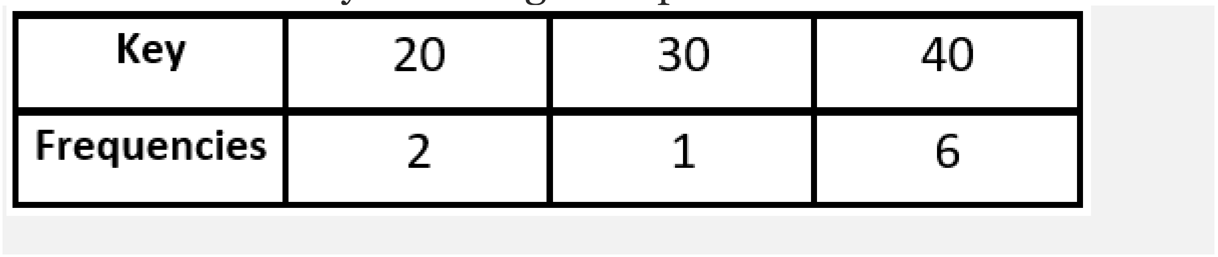
Optimal Binary Search Tree

In *Binary Search Tree (BST)*we know that for each node in the tree, left-sub tree of that particular node contains lesser value than the parent node and similarly for right-sub tree that it contains higher value than the parent node.

Basically, we know the key values of the each node, but let say we also know the frequencies of each node in the process of searching, means that how much time particular node is being searched. So, by knowing the frequencies of each node and their key values we can find the overall cost of searching.

In many applications the cost of searching is very important. So, it is required that the overall cost of searching should be as less as possible. And we know that search time of BST is more than the ***Balanced Binary Search Tree***, as Balanced Binary Search tree has less number of levels than the BST. And there is one way which can further reduce the cost than the Balanced BST, which is ***Optimal Binary Search Tree*** . Let us understand that by following example.



Key	20	30	40
Frequencies	2	1	6

Figure: 1

As there are 3 different keys, so we can have total 5 various BST by changing order of keys. And which van be found by, where “n” is the number of keys.

$$\frac{\binom{2n}{n}}{n+1}$$

Formula: 1

So following are the various possible Binary Search Trees of the above data. And also the overall cost for searching for each BST.

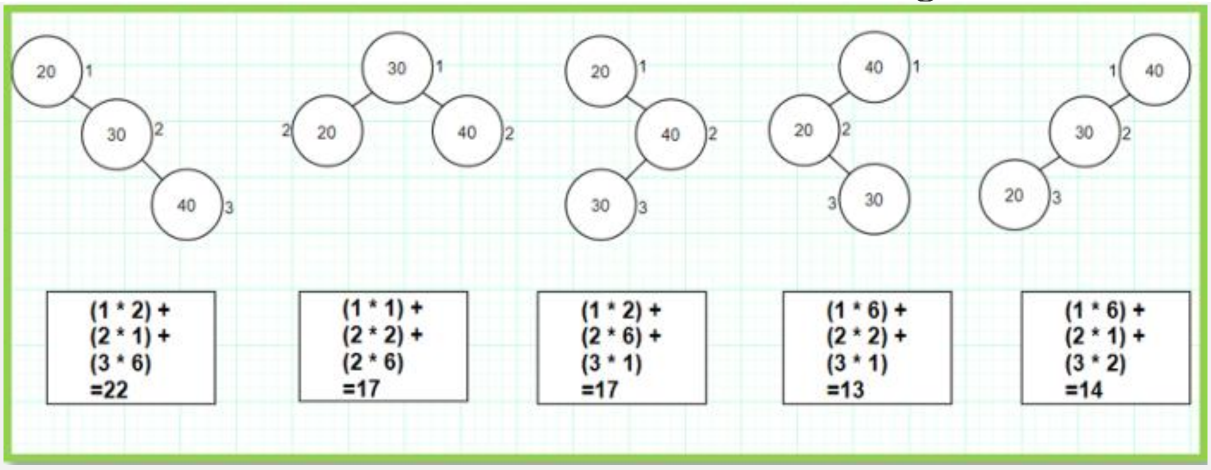


Figure: 2

The cost is computed by multiplying the each node’s frequency with the level of tree(Here we are assuming that the tree starts from level 1) and then add them to compute the overall cost of BST.

