Optimal BST & its Analysis

V. Balasubramanian
SSN College of Engineering

BST

- A binary search tree is one of the most important data structures in computer science. One of its principal applications is to implement a dictionary, a set of elements with the operations of searching, insertion, and deletion.

Example

- As an 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.

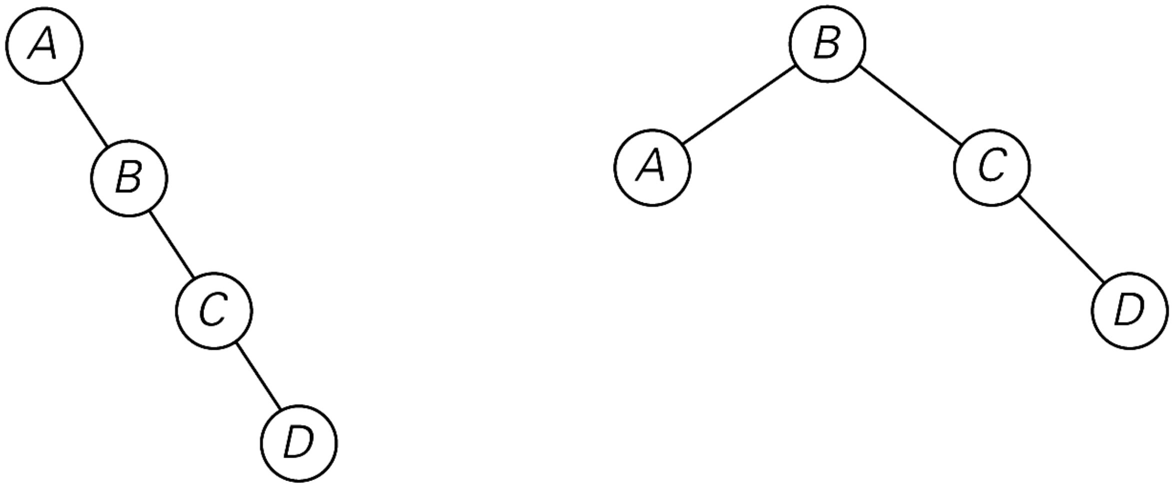

FIGURE 8.8 Two out of 14 possible binary search trees with keys A, B, C, and D

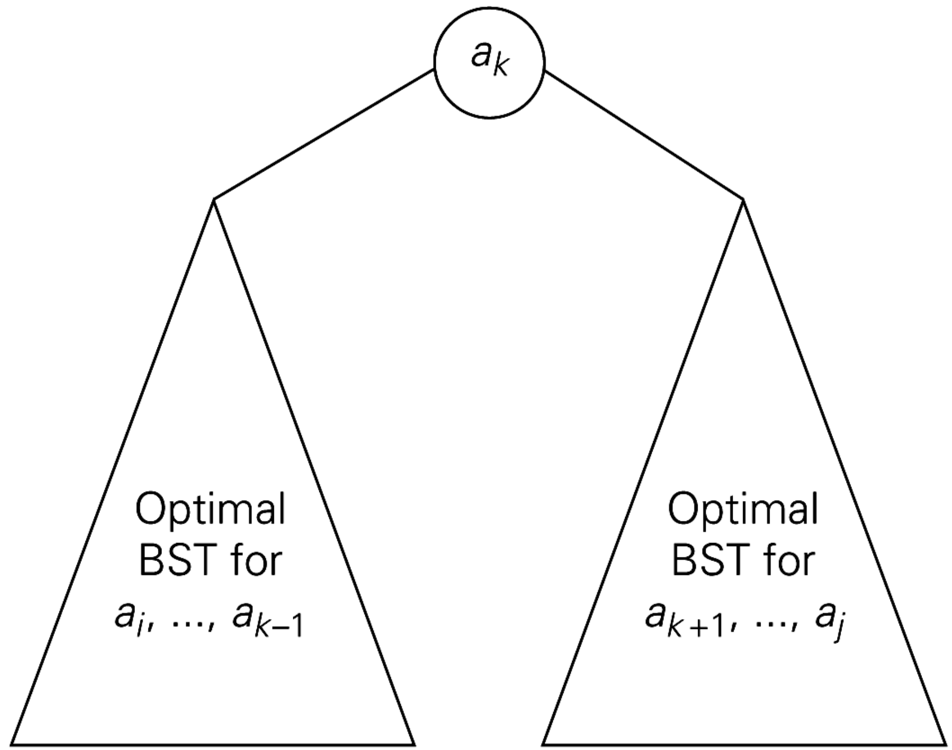

FIGURE 8.9 Binary search tree (BST) with root a_k and two optimal binary search subtrees T_i^{k-1} and T_{k+1}^{j}

Recurrence relation

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

Basis condition

We assume in formula (8.8) that $C(i, i-1) = 0$ for $1 \le i \le n+1$, which can be interpreted as the number of comparisons in the empty tree. Note that this formula implies that

$$C(i, i) = p_i \quad \text{for } 1 \le i \le n,$$

as it should be for a one-node binary search tree containing a_i.

