# [0/1 Knapsack Problem | Dynamic Programming | Example](https://www.gatevidyalay.com/0-1-knapsack-problem-using-dynamic-programming-approach/)

[Design & Analysis of Algorithms](https://www.gatevidyalay.com/category/subjects/design-analysis-of-algorithms/)

## ****Knapsack Problem-****

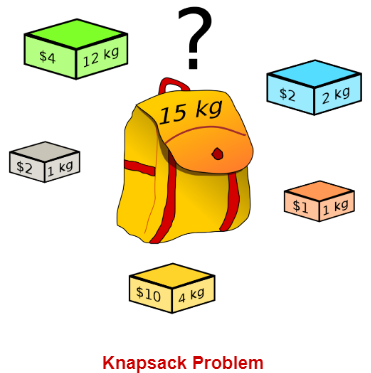
You are given the following-

* A knapsack (kind of shoulder bag) with limited weight capacity.
* Few items each having some weight and value.

**The problem states-**

Which items should be placed into the knapsack such that-

* The value or profit obtained by putting the items into the knapsack is maximum.
* And the weight limit of the knapsack does not exceed.



## ****Knapsack Problem Variants-****

Knapsack problem has the following two variants-

1. Fractional Knapsack Problem
2. 0/1 Knapsack Problem

In this article, we will discuss about 0/1 Knapsack Problem.

## ****0/1 Knapsack Problem-****

In 0/1 Knapsack Problem,

* As the name suggests, items are indivisible here.
* We can not take the fraction of any item.
* We have to either take an item completely or leave it completely.
* It is solved using dynamic programming approach.

**Also Read-** **[Fractional Knapsack Problem](https://www.gatevidyalay.com/fractional-knapsack-problem-using-greedy-approach/" \t "_blank)**

## ****0/1 Knapsack Problem Using Dynamic Programming-****

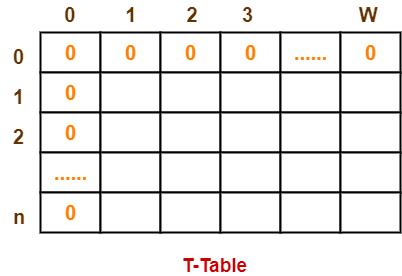
Consider-

* Knapsack weight capacity = w
* Number of items each having some weight and value = n

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

### ****Step-01:****

* Draw a table say ‘T’ with (n+1) number of rows and (w+1) number of columns.
* Fill all the boxes of 0th row and 0th column with zeroes as shown-



### ****Step-02:****

Start filling the table row wise top to bottom from left to right.

Use the following formula-

**T (i , j) = max { T ( i-1 , j ) , valuei + T( i-1 , j – weighti) }**

Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

* This step leads to completely filling the table.
* Then, value of the last box represents the maximum possible value that can be put into the knapsack.

### ****Step-03:****

To identify the items that must be put into the knapsack to obtain that maximum profit,

* Consider the last column of the table.
* Start scanning the entries from bottom to top.
* On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
* After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

## ****Time Complexity-****

* Each entry of the table requires constant time θ(1) for its computation.
* It takes θ(nw) time to fill (n+1)(w+1) table entries.
* It takes θ(n) time for tracing the solution since tracing process traces the n rows.
* Thus, overall θ(nw) time is taken to solve 0/1 knapsack problem using dynamic programming.

## ****PRACTICE PROBLEM BASED ON 0/1 KNAPSACK PROBLEM-****

## ****Problem-****

For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack problem making use of dynamic programming approach.

|  |  |  |
| --- | --- | --- |
| **Item** | **Weight** | **Value** |
| 1 | 2 | 3 |
| 2 | 3 | 4 |
| 3 | 4 | 5 |
| 4 | 5 | 6 |

**OR**

Find the optimal solution for the 0/1 knapsack problem making use of dynamic programming approach. Consider-

n = 4

w = 5 kg

(w1, w2, w3, w4) = (2, 3, 4, 5)

(b1, b2, b3, b4) = (3, 4, 5, 6)

**OR**

A thief enters a house for robbing it. He can carry a maximal weight of 5 kg into his bag. There are 4 items in the house with the following weights and values. What items should thief take if he either takes the item completely or leaves it completely?

|  |  |  |
| --- | --- | --- |
| **Item** | **Weight (kg)** | **Value ($)** |
| Mirror | 2 | 3 |
| Silver nugget | 3 | 4 |
| Painting | 4 | 5 |
| Vase | 5 | 6 |

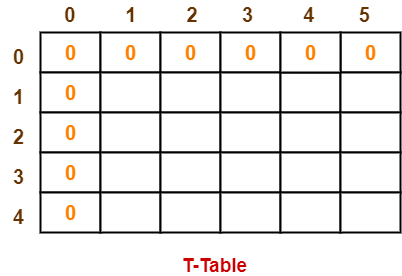
## ****Solution-****

### ****Given-****

* Knapsack capacity (w) = 5 kg
* Number of items (n) = 4

### ****Step-01:****

* Draw a table say ‘T’ with (n+1) = 4 + 1 = 5 number of rows and (w+1) = 5 + 1 = 6 number of columns.
* Fill all the boxes of 0th row and 0th column with 0.



### ****Step-02:****

Start filling the table row wise top to bottom from left to right using the formula-

