Optimal Binary Search Tree (OBST)

Dynamic Programming

Dr. S. Kannimuthu

Preliminaries

BST

- One of the most important data structure in computer science.
- Application: Implementing a dictionary
  - Set of elements with the operations of searching, insertion and deletion

Preliminaries

If the probabilities of searching for elements of a set are known, It is basic to pose a question about an OBST for which the average number of comparisons in a search is the smallest possible.

Example-1

Input: keys[] = {10, 12}, freq[] = {34, 50}

There can be following two possible BSTs

```
    10              12
     \             /
      12          10
     I            II
```

Frequency of searches of 10 and 12 are 34 and 50 respectively.

The cost of tree I is 34*1 + 50*2 = 134

The cost of tree II is 50*1 + 34*2 = 118

Example-2

Input: keys[] = {10, 12, 20}, freq[] = {34, 8, 50}

There can be following possible BSTs

```
   10           12          20      10         20
    \          /  \        /          \       /
     12      10    20    12            20   10
       \                /            /        \
        20            10           12          12
   I           II          III        IV        V
```

Among all possible BSTs, cost of the fifth BST is minimum.

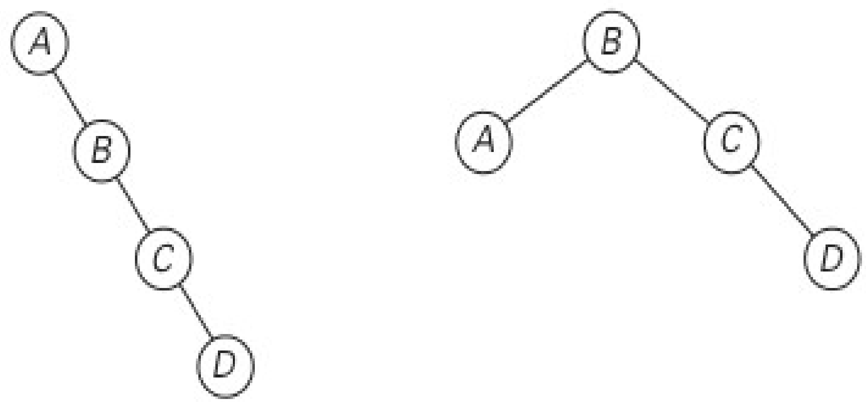
Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142

Example (in other ways)

- Another Example:
- Consider four keys A, B, C, and D to be searched for with probabilities 0.1, 0.2, 0.4, and 0.3, respectively.

Example

Two out of 14 possible binary search trees with keys A, B, C, and D.



The average number of comparisons in a successful search in the first of these trees is 0.1 . 1 + 0.2 . 2 + 0.4 .3 + 0.3 . 4 = 2.9

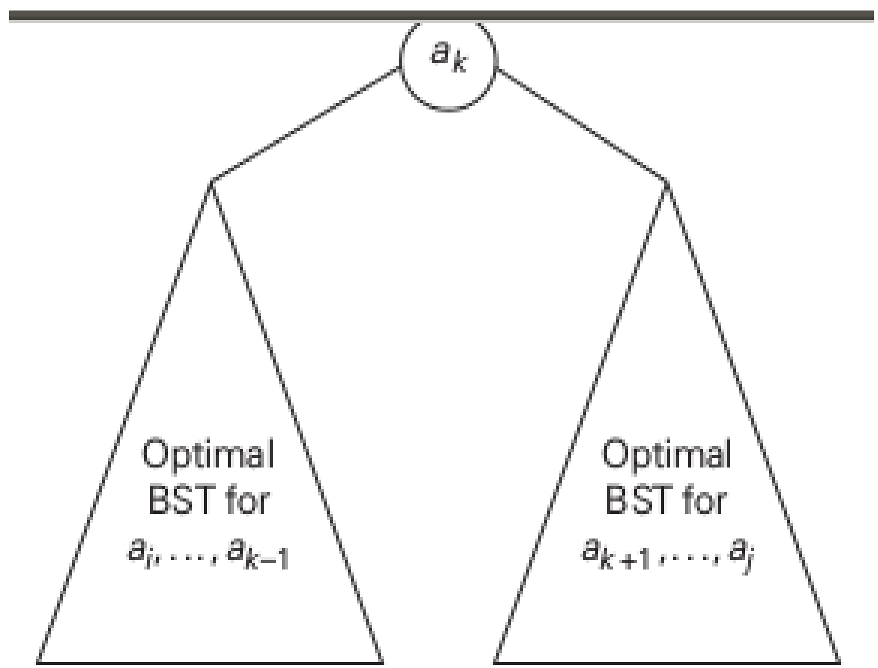
And for the second one it is 0.1 . 2 + 0.2 . 1 + 0.4 . 2 + 0.3 . 3 = 2.1.

