from which the postal van starts and to which it must return.

Edge $\langle i, j \rangle$ is assigned a cost equal to the distance from site i to site j.

The route taken by the postal van is a tour and it should have minimum length.

Clearly,

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

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

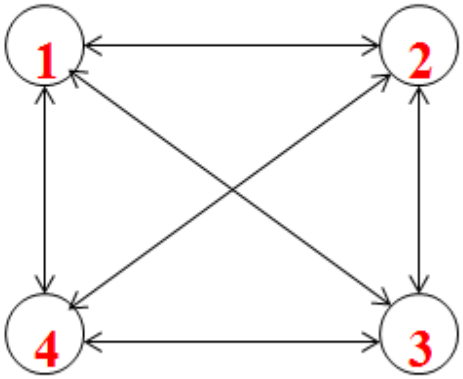
Let g(i,S) be the length of a shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1.

$\min_{j \in S}$ **is considered as the intermediate node g(j, S - { j }) means j is already visited. So, next we have to traverse s - { j } with j as starting point.**

The *length* of an optimal tour :

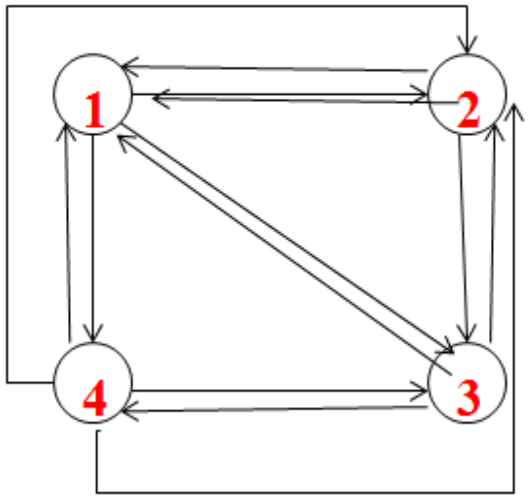
$$g(1, V - \{1\}) = \min_{2 \le k \le n} \{c_{1k} + g(k, V - \{1, k\})\}$$

Example 1: The graph is



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

Solution: The Given graph is



The Distance Matrix is

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

First, we will select any vertex arbitrary , select 1.

Now process for intermediate sets with increasing size.

Clearly $g(i, \emptyset) = C_{i,1}$ $1 \leq i \leq n$

$g(1, \emptyset) = C_{11} = 0$

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

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

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

$|S| = 4$

It means set contains only one element. Before solving this problem, we make an assumption that the sales person starts at vertex 1, from that he can move to vertex 2.

If he visits vertex 2, from that vertex, he next visit either vertex 3 or vertex 4. since $|S| = 1$.

i. $g(2,3) = \min_{j \in S} \{ C_{23} + g(3, \emptyset)\}$

$= 9 + 6 = 15$

ii. $g(2,4) = \min_{j \in S} \{ C_{24} + g(4, \emptyset)\}$

$= 10 + 8 = 18$

From vertex 1, next he can visit vertex 3 instead of vertex 2. In this case, from vertex 3, next he can visit either 2 or 4.

iii. $g(3,2) = \min_{j \in S} \{ C_{32} + g(2, \emptyset)\}$

$= 13 + 5 = 18$

iv. $g(3,4) = \min_{j \in S} \{ C_{34} + g(4, \emptyset)\}$

$= 12 + 8 = 20$

From vertex 1, next he can visit vertex 4 instead of vertex 3. In this case, from vertex 4, next he can visit either 2 or 3.

$$g(1,\{2,3,4\}) = \min_{j \in s} \{ C_{12} + g(2,\{3,4\}), C_{13} + g(3,\{2,4\}), C_{14} + g(4,\{2,3\}) \}$$

$$= \min_{j \in s} \{ 10 + 25, 15 + 25, 20 + 23 \}$$

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

$$= 35$$

Let J(1, S) be the value J(1, {2, 3, 4}) = 2. thus the tour starts from 1 and goes to 2.