**T (i , j) = max { T ( i-1 , j ) , valuei + T( i-1 , j – weighti) }**

### ****Finding T(1,1)-****

We have,

* i = 1
* j = 1
* (value)i = (value)1 = 3
* (weight)i = (weight)1 = 2

Substituting the values, we get-

T(1,1) = max { T(1-1 , 1) , 3 + T(1-1 , 1-2) }

T(1,1) = max { T(0,1) , 3 + T(0,-1) }

T(1,1) = T(0,1) { Ignore T(0,-1) }

T(1,1) = 0

### ****Finding T(1,2)-****

We have,

* i = 1
* j = 2
* (value)i = (value)1 = 3
* (weight)i = (weight)1 = 2

Substituting the values, we get-

T(1,2) = max { T(1-1 , 2) , 3 + T(1-1 , 2-2) }

T(1,2) = max { T(0,2) , 3 + T(0,0) }

T(1,2) = max {0 , 3+0}

T(1,2) = 3

### ****Finding T(1,3)-****

We have,

* i = 1
* j = 3
* (value)i = (value)1 = 3
* (weight)i = (weight)1 = 2

Substituting the values, we get-

T(1,3) = max { T(1-1 , 3) , 3 + T(1-1 , 3-2) }

T(1,3) = max { T(0,3) , 3 + T(0,1) }

T(1,3) = max {0 , 3+0}

T(1,3) = 3

### ****Finding T(1,4)-****

We have,

* i = 1
* j = 4
* (value)i = (value)1 = 3
* (weight)i = (weight)1 = 2

Substituting the values, we get-

T(1,4) = max { T(1-1 , 4) , 3 + T(1-1 , 4-2) }

T(1,4) = max { T(0,4) , 3 + T(0,2) }

T(1,4) = max {0 , 3+0}

T(1,4) = 3

### ****Finding T(1,5)-****

We have,

* i = 1
* j = 5
* (value)i = (value)1 = 3
* (weight)i = (weight)1 = 2

Substituting the values, we get-

T(1,5) = max { T(1-1 , 5) , 3 + T(1-1 , 5-2) }

T(1,5) = max { T(0,5) , 3 + T(0,3) }

T(1,5) = max {0 , 3+0}

T(1,5) = 3

### ****Finding T(2,1)-****

We have,

* i = 2
* j = 1
* (value)i = (value)2 = 4
* (weight)i = (weight)2 = 3

Substituting the values, we get-

T(2,1) = max { T(2-1 , 1) , 4 + T(2-1 , 1-3) }

T(2,1) = max { T(1,1) , 4 + T(1,-2) }

T(2,1) = T(1,1) { Ignore T(1,-2) }

T(2,1) = 0

### ****Finding T(2,2)-****

We have,

* i = 2
* j = 2
* (value)i = (value)2 = 4
* (weight)i = (weight)2 = 3

Substituting the values, we get-

T(2,2) = max { T(2-1 , 2) , 4 + T(2-1 , 2-3) }

T(2,2) = max { T(1,2) , 4 + T(1,-1) }

T(2,2) = T(1,2) { Ignore T(1,-1) }

T(2,2) = 3

### ****Finding T(2,3)-****

We have,

* i = 2
* j = 3
* (value)i = (value)2 = 4
* (weight)i = (weight)2 = 3

Substituting the values, we get-

T(2,3) = max { T(2-1 , 3) , 4 + T(2-1 , 3-3) }

T(2,3) = max { T(1,3) , 4 + T(1,0) }

T(2,3) = max { 3 , 4+0 }

T(2,3) = 4

### ****Finding T(2,4)-****

We have,

* i = 2
* j = 4
* (value)i = (value)2 = 4
* (weight)i = (weight)2 = 3

Substituting the values, we get-

T(2,4) = max { T(2-1 , 4) , 4 + T(2-1 , 4-3) }

T(2,4) = max { T(1,4) , 4 + T(1,1) }

T(2,4) = max { 3 , 4+0 }

T(2,4) = 4

### ****Finding T(2,5)-****

We have,

* i = 2
* j = 5
* (value)i = (value)2 = 4
* (weight)i = (weight)2 = 3

Substituting the values, we get-

T(2,5) = max { T(2-1 , 5) , 4 + T(2-1 , 5-3) }

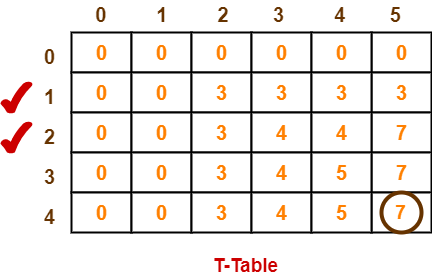
T(2,5) = max { T(1,5) , 4 + T(1,2) }

T(2,5) = max { 3 , 4+3 }

T(2,5) = 7

Similarly, compute all the entries.

After all the entries are computed and filled in the table, we get the following table-



* The last entry represents the maximum possible value that can be put into the knapsack.
* So, maximum possible value that can be put into the knapsack = 7.

### ****Identifying Items To Be Put Into Knapsack-****

Following Step-04,

* We mark the rows labelled “1” and “2”.
* Thus, items that must be put into the knapsack to obtain the maximum value 7 are-

OR

# (SHORTCUT METHOD)How to Solve The 0/1 Knapsack Problem Using Dynamic Programming

## Understand the 0/1 knapsack problem and learn how to solve it with dynamic programming

The **0/1 knapsack problem** is a classical [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming) problem. The [knapsack problem](https://en.wikipedia.org/wiki/Knapsack_problem) is a popular mathematical problem that has been studied for more than a century. 0/1 knapsack is one variant of this problem. Dynamic programming is a commonly studied topic in Computer Science. And 0/1 knapsack is one of the most popular dynamic programming practice problems that is frequently asked in coding interviews.

In this article, we will understand the 0/1 knapsack problem. We will know how to solve it using dynamic programming. Also, we will learn to implement it in Python.

Let’s get started!

**Table of Contents:**· [Dynamic Programming](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#badd)  
· [The 0/1 Knapsack Problem](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#2bf0)  
 ∘ [Let’s understand the problem with an example:](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#235c)  
· [The Naive Approach](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#2b4d)  
· [The Dynamic Programming Approach](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#8922)  
· [Finding Out The Solution](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#5104)  
· [Implementing in Python](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#dade)  
 ∘ [The naive recursive implementation:](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#f6ef)  
 ∘ [The improved implementation using Dynamic Programming:](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#6a6c)  
· [Conclusion](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#0598)  
· [Resources](https://medium.com/geekculture/how-to-solve-the-0-1-knapsack-problem-using-dynamic-programming-9f22e38f9916#eff5)

# Dynamic Programming

Dynamic Programming(DP) is a problem-solving technique to solve [optimization](https://en.wikipedia.org/wiki/Optimization_problem#:~:text=In%20mathematics%2C%20computer%20science%20and,solution%20from%20all%20feasible%20solutions.) and [counting](https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/counting-problem) problems. It is a technique to break a problem into similar subproblems. But never to solve a subproblem twice.

I’ve written an article on how to solve the Fibonacci number sequence using dynamic programming. In this article, I talked about dynamic programming in more detail. You can read it to clear the concept of dynamic programming:

# The 0/1 Knapsack Problem

Knapsack basically means a waterproof bag that soldiers or hikers use. In the 0/1 knapsack problem, we have a bag of given capacity C. We need to pack n items in the bag. Each item has a value v and weight w. The objective is to maximize the profit.

* The capacity of the bag is C.
* We have n items.
* The ith item has value vi and weight wi.
* Our target is to maximize the profit by taking the most valuable items but we can not exceed the bag capacity C.
* We can either carry an item or can not carry it. That means we can not take a fraction of the item. We either take it or leave it. The name 0/1 comes from here. If we can carry a fraction of the item, then it will be called the [fractional knapsack](https://en.wikipedia.org/wiki/Continuous_knapsack_problem) problem.
* We can not take an item more than once.

## Let’s understand the problem with an example:

Suppose we have a bag of maximum capacity C = 8units. We have a total of n = 4 items to choose from. The values of each item are given as a list v = [1, 2, 5, 6] and their corresponding weights are given as a list w = [2, 3, 4, 5]. Since this is a 0/1 knapsack problem, we can either include an item in the bag or can not include it. We need to take the items in such a way so that the total profit is maximized and the total weight of items taken is less than or equal to the capacity C.

# The Naive Approach

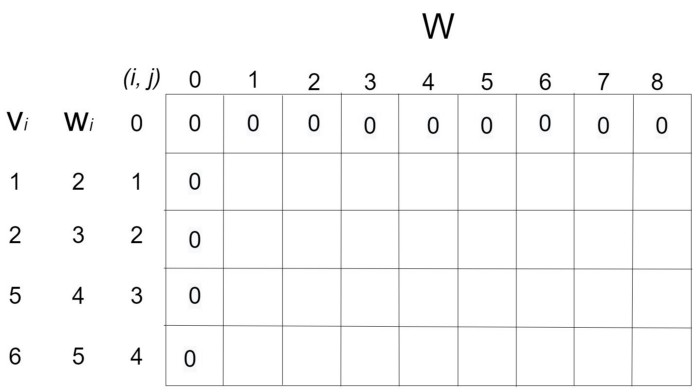
In the 0/1 knapsack problem, we can either include an item in the bag or we can not include it. Let’s represent including an item with 1 and not including an item with 0. In the naive approach, we have to consider all possible solutions.

items: 1st 2nd 3rd 4th 5thincluded of not: 1 0 0 0 0  
 0 1 0 0 0  
 0 0 1 0 0  
 ..........  
 .............  
 ..............  
 1 1 1 1 1

And this will generate 2⁴ possible solutions. So for n items, if we consider all possible solutions we will have 2^n possible solutions. It will give us a complexity of O(2^n). Which is an exponential complexity. Using dynamic programming, we will improve the complexity and make the solution more efficient.

