CSCI3160 Design and Analysis of Algorithms (2025 Fall)

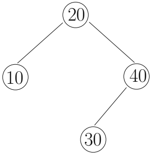
Dynamic Programming 5: Optimal BST

Instructor: Xiao Liang[1]

Department of Computer Science and Engineering
The Chinese University of Hong Kong

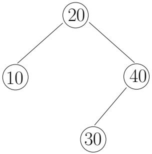
[1] These slides are primarily based on materials prepared by Prof. Yufei Tao (please refer to Prof. Tao's version from 2024 Fall for the original content). Some modifications have been made to better align with this year's teaching progress, incorporating student feedback, in-class interactions, and my own teaching style and research perspective.

Review: Binary Search Tree (BST)



- Each node stores a **key**.
- The key of an internal node u is larger than any key in the left subtree of u, and smaller than any key in the right subtree of u.

Review: Binary Search Tree (BST)



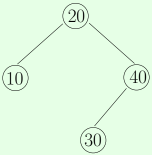
- The **level** of a node u in a BST T — denoted as $level_T(u)$ — equals the number of edges on the path from the root to u.
  - The level of the root is 0.
- The **depth** of a tree is the maximum level of the nodes in the tree.
- Searching for a node u incurs cost proportional to $1 + level_T(u)$.

Let S be a set of n integers. We have learned (from CSCI2100) that a balanced BST on S has depth $O(\log n)$. This is good if all the integers in S are searched with **equal probabilities**.

In practice, not all keys are equally important: some are searched **more often than others**. This gives rise to an interesting question:

> If we know the search frequencies of the integers in S, how to build a better BST to minimize the average search cost?

Example:



Suppose that the search frequencies of 10, 20, 30, and 40 are 40%, 15%, 35%, and 10%, respectively. Then, the average cost of searching for a key in the BST equals:

$$
\begin{aligned}
& \mathit{freq}(10) \cdot \mathit{cost}(10) + \mathit{freq}(20) \cdot \mathit{cost}(20) + \\
& \mathit{freq}(30) \cdot \mathit{cost}(30) + \mathit{freq}(40) \cdot \mathit{cost}(40) \\
= \;& 40\% \cdot 2 + 15\% \cdot 1 + 35\% \cdot 3 + 10\% \cdot 2 \\
= \;& 2.2.
\end{aligned}
$$

The Optimal BST Problem

Input:

- A set S of n integers: $\{1, 2, ..., n\}$;
- An array W where $W[i]$ $(1 \leq i \leq n)$ stores a positive integer weight.

Output: A BST T on S with the smallest **average cost**

$$avgcost(T) \quad = \quad \sum_{i=1}^{n} W[i] \cdot cost_T(i).$$

where $cost_T(i) = 1 + level_T(i)$ is the number of nodes accessed to find the key i in T.

We will solve a more general version of the problem.

Input:

- S and W same as before;
- Integers a, b satisfying $1 \leq a \leq b \leq n$.

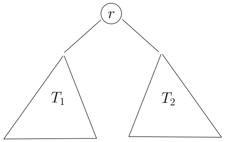
Output: A BST T on $\{a, a+1, ..., b\}$ with the smallest **average cost**:

$$avgcost(T) \quad = \quad \sum_{i=a}^{b} W[i] \cdot cost_T(i).$$

Fact: The root of T must have a key $r \in [a, b]$.

After the root key r is fixed, we know:

- the root's left subtree is a BST T_1 on $S_1 = \{a, ..., r-1\}$, and
- the root's right subtree is a BST T_2 on $S_2 = \{r+1, ..., b\}$.



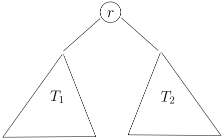
Lemma: Let T, T_1, and T_2 be defined as above. Then:

$$avgcost(T) = \left(\sum_{i=a}^{b} W[i]\right) + avgcost(T_1) + avgcost(T_2).$$

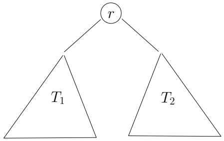
Proof:

$$
\begin{aligned}
& avgcost(T) \\
= \;& \sum_{i=a}^{b} W[i] \cdot cost_T(i) = \sum_{i=a}^{b} W[i] \cdot (1 + level_T(i)) \\
= \;& \left(\sum_{i=a}^{b} W[i]\right) + \sum_{i=a}^{b} W[i] \cdot level_T(i) \\
= \;& \left(\sum_{i=a}^{b} W[i]\right) + \left(\sum_{i=a}^{r-1} W[i] \cdot level_T(i)\right) + \left(\sum_{i=r+1}^{b} W[i] \cdot level_T(i)\right)
\end{aligned}
$$

(Continued on the next slide)



$$
\begin{aligned}
&= \left(\sum_{i=a}^{b} W[i]\right) + \\
&\quad \left(\sum_{i=a}^{r-1} W[i] \cdot (1 + level_{T_1}(i))\right) + \left(\sum_{i=r+1}^{b} W[i] \cdot (1 + level_{T_2}(i))\right) \\
&= \left(\sum_{i=a}^{b} W[i]\right) + \left(\sum_{i=a}^{r-1} W[i] \cdot cost_{T_1}(i)\right) + \left(\sum_{i=r+1}^{b} W[i] \cdot cost_{T_2}(i)\right) \\
&= \left(\sum_{i=a}^{b} W[i]\right) + avgcost(T_1) + avgcost(T_2).
\end{aligned}
$$



□

Define *optavg*(a, b) as

- 0, if $a > b$;
- the smallest average cost of a BST on $\{a, a+1, ..., b\}$, otherwise.

Define *optavg*$(a, b \mid r)$ as the optimal average cost of a BST, **on condition that** the BST has $r \in [a, b]$ as the key of the root.

By the previous lemma, we have:

$$
\begin{aligned}
& optavg(a, b \mid r) \\
= \quad & \left(\sum_{i=a}^{b} W[i]\right) + optavg(a, r-1) + optavg(r+1, b).
\end{aligned}
$$

Example: $S = \{1, 2, 3, 4\}$; $W = (40, 15, 35, 10)$.

Consider choosing 2 as the root key.

$$\begin{aligned} & \mathit{optavg}(1, 4 \mid 2) \\ = \; & \left(\sum_{i=1}^{4} W[i] \right) + \mathit{optavg}(1, 1) + \mathit{optavg}(3, 4) \\ = \; & 100 + 40 + 55 = 195. \end{aligned}$$

Hence, **among all BSTs with root key 2**, the best BST has average cost 195.

The **recursive structure** of the problem:

$$
\begin{aligned}
& optavg(a, b) \\
= \;& \min_{r=a}^{b} optavg(a, b \mid r) \\
= \;& \left(\sum_{i=a}^{b} W[i] \right) + \min_{r=a}^{b} \left\{ optavg(a, r-1) + optavg(r+1, b) \right\}.
\end{aligned}
$$

With dynamic programming, we can compute $optavg(1, n)$ in $O(n^3)$ time (left as a special exercise).

Strictly speaking, we have not produced the optimal BST yet. However, fixing the issue should be fairly standard to you at this moment: the piggyback technique allows you to build the tree in the same time complexity as computing $opt(1, n)$. This is left as a special exercise.