The remaining tour can be obtained from g(2, {3, 4}).

So, J(2, {3, 4}) = 4. thus the next edge is <2, 4>.

The remaining tour can be obtained from g(4, {3, Ø}).

So, J(4, {3, Ø}) = 3

Therefore the optimal tour is 1 – 2 – 4 – 3 - 1.

Time Complexity:

Let N be the number of g(i, S)'s that have to be computed.

For each value of |S| there are n-1 choices for i.

The no. of distinct sets S of size k not including 1 and i is $(n-2)C_k$

Hence $N = \sum_{K=0}^{n-2} (n-k-1)\,(n-2)C_k = (n-1)\,2^{n-2}$

$$N = (n-1) \sum_{K=0}^{n-2} (n-2)_{C_k}$$

$$= (n-1).\ ((n-2)_{C_0} + (n-2)_{C_1} + (n-2)_{C_2} + \ldots\ldots\ldots + (n-2)_{C_{n-2}})$$

$$= (n-1).\ 2^{n-2} \quad (\text{Since } n_{C_0} + n_{C_1} + n_{C_2} + \ldots\ldots + n_{C_n} = 2^n)$$

$$n.N = n(n-1).\ (2^n/4) \text{ for n-stages}$$

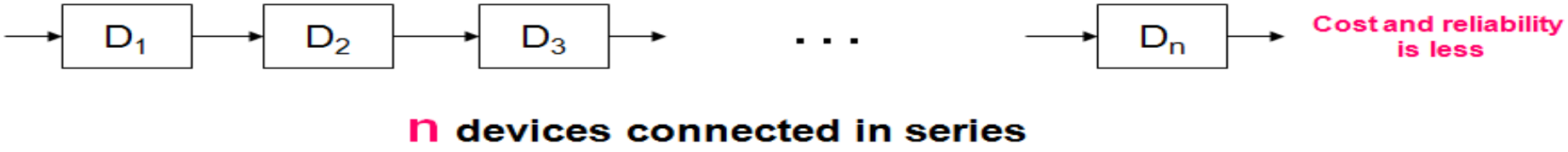
$$= n^2 - 2^n/4 - n.2^n/4$$

$$\Rightarrow O(n^2.2^n)$$

The time complexity of Travelling Sales Person Problem is $O(n^2.2^n)$

4.2.6. Reliability Design

The problem is to design a system that is composed of several devices connected in series.



Let r_i be the reliability of devices D_i(that is, r_i is the probability that device i will function properly), then the reliability of entire system is πr_i.

Even if the individual devices are very reliable(the r_i's are very close to one), the reliability of the system may not be very good.

Multiple copies of the same device type are connected in parallel through the use of switching circuits. The switching circuits determine which devices in any given group are functioning properly.

Example 1: Design a three stage system with device types D_1, D_2 and D_3. the costs are Rs. 30, Rs. 15, and Rs. 20 respectively. The cost of the system is to be no more than Rs. 105. the reliability of each device type is 0.9, 0.8 and 0.5 respectively.

Solution:

No. of stages : 3

$c_1 = 30$ $r_1 = 0.9$

$c_2 = 15$ $C = 105$ $r_2 = 0.8$

$c_3 = 20$ $r_3 = 0.5$

We will first compute u_1, u_2, u_3 using

$$u_i = \left[(C + c_i - \sum_{1 \le j \le n} c_j) / c_i \right]$$

u_1 $= [(105 + 30 - (30 + 15 + 20)) / 30]$

$= [(135 - 65) / 30]$

$= [70 / 30]$

$= [2.333] = 2$

u_2 $= [(105 + 15 - (30 + 15 + 20)) / 15]$

$= [(120 - 65) / 15]$

$= [55 / 15]$

$= [3.666] = 3$

u_3 $= [(105 + 20 - (30 + 15 + 20)) / 20]$

$= [(125 - 65) / 20]$

$= [60 / 20]$

$= [3] = 3$

$\therefore (u_1, u_2, u_3) = (2, 3, 3)$

Now we will start computing sequences

$\Phi_i(m_i) = 1 - (1 - r_i)^{m_i}$, $\Phi_i(m_i)$ is a reliable function

Each pair in this is represented by (r, x)

Here r is reliability and x is cost.

