UNIT III Dynamic Programming

Dynamic Programming(DP) :

Dynamic Programming is a technique where, if any problem can be divided into subproblems, which in turn are divided into smaller subproblems, and if there are overlapping among these subproblems, then the solutions to these subproblems can be saved for future reference. In this way, efficiency of the CPU can be enhanced.

Dynamic Programming = Memorization/Memoization + Recursion

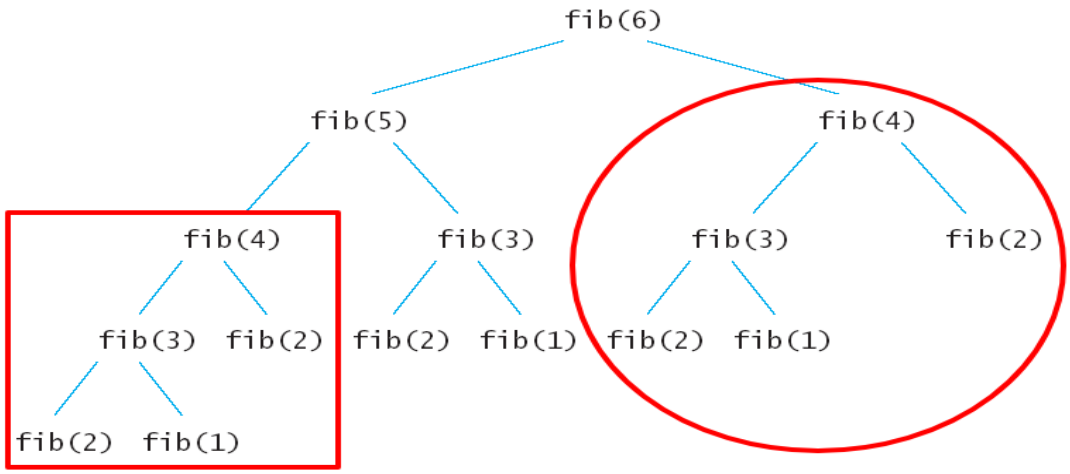
for better understanding of Dynamic Programming, let's revisit the recursion and then try to understand how DP works.

Recursion : Recursion is a process in which the function calls itself in order to solve the main problem

Example : Fibonacci series

```
#include<stdio.h>

void main() {
    printf("The Fibonacci number for index 6 = %d",fib(6));
}
int fib (int n) {
        if (n < 2)
            return 1;
        return fib(n-1) + fib(n-2);
}
```



In the above problem of solving fib(6), it was divided into sub problems fib(5) and fib(4), fib(5) is again divided into fib(4) and fib(3) and so on..

we can see that the same subproblem fib(4) is executed for more than one time making inefficient utilization of CPU.

'If the sub problems are being executed more than one time why don't we store the result somewhere on first computation itself and retrieve the same result when required' this is how the Dynamic programming works

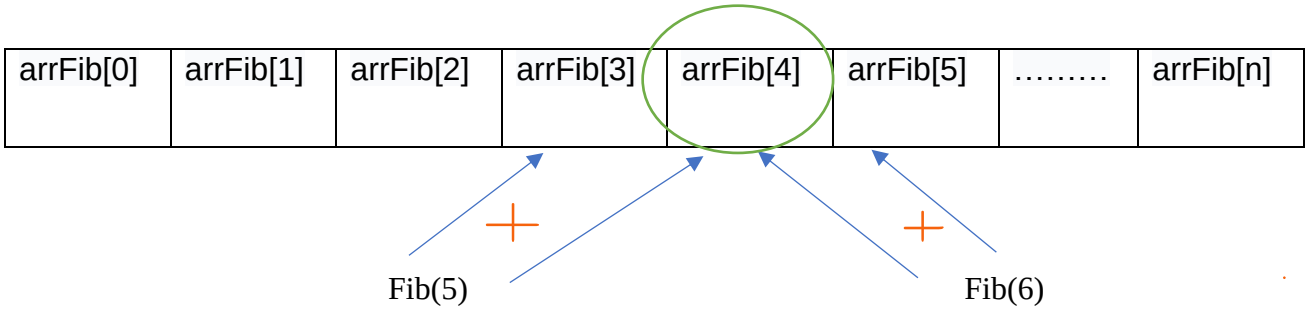
DP is generally used to solve problems which involve the following steps

- Split the problem into multiple small subproblems and store results of the sub problem.
- While solving each subproblem, check if the same problem was solved earlier. If yes, then take the stored result instead of solving the same subproblem again.
- Merge the subproblem results into the final result.

As we are storing the answer of every subproblem for future use, it requires extra memory to save the data. This process is called as **memoization / memorization**.

```
include<stdio.h>
void main() {
    printf("The Fibonacci number for index 6 = %d",fib(6));
}
int fib (int n) {
        arrFib[100] = {0};
        arrFib[0] = 1;
        arrFib[1] = 1;
        for (int i = 2; i<n; i++)
           arrFib[i] = arrFib[i-1] + arrFib[i-2];
       return arrFib[n-1]
    }
```

Instead of calling the functions recursively again and again, we are calculating the value of the Fibonacci series and storing it in an array.



All-Pair shortest Path - Floyd-Warshall algorithm

Step 1: Store the graph in adjacency matrix, where d(i,j) represents distance between vertex i & j

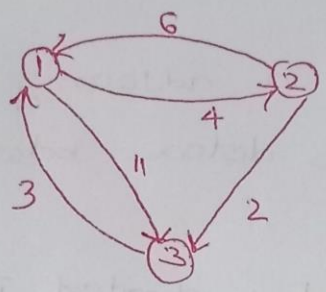
Step 2: select a vertex, Find shortest Path to all other vertices from this vertex & update adjacency matrix

Step 3: Repeat step 2 for all other vertices

Step 4: The Final adjacency matrix, contains all Pairs shortest Path

```
for k = 1 to n                    // n -> no of vertices
{
    for i = 1 to n
    {
        for j = 1 to n
        {
            d(i,j) = min{ d(i,j), d(i,k) + d(k,j) }
        }
    }
}
```