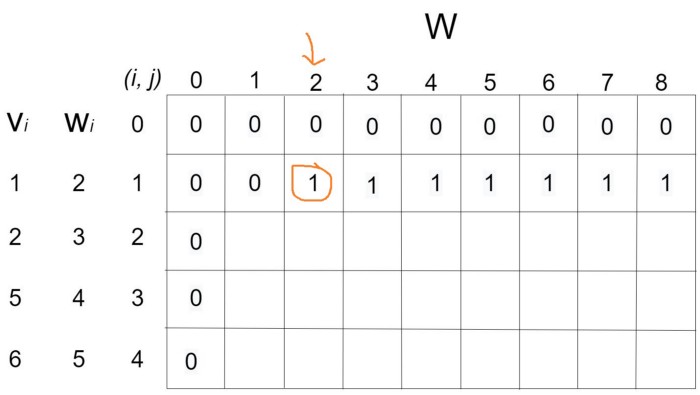
# The Dynamic Programming Approach

We have a total capacity of 8 units, and we have 4 items. So we will create a table as follows:



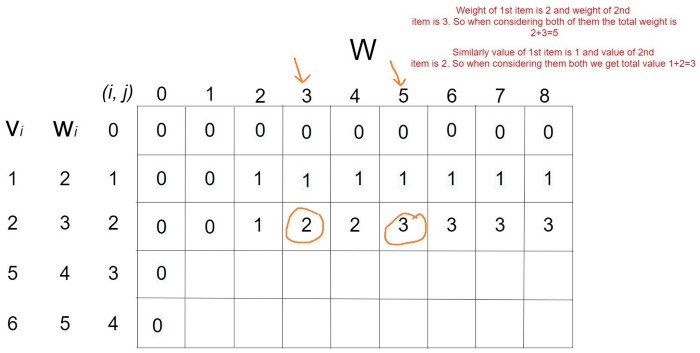
The initial state of the table | Image by author

Now let’s take the first item that has weight w = 2 units and value v = 1 units. So we can put it in the 3rd column of the 2nd row.



Considering the 1st item | Image by author

Now let’s move on to the 2nd item. Note that when considering the 2nd item we also have to consider the 1st item. The weight of the 2nd item is w = 3 units and the value v = 2 units. So we can put it in the 4th column of the 3rd row. And also considering the previous item their total weight w = 5 and value v = 3. So we have to put their total value in the 6th column of the 3rd row.

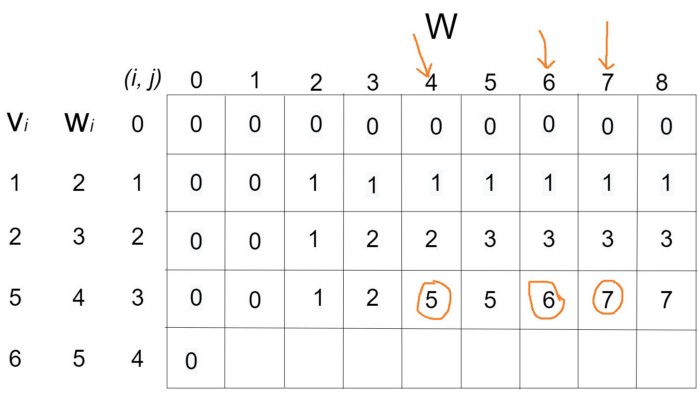


Considering the 2nd item | Image by author

Now let’s move on to the 3rd item. When considering the 3rd item there will be four scenarios:

1. Take only the 3rd item. The weight of the 3rd item is 4 and the value is 5.
2. Take the 3rd and the 1st item. The total weight will be 4+2=6 and the total value will be 5+1=6.
3. Take the 3rd and the 2nd item. The total weight will be 4+3=7 and the total value will be 5+2=7
4. Take them all. The total weight will be 4+3+2=9. Which is greater than the bag capacity C = 8. So we can not take all three items.

Let's fill up the 4th row considering the first three options accordingly:



Considering the 3rd item | Image by author

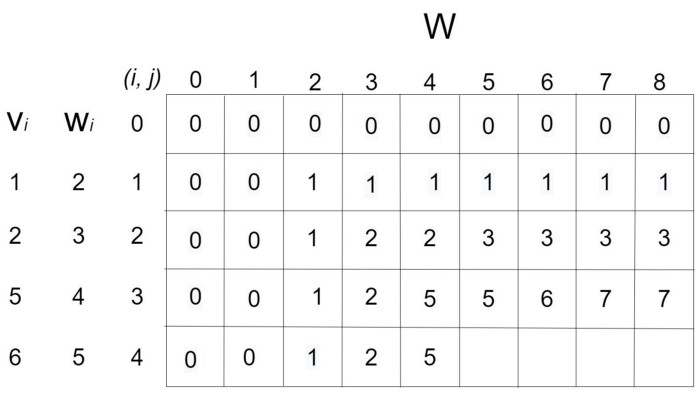
We can develop a formula now. If we represent the table as T, then

T[i, j] = max( T[i-1, j], {T[i-1, j-w[i]] + v[i]} ); where i is the column value and j is the row value. Now let’s fill up the last row using this formula:

T[4, 1] = max( T[3, 1], T[3, 1–5] + 6) = max(T[3, 1], T[3, -4] + 6)

As you can see this is giving us a negative position. So we have to put the value of the previous row T[3, 1] = 0.

Similarly, we will get negative positions up to the 5th column. We will put previous values till the 5th column.



We put previous values till the 5th column | Image by author

After that we need to calculate the values using the formula:

* T[4, 5] = max(T[3, 5], T[3, 5–5] + 6) = max(5, 0+6) = max(5,6) = 6
* T[4, 6] = max(T[3, 6], T[3, 6–5] + 6) = max(6, 0+6) = max(6, 6) = 6
* T[4, 7] = max(T[3, 7], T[3, 7–5] + 6) = max(7, 1+6) = max(7, 7) = 7
* T[4, 8] = max(T[3, 8], T[3, 8–5] + 6) = max(7, 2+6) = max(7, 8) = 8

And now we have the complete table:



The final table | Image by author

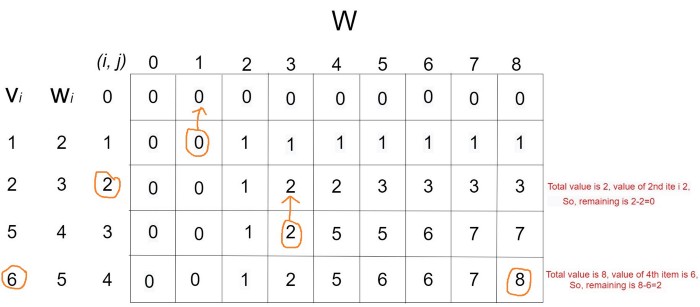
# Finding Out The Solution

From the table, we can say that the maximum profit is 8. But which items did we include? Let’s find it out:

We found 8 in the last row. Is it in any of the previous rows? No, there is no 8 in the previous rows. So we can say we got 8 by including the last item, which means the 4th item. The value of the 4th item was 6. So the remaining profit is 8–6=2.