Using S_j^i to represent all tuples, obtained from S^{i-1} by choosing $m_i = j$.

Initially,

$$S^0 = \{ (1, 0) \}$$

Since $1 \le m_i \le u_i$

That is, $1 \le m_1 \le 2$ ($\because u_i = 2$)

$\therefore$ $m_1 = 1$ or $m_2 = 2$

It means $j = 1$ or $j = 2$

By using S_j^i, calculate S_1^1.

Here i = 1, j = 1 and $m_1 = 1$

$$\Phi_i(m_i) = 1 - (1 - r_i)^{m_i}$$

$$= 1 - (1 - r_1)^{m_1}$$

$$= 1 - (1 - r_1)^{1}$$

$$= 1 - (1 - 0.9)^{1}$$

$$= 1 - (0.1)^{1}$$

$$= 0.9$$

And $S^0 = \{ (1, 0) \}$

In each tuple (r, x), reliability is multiplied with previous reliability and cost is added to previous cost.

$$S_1^1 = \{ 0.9 \times 1, 30 + 0 \}$$

$$= \{ (0.9, 30) \}$$

Now to compute S_2^1 ,we will include first tuple (0.9, 30)

For S_2^1

Here i = 1, j = 2 , m_1 = j = 2 and r_1 = 0.9

$\Phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-r_1)^{m_1}$

$= 1-(1-r_1)^{2}$

$= 1-(1-0.9)^{2}$

$= 1-(0.1)^{2}$

$= 1-0.01$

$= 0.99$

And $S^0 = \{(1, 0)\}$

Now 0.99 is multiplied with 1 and cost $2xc_1 = 2x30 = 60$ (Since j = 2) is added to 0.

$S_2^1 = \{ (0.99 \times 1, 60 + 0) \}$

$= \{ (0.99, 60) \}$

We can't calculate S_2^1, since upper bound is 2. that is $u_1 = 2$.

So, S^1 can be obtained by merging the sets S_1^1 and S_2^1.

$S^1 = S_1^1 \cup S_2^1$.

$= \{ (0.9, 30) \} \cup \{ (0.99, 60) \}$.

$= \{ (0.9, 30), (0.99, 60) \}$.

Similarly, S_1^2, S_2^2 and S_2^2 can be calculated from S^1.

Since $1 \le m_i \le u_i$

That is, $1 \le m_2 \le u_2$

$1 \le m_2 \le 3 \ (\because u_2 = 3)$

$\therefore \ m_2 = 1$ or $m_2 = 2$ or $m_2 = 3$

It means $j = 1$ or $j = 2$ or $j = 3$.

For S_1^2

Here i = 2, j = 1 , m_2 = j = 1 and r_2 = 0.8

$$\pi_{i \le n} \Phi_2(m_2) = 1 - (1 - r_i)^{m_i}$$

$$= 1 - (1 - r_2)^{m_2}$$

$$= 1 - (1 - r_2)^{1}$$

$$= 1 - (1 - 0.8)^{1}$$

$$= 1 - (0.2)^{1}$$

$$= 1 - 0.2$$

$$= 0.8$$

And $S^1 = \{ (0.9, 30), (0.99, 60) \}$.

$$S_1^2 = \{ (0.9 \times 0.8, 30 + 15), (0.99 \times 0.8, 60 + 15) \}$$

$$= \{ (0.72, 45), (0.792, 75) \}$$

For S_2^2

Here i = 2, j = 2 , m_2 = j = 2 and r_2 = 0.8

$\pi \Phi_2 (m_2) = 1- (1 - r_i)^{m_i}$

$= 1- (1 - r_2)^{m_2}$

$= 1- (1 - r_2)^{2}$

$= 1- (1 - 0.8)^{2}$

$= 1- (0.2)^{2}$

$= 1 - 0.04$

$= 0.96$

And $S^1 = \{ (0.9, 30), (0.99, 60) \}$.