Find All Pairs shortest Path for following graph



For Given graph, the initial cost adjacency matrix

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

All Pairs shortest Path

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

Here No of vertices = 4

$\Rightarrow$ k runs from 1 to 4

$\rightarrow$ Set k = 1, Select any vertex say ① ~~first~~ find shortest Path between all Pairs of vertices with ① as intermediate vertex

$$A^1 = \min \left\{ \{ A^0(i,1) + A^0(1,j) \}, C(i,j) \right\}$

$A^1(1,1) = 0$, $A^1(2,2) = 0$, $A^1(3,3) = 0$

$A^1(1,2) = 4$

$A^1(1,3) = 11$

} No change in distance because 1 is source as well as the intermediate vertex

$A^1(2,1) = 6$

$A^1(3,1) = 3$

} No change in distance because 1 is intermediate as well as the End vertex

$A^1(2,3)$

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

$C(2,3)$

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

$= 2$ // No need to update distance

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

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

$= 7$ // Need to update

Now

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

Now Set K = 2, select any vertex say ② find shortest Path to other vertices with ② as intermediate

$$A^2 = \min\{\{A'(i,2) + A'(2,j)\}, C(i,j)\}$$

$$A^2(1,1) = A^2(2,2) = A^2(3,3) = 0$$

$A^2(2,1) = 6$
$A^2(2,3) = 2$
} ② is source / intermediate
No change in distance

$A^2(1,2) = 4$
$A^2(3,2) = 7$
} ② is intermediete / End
No change in distance

$$A^2(1,3) = \min\{\{A'(1,2) + A'(2,3)\}, C(1,3)\}$$
$$= \min\{(4+2), 11\}$$
$$= 6 \quad \text{// update}$$

$$A^2(3,1) = \min\{\{A'(3,2) + A'(2,1)\}, C(3,1)\}$$
$$= \min\{(7+6), 3\}$$
$$= 3 \quad \text{// No need to update}$$

Now

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & \underline{6} \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

Now set $k = 3$, Select the last vertex (3) and find Shortest Path to all vertices with (3) as intermediate

the A^3 =

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

We have calculated shortest Path from ~~Each~~ Each vertex to Every other vertex and $k=3$ is the cost matrix with All Pairs shortest Path

Single Source Shortest paths General Weights / Single Source Shortest path using Dynamic Programming / Single Source Shortest path using Bellman Ford Algorithm :

Like Dijkstra's shortest path algorithm, the Bellman-Ford algorithm is also used to find the shortest path from given source vertex to all other vertices in the graph G, with V vertices and E edges.. Though it is slower than Dijkstra's algorithm, Bellman-Ford is capable of handling graphs that contain negative edge weights.

Note : if there exists a negative cycle in the graph, then there is no shortest path

The pseudo-code for the Bellman-Ford algorithm

```
        for v in V:
                distance[v]= infinity
        source.distance = 0
        for i from 1 to |V| - 1:
                for (u, v) in E:
                        relax(u, v)
```

The first for loop sets the distance to each vertex in the graph to infinity. This is later changed for the source vertex to equal zero.

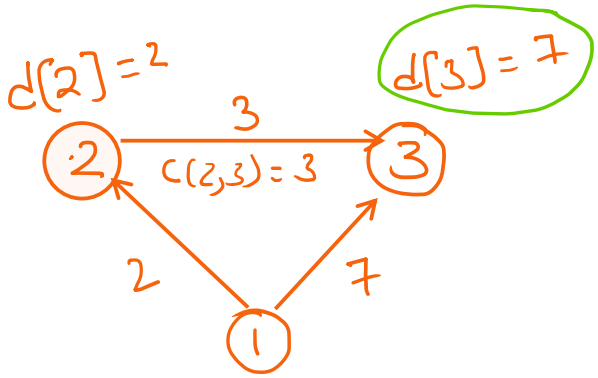
The next for loop is used to relax each edge (u, v) in E. This process is done |V| -1 times

```
relax(u, v):
    if distance[v]> distance[u]+ cost(u, v):
        distance[v] = distance[u] + cost(u, v)
```

Relaxation – Relaxing an Edge (u,v), d[u],d[v] are distance from source vertex to u and v, c(u,v) represents cost of edge between u and v

if (d[v] > d[u]+c(u,v))

d[v] = d[u]+c(u,v)



1 -> source vertex

Relaxing Edge (2,3) u = 2, v = 3

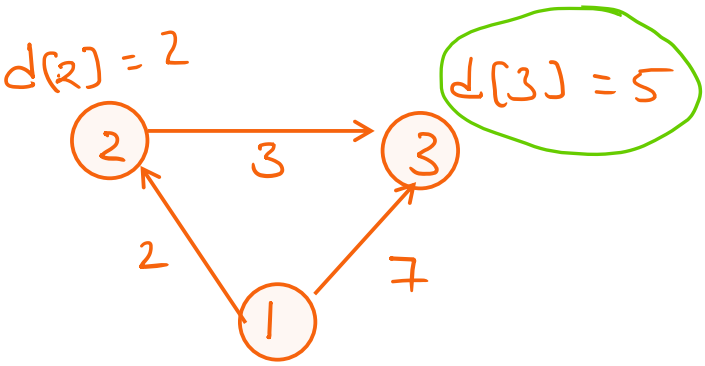
$$d[v] > d[u] + c(u,v)$$

$$7 > 2 + 3$$

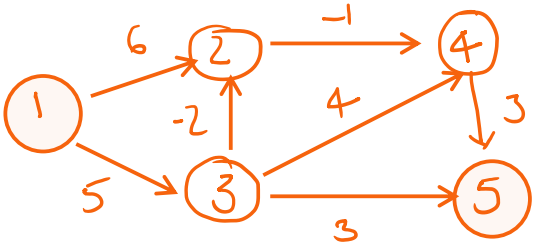
$$\Rightarrow \quad d[v] = d[u] + c(u,v)$$

$$\Rightarrow \quad d[3] = d[2] + c(2,3) = 2 + 3 = 5$$

$\Rightarrow$



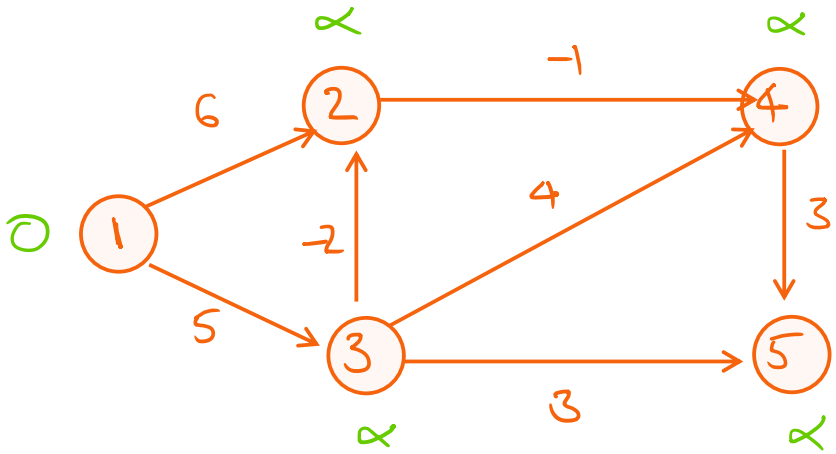
Example



Step 1 : List all the edges

(1,2). (1,3), (2,4,), (3,2,), (3,4), (3,5), (4,5)

