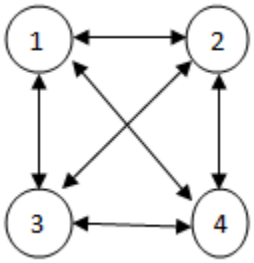
Travelling sales person problem: Here some weighted graph is given and this the cost adjacency matrix for the graph



There is an edge bi-directional shown here. These are actually parallel edges but instead of avoiding too many edges, single edge with either direction is given here.
So, the cost is an either direction is different.

Now the problem is we have to select any one of the vertex as the starting vertex and find out a path such that it is giving through the entire vertex and returning back to the starting vertex.

"So, it should form a cycle covering all the vertices".

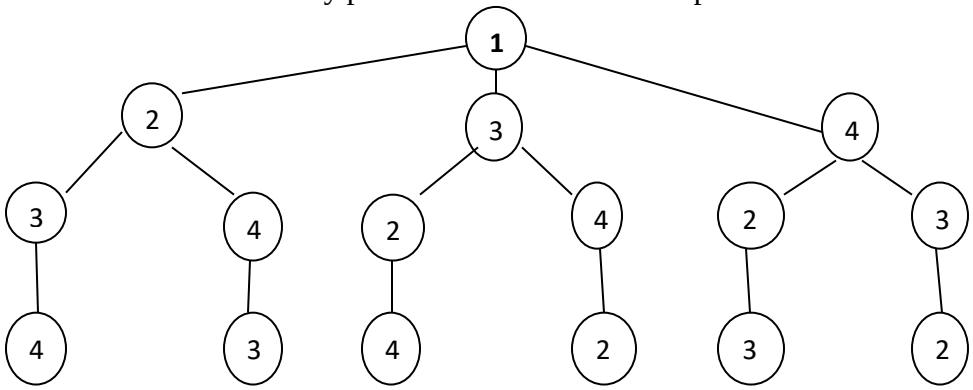
Let us see how Brute force approach and Dynamic programming can be used for solving the travelling sales person.

Brute force approach:

Example:

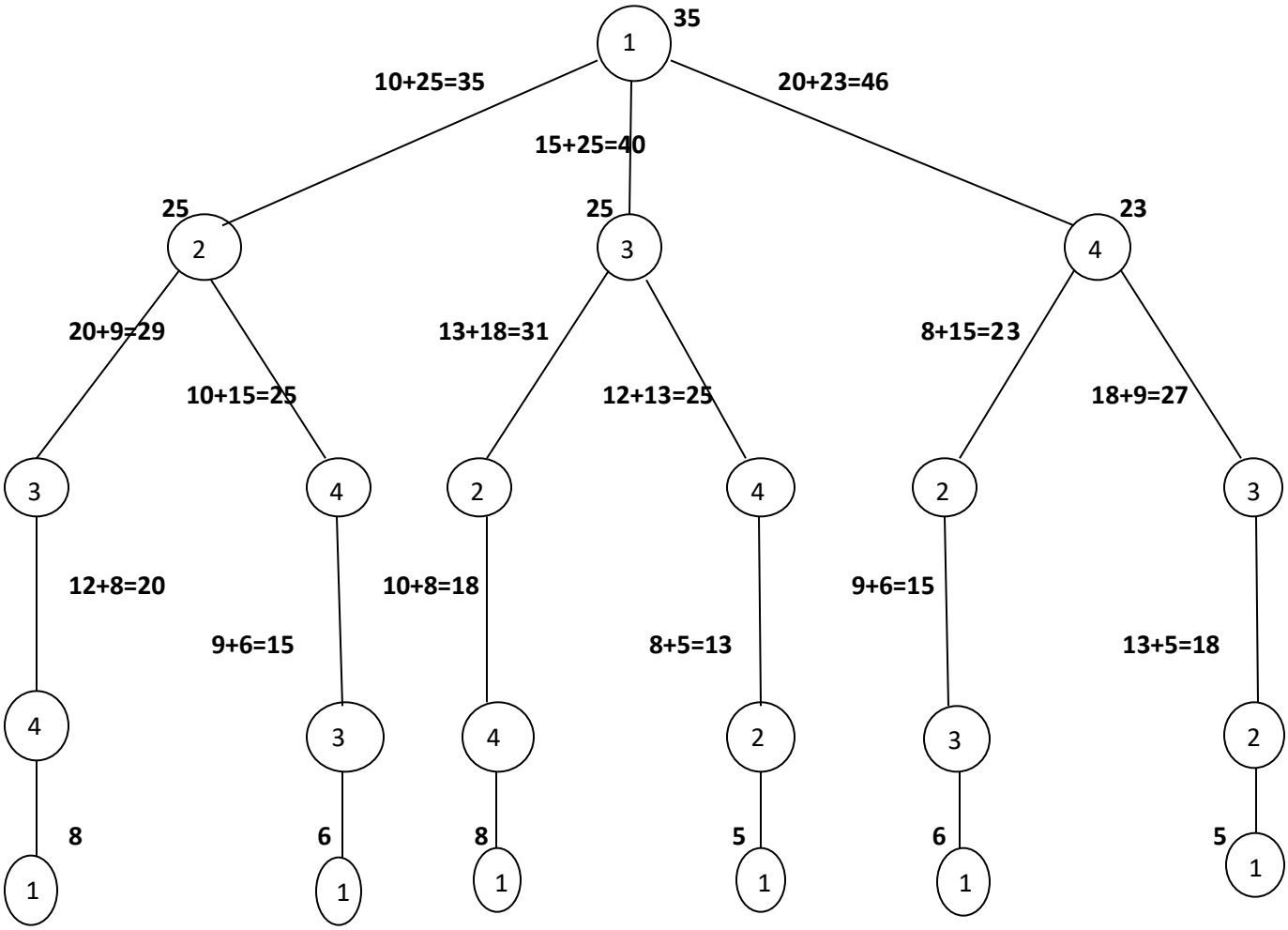
	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

First we need to check how many possible routes that a sales person can take.



- After that he has to return back to the starting vertex 1 only.

Solution:



- The cost of shortest route is 35 using Brute force approach.

Now, let us see how Dynamic programming works

Formulae: $g(i,s) = \min_{k \in s} \{c_{i\,k} + g(k, s - \{k\})\}$

- Cost [i] some starting vertex to set of vertices then minimum of k belongs to set of vertices. Then this c (i , k) + cost of k to set and from this set i should subtract the k^{th} vertex.

Solution using Dynamic programming,

g (2,Ø) = 5 g (3, Ø) = 6 g(4, Ø) = 8

Now, set generated with that

g (2, {3}) = 15 g (2,{4}) = 18 g (3,{2}) = 18 g (3,{4}) = 20

g(4,{2}) = 13 g(4,{3}) = 15

g(2,{3,4}) = 25 g(3,{2,4}) = 25 g(4,{2,3}) = 23

Step 4: g (1{2,3,4}) = min {C_{12} + g(2,{3,4}) , C_{13} + g(3,{2,4}) , C_{14} + g(4, {2,3}) }

= 10+25 = 35 15+25 = 40 20+23 = 43

Therefore, the minimum path is : 1 → 2 → 4 → 3

So, finally we will solve the problem by taking sequence of decisions.

Rod Cutting Problem:

- Cutting the Rod to maximize the profit.
- Given a rod of certain length and given the process of different lengths which is selling in the market.
- How do you cut the rod? So that you can maxima as a profit.