As we can see in Figure 2 there are various 5 types of arrangements are possible. And as we have discussed earlier that, it is possible to further reduce the cost of Balanced BST which is specified in following figure 3.

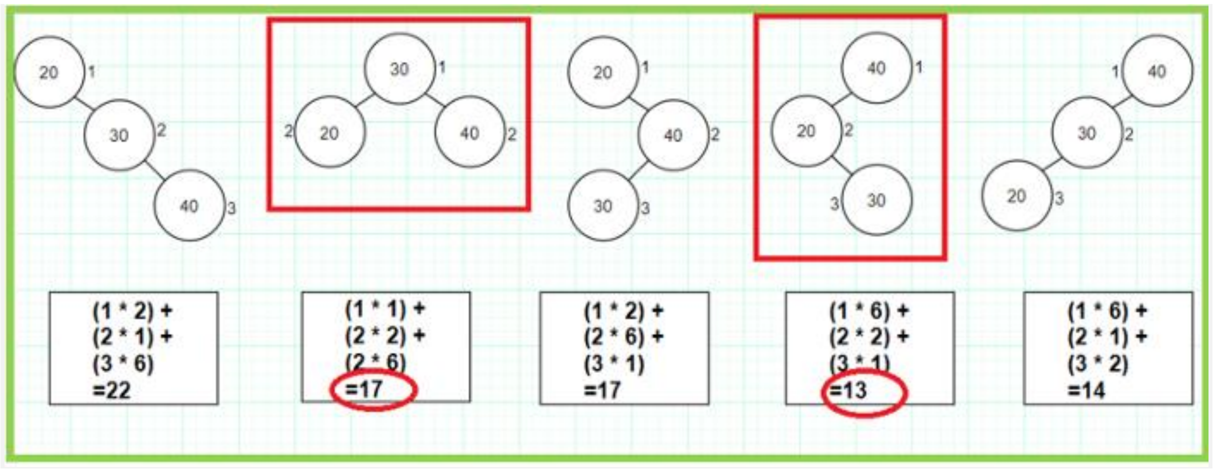


Figure: 3

As it is shown in above figure that 2nd BST is balanced and the 4th BST is not balanced, though it's cost is less than the cost of Balanced BST and its cost is the least among all, so it is our Optimal Binary Search Tree for the given data in Figure 1. So here our idea to generate the Optimal Binary Search Tree is that, the nodes whose frequencies are more should appear in the lower levels of Tree .i.e, In our example node with key 40 is having highest frequency and which appears at level 1 which is our Optimal BST.

Here note that: if the number of nodes are less then we can find optimal BST by checking all possible arrangements, but if the nodes are greater than 3 like 4,5,6..... then respectively 14,42,132..... , different BSTs are possible so by checking all arrangements to find Optimal Cost may lead to extra overhead. So we will see now another approach to solve the problem of Optimal BST using Dynamic Programming Approach.

Using Dynamic Approach:-

$$c[i,j] = \begin{cases} \min\limits_{i \le k \le j} \{ c[i,k-1] + C(k+1,j)\} + \sum\limits_{l=i}^{j} p_l & \text{if } i < j \\ p_i & \text{if } i = j \\ 0 & \text{if } i > j \end{cases}$$