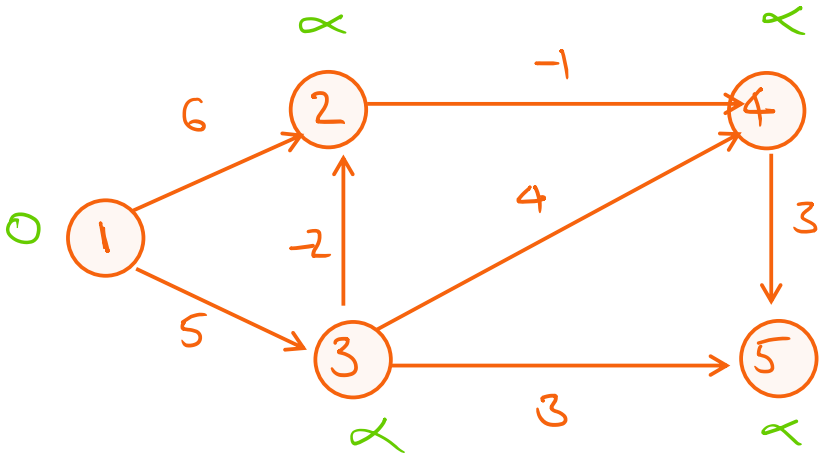
Step 2 : Initialize the distance for source vertex to 0 and other vertices to ∞ (infinite)



	1	2	3	4	5
i=0	0	∝	∝	∝	∝

Step 3 : Relax all the edges for V-1 times

i) Relaxing edges for 1st time



Edge (1,2) u = 1 , v = 2

d[v] < d[u] + c(u,v)

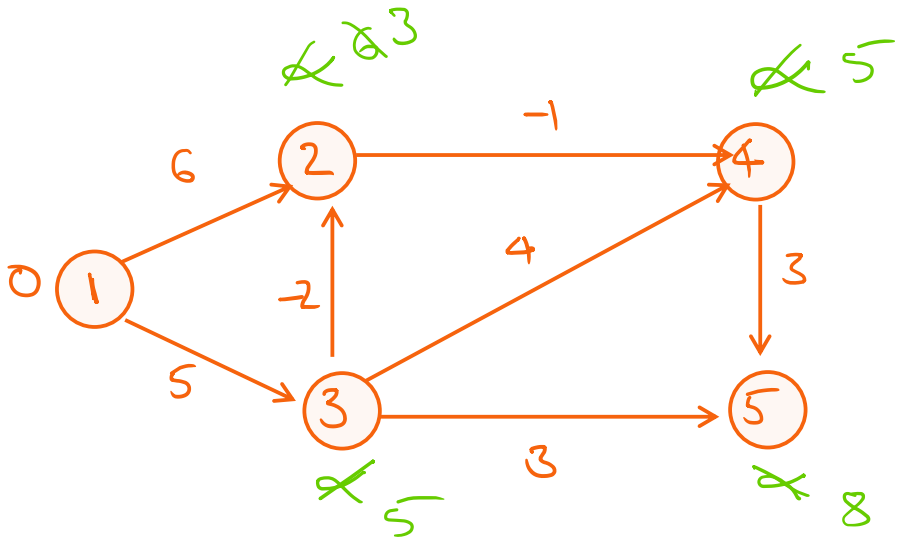
d[v] = d[u] + c(u,v)

$$d(2) > d(1) + c(1,2)$$

$$\infty > 0 + 6$$

$$\Rightarrow d[2] = 6$$

Similarly relax all the edges then updated distances will be



	1	2	3	4	5
i = 0	0	∞	∞	∞	∞
i = 1	0	3	5	5	8 ✓

ii) Relax all the edges for second time on previously updated distances

Edges (1,2). (1,3), (2,4,), (3,2,), (3,4), (3,5), (4,5)

For first edge (1,2)

d[1] = 0 and d[2] = 3

d[2] > d[1] + c(1,2)

3 > 0+6 is false so no need to update distance of d[2]

Similarly in edge (1,3) d[3] will not be updated

Edge (2,4) d[2] = 3 and d[4] = 2

d[3] > d[2] + c(2.3)

5 > 3 – 1

5 > 2 is true so update d[3] = 2

In Edges (3,2) (3,4) and (3,5) distances will not be updated

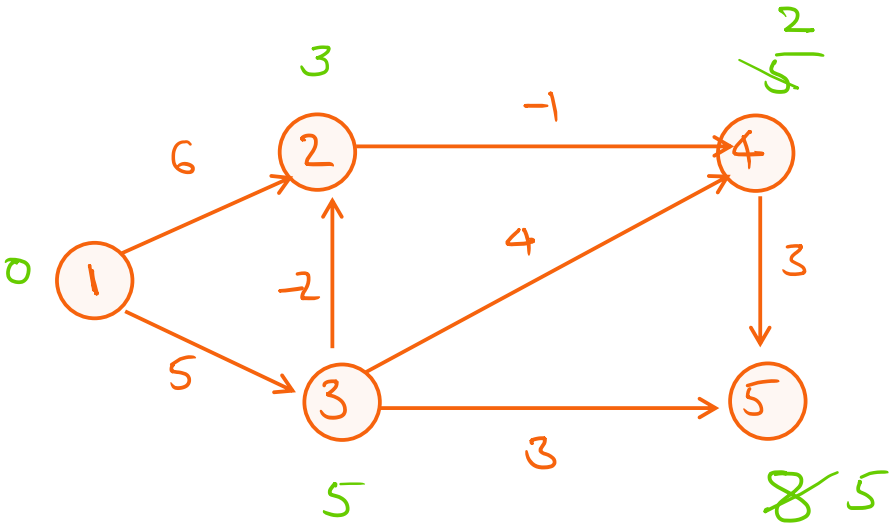
In last edge (4,5) d[4] = 2 and d[5] =8

d[5] > d[4] + c(4,5)

8 > 2 + 3

8 > 5 is true so update d[5] = 5

All the edges were relaxed, updated graph and distances will be

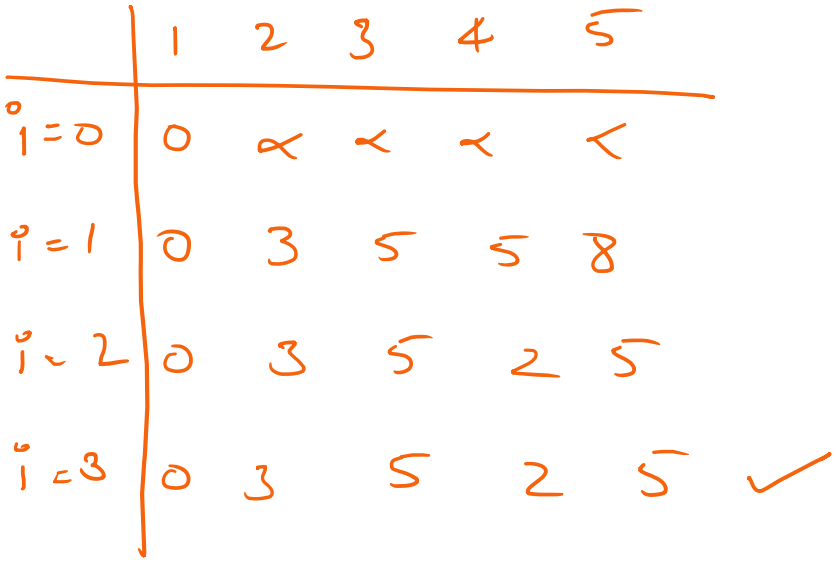


	1	2	3	4	5
i = 0	0	∞	∞	∞	∞
i = 1	0	3	5	5	8
i = 2	0	3	5	2	5 ✓

iii) Now Relaxing all the edges for third time

Edges (1,2). (1,3), (2,4,), (3,2,), (3,4), (3,5), (4,5)