Example: **length = 5 1 = 2, 2 = 5, 3 = 7, 4 = 8**

- In this example we have a Rod of length 5 and then we have a length and value for each length.
- In this we have two possible solutions i.e., (2 , 3) = 5+7 =12 (1 , 2, 2) = 2+5+5 = 12
- By using dynamic programming we can solve this problem

Solution:

P	L
2	1
5	2
7	3
8	4

0	1	2	3	4	5
0	2	4	6	8	10
0	2	5	7	9	12
0	2	5	7	9	12
0	2	5	7	9	12

Algorithm:

```
if ( j >= 1)
{
T[i] [ j] = max ( t[i-1] [ j] , Val [i] + T [i] [ j-i] );
Else
{
  T[i] [ j] =  T[i-1] [ j];
}
}
```

Elements of Dynamic Programming:

Dynamic Programming posses two important elements which are given below

1. **Overlapping sub problem:** One of the main characteristics is to split the problem into sub problem, as similar as divide and conquer approach. The overlapping sub problem is found in that problem where bigger problems share the smaller problem. However unlike divide and conquer there are many sub problems in which overlap cannot be treated distinctly or independently. Basically, there are two ways for handling the overlapping sub problems.

   a. Top down approach: It is also termed as memoization technique. In this, the problem is broken into sub problem and these sub problems are solved ant he solutions are remembered, in case if they need to solved in future. This means that the values are stored in a data structure, which will help us to reach them efficiently when the same problem will occur during the program execution.

   b. Bottom up approach: It is also termed as tabulation technique. In this, all sub problems are needed to be solved in advance and then used to build up a solution to the larger problem.

2. **Optimal sub structure:** It implies that the optimal solution can be obtained from the optimal solution of its sub problem. So optimal substructure is simply an optimal selection among all the possible substructures that can help to select the best structure of the same kind to exist.

Matrix chain multiplication: Matrix chain multiplication is an optimization problem that can be solved using dynamic programming. Given a sequence of matrices, the goal is to find out the most efficient way to multiply these matrices.

So, let us start with matrix multiplication. First of all if you have matrix 2 matrices and you want to multiply them and their dimensions are suppose (5 * **4 4** * 3)

The condition for multiplying two matrices the number of columns of the first matrix should be equal to number of rows of second matrix then only multiplication can be done.

If I write other way B*A then their dimensions (4 * **3 5** * 4) they cannot be multiplied because they are not same.

However, the order in which the product is parenthesized affects the number of simple arithmetic operations needed to complete the product that is the computational complexity. For example, if A is a 5 * 4 matrix, B is a 4 * 3 matrix, then

Computing AB = C needs (5 * 4 4 * 3) = 5 * 4 * 3 =60. Generating C matrix I have to make 60 multiplications.

Example: A1 A2 A3 A4

5 * **4 4** * **6 6** * **2 2** * 7

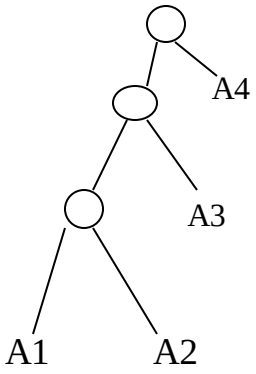
Here 4 matrices are given and dimensions also given. They are arranged such that they can be multiplied because they are same.

But, the question is not to multiply and get the answer. I have to take a pair at a time and multiply so the question is which pair should select such that the cost of multiplying all of them is Minimum.

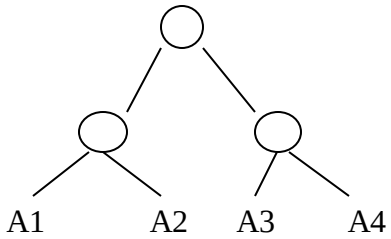
So, the problem is not to multiply them the problem is to know how to multiply them.

This is a matrix chain multiplication. I should parenthesize them.

((A1 * A2) A3) A4)



(A1 * A2) (A3 * A4)



So, how to try all possibilities it is difficult for every problem even they may be missing some possibilities also so without missing anything how to get the perfect answer by trying all possibilities.