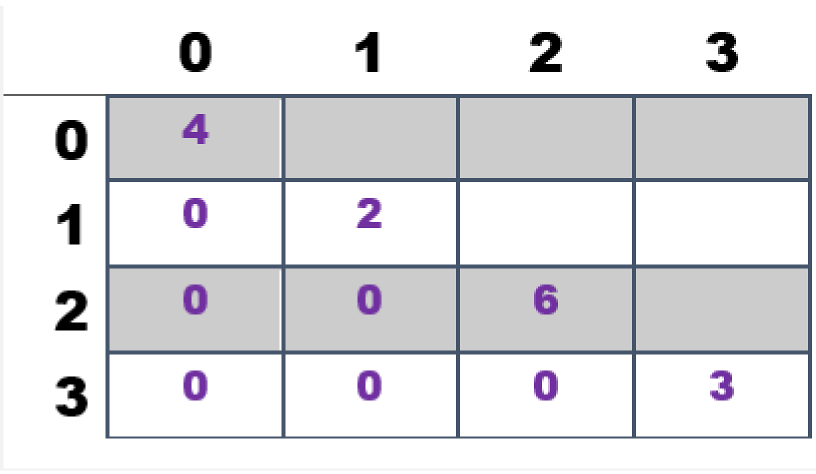
Formula: 2

where c[i , j] in which c stands for cost, “ pi ” stands for frequency of node i. We will use matrix form to solve this problem.Problem Statement is as following and we have to find Optimal BST for that:

Node	0	1	2	3
Key	10	12	16	21
Freq.	4	2	6	3

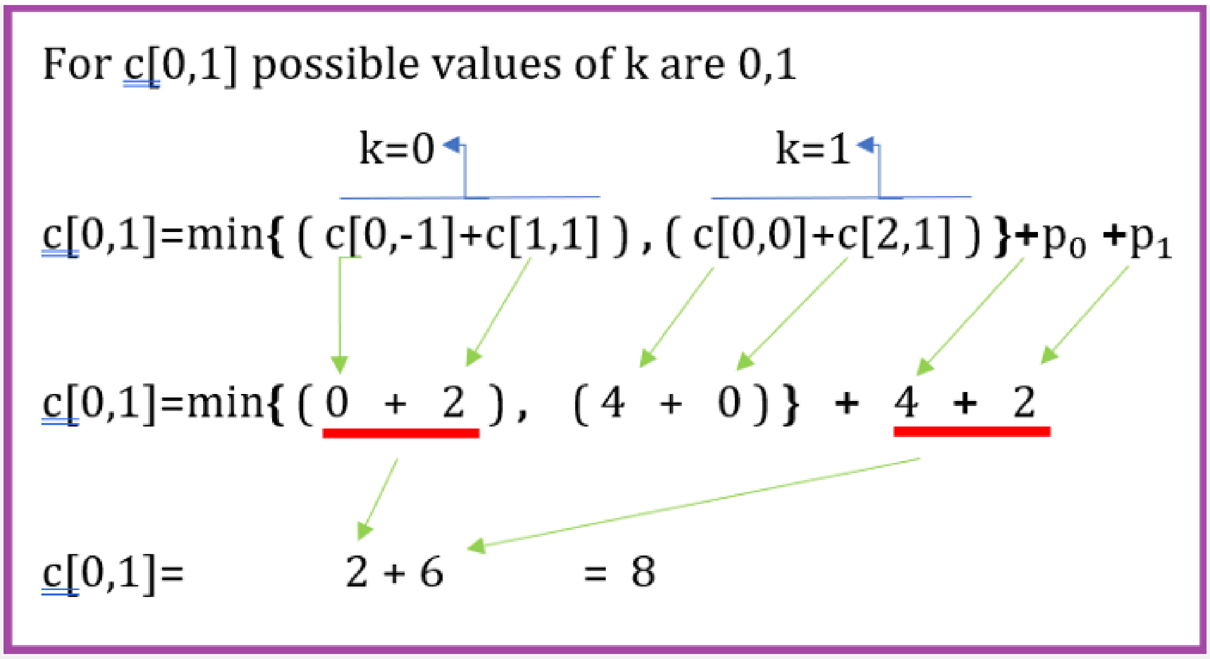
SOLUTION

Step: 1 :: According to above formula: 2 we derived the following matrix, that c[i,i] = Pi and for i>j → c[i,j] = 0;



	0	1	2	3
0	4			
1	0	2		
2	0	0	6	
3	0	0	0	3

Step: 2 :: So now compute value of C[0,1] as following and put it in table.