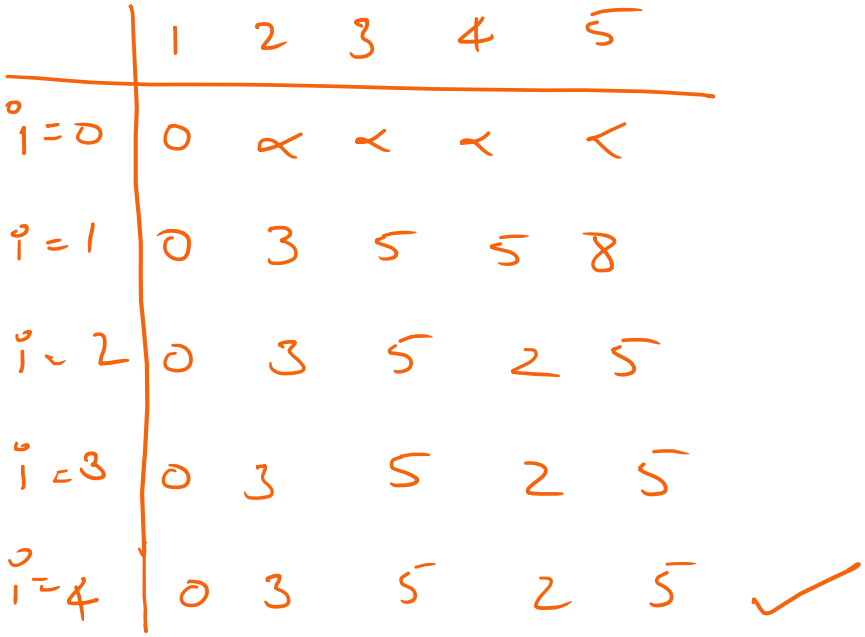
No updates were found the distances will be same as previous iteration



iv) Now Relaxing all the edges for fourth time

Edges (1,2). (1,3), (2,4,), (3,2,), (3,4), (3,5), (4,5)

No updates were found the distances will be same as previous iteration



All the edges in Graph were relaxed for 4 time i.e., V-1 times where no of vertices V = 5 the distances obtained in V-1th were the final shortest path distances from source vertex 0 to all other vertices

Optimal Binary Search Trees(OBST) : An optimal binary search tree is a BST, which has **minimal cost of finding each node.** A Binary Search Tree in which placing the most frequently used data in the root and closer to the root element, while placing the least frequently used data near leaves and in leaves to improve the search time is known to be OBST.

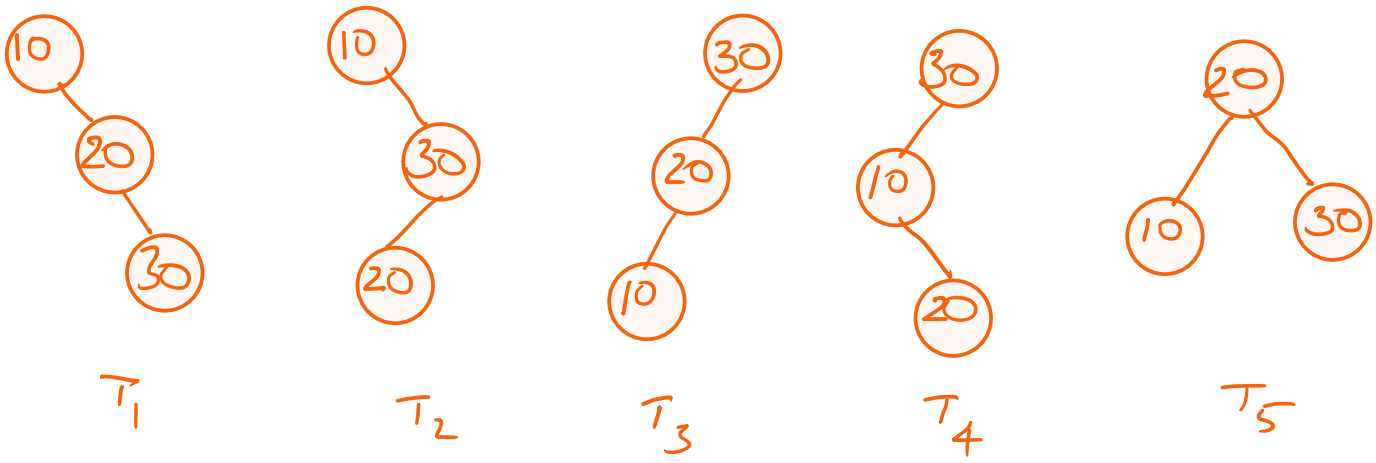
Construct all possible Binary search trees for following elements

10, 20, 30

No of possible BST will be $\frac{2nc_n}{n+1}$

n = no of Elements = 3

$$\frac{2 \times 3C_3}{n+1} = \frac{6C_3}{4} = 5$$



Here T5 will be optimal Binary Search Tree, when no of Search operations are almost same for all the keys