Solution: A1 A2 A3 A4

5 * **4 4** * **6 6** * **2 2** * 7

As dynamic programming uses tabulation method. In tabulation method we will follow bottom up approach. We should take smaller values and fill the table first.

Formulae : $m[i , j] = \min \{ m[i , k] + m [k+1, j] + d_{i-1} * d_k * d_j \}$

Step 1: A1 is not multiplied with anything. So the cost of multiplication is Zero

A1 = m [1 , 1] = 0 Similarly, A2 = m [2 , 2] = 0 , A3 = m[3 , 3] = 0 , A4 = m[4 , 4] = 0

Step 2: i) Let us find out m [1 , 2]

A1 is 5 * 4 A2 is 4 * 6 = 5 * 4 * 6 = 120

ii) Let us find out m[2 , 3]

A2 is 4 * 6 A3 is 6 * 2 = 4 * 6 * 2 = 48

iii) Let us find out m[3 , 4]

A3 is 6 * 2 A4 is 2 * 7 = 6 * 2 * 7 = 84

Step 3: i) Let us find out m[1 , 3]

A1 (A2 * A3)	(A1 * A2) A3
5 * **4** **4** * 6 6 * **2**	**5** * 4 4 * **6** **6** * **2**
m[1,1] + m[2,3] + 5*4*2	m[1,2] + m[3,3] + 5*6*2
0 + 48 + 40 = **88**	120 + 0 + 60 = 180

ii) Let us find out m[2 , 4]

A2 (A3 * A4)	(A2 * A3) A4
4 * **6** **6** * 2 2 * **7**	**4** * 6 6 * **2** **2** * **7**
m[2,2] + m[3,4] + 4*6*7	m[2,3] + m[4,4] + 4*2*7
0 + 84 +168 = 252	48 + 0 + 56 =**104**

Step 4: Let us find out m[1 , 4]

A1 (A2 A3 A4)	(A1 A2) (A3 A4)	(A1 A2 A3) A4
5***4** **4***6 6*2 2***7**	**5***4 4***6** **6***2 2***7**	**5***4 4*6 6***2** **2*****7**

m[1,4] = min { m[1,1] + m[2,4] + 5*4*7 , m[1,2] + m[3,4] + 5*6*7 , m[1,3] + m[4,4] + 5*2*7 }

= min { 0 + 104 +140 , 120 + 84 + 210 , 88 + 0 + 70 }

= min{ 244 , 314 , **158** }

	1	**2**	**3**	**4**
1	0	120	88	158
2		0	48	104
3			0	84
4				0

	1	**2**	**3**	**4**
1		1	1	3
2			2	3
3				3
4				

Time complexity: $n(n-1) / 2 = n^2$

But each element how we are getting we are calculating all and finding the minimum. So how much time it takes n. So n2 *** n = n^3**

Longest Common Subsequence:

The Longest common subsequence (LCS) problem is the problem of finding the longest sequence which exists in both the given strings.

Common Subsequence: Suppose, X and Y are tow subsequences over a finite set of elements. We can say that Z is a common subsequence of X and Y, if Z is a subsequence of both X and Y.

Longest common subsequence: if a set of sequences are given, the LCS problem is to find a common subsequence of all the sequences that is of maximal length.

Example: S = abcd (no of characters in a string)

Now, we are supposed to find out the subsequence of this string.

S = ab, ac, ad, bc, bd, cd, abc, abd, acd, abcd = $2^n = 2^4 = 16$ number of subsequences is possible.