Now 0.96 is multiplied with previous reliability and cost $2xc_2 = 2x15 = 30$ (Since j = 2) is added to previous cost.

$S_2^2 = \{ (0.9 \times 0.96, 30 + 30), (0.99 \times 0.96, 60 + 30) \}$

$= \{ (0.864, 60), (0.9504, 90) \}$

For S_3^2

Here i = 2, j = 3 , m_2 = j = 3 and r_2 = 0.8

$\pi \Phi_2 (m_2) = 1 - (1 - r_i)^{m_i}$

$= 1 - (1 - r_2)^{m_2}$

$= 1 - (1 - r_2)^{3}$

$= 1 - (1 - 0.8)^{3}$

$= 1 - (0.2)^{3}$

$= 1 - 0.008$

$= 0.992$

And $S^1 = \{ (0.9, 30), (0.99, 60) \}$.

Now 0.992 is multiplied with previous reliability and cost $3xc_2 = 3x15 = 45$ (Since j = 3) is added to previous cost.

$S_3^2 = \{ (0.9 \times 0.992, 30 + 45), (0.99 \times 0.992, 60 + 45) \}$

$= \{ (0.8928, 75), (0.98208, 105) \}$

So, S^2 can be obtained by merging the sets S_1^2, S_2^2 and S_3^2.

$S^2 = S_1^2 \cup S_2^2 \cup S_3^2$.

$= \{ (0.72, 45), (0.792, 75) \} \cup \{ (0.864, 60), (0.9504, 90) \} \cup \{ (0.8928, 75), (0.98208, 105) \}$

$= \{ (0.72, 45), (0.792, 75), (0.864, 60), (0.9504, 90), (0.8928, 75), (0.98208, 105) \}$.

Now applying Purging Rule or Dominance Rule:

The pair (0.792, 75) is discarded. Because $45 < 75 < 60$. So,

$S^2 = \{ (0.72, 45), (0.864, 60), (0.9504, 90), (0.8928, 75), (0.98208, 105) \}$.

Observe the pairs, cost is in increasing order. That is $45<60<90<75<105$

The pair (0.9504, 75) is not in appropriate order. So, discard from S^2.

$S^2 = \{ (0.72, 45), (0.864, 60), (0.8928, 75), (0.98208, 105) \}$.

For S_1^3, Here i = 3, j = 1 , m_3 = j = 1 and r_3 = 0.5

$$\pi_{i \le n} \Phi_3(m_3) = 1-(1-r_i)^{m_i}$$

$$= 1-(1-r_3)^{m_3}$$

$$= 1-(1-r_3)^{1}$$

$$= 1-(1-0.5)^{1}$$

$$= 1-(0.5)^{1}$$

$$= 1-0.5$$

$$= 0.5$$

Similarly, S_1^3 , S_2^3 and S_3^3 can be calculated from S^2.

Since $1 \le m_i \le u_i$

That is, $1 \le m_3 \le u_3$

$1 \le m_3 \le 3$ ($\because u_3 = 3$)

$\therefore$ $m_3 = 1$ or $m_3 = 2$ or $m_3 = 3$

It means j = 1 or j = 2 or j = 3.

And $S^2 = \{ (0.72, 45), (0.864, 60), (0.8928, 75), (0.98208, 105) \}$.

$S_1^3 = \{ (0.72 \times 0.5, 45 + 20), (0.864 \times 0.5, 60 + 20), (0.8928 \times 0.5, 75 + 20), (0.98208 \times 0.5, 105 + 20) \}$

$S_1^3 = \{ (0.36, 65), (0.432, 80), (0.4464, 95), (0.49104, 125) \}$

Last tuple is removed from S_1^3. Since cost 125 exceeding the given cost 105.