Is 2 is in the 3rd row? Yes, it is in the 3rd row. But we can find 2 also in the 2nd row. So, we found value 2 not by including the 3rd item, but by including the 2nd item. The value of the 2nd item is 2. So the remaining profit is 2–2=0.

Now we don’t have any item with the value of 0. So no further item was included.



The items included | Image by author

So we found the maximum profit of 8 by including the 2nd and the 4th item. Here is the final solution:

1 means included  
 0 means not included items: 1st 2nd 3rd 4th 5thincluded of not: 0 1 0 1 0maximum profit = value of the 2nd item + value of the 4th item  
 = 2 + 6  
 = 8

This is the solution to our 0/1 knapsack problem.

Optimal Binary Search Tree

As we know that in binary search tree, the nodes in the left subtree have lesser value than the root node and the nodes in the right subtree have greater value than the root node.

We know the key values of each node in the tree, and we also know the frequencies of each node in terms of searching means how much time is required to search a node. The frequency and key-value determine the overall cost of searching a node. The cost of searching is a very important factor in various applications. The overall cost of searching a node should be less. The time required to search a node in BST is more than the balanced binary search tree as a balanced binary search tree contains a lesser number of levels than the BST. There is one way that can reduce the cost of a [binary search tree](https://www.javatpoint.com/binary-search-tree)

is known as an **optimal binary search tree**.

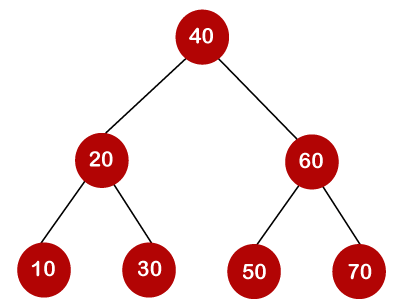
**Let's understand through an example.**

If the keys are 10, 20, 30, 40, 50, 60, 70

20.5M

338

Hello Java Program for Beginners

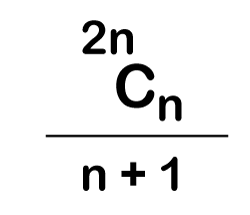


In the above tree, all the nodes on the left subtree are smaller than the value of the root node, and all the nodes on the right subtree are larger than the value of the root node. The maximum time required to search a node is equal to the minimum height of the tree, equal to logn.

Now we will see how many binary search trees can be made from the given number of keys.

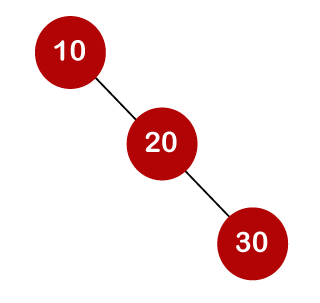
For example: 10, 20, 30 are the keys, and the following are the binary search trees that can be made out from these keys.

The Formula for calculating the number of trees:

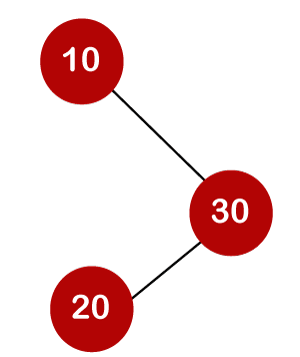


When we use the above formula, then it is found that total 5 number of trees can be created.

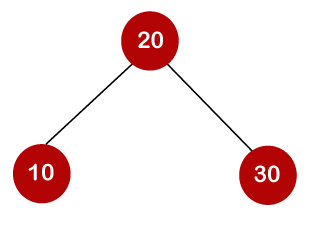
The cost required for searching an element depends on the comparisons to be made to search an element. Now, we will calculate the average cost of time of the above binary search trees.



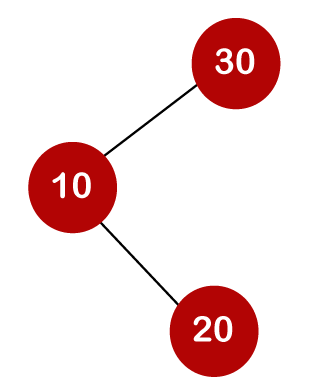
In the above tree, total number of 3 comparisons can be made. The average number of comparisons can be made as:

Optimal Binary Search Tree  


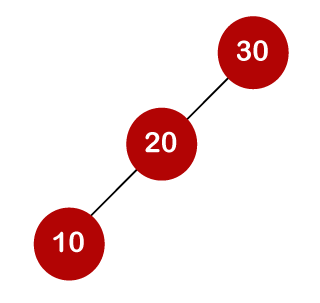
In the above tree, the average number of comparisons that can be made as:

Optimal Binary Search Tree  


In the above tree, the average number of comparisons that can be made as:

Optimal Binary Search Tree  


In the above tree, the total number of comparisons can be made as 3. Therefore, the average number of comparisons that can be made as:

Optimal Binary Search Tree  


In the above tree, the total number of comparisons can be made as 3. Therefore, the average number of comparisons that can be made as:

Optimal Binary Search Tree

In the third case, the number of comparisons is less because the height of the tree is less, so it's a balanced binary search tree.

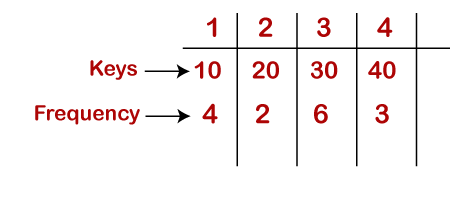
Till now, we read about the height-balanced binary search tree. To find the optimal binary search tree, we will determine the frequency of searching a key.

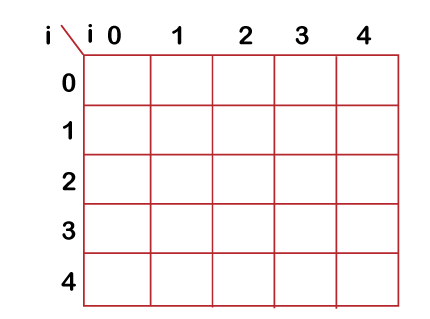
Let's assume that frequencies associated with the keys 10, 20, 30 are 3, 2, 5.

The above trees have different frequencies. The tree with the lowest frequency would be considered the optimal binary search tree. The tree with the frequency 17 is the lowest, so it would be considered as the optimal binary search tree.

Dynamic Approach

Consider the below table, which contains the keys and frequencies.





General formula for calculating the minimum cost is:

C[i,j] = min{c[i, k-1] + c[k,j]} + w(i,j)

I=1,J=3

I=0,j=2,

I=2,j=4

**First, we will calculate the values where j-i is equal to zero.**

When i=0, j=0, then j-i = 0

When i = 1, j=1, then j-i = 0

When i = 2, j=2, then j-i = 0

When i = 3, j=3, then j-i = 0

When i = 4, j=4, then j-i = 0

Therefore, c[0, 0] = 0, c[1 , 1] = 0, c[2,2] = 0, c[3,3] = 0, c[4,4] = 0

**Now we will calculate the values where j-i equal to 1.**

When j=1, i=0 then j-i = 1

When j=2, i=1 then j-i = 1

When j=3, i=2 then j-i = 1

When j=4, i=3 then j-i = 1

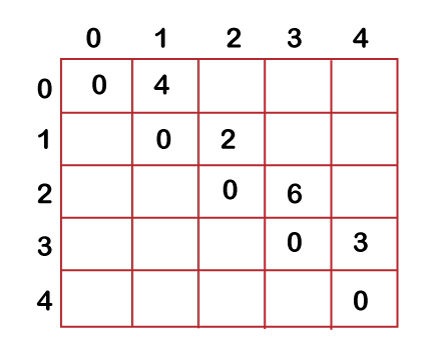
Now to calculate the cost, we will consider only the jth value.