S1 = abcd

S2 = bcd find the common subsequence

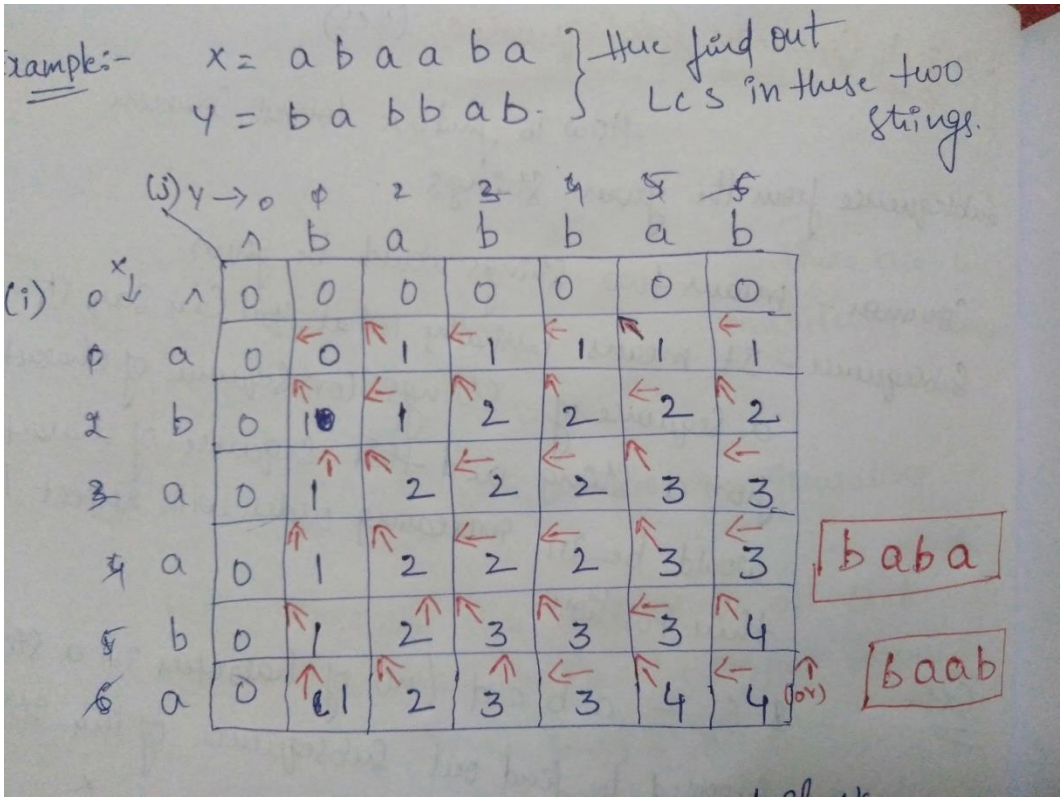
S3 = bcd, cd, bc (bcd is the longest common subsequence)

Formulae: if(x[i] = = y[j]), c[i] [j] = 1+C [i-1] [j-1]

Otherwise C[i] [j] = max (c[i] [j-1], c [i-1] [j])

Example: X = a b a a b a

Y = b a b b a b



- If characters are same fill with 1 and check diagonally and choose the max value to place it.
- If characters are not same fill with 0 and check left and upper values and choose the max value to place it.
- Checkout the last box the intersection of last row and column i.e., 4 then the length of common subsequence would be "4"
- The sequence of the characters are **b a b a** and **b a a b**

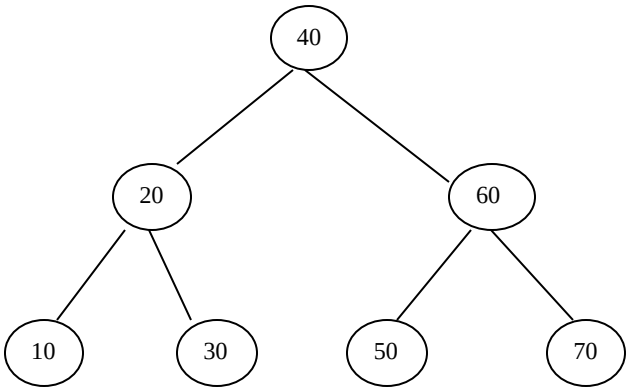
Optimal Binary Search Tree:

Let's start with binary search tree, if the set of keys are given here and the purpose is to search the keys.

I want to search some element and there are different search methods like linear search, Binary search and also have a data structure.

