Dynamic Programming 2: Optimal BST

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
Designing a dynamic programming algorithm, in general, requires discovering a **recursive structure** of the underlying problem. Next, we will illustrate this through the **optimal BST problem**.
Each node stores a key.

The key of an internal node $u$ is larger than any key in its left subtree, and smaller than any key in its right subtree.
The **level** of a node $u$ in a BST $T$ — denoted as $\text{level}_T(u)$ — 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 $1 + \text{level}_T(u)$.

- How many nodes do you need to access to search for node 10, 20, 30, and 40, respectively?
Let $S$ be a set of $n$ integers. We know that a balanced BST on $S$ has depth $O(\log n)$. This is good if we assume that 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 we know the 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:

\[
\text{freq}(10) \cdot \text{cost}(10) + \text{freq}(20) \cdot \text{cost}(20) + \\
\text{freq}(30) \cdot \text{cost}(30) + \text{freq}(40) \cdot \text{cost}(40) \\
= 40\% \cdot 2 + 15\% \cdot 1 + 35\% \cdot 3 + 10\% \cdot 2 \\
= 2.2
\]

where \( \text{freq}(k) \) denotes the search frequency of key \( k \), and \( \text{cost}(k) \) denotes the cost of searching for \( k \) in the tree.
The Optimal BST Problem

Input:
- A set $S$ of $n$ integers: $\{1, 2, \ldots, n\}$;
- 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:

$$\text{avgcost}(T) = \sum_{i=1}^{n} W[i] \cdot \text{cost}_T(i).$$

where $\text{cost}_T(i) = 1 + \text{level}_T(i)$ is the number of nodes accessed to find the key $i$ in $T$.

Think: here we consider that the keys are $1, 2, \ldots, n$, respectively; do we lose any generality?
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 $\{a, a+1, \ldots, b\}$ with the smallest **average cost**:

$$\text{avgcost}(T) = \sum_{i=a}^{b} W[i] \cdot \text{cost}_T(i).$$

where $\text{cost}_T(i) = 1 + \text{level}_T(i)$ is the number of nodes accessed to find the key $i$ in $T$. 
As mentioned, an important step in designing a dynamic programming algorithm is to figure out the **recursive structure** of the underlying problem. Typically, this involves three steps:

1. identify **all** the possible options for the “**first**” choice;
2. **conditioned on** the first choice, find the optimal solution;
3. take the first choice that leads to the **overall best** solution.

Next, we will explain how to do so for the optimal BST problem.
1. Find all the Options for the First Choice

**First Choice:** Key at the root of $T$?

Clearly, we have $b - a + 1$ options: we can put $a, a + 1, \ldots, b$ as the key at the root.

Suppose that we put $r$ as the key at the root for some $r \in [a, b]$. Then, its left subtree must be a BST $T_1$ on $S_1 = \{a, \ldots, r - 1\}$, and its right subtree must be a BST $T_2$ on $S_2 = \{r + 1, \ldots, b\}$.
**Example:** \( S = \{1, 2, 3, 4\}; \ W = (40, 15, 35, 10) \).

Consider the option of putting 2 at the root. The left subtree must contain just a single leaf with the key 1.

The right subtree, on the other hand, has two choices:

\[
\begin{array}{c}
4 \\
| \\
3
\end{array}
\quad \text{or} \quad
\begin{array}{c}
3 \\
| \\
4
\end{array}
\]
2. Conditioned on the First Choice, Find the Optimal Solution:
Put \( r \) at the root of \( T \). Next, we will show that, to minimize the average cost of \( T \), we should choose the best trees for \( T_1 \) and \( T_2 \).

\[
\text{avgcost}(T) = \sum_{i=a}^{b} W[i] \cdot \text{cost}_T(i) = \sum_{i=a}^{b} W[i] \cdot (1 + \text{level}_T(i))
\]

\[
= \left( \sum_{i=a}^{b} W[i] \right) + \sum_{i=a}^{b} W[i] \cdot \text{level}_T(i)
\]

\[
= \left( \sum_{i=a}^{b} W[i] \right) + \left( \sum_{i=a}^{r-1} W[i] \cdot \text{level}_T(i) \right) + \left( \sum_{i=r+1}^{b} W[i] \cdot \text{level}_T(i) \right)
\]