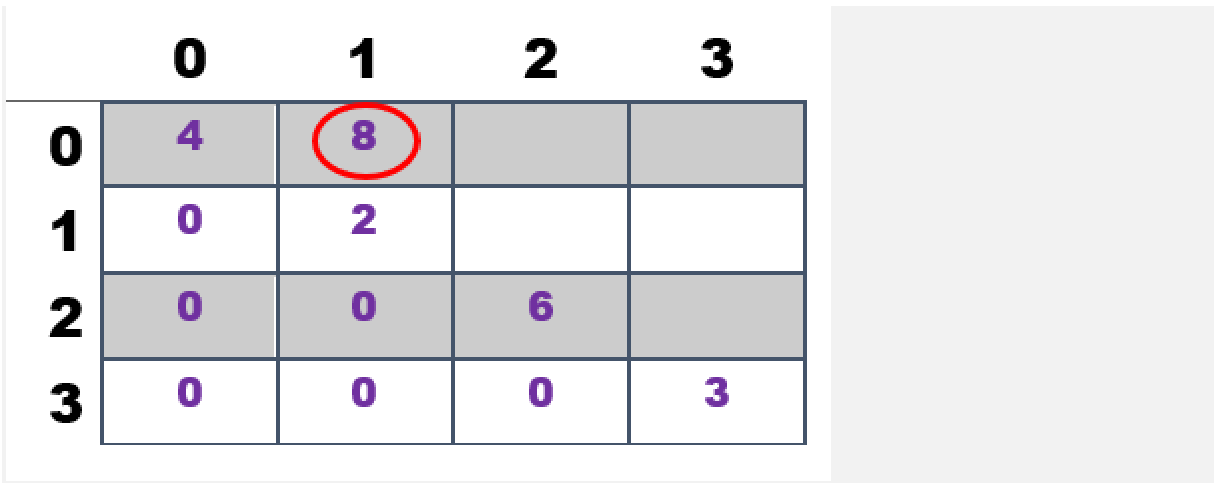
For c[0,1] possible values of k are 0,1

$$c[0,1]=\min\{\overbrace{(c[0,-1]+c[1,1])}^{k=0},\overbrace{(c[0,0]+c[2,1])}^{k=1}\}+p_0+p_1$$

$$c[0,1]=\min\{(\underline{0\ +\ 2}),\ (4\ +\ 0)\}\ +\ \underline{4\ +\ 2}$$

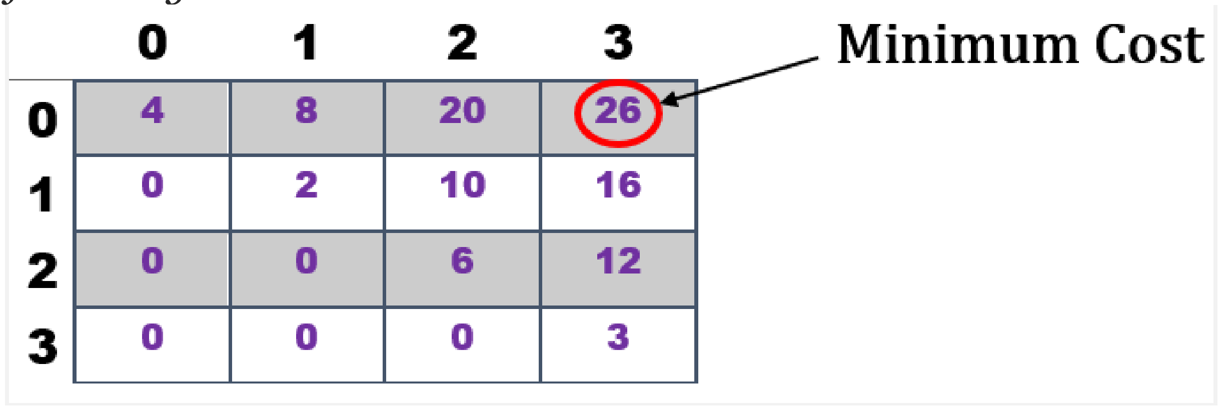
$$c[0,1]=\quad 2+6 \quad = 8$$

and the table after this step will look like...