Can we utilize a binary search tree for searching an element? Yes let's take an example.

Keys: 10, 20, 30, 40, 50, 60, 70 (Arrange the elements in the form of a binary tree)



Height of the tree is **log n**

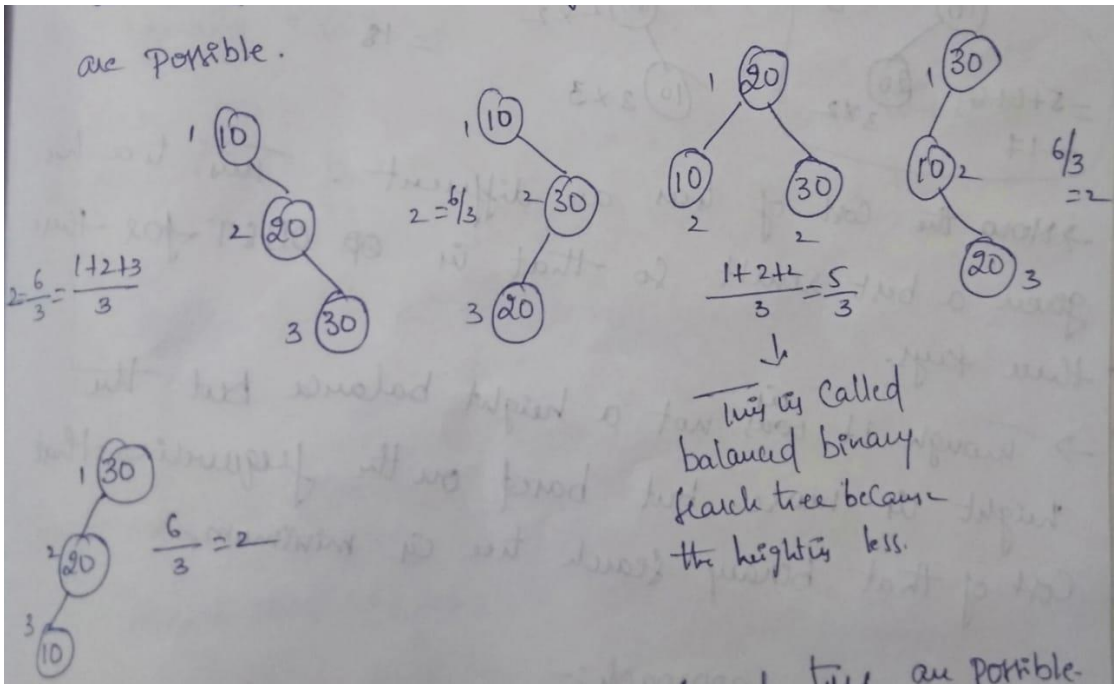
The benefit of arrange this type of binary tree is that it is useful for searching.

The time taken for searching a key element, the maximum time will be if the element is in the leaf. The height of the binary tree is minimum i.e., log n.

Now, I will show you for a given set of element and how many different ways are possible for a given set.

Example: Keys: 10, 20, 30 (let's see how many different binary search trees are possible)

There are five different binary search trees are possible. So for n keys how many different bst's are possible $2nc_n / n+1 = 2*3c_3 / 3+1 = 5$