Observe below searching scenario for keys

Element	10	20	30
Frequency of search	5	40	100

For the above case, which tree will perform better

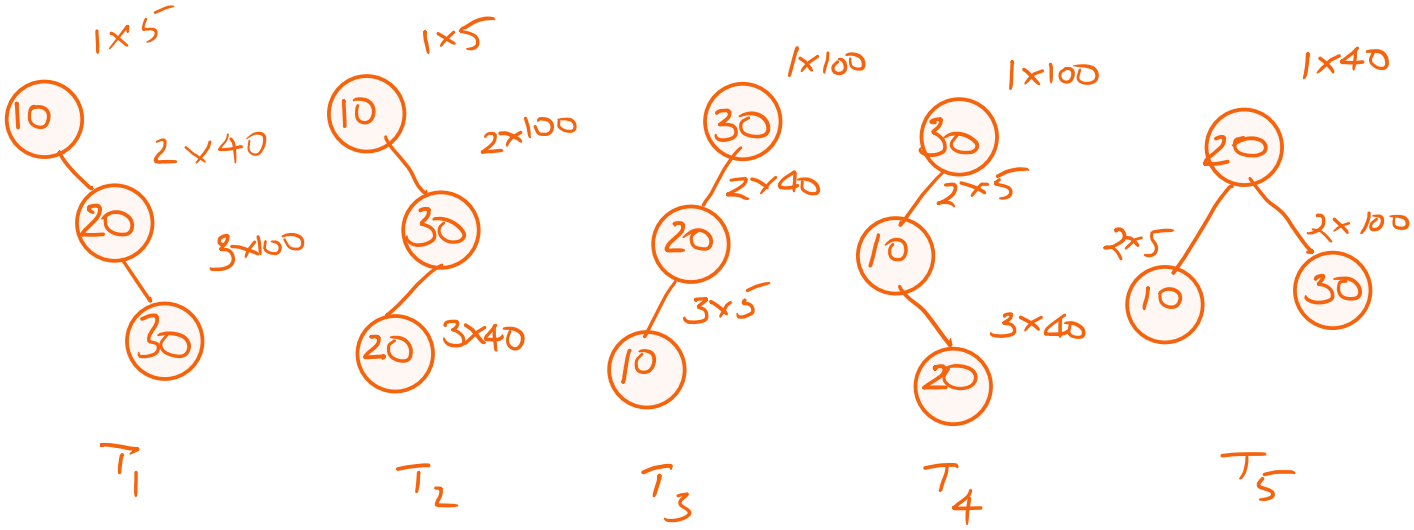
T1 = 1*5 + 2* 40 + 3*100 = 385

T2 = 1*5 + 2* 100 + 3*40 = 325

T3 = 1*100 + 2* 40 + 3*5 = 195

T4 = 1*100 + 2*5 + 3*40 = 230

T5 = 1*40 + 2* 5 + 2*100 = 290



Constructing All possible BST will be give us $\frac{2nc_n}{n+1}$

It will take huge time to construct all BST and then finding optimal one.

We can know the Optimal Binary Search Tree using the below procedure

- Here we assume, the probability of accessing a key K_i is p_i.
- Some dummy keys ($d_0, d_1, d_2, \ldots d_n$) are added as some searches may be performed for the values which are not present in the Key set K.
- We assume, for each dummy key d_i probability of access is q_i.
- W(i, j) denotes weight which can be defined using the following formula:

$$W(i, j) = P(j) + Q(j) + w(i, j-1)$$

- C(i, j) denotes the cost matrix for OBST C(i, j) can be defined recursively, in the following manner:

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

- R(i,j) = k value chosen among C(i, j)
- Initially C (i, i) = 0 and w (i, i) = Q (i) for 0 < i < n.

EXAMPLE Let n = 4, and (a_1, a_2, a_3, a_4) = (do, if, need, while) Let P (1: 4) = (3, 3, 1, 1) and

Q (0: 4) = (2, 3, 1, 1, 1)

Solution:

Table for recording W (i, j), C (i, j) and R (i, j):

	Column Row	**0**	**1**	**2**	**3**	**4**
j-i = 0	**0**	$W_{0,0}$ = 2 $C_{0,0}$ = 0 $R_{0,0}$ =0	$W_{1,1}$ = 3 $C_{1,1}$ = 0 $R_{1,1}$ =0	$W_{2,2}$ = 1 $C_{2,2}$ = 0 $R_{2,2}$ =0	$W_{3,3}$ = 1 $C_{3,3}$ = 0 $R_{3,3}$ =0	$W_{4,4}$ = 1 $C_{4,4}$ = 0 $R_{4,4}$ =0
j-i = 1	**1**	$W_{0,1}$ = 8 $C_{0,1}$ = 8 $R_{0,1}$ =1	$W_{1,2}$ = 7 $C_{1,2}$ = 7 $R_{1,2}$ =2	$W_{2,3}$ = 3 $C_{2,3}$ = 3 $R_{2,3}$ =3	$W_{3,4}$ = 3 $C_{3,4}$ = 3 $R_{3,4}$ =4	
j-i = 2	**2**	$W_{0,2}$ = 12 $C_{0,2}$ = 19 $R_{0,2}$ =1	$W_{1,3}$ = 9 $C_{1,3}$ = 12 $R_{1,3}$ =2	$W_{2,4}$ = 5 $C_{2,4}$ = 8 $R_{2,4}$ =3		
j-i = 3	**3**	$W_{0,3}$ = 14 $C_{0,3}$ = 25 $R_{0,3}$ =2	$W_{1,4}$ = 11 $C_{1,4}$ = 19 $R_{1,4}$ =2			
j-i = 4	**4**	$W_{0,4}$ = 16 $C_{0,4}$ = 32 $R_{0,4}$ =2				

This computation is carried out row-wise from row 0 to row 4.
Initially, W (i, i) = Q(i) and C (i, i) = 0 and R (i, i) = 0

j-i = 0	$W_{0,0}$ = 2 $C_{0,0}$ = 0 $R_{0,0}$ =0	$W_{1,1}$ = 3 $C_{1,1}$ = 0 $R_{1,1}$ =0	$W_{2,2}$ = 1 $C_{2,2}$ = 0 $R_{2,2}$ =0	$W_{3,3}$ = 1 $C_{3,3}$ = 0 $R_{3,3}$ =0	$W_{4,4}$ = 1 $C_{4,4}$ = 0 $R_{4,4}$ =0

Solving for C (0, n):

FIRST ROW, computing all C (i, j) such that j - i = 1
Possible elements are (0,1), (1,2), (2,3) and (3,4)

j-i = 1	$W_{0,1} = 8$ $C_{0,1} = 8$ $R_{0,1} = 1$	$W_{1,2} = 7$ $C_{1,2} = 7$ $R_{1,2} = 2$	$W_{2,3} = 3$ $C_{2,3} = 3$ $R_{2,3} = 3$	$W_{3,4} = 3$ $C_{3,4} = 3$ $R_{3,4} = 4$

Start with i = 0 and j = 1, Min i < k ≤ J.
the possible value for k = 1

W (0, 1) = P (1) + Q (1) + W (0, 0) = 3 + 3 + 2 = 8
C (0, 1) = W (0, 1) + min {C (0, 0) + C (1, 1)} = 8
R (0, 1) = 1 (value of 'K' that is minimum in the above equation).

with i = 1 and j = 2, Min i < k ≤ J the possible value for k = 2

W (1, 2) = P (2) + Q (2) + W (1, 1) = 3 + 1 + 3 = 7
C (1, 2) = W (1, 2) + min {C (1, 1) + C (2, 2)} = 7
R (1, 2) = 2

with i = 2 and j = 3, Min i < k ≤ J the possible value for k = 3

W (2, 3) = P (3) + Q (3) + W (2, 2) = 1 + 1 + 1 = 3
C (2, 3) = W (2, 3) + min {C (2, 2) + C (3, 3)} = 3 + [(0 + 0)] = 3
R (2, 3) = 3

with i = 3 and j = 4, Min i < k ≤ J the possible value for k = 4

(3, 4) = P (4) + Q (4) + W (3, 3) = 1 + 1 + 1 = 3
C (3, 4) = W (3, 4) + min {[C (3, 3) + C (4, 4)]} = 3 + [(0 + 0)] = 3
R (3, 4) = 4

Second, Computing all C (i, j) such that j - i = 2

j-i = 2	$W_{0,2} = 12$ $C_{0,2} = 19$ $R_{0,2} = 1$	$W_{1,3} = 9$ $C_{1,3} = 12$ $R_{1,3} = 2$	$W_{2,4} = 5$ $C_{2,4} = 8$ $R_{2,4} = 3$

Start with i = 0; so j = 2; as i < k ≤ J, so the possible values for k =1 and 2.