	0	1	2	3
0	4	8		
1	0	2		
2	0	0	6	
3	0	0	0	3

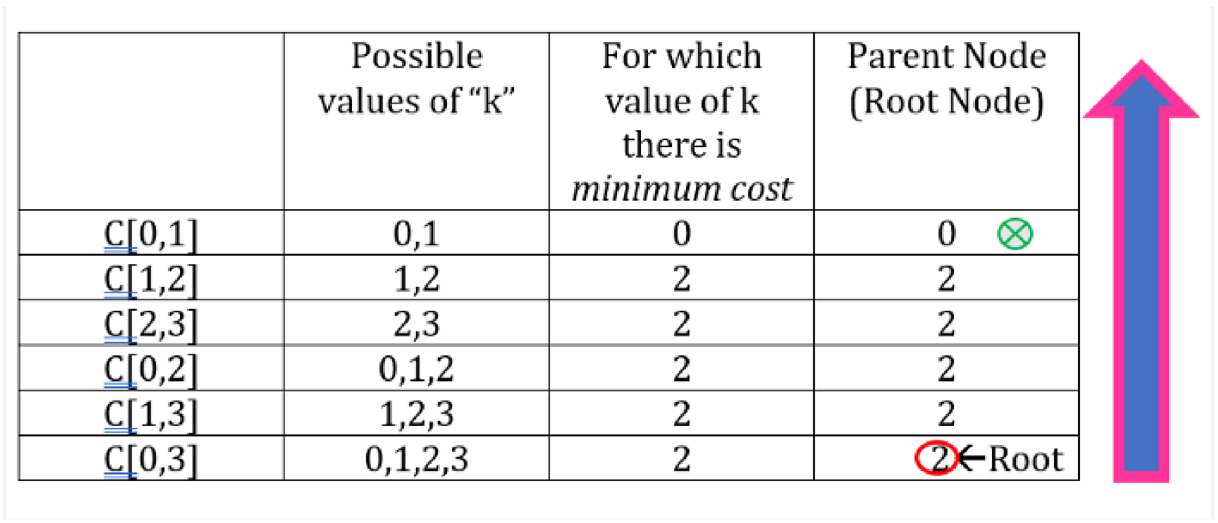
Step: 3 *:: Do same for all elements like c[1,2],c[2,3],c[0,2],c[1,3],c[0,3] and check your answer with the following table........*



	0	1	2	3
0	4	8	20	26
1	0	2	10	16
2	0	0	6	12
3	0	0	0	3

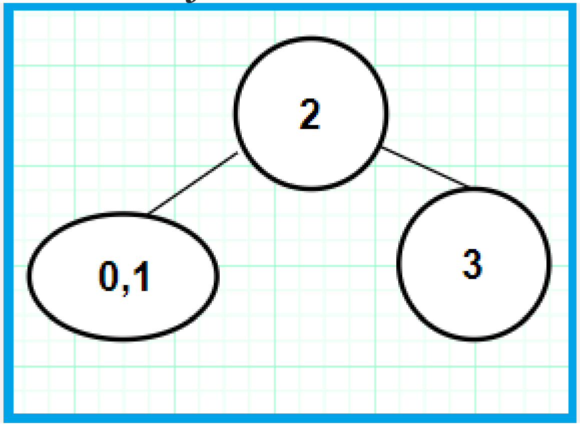
Minimum Cost

Step:4 :: So now we want to arrange the nodes in such a way that they will cost the minimum value which we derived previously. For that we have to choose root(parent node)in every group of nodes. See in step 2: for node 0 and node 1 the parent node will be decided by the value of K for which we took the minimum cost, in our example for value of k=0, we are able to find the minimum value SO parent node will be node 0 and similarly do for others and make table as following which will help you to arrange nodes very easily.