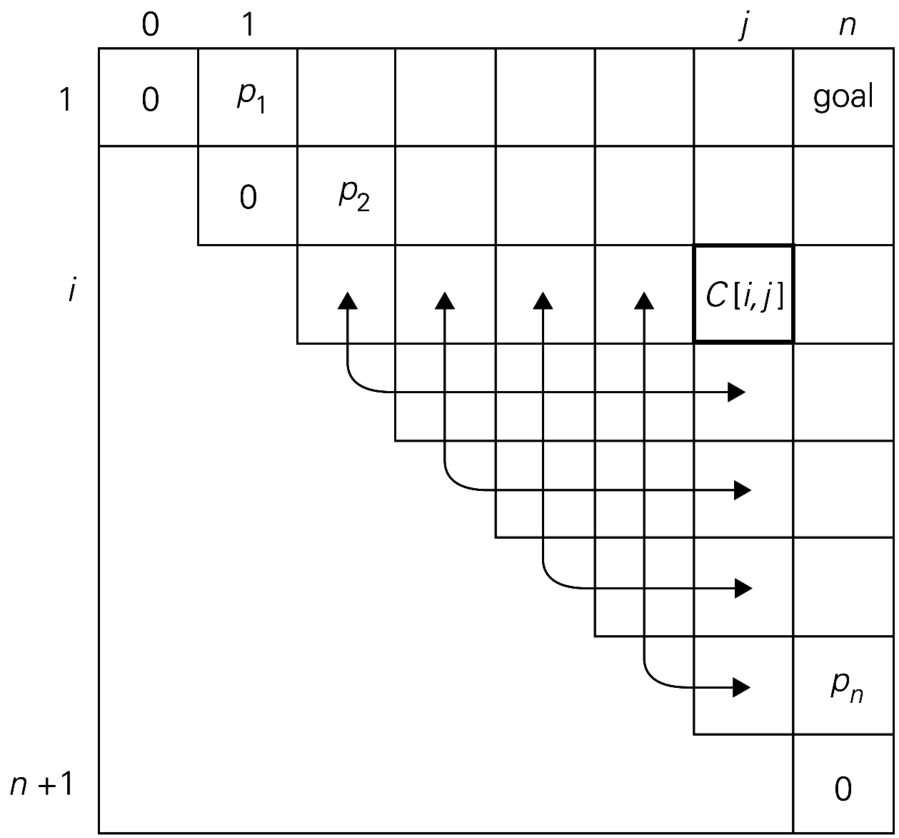

FIGURE 8.10 Table of the dynamic programming algorithm for constructing an optimal binary search tree

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 $C(1, 2)$:

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

$$= 0.4.$$

$$C[1,2] = \min \begin{array}{ll} k=1: & C[1,0] + C[2,2] + \sum_{s=1}^{2} p_s = 0 + 0.2 + 0.3 = 0.5 \\ k=2: & C[1,1] + C[3,2] + \sum_{s=1}^{2} p_s = 0.1 + 0 + 0.3 = \mathbf{0.4} \end{array} = 0.4$$

$$C[2,3] = \min \begin{array}{ll} k=2: & C[2,1] + C[3,3] + \sum_{s=2}^{3} p_s = 0 + 0.4 + 0.6 = 1.0 \\ k=3: & C[2,2] + C[4,3] + \sum_{s=2}^{3} p_s = 0.2 + 0 + 0.6 = \mathbf{0.8} \end{array} = 0.8$$

$$C[3,4] = \min \begin{array}{ll} k=3: & C[3,2] + C[4,4] + \sum_{s=3}^{4} p_s = 0 + 0.3 + 0.7 = \mathbf{1.0} \\ k=4: & C[3,3] + C[5,4] + \sum_{s=3}^{4} p_s = 0.4 + 0 + 0.7 = 1.1 \end{array} = 1.0$$

$$C[1,3] = \min \begin{array}{ll} k=1: & C[1,0] + C[2,3] + \sum_{s=1}^{3} p_s = 0 + 0.8 + 0.7 = 1.5 \\ k=2: & C[1,1] + C[3,3] + \sum_{s=1}^{3} p_s = 0.1 + 0.4 + 0.7 = 1.2 \\ k=3: & C[1,2] + C[4,3] + \sum_{s=1}^{3} p_s = 0.4 + 0 + 0.7 = \mathbf{1.1} \end{array} = 1.1$$