The cost of c[0,1] is 4 (The key is 10, and the cost corresponding to key 10 is 4).

The cost of c[1,2] is 2 (The key is 20, and the cost corresponding to key 20 is 2).

The cost of c[2,3] is 6 (The key is 30, and the cost corresponding to key 30 is 6)

The cost of c[3,4] is 3 (The key is 40, and the cost corresponding to key 40 is 3)



**Now we will calculate the values where j-i = 2**

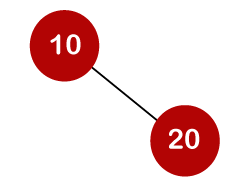
When j=2, i=0 then j-i = 2

When j=3, i=1 then j-i = 2

When j=4, i=2 then j-i = 2

In this case, we will consider two keys.

* When i=0 and j=2, then keys 10 and 20. There are two possible trees that can be made out from these two keys shown below:



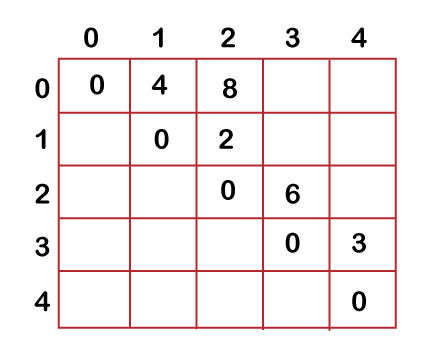
In the first binary tree, cost would be: 4\*1 + 2\*2 = 8

In the second binary tree, cost would be: 4\*2 + 2\*1 = 10

The minimum cost is 8; therefore, c[0,2] = 8

General formula for calculating the minimum cost is:

C[i,j] = min{c[i, k-1] + c[k,j]} + w(i,j)



* When i=1 and j=3, then keys 20 and 30. There are two possible trees that can be made out from these two keys shown below:

In the first binary tree, cost would be: 1\*2 + 2\*6 = 14

In the second binary tree, cost would be: 1\*6 + 2\*2 = 10

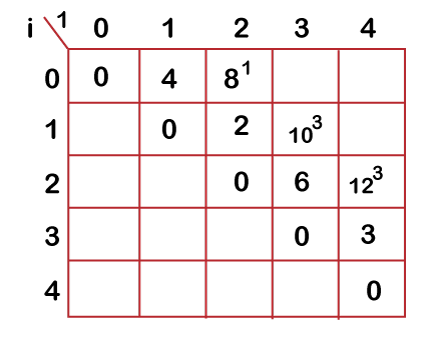
The minimum cost is 10; therefore, c[1,3] = 10

* When i=2 and j=4, we will consider the keys at 3 and 4, i.e., 30 and 40. There are two possible trees that can be made out from these two keys shown as below:

In the first binary tree, cost would be: 1\*6 + 2\*3 = 12

In the second binary tree, cost would be: 1\*3 + 2\*6 = 15

The minimum cost is 12, therefore, c[2,4] = 12



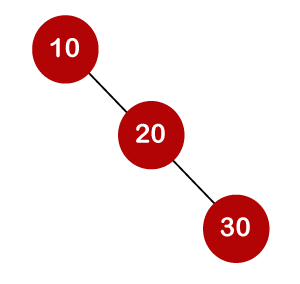
Now we will calculate the values when j-i = 3

When j=3, i=0 then j-i = 3

When j=4, i=1 then j-i = 3

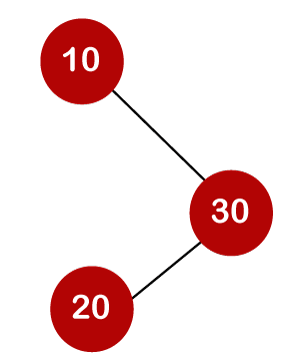
* When i=0, j=3 then we will consider three keys, i.e., 10, 20, and 30.

The following are the trees that can be made if 10 is considered as a root node.



In the above tree, 10 is the root node, 20 is the right child of node 10, and 30 is the right child of node 20.

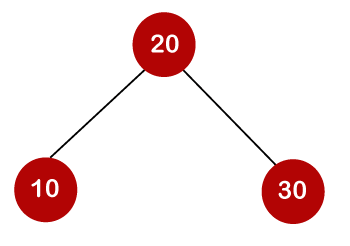
Cost would be: 1\*4 + 2\*2 + 3\*6 = 26



In the above tree, 10 is the root node, 30 is the right child of node 10, and 20 is the left child of node 20.

Cost would be: 1\*4 + 2\*6 + 3\*2 = 22

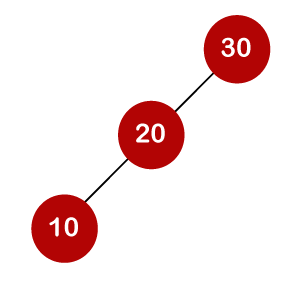
The following tree can be created if 20 is considered as the root node.



In the above tree, 20 is the root node, 30 is the right child of node 20, and 10 is the left child of node 20.

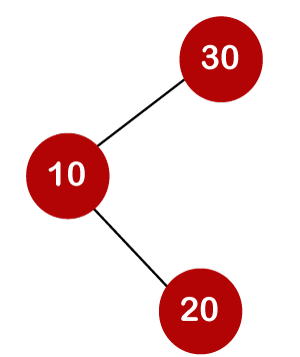
Cost would be: 1\*2 + 4\*2 + 6\*2 = 22

The following are the trees that can be created if 30 is considered as the root node.



In the above tree, 30 is the root node, 20 is the left child of node 30, and 10 is the left child of node 20.

Cost would be: 1\*6 + 2\*2 + 3\*4 = 22



In the above tree, 30 is the root node, 10 is the left child of node 30 and 20 is the right child of node 10.

Cost would be: 1\*6 + 2\*4 + 3\*2 = 20

Therefore, the minimum cost is 20 which is the 3rd root. So, c[0,3] is equal to 20.

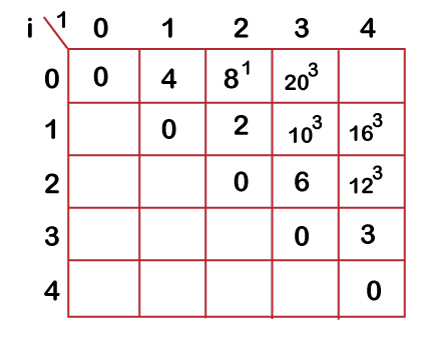
* When i=1 and j=4 then we will consider the keys 20, 30, 40

c[1,4] = min{ c[1,1] + c[2,4], c[1,2] + c[3,4], c[1,3] + c[4,4] } + 11

= min{0+12, 2+3, 10+0}+ 11

= min{12, 5, 10} + 11

The minimum value is 5; therefore, c[1,4] = 5+11 = 16



* **Now we will calculate the values when j-i = 4**

When j=4 and i=0 then j-i = 4

In this case, we will consider four keys, i.e., 10, 20, 30 and 40. The frequencies of 10, 20, 30 and 40 are 4, 2, 6 and 3 respectively.