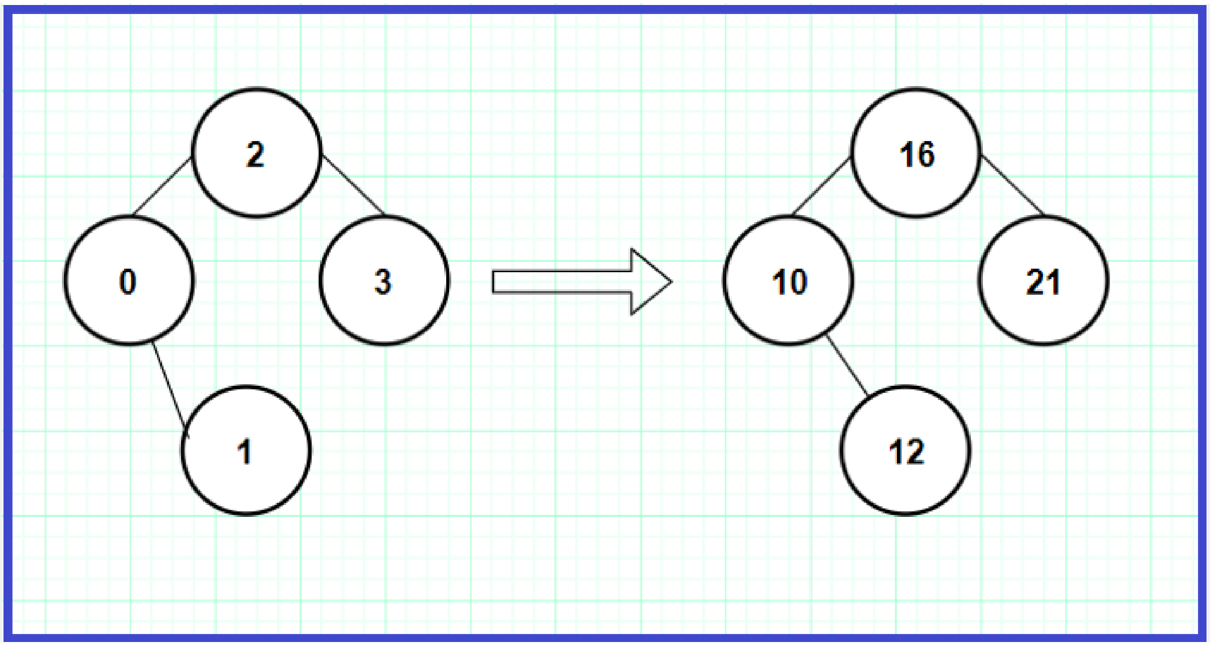


	Possible values of “k”	For which value of k there is *minimum cost*	Parent Node (Root Node)
C[0,1]	0,1	0	0 ⊗
C[1,2]	1,2	2	2
C[2,3]	2,3	2	2
C[0,2]	0,1,2	2	2
C[1,3]	1,2,3	2	2
C[0,3]	0,1,2,3	2	2←Root

Step: 5:: After creating table as shown above go from BOTTOM → UP. As shown in table Node 2 appears at bottom of the table so it is root of our BST. In our example the keys are sorted so we can say that node 3 will appear at the right sub-tree of node 2 , and node 0 and node 1 will appear at the left sub-tree of the node 2 as shown below.



So now there is one question that “ Which node is parent node between node 0 and node 1??” *For that again Go from Bottom →Up and see that node 0 is parent node , because when we were computing value of c[0,1] at that time we came to know that between 0,1 node 0 is parent node. And as in our example the keys are sorted so node 1 will appear in the right sub-tree of* *node 0(because key value of node 1 is higher than the key value of node 0). So our Optimal Binary Search Tree for our example will look like.......*



Above is our Optimal Binary Search Tree with cost 26.

Algorithm to Compute the minimum cost::

```
function $C(i,j)$
    if $C(i,j)$ already computed then
        return $C(i,j)$
    end if
    if $i > j$ then
        return $0$
    else if $i = j$ then
        return $p_i$
    else
        return $\min_{i \le k \le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{l=i}^{j} p_l$
    end if
end function
```

For actual implementation of BST, for each sub-problem we can also store the root(parent node), and the hint for the same is as follow:

$$r(i,j) = \underset{i \le k \le j}{\operatorname{argmin}} \{C(i, k-1) + C(k+1, j)\}.$$

in which r(i,j) stores root(parent node) for node i to j.