(Continuing on the next slide)
\[
\begin{align*}
&= \left( \sum_{i=a}^{b} W[i] \right) + \left( \sum_{i=a}^{r-1} W[i] \cdot (1 + \text{level}_{T_1}(i)) \right) + \\
&\quad \left( \sum_{i=r+1}^{b} W[i] \cdot (1 + \text{level}_{T_2}(i)) \right) \\
&= \left( \sum_{i=a}^{b} W[i] \right) + \left( \sum_{i=a}^{r-1} W[i] \cdot \text{cost}_{T_1}(i) \right) + \left( \sum_{i=r+1}^{b} W[i] \cdot \text{cost}_{T_2}(i) \right) \\
&= \left( \sum_{i=a}^{b} W[i] \right) + \text{avgcost}(T_1) + \text{avgcost}(T_2)
\end{align*}
\]

Clearly, we should minimize \( \text{avgcost}(T_1) \) and \( \text{avgcost}(T_2) \), namely, building optimal BSTs on \( S_1 \) and \( S_2 \), recursively.
Example: $S = \{1, 2, 3, 4\}; \ W = (40, 15, 35, 10)$.

Consider the option of putting 2 at the root. As mentioned, the right subtree has two choices:

```
    4
   / \  or
  3   3
     /
    4
```

We know from the above discussion that the right subtree should be an optimal BST on $\{3, 4\}$. Which of the above two choices is optimal on $\{3, 4\}$?

The answer is the second one: it has an average cost of $35 \cdot 1 + 10 \cdot 2 = 55$. 
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 $r$ as the key of the root.

The previous discussion has essentially proved:

$$optavg(a, b \mid r) = \left( \sum_{i=a}^{b} W[i] \right) + optavg(a, r - 1) + optavg(r + 1, b).$$
Example: \( S = \{1, 2, 3, 4\} \); \( W = (40, 15, 35, 10) \).

Consider the option of putting 2 at the root.

\[
\text{optavg}(1, 4 \mid 2) = \left( \sum_{i=1}^{4} W[i] \right) + \text{optavg}(1, 1) + \text{optavg}(3, 4)
\]
\[
= 100 + 40 + 55 = 195.
\]

Hence, if we want to put 2 at the root, the best BST we can construct has average cost 195.
3. **Selecting the Best First Choice:** The best choice for $r$ is the one that leads to the smallest average cost, namely:

\[
\text{optavg}(a, b) = \min_{r=a}^{b} \text{optavg}(a, b \mid r)
\]

\[
= \left( \sum_{i=a}^{b} W[i] \right) + \min_{r=a}^{b} \left\{ \text{optavg}(a, r - 1) + \text{optavg}(r + 1, b) \right\}.
\]

This is the recursive structure of the problem.
**Example:** $S = \{1, 2, 3, 4\}$; $W = (40, 15, 35, 10)$. The optimal tree is actually:

```
  3
 / \
1   4

/ \\   \\
2   1
```

$$
\text{optavg}(1, 4) = \text{optavg}(1, 4 \mid 3)
= \left( \sum_{i=1}^{4} W[i] \right) + \text{optavg}(1, 2) + \text{optavg}(4, 4)
= 100 + \text{optavg}(1, 2) + 10 = 110 + \text{optavg}(1, 2)
= 110 + \text{optavg}(1, 2 \mid 1)
= 110 + \left( \sum_{i=1}^{2} W[i] \right) + \text{optavg}(1, 0) + \text{optavg}(2, 2)
= 110 + 55 + 0 + 15 = 180.
$$
Putting Everything Together

We have converted the optimal BST problem into the following problem:

**Input**: An array $W$ of $n$ integers.

**Output** Compute $\text{optavg}(1, n)$ where for any $a, b \in [1, n]$:

$$\text{optavg}(a, b) =
\begin{cases}
  0, & \text{if } a > b \\
  \left( \sum_{i=a}^{b} W[i] \right) + \min_{r=a}^{b} \left\{ \text{optavg}(a, r-1) + \text{optavg}(r+1, b) \right\}, & \text{otherwise}
\end{cases}$$

This is precisely the problem we studied in the previous lecture! Recall that with dynamic programming, we can compute $\text{optavg}(1, n)$ in $O(n^3)$ time.
Strictly speaking, there is one more step: although we have calculated $\text{optavg}(1, n)$, we still have not produced the optimal BST yet!

This is, in fact, rather trivial — you can do so in $O(n)$ time after computing $\text{optavg}(1, n)$ with dynamic programming. This will be left as a regular exercise.