w[0, 4] = 4 + 2 + 6 + 3 = 15

If we consider 10 as the root node then

C[0, 4] = min {c[0,0] + c[1,4]}+ w[0,4]

= min {0 + 16} + 15= 31

If we consider 20 as the root node then

C[0,4] = min{c[0,1] + c[2,4]} + w[0,4]

= min{4 + 12} + 15

= 16 + 15 = 31

If we consider 30 as the root node then,

C[0,4] = min{c[0,2] + c[3,4]} +w[0,4]

= min {8 + 3} + 15

= 26

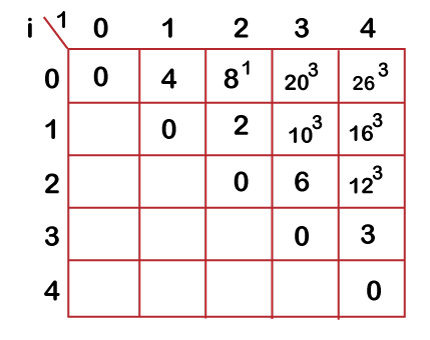
If we consider 40 as the root node then,

C[0,4] = min{c[0,3] + c[4,4]} + w[0,4]

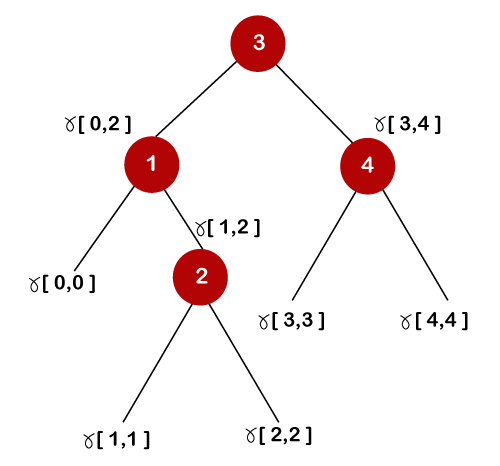
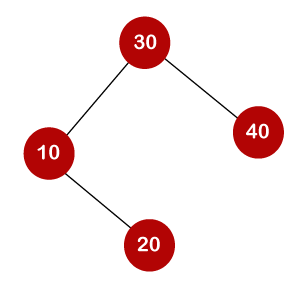
= min{20 + 0} + 15

= 35

In the above cases, we have observed that 26 is the minimum cost; therefore, c[0,4] is equal to 26.



The optimal binary tree can be created as:

General formula for calculating the minimum cost is:

C[i,j] = min{c[i, k-1] + c[k,j]} + w(i,j)

OR

https://www.kodnest.com/free-online-courses/algorithm-2/lessons/all-pairs-shortest-paths/topic/optimal-binary-search-trees/

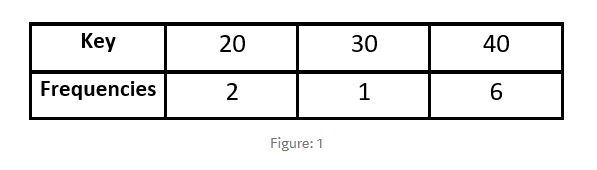
# Optimal Binary Search Trees

A Binary Search Tree (BST) is a tree where the key values are stored in the internal nodes. The external nodes are null nodes. The keys are ordered lexicographically, i.e. for each internal node all the keys in the left sub-tree are less than the keys in the node, and all the keys in the right sub-tree are greater.

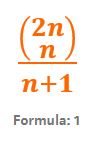
When we know the frequency of searching each one of the keys, it is quite easy to compute the expected cost of accessing each node in the tree. An optimal binary search tree is a BST, which has minimal expected cost of locating each node.

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.

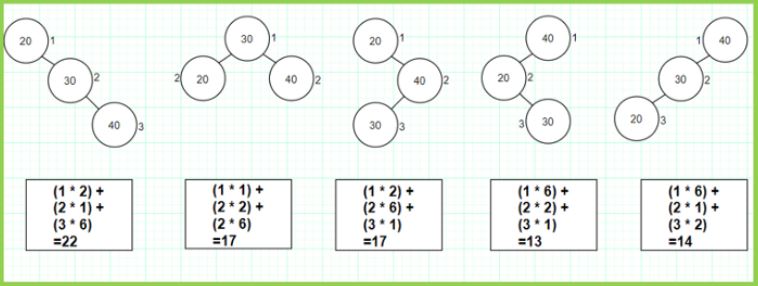
In many applications the cost of searching is very important. So, it is required that the overall cost of searching should be as less as possible. And we know that search time of BST is more than the **Balanced Binary Search Tree,** as Balanced Binary Search tree has less number of levels than the BST. 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.



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.

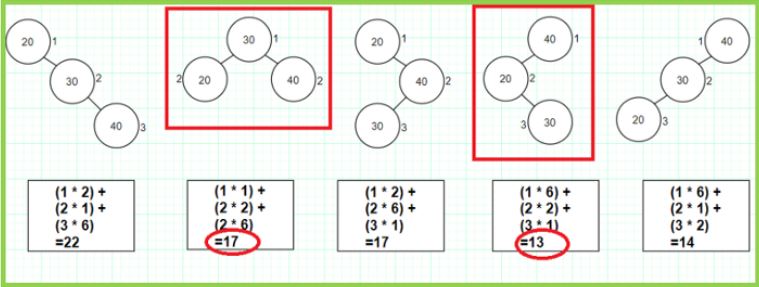


So following are the various possible Binary Search Trees of the above data. And also the overall cost for searching for each BST.



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.

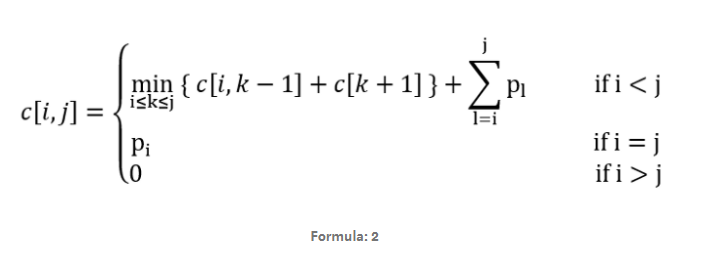
As we can see in Figure 2 there are various 5 types of arrangements are possible. And as we have discussed earlier that, it is possible to further reduce the cost of Balanced BST which is specified in following figure 3.