W (0, 2) = P (2) + Q (2) + W (0, 1) = 3 + 1 + 8 = 12
C (0, 2) = W (0, 2) + min {(C (0, 0) + C (1, 2)), (C (0, 1) + C (2, 2))}
= 12 + min {(0 + 7, 8 + 0)} = 19
R (0, 2) = 1

Next, with i = 1; so j = 3; as i < k ≤ j, so the possible value for k = 2 and 3.
W (1, 3) = P (3) + Q (3) + W (1, 2) = 1 + 1+ 7 = 9
C (1, 3) = W (1, 3) + min {[C (1, 1) + C (2, 3)], [C (1, 2) + C (3, 3)]}
= W (1, 3) + min {(0 + 3), (7 + 0)} = 9 + 3 = 12
R (1, 3) = 2

Next, with i = 2; so j = 4; as i < k ≤ j, so the possible value for k = 3 and 4.

W (2, 4) = P (4) + Q (4) + W (2, 3) = 1 + 1 + 3 = 5
C (2, 4) = W (2, 4) + min {[C (2, 2) + C (3, 4)], [C (2, 3) + C (4, 4)]

$= 5 + \min \{(0 + 3), (3 + 0)\} = 5 + 3 = 8$

$R(2, 4) = 3$

Third, Computing all C (i, j) such that J - i = 3; j = i + 3 and as $0 \leq i < 2$; i = 0, 1;i < k ≤ J.

j-i = 3	$W_{0,3} = 14$ $C_{0,3} = 25$ $R_{0,3} = 2$	$W_{1,4} = 11$ $C_{1,4} = 19$ $R_{1,4} = 2$

Start with i = 0; so j = 3; as i < k ≤ j, so the possible values for k = 1, 2 and 3.

$W(0, 3) = P(3) + Q(3) + W(0, 2) = 1 + 1 + 12 = 14$

$C(0, 3) = W(0, 3) + \min \{[C(0, 0) + C(1, 3)], [C(0, 1) + C(2, 3)], [C(0, 2) + C(3, 3)]\}$

$= 14 + \min \{(0 + 12), (8 + 3), (19 + 0)\} = 14 + 11 = 25$

$R(0, 3) = 2$

Start with i = 1; so j = 4; as i < k ≤ j, so the possible values for k = 2, 3 and 4.W

$(1, 4) = P(4) + Q(4) + W(1, 3) = 1 + 1 + 9 = 11$

$C(1, 4) = W(1, 4) + \min \{[C(1, 1) + C(2, 4)], [C(1, 2) + C(3, 4)], [C(1, 3) + C(4, 4)]\}$

$= 11 + \min \{(0 + 8), (7 + 3), (12 + 0)\} = 11 + 8 = 19$

$R(1, 4) = 2$

Fourth, Computing all C (i, j) such that j - i = 4; j = i + 4 and as $0 \leq i < 1$; i = 0;i < k ≤ J.

j-i = 4	$W_{0,4} = 16$ $C_{0,4} = 32$ $R_{0,4} = 2$

Start with i = 0; so j = 4; as i < k ≤ j, so the possible values for k = 1, 2, 3 and 4.

$W(0, 4) = P(4) + Q(4) + W(0, 3) = 1 + 1 + 14 = 16$

$C(0, 4) = W(0, 4) + \min \{[C(0, 0) + C(1, 4)], [C(0, 1) + C(2, 4)], [C(0, 2) + C(3, 4)], [C(0, 3) + C(4, 4)]\}$

$= 16 + \min [0 + 19, 8 + 8, 19+3, 25+0] = 16 + 16 = 32$

$R(0, 4) = 2$

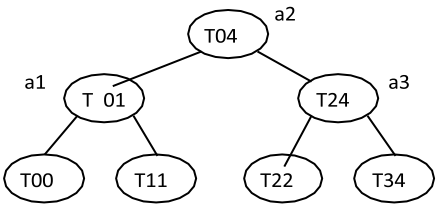
From the table we see that C (0, 4) = 32 is the minimum cost of a binary search tree for (a_1, a_2, a_3, a_4). The root of the tree 'T_{04}' is 'a_2'.

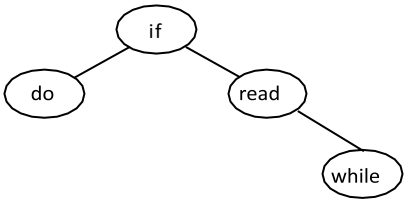
Hence the left sub tree is 'T_{01}' and right sub tree is T_{24}. The root of 'T_{01}' is 'a_1' and the root of 'T_{24}' is a_3.

The left and right sub trees for 'T_{01}' are 'T_{00}' and 'T_{11}' respectively. The root of T_{01} is 'a_1'

The left and right sub trees for T_{24} are T_{22} and T_{34} respectively.The root of T_{24} is 'a_3'.

The root of T_{22} is null The root of T_{34} is a_4.





Example 2: Consider four elements a1, a2, a3 and a4 with Q0 = 1/8, Q1 = 3/16, Q2 = Q3 = Q4 = 1/16 and p1 = 1/4, p2 = 1/8, p3 = p4 =1/16. Construct an optimal binary search tree.

STRING EDITING

The minimum number of "character edit operations" needed to turn one sequence into the other is known as **String Edit Distance**.

Consider the strings X= Amdrewz, Y= Andrew

To convert X to Y

1. substitute m to n
2. delete the z

Therefore Distance = 2

Given strings s and t

Distance is the shortest sequence of edit commands that transform s to t, (or equivalently s to t).