Important to note

- The total number of binary search trees with n keys is equal to the nth Catalan number:

$$\frac{1}{n+1} \boxed{{}_{(2n)}C_{(n)}}$$

How to solve?



Forming Recurrences

$$C(i, j) = \min_{i \le k \le j} \{p_k \cdot 1 + \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \text{ in } T_i^{k-1} + 1)$$

$$+ \sum_{s=k+1}^{j} p_s \cdot (\text{level of } a_s \text{ in } T_{k+1}^{j} + 1)\}$$

$$= \min_{i \le k \le j} \{\sum_{s=i}^{k-1} p_s \cdot \text{level of } a_s \text{ in } T_i^{k-1} + \sum_{s=k+1}^{j} p_s \cdot \text{level of } a_s \text{ in } T_{k+1}^{j} + \sum_{s=i}^{j} p_s\}$$

$$= \min_{i \le k \le j} \{C(i, k-1) + C(k+1, j)\} + \sum_{s=i}^{j} p_s.$$

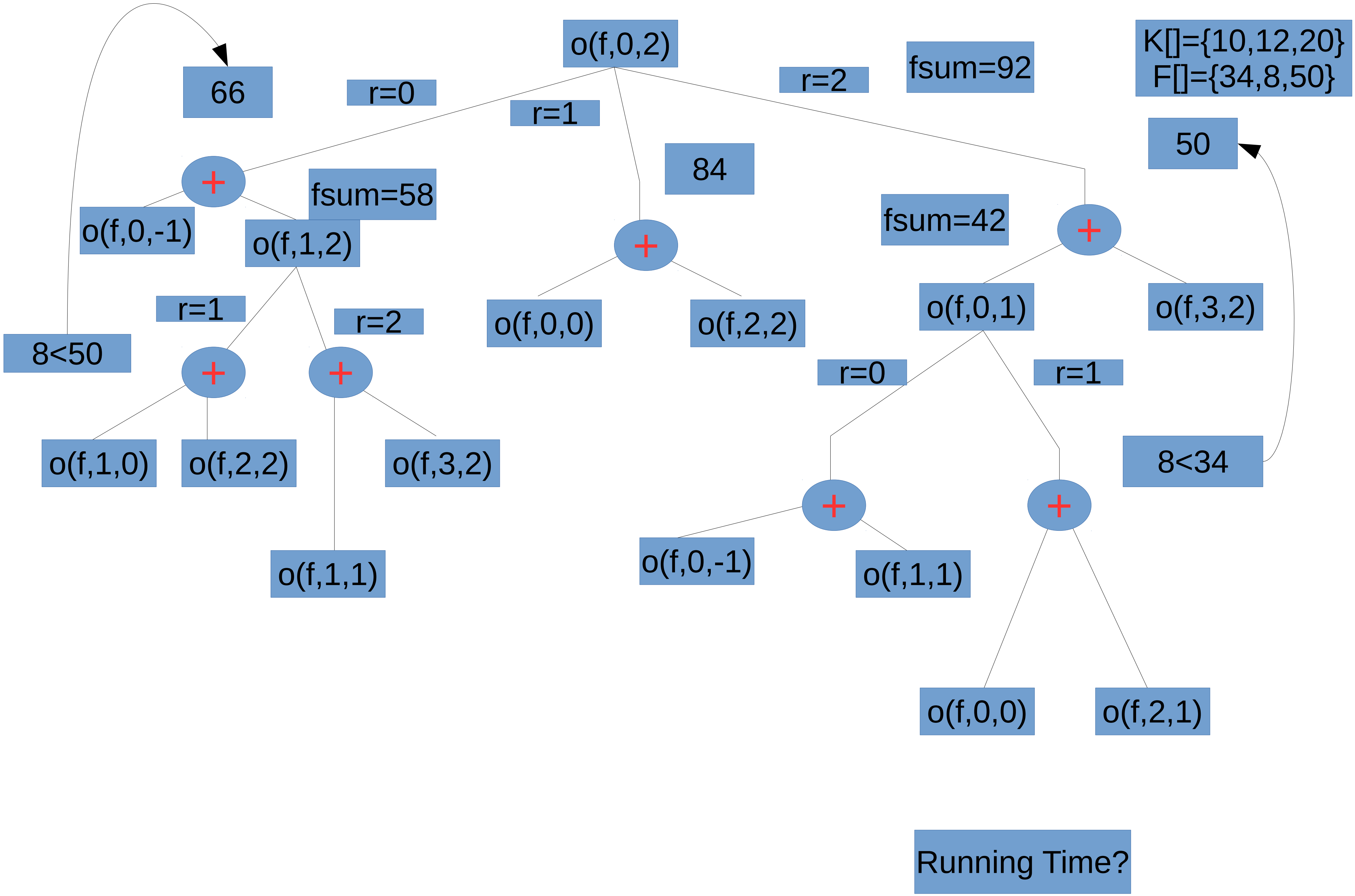
Thus, we have the recurrence

$$C(i, j) = \min_{i \le k \le j} \{C(i, k-1) + C(k+1, j)\} + \sum_{s=i}^{j} p_s \quad \text{for } 1 \le i \le j \le n. \qquad \textbf{(8.8)}$$

Recursive Solution

```
int optCost(int freq[], int i, int j)
{
   if (j < i)      // no elements in this subarray
     return 0;
   if (j == i)     // one element in this subarray
     return freq[i];
   int fsum = sum(freq, i, j);
   int min = INT_MAX;
   for (int r = i; r <= j; ++r)
   {
       int cost = optCost(freq, i, r-1) +
                  optCost(freq, r+1, j);
       if (cost < min)
           min = cost;
   }
   return min + fsum;
}
int optimalSearchTree(int keys[], int freq[], int n)
{
return optCost(freq, 0, n-1);
}
```

•



o(f,0,2)
66
r=0