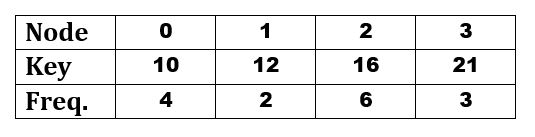
As it is shown in above figure that 2nd BST is balanced and the 4th BST is not balanced, though it’s cost is less than the cost of Balanced BST and its cost is the least among all, so it is our Optimal Binary Search Tree for the given data in Figure 1. So here our idea to generate the Optimal Binary 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:- correct error replace c[k+1] by c[k+1,j]

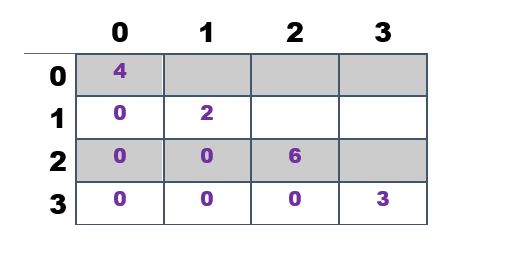


where c[ i , j ] in which c stands for cost, “ pi ” stands for frequency of node i. We will use matrix form to solve this problem.Problem Statement is as following and we have to find Optimal BST for that:

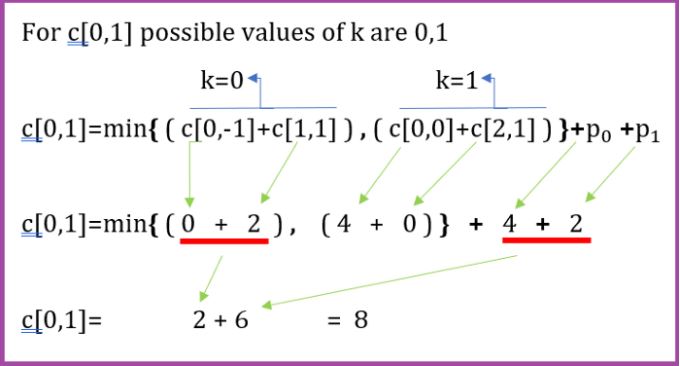


**SOLUTION**

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;



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

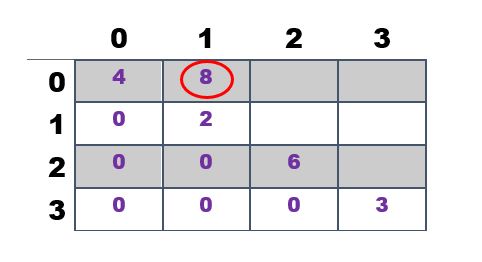


General formula for calculating the minimum cost is:

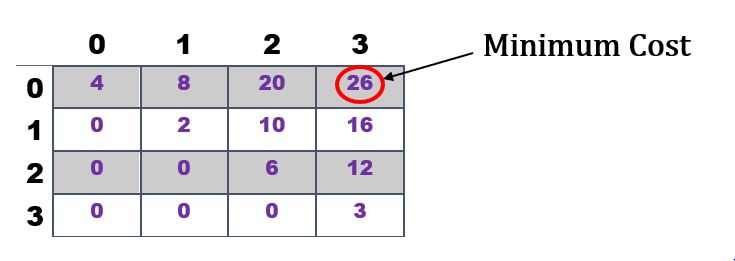
C[i,j] = min{c[i, k-1] + c[k+1,j]} + w(i,j) ,i<=k<=j.

K=0

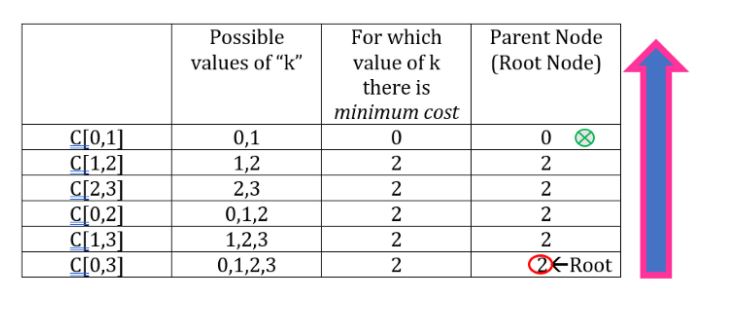
and the table after this step will look like…



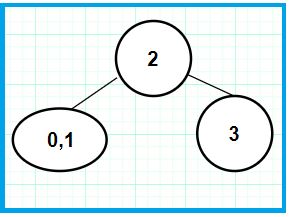
**Step: 3**:: Do same for all elements like c[1,2],c[2,3],c[0,2],c[1,3],c[0,3] and check your answer with the following table……..



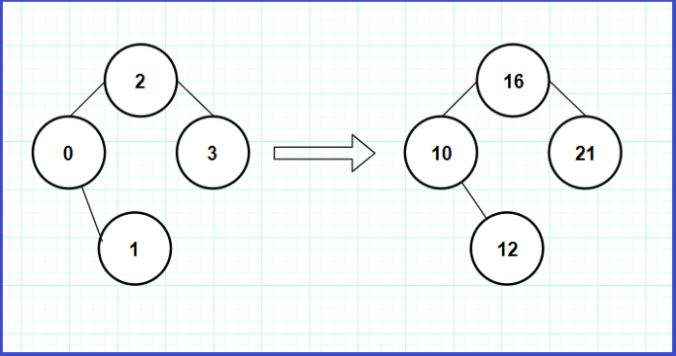
Step:4 :: So now we want to arrange the nodes in such a way that they will cost the minimum value which we derived previously. For that we have to choose root(parent node)in every group of nodes. See in step 2: for node 0 and node 1 the parent node will be decided by the value of K for which we took 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.



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.

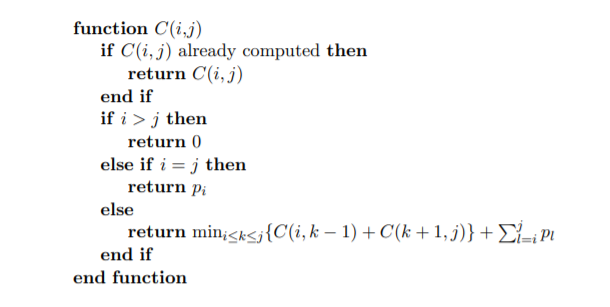


So now there is one question that “ Which node is parent node between node 0 and node 1??” For that again Go from Bottom →Up and see that node 0 is parent node , because when we were computing value of c[0,1] at that time we came to know that between 0,1 node 0 is parent node. And as in our example the 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:



OR <https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_optimal_cost_binary_search_trees.htm>

## Problem Statement (TRAVELING SALES MAN PROBLEM ) (<https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm>)

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

## Solution

Travelling salesman problem is the most notorious computational problem. We can use brute-force approach to evaluate every possible tour and select the best one. For **n** number of vertices in a graph, there are **(*n* - 1)!** number of possibilities.

Instead of brute-force using dynamic programming approach, the solution can be obtained in lesser time, though there is no polynomial time algorithm.

Let us consider a graph ***G = (V, E)***, where ***V*** is a set of cities and ***E*** is a set of weighted edges. An edge ***e(u, v)*** represents that vertices ***u*** and ***v*** are connected. Distance between vertex ***u*** and ***v*** is ***d(u, v)***, which should be non-negative.

Suppose we have started at city ***1*** and after visiting some cities now we are in city ***j***. Hence, this is a partial tour. We certainly need to know ***j***, since this will determine which cities are most convenient to visit next. We also need to know all the cities visited so far, so that we don't repeat any of them. Hence, this is an appropriate sub-problem.

For a subset of cities ***S Є {1, 2, 3, ... , n}*** that includes ***1***, and ***j Є S***, let ***C(S, j)*** be the length of the shortest path visiting each node in **S** exactly once, starting at ***1*** and ending at ***j***.

When |***S***| > 1, we define ***C(S, 1)*** = ∝ since the path cannot start and end at **1**.