- Simple set of operations:
  - Copy character from s over to t (cost 0)
  - Delete a character in s (cost 1)
  - Insert a character in t (cost 1)
  - Replace one character for another (cost 1)

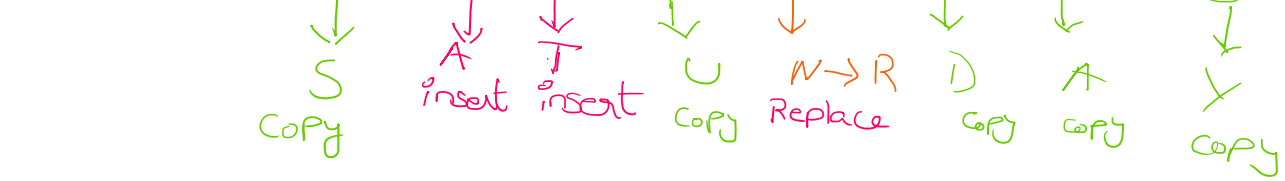
This is called to be "Levenshtein distance"

```
Algorithm stringEdit()
{
        for(i=0; i<len(str1)+1; i++)
        {
                for (i=0; i<len(str1)+1; i ++)
                {
                        if(i==0 && j==0)
                        T[i][j] = 0;
                        elseif (i==0)
                                T[i][j] = T[i][j-1] + 1
                        elseif (j==0)
                                T[i][j] = T[i-1][j] + 1
                        else {
                                T[i][j] = 1+ min { T[i][j-1], T[i-1][j-1], T[i-1][j]
                            }
                        } //end inner for
            } //end outer for
} // End Algo
```

Example

Edit the string Sunday to Saturday

	Null	S	A	T	U	R	D	A	Y
Null	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
U	2	1	1	2	2	3	4	5	6
N	3	2	2	2	3	3	4	5	6
D	4	3	3	3	3	4	3	4	5
A	5	4	3	4	4	4	4	3	4
Y	6	5	4	4	5	5	5	4	3



String Edit distance = 3

1. Insert ‘A’ 2. Insert ‘T’ and 3. Replace ‘N’ with ‘R’

Remaining Letters ‘S’, ‘U’, ‘D’, ‘A’, ‘Y’ are copied

0/1 Knapsack

In 0/1 Knapsack Problem,

As the name suggests, We have to either take an item completely or leave it completely. We cannot take the fraction of any item. It is solved using dynamic programming approach.

Example

Let say there are 2 objects and Knapsack with weight M = 20

	Object 1	Object 2
Weight	15	20
Profit	10	8

Here the weight of objects together is 35, whereas given knapsack weight is 20.

As object1 gives highest profit consider Object 1 into the knapsack, then knapsack size will be reduced to 20-15 = 5. The available space of knapsack is now 5.

The weight of second object is 20, which is greater than the available space 5. so we can not place second object into knapsack we have to leave it.

The total profit = 10 only

Note : If the above problem is fractional knapsack unlike 0/1 knapsack, we can consider the fractional part of the second object which is 5/20 =0.25 which gives total profit of 10 + 0.25(8) = 12.

0/1 Knapsack Problem Using Dynamic Programming

Knapsack weight capacity = w

Number of objects = n

Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns.

Fill the Table of 0th row and 0th column with zeroes as shown

	0	1	2	……	W
0	0	0	0	0	0
1	0				
2	0				
…	0				
N	0				

Here objects weights are represented row wise by W_i where i runs from 1 to n

Like W2 represents weight of object2

W3 represents weight of object3

Wn represents weight of object last object N

1					
2					
...					
N					

Knapsack weight is represented column wise as Wsum, Wsum runs from 1 to W(Kanpsack weight)

1	2	.....	W

Now fill the table row by row from left to right based the below conditions.

If (Wi > Wsum) ------> EQ1

T[i][Wsum] = T[i-1][Wsum]

elseif (Wi<= Wsum) ------> EQ2

T[i][Wsum] = Max {

T[i-1][Wsum] ,

P[i] + T[i-1][Wsum - Wi]

}

Note – Pi represents the profit of the object i

Example

Solve the below 0/1 Knapsack problem

Weight of objects Wi = {2, 3, 5, 4}

Profits of objects Pi = {3, 4, 6, 5}

Knapsack capacity W = 5

Solution :

Sort the Objects in ascending order w.r.t weights

Draw Table of size -- (no of objects + 1) * (Knapsack size + 1)

Fill first row as well as first column with zeros

Object	Profit	Weight	0	1	2	3	4	5
0	0	0	0	0	0	0	0	0
1	3	2	0					
2	4	3	0					
3	5	4	0					
4	6	5	0					

Now find the values for each cell

First Object W1 = 2, and i =1

Wsum = 1,

here W1 > Wsum so according to EQ1 copy the profit value from above cell

T[i][Wsum] = T[i-1][Wsum] = T[0][1] = 0

Wsum = 2

here W1 <= Wsum so according to EQ2

T[i][Wsum] = Max { T[i-1][Wsum], P[i] + T[i-1][Wsum-Wi] }

= Max { T[0][2] , 2 + T[0][2-2] }

= Max {0 , 2+ 0}

= 2

Similarly when Wsum = 3, 4 and 5

T[1][Wsum] = 2

Second Object W2 = 3 and i =2

Wsum = 1,

here W2 > Wsum so according to EQ1 copy the profit value from above cell

T[i][Wsum] = T[i-1][Wsum] = T[1][1] = 0

Wsum = 2,

W2 > Wsum,

T[i][Wsum] = T[i-1][Wsum] = T[1][2] = 3

Wsum = 3,

W3 <= Wsum,

T[i][Wsum] = Max { T[i-1][Wsum], P[i] + T[i-1][Wsum-Wi] }

= Max { T[1][3] , 4 + T[1][3-3] }

= Max {3 , 4+ 0}

= 4

Wsum = 4,

W3 <= Wsum,

T[i][Wsum] = Max { T[i-1][Wsum], P[i] + T[i-1][Wsum-Wi] }

= Max { T[1][4] , 4 + T[1][4-3] }

= Max {3 , 4+ 0}

= 4

Wsum = 3,

W3 <= Wsum,

T[i][Wsum] = Max { T[i-1][Wsum], P[i] + T[i-1][Wsum-Wi] }

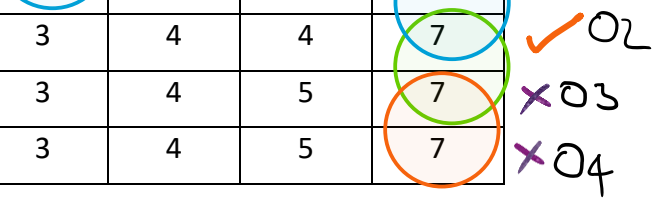
= Max { T[1][5] , 4 + T[1][5-3] }

= Max {3 , 4+ 3}

= 7

Similarly calculate values for all the cells

Object	Profit	Weight	0	1	2	3	4	5
0	0	0	0	0	0	0	0	0
1	3	2	0	0	3	3	3	3
2	4	3	0	0	3	4	4	7
3	5	4	0	0	3	4	5	7
4	6	5	0	0	3	4	5	7



Objects included = O1 and O2

So profits = 3 + 4 =7

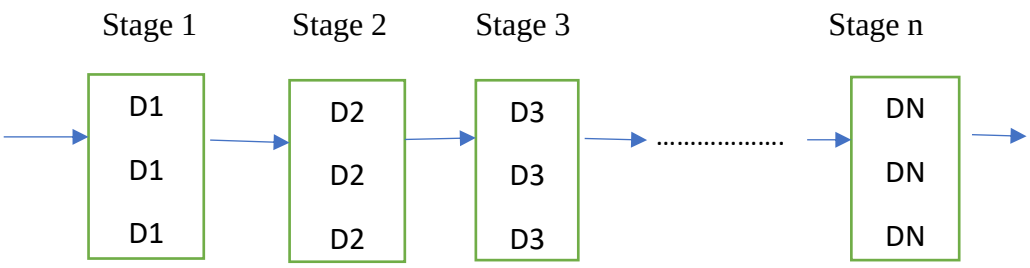
Reliability Design

Reliability Design aims to build a system with high reliability, Where the devices are connected in series.