r=1
r=2
fsum=92
K[]={10,12,20}
F[]={34,8,50}
50
84
fsum=58
fsum=42
o(f,0,-1)
o(f,1,2)
r=1
r=2
8<50
o(f,0,0)
o(f,2,2)
o(f,0,1)
o(f,3,2)
r=0
r=1
o(f,1,0)
o(f,2,2)
o(f,3,2)
8<34
o(f,1,1)
o(f,0,-1)
o(f,1,1)
o(f,0,0)
o(f,2,1)
Running Time?

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

The initial tables look like this:

main table

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

root table

	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

Let us compute $C(1, 2)$:

$$C(1, 2) = \min \begin{Bmatrix} k = 1: & C(1, 0) + C(2, 2) + \sum_{s=1}^{2} p_s = 0 + 0.2 + 0.3 = 0.5 \\ k = 2: & C(1, 1) + C(3, 2) + \sum_{s=1}^{2} p_s = 0.1 + 0 + 0.3 = 0.4 \end{Bmatrix}$$

$$= 0.4.$$

main table

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

root table

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					

DP-Solution

```
int optimalSearchTree(int
keys[], int freq[], int n)
{
    int cost[n][n];
   for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];
 for (int L=2; L<=n; L++)
    {
        for (int i=0; i<=n-L+1; i++)
        {
            int j = i+L-1;
            cost[i][j] = INT_MAX;
             for (int r=i; r<=j; r++)
            {
               int c = ((r > i)? cost[i][r-1]:0) +
                       ((r < j)? cost[r+1][j]:0) +
                       sum(freq, i, j);
               if (c < cost[i][j])
                  cost[i][j] = c;
            }
        }
    }
    return cost[0][n-1];
}
```

THANK YOU