Now, let express **C(S, j)** in terms of smaller sub-problems. We need to start at ***1*** and end at **j**. We should select the next city in such a way that

C(S,j)=minC(S−{j},i)+d(i,j)wherei∈Sandi≠jc(S,j)=minC(s−{j},i)+d(i,j)wherei∈Sandi≠jC(S,j)=minC(S−{j},i)+d(i,j)wherei∈Sandi≠jc(S,j)=minC(s−{j},i)+d(i,j)wherei∈Sandi≠j

**Algorithm: Traveling-Salesman-Problem**

C ({1}, 1) = 0

for s = 2 to n do

for all subsets S Є {1, 2, 3, … , n} of size s and containing 1

C (S, 1) = ∞

for all j Є S and j ≠ 1

C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}

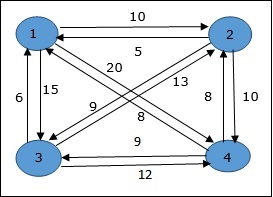
Return minj C ({1, 2, 3, …, n}, j) + d(j, i)

## Analysis

There are at the most 2n.n2..n.n sub-problems and each one takes linear time to solve. Therefore, the total running time is O(2n.n2)O(2n.n2).

## Example

In the following example, we will illustrate the steps to solve the travelling salesman problem.



From the above graph, the following table is prepared.

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

### S = Φ

### Cost(i,s)=min{Cost(j ,s–(j))+d[i,j]}

### I=1

### S={2,3,4}

### K=0

### K=0

Cost(2,Φ,1)=d(2,1)=5

Cost(2,Φ,1)=d(2,1)=5Cost(2,Φ,1)=d(2,1)=5Cost(2,Φ,1)=d(2,1)=5

Cost(3,Φ,1)=d(3,1)=6

Cost(3,Φ,1)=d(3,1)=6Cost(3,Φ,1)=d(3,1)=6Cost(3,Φ,1)=d(3,1)=6

Cost(4,Φ,1)=d(4,1)=8

Cost(4,Φ,1)=d(4,1)=8Cost(4,Φ,1)=d(4,1)=8Cost(4,Φ,1)=d(4,1)=8

### S = 1 , j or k same here

Cost(i,s)=min{Cost(j ,s–(j))+d[i,j]}

{2,3,4}-{2}

, Cost(2,{3},1)=d[2,3]+Cost(3,Φ,1)=9+6=15

cost(2,{3},1)=d[2,3]+cost(3,Φ,1)=9+6=15Cost(2,{3},1)=d[2,3]+Cost(3,Φ,1)=9+6=15cost(2,{3},1)=d[2,3]+cost(3,Φ,1)=9+6=15

Cost(2,{4},1)=d[2,4]+Cost(4,Φ,1)=10+8=18cost(2,{4},1)=d[2,4]+cost(4,Φ,1)=10+8=18Cost(2,{4},1)=d[2,4]+Cost(4,Φ,1)=10+8=18cost(2,{4},1)=d[2,4]+cost(4,Φ,1)=10+8=18

Cost(3,{2},1)=d[3,2]+Cost(2,Φ,1)=13+5=18cost(3,{2},1)=d[3,2]+cost(2,Φ,1)=13+5=18Cost(3,{2},1)=d[3,2]+Cost(2,Φ,1)=13+5=18cost(3,{2},1)=d[3,2]+cost(2,Φ,1)=13+5=18

Cost(3,{4},1)=d[3,4]+Cost(4,Φ,1)=12+8=20cost(3,{4},1)=d[3,4]+cost(4,Φ,1)=12+8=20Cost(3,{4},1)=d[3,4]+Cost(4,Φ,1)=12+8=20cost(3,{4},1)=d[3,4]+cost(4,Φ,1)=12+8=20

Cost(4,{3},1)=d[4,3]+Cost(3,Φ,1)=9+6=15cost(4,{3},1)=d[4,3]+cost(3,Φ,1)=9+6=15Cost(4,{3},1)=d[4,3]+Cost(3,Φ,1)=9+6=15cost(4,{3},1)=d[4,3]+cost(3,Φ,1)=9+6=15

Cost(4,{2},1)=d[4,2]+Cost(2,Φ,1)=8+5=13cost(4,{2},1)=d[4,2]+cost(2,Φ,1)=8+5=13Cost(4,{2},1)=d[4,2]+Cost(2,Φ,1)=8+5=13cost(4,{2},1)=d[4,2]+cost(2,Φ,1)=8+5=13

### S = 2

Cost(2,{3,4},1)=⎧⎩⎨d[2,3]+Cost(3,{4},1)=9+20=29d[2,4]+Cost(4,{3},1)=10+15=25=25Cost(2,{3,4},1){d[2,3]+cost(3,{4},1)=9+20=29d[2,4]+Cost(4,{3},1)=10+15=25=25Cost(2,{3,4},1)={d[2,3]+Cost(3,{4},1)=9+20=29d[2,4]+Cost(4,{3},1)=10+15=25=25Cost(2,{3,4},1){d[2,3]+cost(3,{4},1)=9+20=29d[2,4]+Cost(4,{3},1)=10+15=25=25

Cost(3,{2,4},1)=⎧⎩⎨d[3,2]+Cost(2,{4},1)=13+18=31d[3,4]+Cost(4,{2},1)=12+13=25=25Cost(3,{2,4},1){d[3,2]+cost(2,{4},1)=13+18=31d[3,4]+Cost(4,{2},1)=12+13=25=25Cost(3,{2,4},1)={d[3,2]+Cost(2,{4},1)=13+18=31d[3,4]+Cost(4,{2},1)=12+13=25=25Cost(3,{2,4},1){d[3,2]+cost(2,{4},1)=13+18=31d[3,4]+Cost(4,{2},1)=12+13=25=25

Cost(4,{2,3},1)=⎧⎩⎨d[4,2]+Cost(2,{3},1)=8+15=23d[4,3]+Cost(3,{2},1)=9+18=27=23Cost(4,{2,3},1){d[4,2]+cost(2,{3},1)=8+15=23d[4,3]+Cost(3,{2},1)=9+18=27=23Cost(4,{2,3},1)={d[4,2]+Cost(2,{3},1)=8+15=23d[4,3]+Cost(3,{2},1)=9+18=27=23Cost(4,{2,3},1){d[4,2]+cost(2,{3},1)=8+15=23d[4,3]+Cost(3,{2},1)=9+18=27=23

### S = 3

Cost(1,{2,3,4},1)=⎧⎩⎨⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪⎪d[1,2]+Cost(2,{3,4},1)=10+25=35d[1,3]+Cost(3,{2,4},1)=15+25=40d[1,4]+Cost(4,{2,3},1)=20+23=43=35cost(1,{2,3,4}),1)d[1,2]+cost(2,{3,4},1)=10+25=35d[1,3]+cost(3,{2,4},1)=15+25=40d[1,4]+cost(4,{2,3},1)=20+23=43=35Cost(1,{2,3,4},1)={d[1,2]+Cost(2,{3,4},1)=10+25=35d[1,3]+Cost(3,{2,4},1)=15+25=40d[1,4]+Cost(4,{2,3},1)=20+23=43=35cost(1,{2,3,4}),1)d[1,2]+cost(2,{3,4},1)=10+25=35d[1,3]+cost(3,{2,4},1)=15+25=40d[1,4]+cost(4,{2,3},1)=20+23=43=35

The minimum cost path is 35.

Start from cost **{1, {2, 3, 4}, 1}**, we get the minimum value for **d [1, 2]**. When **s = 3**, select the path from 1 to 2 (cost is 10) then go backwards. When **s = 2**, we get the minimum value for **d [4, 2]**. Select the path from 2 to 4 (cost is 10) then go backwards.

When **s = 1**, we get the minimum value for **d [4, 3]**. Selecting path 4 to 3 (cost is 9), then we shall go to then go to **s = Φ** step. We get the minimum value for **d [3, 1]** (cost is 6).

Values