N Devices connected in Series

In the above figure only one copy of each device is used to define the system. If any one the devices fail the entire system will fail. In this system the reliability and cost of system both are less.

The Reliability of the system can be increased by increasing the no of copies of each device. For example, in case one copy fails the other copies of same device can used that leads to improved reliability of the system as shown below



So, if we duplicate the devices at each stage then the reliability of the system can be increased.

It can be said that multiple copies of the same device type are connected in parallel through the use of switching circuits. Here, switching circuit determines which devices in any given group are functioning properly. Then they make use of such devices at each stage, that result is increase in reliability at each stage. If at each stage, there are **mi** similar types of devices **Di**, then the probability that all **mi** have a malfunction is **(1 - ri)^mi**, which is very less, and the reliability of the stage **i** becomes **(1 – (1 - ri) ^mi)**.

Thus, if **ri = 0.99** and **mi = 2**, then the stage reliability becomes

$$1 - (1 - 0.99)^2 = 0.9999$$

which is almost equal to **1**. Which is much better than that of the previous case.

In reliability design, we try to use device duplication to maximize reliability. But this maximization should be considered along with the cost.

Let **c** is the maximum allowable cost and **ci** be the cost of each unit of device **i**. Then the maximization problem can be given as follows:

Maximize π Øi (mi) for 1 <= I <= n Subject to $\sum_{i=1}^{n} c_i m_i <= C$

Dominance Rule : Consider two set of pairs (R1, C1) and (R2, C2)

If R1 > R2 and C1 < C2 then we can discard the pair having less reliability and high cost

Example – (R1,C1) = (0.9, 20) and (R2, C2) = (0.5, 60)

Here 0.9 > 0.5 and 20 < 60 so discard the pair (0.5, 60)

Example

we need to design 3 stage systems with devices D1, D2, and D3 and costs of devices are $30, $15, and $20, respectively. The cost of this system is to be no more than $105. The reliability of each device type is 0.9, 0.8, and 0.5. Find the optimal reliability for the entire system in given cost.

	Cost	Reliability
D1	30	0.9
D2	15	0.8
D3	20	0.5

Solution :

Given allowable cost C = 105,

Let UBi be the upper bound(Maximum) no of copies for Device i

$$UBi = \frac{c - \sum_{j=1}^{n} c_j}{c_i} + 1$$

$\sum_{j=1}^{n} C_j$ = The cost for one copy of each device = C1 + C2 + C3

= 30 + 15 + 20

= 65

$C - \sum_{j=1}^{n} C_j = 105 - 65 = 40$

Maximum number of additional D1 devices we can purchase

= 40/30 + 1 = 2

Maximum number of additional D2 devices we can purchase

= 40/15 + 1 = 3

Maximum number of additional D3 devices we can purchase

= 40/20 + 1 = 3

	Cost	Reliability	UBi
D1	30	0.9	2
D2	15	0.8	3
D3	20	0.5	3

No of Devices D1 can be purchased = 2

No of Copies	Reliability	Cost	
D1 – 1 copy	0.9	30	(0.9,30)
D1 – 2 copies	0.99	60	(0.99,60)

D1 – 2 copies reliability $= 1 – (1 – 0.9)^2$

$= 0.99$

D1 – 2 copies cost $= 30 + 30 = 60$

No of Devices D2 can be purchased = 3

No of Copies	Reliability	Cost	
D2 – 1 copy	0.8	15	(0.9,15)
D2 – 2 copies	0.96	30	(0.96,30)
D2 – 3 copies	0.992	45	(0.992,45)

D2 – 2 copies reliability $= 1 – (1 – 0.8)^2$
$= 0.96$

D2 – 2 copies cost $= 15 + 15 = 30$

D2 – 3 copies reliability $= 1 – (1 – 0.8)^3$
$= 0.992$

D2 – 3 copies cost $= 15 + 15 + 15 = 45$

No of Devices D3 can be purchased = 3

No of Copies	Reliability	Cost	
D2 – 1 copy	0.5	20	(0.5,20)
D2 – 2 copies	0.75	40	(0.75,40)
D2 – 3 copies	0.875	60	(0.875,60)

D2 – 2 copies reliability $= 1 – (1 – 0.8)^2$
$= 0.96$

D2 – 2 copies cost $= 15 + 15 = 30$

D2 – 3 copies reliability $= 1 – (1 – 0.8)^3$
$= 0.992$

D2 – 3 copies cost $= 15 + 15 + 15 = 45$

Let $S_0 = \{1, 0\}$

$S_1^1 = \{(0.9, 30)\}$

$S_1^2 = \{(0.99, 60)\}$

$S_1 = \{(0.9, 30), (0.99, 60)\}$

$S_2^1 = \{(0.72, 45), (0.792, 85)\}$

$(0.8, 15) \times S_1$

$S_2^2 = \{(0.864, 60), (0.9504, 90)\}$

$(0.96, 30) \times S_1$

$S_2^3 = \{(0.892, 75), (0.9564, 105)\}$

The Pairs $(0.9504, 90)$ and $(0.9564, 105)$ were Eliminated because we cannot buy atleast one copy of device having cost 20 as $90+20$ and $105+20 > 105$ (Total Available)

$S_2 = \{(0.72, 45), (0.792, 85), (0.864, 60), (0.892, 75)\}$ By Dominance Rule

$$\therefore S_2 = \{(0.72, 45), (0.864, 60), (0.892, 75)\}$$

$$S_3^1 = \{(0.36, 65), (0.432, 80), (0.4462, 95)\}$$

$$(0.5, 20) \times S_2$$

$$S_3^2 = \{(0.54, 85), (0.648, 100), (0.6696, 115)\}$$

$$(0.75, 40) \times S_2$$

Total available cost = 105 only

$$S_3^3 = \{(0.63, 105), (\quad, 120), (\quad, 135)\}$$

By Dominance Rule

$$S_3 = \{(0.36, 65), (0.432, 80), (0.4462, 95), (0.54, 85), (0.648, 100), (0.63, 105)\}$$

$$\therefore S_3 = \{(0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100), (0.63, 105)\}$$

The pair having highest Reliability

$\therefore$ Maximum Reliability = 0.648 with cost = 100

If we verify the Path for pair (0.648, 100) we can observe that D_1 - one device and D_2, D_3 two devices were considered