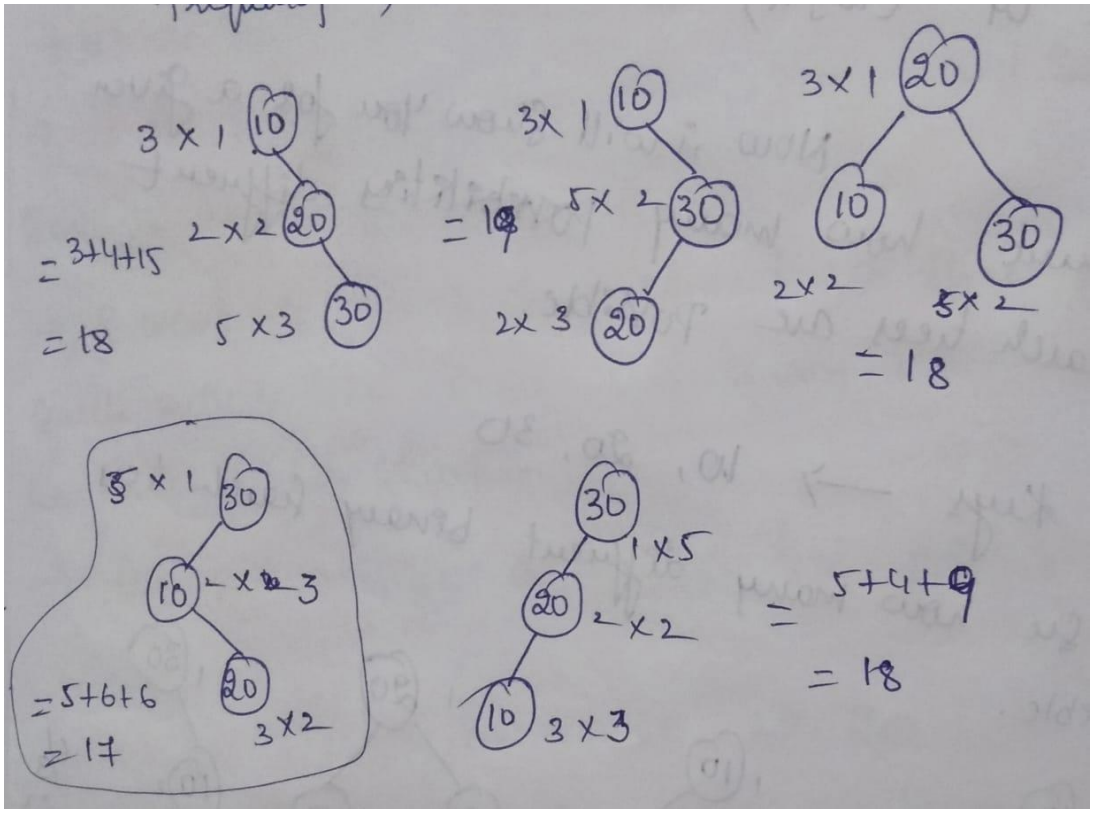


Now we will see what the cost of searching it in each tree is.

Now let us come to know optimal binary search tree. Here, we also consider the frequency of searching of those keys.

Keys: 10, 20, 30

Frequency: 3, 2, 5



Now the cost of trees are different, this tree has given the given a best result. Though it is not a height balanced but the height is more. Based on the frequencies the cost of that binary search tree is Minimum.

Dynamic programming approach:

1 2 3 4

Keys: 10, 20, 30, 40

Frequency: 4 2 6 3

i

j

	0	**1**	**2**	**3**	**4**
0	0	4	8^{1}	20^{3}	26^{3}
1		0	2	10^{3}	16^{3}
2			0	6	12^{3}
3				0	3
4					0

Formulae: c[i,j] = min{ c[i, k-1] + c[k, j]} + w[i,j]

i<k<=j

Step:1 Now first of all we will find out those values which are $l = j - i$

For which values it will be $j - i = 0$ **c [0,0] = 0-0 =0, c[1-1] = 1-1 = 0, c[2-2] = 2-2 = 0, c[3,3] = 3-3 = 0, c[4,4] = 4-4 =0.**

Step2: Now I will find out this diagonal. $l = j - i = 1$

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

c[0,1] = 4 c[1,2] = 2 c[2,3] = 6 c[3,4] =3

Key: 10 20 30 40

Frequency: 4 2 6 3

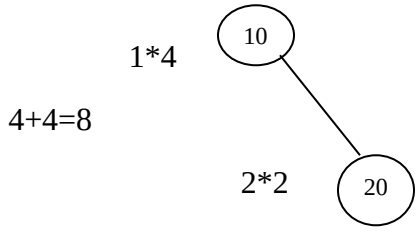
Step3: $l = j - i = 2$ c[0,2] = 2-0 = 2, c[1,3] = 3-1 = 2, c[2,4] = 4-2 = 2

