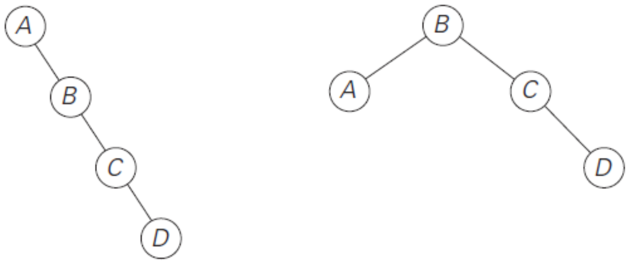
OPTIMAL BINARY SEARCH TREES

- A **Binary Search Tree** is one of the most important data structures which contain a set of elements with the operations of searching, insertion, and deletion.
- An **Optimal Binary Search Tree** is the one for which the average number of comparisons in a search is the smallest possible, if the probability of searching each elements is given.
- Consider four keys A, B, C, and D to be searched for with probabilities 0.1, 0.2, 0.4, and 0.3, respectively.
- The below figure depicts two out of 14 possible binary search trees containing these keys.



- The average number of comparisons in a successful search in the first of these trees is

  $0.1 \cdot 1 + 0.2 \cdot 2 + 0.4 \cdot 3 + 0.3 \cdot 4 = 2.9,$

  and for the second one it is

  $0.1 \cdot 2 + 0.2 \cdot 1 + 0.4 \cdot 2 + 0.3 \cdot 3 = 2.1.$
- Neither of these two trees is, in fact, optimal.
- The total number of binary search trees with n keys is equal to the nth **Catalan number**,

$$c(n) = \frac{1}{n+1}\binom{2n}{n} \quad \text{for } n > 0, \quad c(0) = 1,$$

- Let $a_1, \ldots, a_n$ be distinct keys ordered from the smallest to the largest and let $p_1, \ldots, p_n$ be the probabilities of searching for them.
- Let C(i, j) be the smallest average number of comparisons made in a successful search in a binary search tree T_i^j.
- Suppose the root contains key a_k, the left subtree T_i^{k-1} contains keys $a_i, \ldots, a_{k-1}$ optimally arranged, and the right subtree T_j^{k+1} contains keys $a_{k+1}, \ldots, a_j$ also optimally arranged.
- If we count tree levels starting with 1 to make the comparison numbers equal the keys' levels, the following recurrence relation is obtained:

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

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

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

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

- The recurrence relation is

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

- Here $C(i, i-1) = 0$ for $1 \le i \le n+1$ [represents empty tree] and

  $C(i, i) = p_i$, for $1 \le i \le n$ [represents an one node tree],
- The algorithm computes $C(1, n)$, the average number of comparisons for successful searches in the optimal binary tree.
- To get the optimal tree, another two-dimensional table to record the value of k for which it is minimum.

<u>EXAMPLE</u>

Construct an optimal binary search tree for the given set of keys

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

Initial tables will be: here $C(i, i-1) = 0$ for $1 \le i \le n+1$ & $C(i, i) = p_i$

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

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

$$C(1,2) = \min\begin{cases} \text{for } k = 1: C[1,0] + C[2,2] + \sum_{s=1}^{2} p_s \\ \text{for } k = 2: C[1,1] + C[3,2] + \sum_{s=1}^{2} p_s \end{cases}$$

= min[0 + 0.2 + 0.3, 0.1 + 0 + 0.3]

= min[0.5, 0.4]

= 0.4

$$C(2,3) = \min\begin{cases} \text{for } k = 2: C[2,1] + C[3,3] + \sum_{s=2}^{3} p_s \\ \text{for } k = 3: C[2,2] + C[4,3] + \sum_{s=2}^{3} p_s \end{cases}$$

= min[0 + 0.4 + 0.6, 0.2 + 0 + 0.6]

= min[1.0, 0.8]