$$C[2,4] = \min \begin{array}{ll} k=2: & C[2,1] + C[3,4] + \sum_{s=2}^{4} p_s = 0 + 1.0 + 0.9 = 1.9 \\ k=3: & C[2,2] + C[4,4] + \sum_{s=2}^{4} p_s = 0.2 + 0.3 + 0.9 = \mathbf{1.4} \\ k=4: & C[2,3] + C[5,4] + \sum_{s=2}^{4} p_s = 0.8 + 0 + 0.9 = 1.7 \end{array} = 1.1$$

$$C[1,4] = \min \begin{array}{ll} k=1: & C[1,0] + C[2,4] + \sum_{s=1}^{s=4} p_s = 0 + 1.4 + 1.0 = 2.4 \\ k=2: & C[1,1] + C[3,4] + \sum_{s=1}^{4} p_s = 0.1 + 1.0 + 1.0 = 2.1 \\ k=3: & C[1,2] + C[4,4] + \sum_{s=1}^{4} p_s = 0.4 + 0.3 + 1.0 = \mathbf{1.7} \\ k=4: & C[1,3] + C[5,4] + \sum_{s=1}^{4} p_s = 1.1 + 0 + 1.0 = 2.1 \end{array} = 1.7$$

the 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

the root table

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

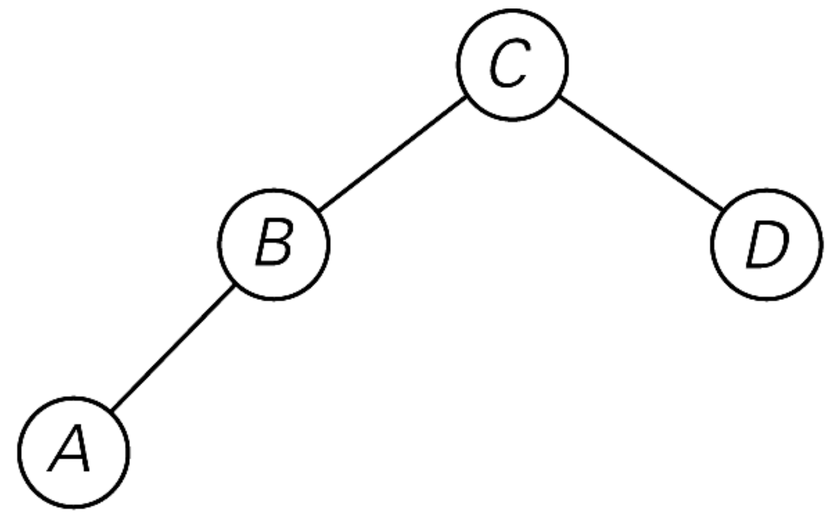

FIGURE 8.11 Optimal binary search tree for the example

```
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
```

Analysis

$$\begin{aligned}
\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}\sum_{k=i}^{i+d} 1 &= \sum_{d=1}^{n-1}\sum_{i=1}^{n-d}(i+d-i+1) = \sum_{d=1}^{n-1}\sum_{i=1}^{n-d}(d+1) \\
&= \sum_{d=1}^{n-1}(d+1)(n-d) = \sum_{d=1}^{n-1}(dn+n-d^2-d) \\
&= \sum_{d=1}^{n-1} nd + \sum_{d=1}^{n-1} n - \sum_{d=1}^{n-1} d^2 - \sum_{d=1}^{n-1} d \\
&= n\frac{(n-1)n}{2} + n(n-1) - \frac{(n-1)n(2n-1)}{6} - \frac{(n-1)n}{2} \\
&= \frac{1}{2}n^3 - \frac{2}{6}n^3 + O(n^2) \in \Theta(n^3).
\end{aligned}$$

b. The algorithm generates the $(n+1)$-by-$(n+1)$ table C and the n-by-n table R and fills about one half of each. Hence, the algorithm's space efficiency is in $\Theta(n^2)$.

Example

- The root of an optimal binary search tree always contains the key with the highest search probability? T / F

- False. counterexample: A(0.3),B(0.3),C(0.4).
- The average number of comparisons in a binary search tree with C in its root is $0.3 \cdot 2 + 0.3 \cdot 3 + 0.4 \cdot 1 = 1.9$, while the average number of comparisons in the binary search tree with B in its root is $0.3 \cdot 1 + 0.3 \cdot 2 + 0.4 \cdot 2 = 1.7$.