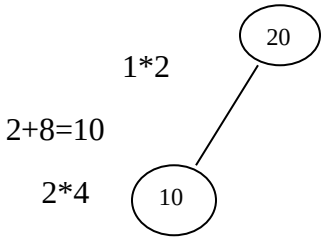
i) C[0 , 2] (0----1-----2)

K: 10 20

F: 2 4

Let us see using these two keys we can find out the minimum cost of binary search tree or optimal binary search tree they can generate by using these two keys.

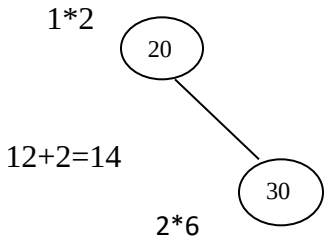


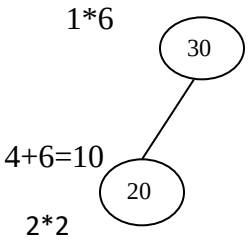


ii) c[1 , 3] (1----2----3)

k: 20 30

f: 2 6

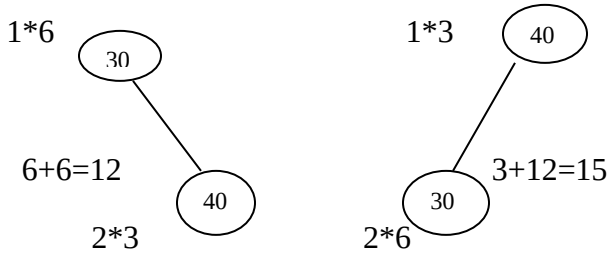




iii) c[2 , 4] (2----3----4)

k: 30 40

f: 6 3



Step4: i) l = j – i =3 c[0 , 3] (0---1---2---3) w[i,j] = [0,3]

K: 10 20 30 w[0,3] = (4+2+6 = 12)

F: 4 2 6

C[0,3] = min { c[0,0] + c[1,3] +w[0,3] , c[0,1] + c[2,3] + w[0,3] , c[0,2] + c[3,3] + w[0,3] }

= min{ 0+10+12 , 4+6+12 , 8+0+12 } = min{ 22 , 22 , **20** }

ii) c[1 , 4] (1---2---3---4) w[i , j] = [1,4]

K: 20 30 40

F: 2 6 3 w[1,4] = (2+6+3 = 11)

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

= min { 0+12+11 , 2+3+11, 10+0+11} = min {23 , 16 , 21}

Step5: l = j – i = 4 , c[0,4] = 4-0 =4

C[0 , 4] (0---1---2---3---4) w[i , j] = [0,4]

K : 10 20 30 40

F: 4 2 6 3 w[0,4] = (4+2+6+3 = 15)

C[0,4] = min{c[0,0] + c[1,4] + 15 , c[0,1] +c[2,4] +15, c[0,2] +c[3,4]+15 , c[0,3]+c[4,4]+15}

= min{0+16+15, 4+12+15, 8+3+15, 20+0+15} = min{31, 31, 26, 35}

Matroid: Matroid is an ordered pair M = (S,I)

- S is a finite set

- I is non empty family of subsets of S, called the independent subsets of S, such that if B ∈ I and A subset B, then A ∈ I. I is hereditary.
- If A ∈ I & B ∈ I, and |A| < |B|, then there exits some element x∈ B-A such that A∪{x} ∈ I. We say M is satisfies the exchange property.

Activity selection problem:

Algorithm:

N = s.length

A={a1}

K=1

For m=2 to n

    if s[m.=f[k]

    A = A ∪ {a_m}

    K = m

return A