There fore, S_1^3 $= \{ (0.36, 65), (0.432, 80), (0.4464, 95) \}$

For S_2^3

Here i = 3, j = 2 , m_3 = j = 2 and r_3 = 0.5

$$\pi \Phi_3 (m_3) = 1 - (1 - r_i)^{m_i}$$

$$= 1 - (1 - r_3)^{m_3}$$

$$= 1 - (1 - r_3)^{2}$$

$$= 1 - (1 - 0.5)^{2}$$

$$= 1 - (0.5)^{2}$$

$$= 1 - 0.25$$

$$= 0.75$$

And $S^2 = \{ (0.72, 45), (0.864, 60), (0.8928, 75), (0.98208, 105) \}$.

Now 0.75 is multiplied with previous reliability and cost $2xc_3 = 2x20 = 40$ (Since j = 2) is added to previous cost.

S_2^3 = { (0.72 x 0.75, 45 + 40), (0.864 x 0.75, 60 + 40), (0.8928 x 0.75, 75 + 40), (0.98208 x 0.75, 105 + 40) }

S_2^3 = { (0.54, 85), (0.648, 100), (0.6696, 115), (0.73656, 145) }

Last two tuples are removed from S_2^3 . Since costs are exceeding the given cost 105.

There fore, S_2^3 = { (0.54, 85), (0.648, 100) }

For S_3^3

Here i = 3, j = 3 , m_3 = j = 3 and r_3 = 0.5

$\pi \Phi_3 (m_3) = 1 - (1 - r_i)^{m_i}$

$= 1 - (1 - r_3)^{m_3}$

$= 1 - (1 - r_3)^3$

$= 1 - (1 - 0.5)^3$

$= 1 - (0.5)^3$

$= 1 - 0.125$

$= 0.875$

And $S^2 = \{ (0.72, 45), (0.864, 60), (0.8928, 75), (0.98208, 105) \}$.

Now 0.875 is multiplied with previous reliability and cost $3xc_3 = 3x20 = 60$ (Since j = 3) is added to previous cost.

$S_3^3 = \{ (0.72 \times 0.875, 45 + 60), (0.864 \times 0.875, 60 + 60), (0.8928 \times 0.875, 75 + 60), (0.98208 \times 0.875, 105 + 60) \}$

$S_3^3 = \{ (0.63, 105), (0.756, 120), (0.7812, 135), (0.85932, 165) \}$

Last three tuples are removed from S_3^3. Since costs are exceeding the given cost 105.

There fore, $S_3^3 = \{(0.63, 105)\}$

So, S^3 can be obtained by merging the sets S_1^3, S_2^3 and S_3^3.

$S^3 = S_1^3 \cup S_2^3 \cup S_3^3$.

$= \{ (0.36, 65), (0.432, 80), (0.4464, 95) \} \cup \{ (0.54, 85), (0.648, 100) \} \cup \{ (0.63, 105) \}$

$S^3 = \{ (0.36, 65), (0.432, 80), (0.4464, 95), (0.54, 85), (0.648, 100), (0.63, 105) \}$

Now applying Purging Rule or Dominance Rule:

The pair (0.4464, 95)is discarded. Because 65 < 80 < 95 < 85 < 100 < 105. So,

$S^3 = \{ (0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100), (0.63, 105) \}$

The pair (0.63, 105) is discarded. Because 0.36 < 0.432 < 0.54 < 0.648 < 0.63. So,

$S^3 = \{ (0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100) \}$

The best design has reliability of 0.648 and a cost of 100.

(0.648, 100) pair present in S_2^3, that is, $i = 3, j = 2, m_3 = 2$.

The (0.648, 100) obtained from (0.864, 60) which is present in S_2^2, that is, $i = 2, j = 2, m_2 = 2$.

The (0.864, 60) is obtained from (0.9, 30) which is present in S_1^1, that is, $i = 1, j = 1, m_1 = 1$.

$\therefore\ m_1 = 1, m_2 = 2$ and $m_3 = 2$