= 0.8

$$C(3,4) = \min\begin{cases} \text{for } k = 3: C[3,2] + C[4,4] + \sum_{s=3}^{4} p_s \\ \text{for } k = 4: C[3,3] + C[5,4] + \sum_{s=3}^{4} p_s \end{cases}$$

= min[0 + 0.3 + 0.7, 0.4 + 0 + 0.7]

= min[1.0, 1.1]

= 1.0

Now the tables becomes

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

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

$$C(1,3) = \min\begin{cases} \text{for } k = 1: C[1,0] + C[2,3] + \sum_{s=1}^{3} p_s \\ \text{for } k = 2: C[1,1] + C[3,3] + \sum_{s=1}^{3} p_s \\ \text{for } k = 3: C[1,2] + C[4,3] + \sum_{s=1}^{3} p_s \end{cases}$$

= min[0 + 0.8 + 0.7, 0.1 + 0.4 + 0.7, 0.4 + 0 + 0.7]

= min[1.5, 1.2, 1.1]

= 1.1

$$C(2,4) = \min\begin{cases} \text{for } k = 2: C[2,1] + C[3,4] + \sum_{s=2}^{4} p_s \\ \text{for } k = 3: C[2,2] + C[4,4] + \sum_{s=2}^{4} p_s \\ \text{for } k = 4: C[2,3] + C[5,4] + \sum_{s=2}^{4} p_s \end{cases}$$

= min[0 + 1.0 + 0.9, 0.2 + 0.3 + 0.9, 0.8 + 0 + 0.9]

= min[1.9, 1.4, 1.7]

= 1.4

Now the tables becomes

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

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

$$C(1,4) = \min\begin{cases} \text{for } k = 1: C[1,0] + C[2,4] + \sum_{s=1}^{4} p_s \\ \text{for } k = 2: C[1,1] + C[3,4] + \sum_{s=1}^{4} p_s \\ \text{for } k = 3: C[1,2] + C[4,4] + \sum_{s=1}^{4} p_s \\ \text{for } k = 4: C[1,3] + C[5,4] + \sum_{s=1}^{4} p_s \end{cases}$$

= min[0 +1.4 +1.0, 0.1 + 1.0 + 1.0, 0.4 + 0.3 + 1.0, 1.1 + 0 + 1.0]

= min[2.4, 2.1, 1.7, 2.1]

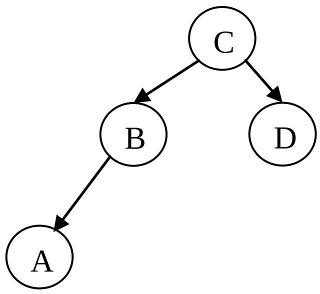
= 1.7

Now the tables becomes

	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

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

- Thus, the average number of key comparisons in the optimal tree is equal to 1.7.
- Since R(1, 4) = 3, the root of the optimal tree contains the third key, i.e., C.
- Since its a binary search tree, Its left subtree is made up of keys A and B, and its right subtree contains just the key D
- To find the specific structure of these subtrees,
- In the root table since R(1, 2) = 2, the root of the optimal tree containing A and B is B, with A being its left child.
- Since R(4, 4) = 4, the root of this one-node optimal tree is its only key D.
- The below figure represents the optimal tree



The pseudocode of this algorithm is given below

ALGORITHM *OptimalBST*$(P[1..n])$

```
//Finds an optimal binary search tree by dynamic programming
//Input: An array P[1..n] of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
//        optimal BST and table R of subtrees' roots in the optimal BST
for i ← 1 to n do
    C[i, i − 1] ← 0
    C[i, i] ← P[i]
    R[i, i] ← i
C[n + 1, n] ← 0
for d ← 1 to n − 1 do    //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j];   kmin ← k
        R[i, j] ← kmin
        sum ← P[i];   for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```

The time efficiency of this algorithm is $\Theta(n^3)$