Optimal Binary Search Trees

Chapter 15

Balanced BST

- Are balanced binary search trees always the most efficient search trees?
- Yes! But only if every key is equally probable

Example

- Dictionary for spell-checking
  - What if the root of a balanced tree is “panentheism”?
    - Occurrence in ordinary text is very low
    - Most searches will waste at least one comparison
  - What if the most common words (“a”, “an”, “the”, etc.) are the leaves?
- Balanced binary search tree is not always the most efficient
  - Problem is that not all words are equally likely
Optimal BST
- In optimal BSTs we store the probability of each node along with its key
- Given sequence $\mathbf{K} = <k_1, k_2, \ldots, k_n>$ of $n$ distinct keys, sorted ($k_1 < k_2 < \ldots < k_n$)
- Want to build a binary search tree from the keys
- For $k_i$, have probability $p_i$ that a search is for $k_i$
- Want BST with minimum expected search cost.

Cost of a Search
- Actual cost = # of items examined.
- For key $k_i$, cost = $\text{depth}_T(k_i) + 1$, where $\text{depth}_T(k_i)$ = depth of $k_i$ in BST $T$.

$$E[\text{search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i$$
$$= \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i$$
$$= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i$$

Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p_i$</th>
<th>$\text{depth}_T(k_i)$</th>
<th>$\text{depth}_T(k_i) \cdot p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>3</td>
<td>0.75</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>5</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Therefore, $E[\text{search cost}] = 2.15$. 
Observations

• Optimal BST might not have smallest height.
• Optimal BST might not have highest probability key at root.

Exhaustive Checking

• Construct each n-node BST.
• For each, put in keys.
• Then compute expected search cost.
• But there are $\Omega(4^n / n^{3/2})$ different BSTs with n nodes.
Solution

• Dynamic Programming

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution bottom-up
4. Construct an optimal solution from the computed information

Step 1: Optimal Solution

• Consider any subtree of a BST. It contains keys in a contiguous range \(k_i, \ldots, k_j\) for some \(1 \leq i \leq j \leq n\).

If \(T\) is an optimal BST and \(T\) contains subtree \(T'\) with keys \(k_i, \ldots, k_j\), then \(T'\) must be an optimal BST for keys \(k_i, \ldots, k_j\).

Step 1: Optimal Solution

• Use optimal substructure to construct an optimal solution to the problem from optimal solutions to subproblems:

• Given keys \(k_i, \ldots, k_j\) (the problem).

• One of them, \(k_r\), where \(i \leq r \leq j\), must be the root.

• Left subtree of \(k_r\) contains \(k_i, \ldots, k_{r-1}\).

• Right subtree of \(k_r\) contains \(k_{r+1}, \ldots, k_j\).
Step 1: Optimal Solution

- If we examine all candidate roots \( k_r \), for \( i \leq r \leq j \), and
  - We determine all optimal BSTs containing \( k_i, \ldots, k_{r-1} \) and containing \( k_{r+1}, \ldots, k_j \)
- then we’re guaranteed to find an optimal BST for \( k_i, \ldots, k_j \)

Step 2: Recursive Solution

Subproblem domain:
- Find optimal BST for \( k_i, \ldots, k_j \), where \( i \geq 1, j \leq n, j \geq i - 1 \).
- When \( j = i - 1 \), the tree is empty.

Define \( c[i, j] \) = expected search cost of optimal BST for \( k_i, \ldots, k_j \).

If \( j = i - 1 \), then \( c[i, j] = 0 \).

If \( j \geq i \):
- Select a root \( k_r \), for some \( i \leq r \leq j \).
- Make an optimal BST with \( k_i, \ldots, k_{r-1} \) as the left subtree.
- Make an optimal BST with \( k_{r+1}, \ldots, k_j \) as the right subtree.
- Note: when \( r = i \), left subtree is \( k_i, \ldots, k_{i-1} \); when \( r = j \), right subtree is \( k_{j+1}, \ldots, k_j \).

Step 2

When a subtree becomes a subtree of a node:
- Depth of every node in subtree goes up by 1.
- Expected search cost increases by

\[
\begin{align*}
  w(i, j) &= \sum_{l=i}^{j} p_l \\
  &\text{ (refer to equation (+))}.
\end{align*}
\]
Step 2

If $k_r$ is the root of an optimal BST for $k_i, \ldots, k_j$:

$$e[i, j] = p_r + (e[i, r - 1] + w[i, r - 1]) + (e[r + 1, j] + w[r + 1, j]).$$

But $w[i, j] = w[i, r - 1] + p_r + w[r + 1, j]$.

Therefore, $e[i, j] = e[i, r - 1] + e[r + 1, j] + w[i, j]$.

This equation assumes that we already know which key is $k_r$.

We don't.

Try all candidates, and pick the best one:

$$e[i, j] = \begin{cases} 0 & \text{if } j = i - 1, \\ \min_{1 \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w[i, j]\} & \text{if } i \leq j. \end{cases}$$

Step 3

**OPTIMAL-BST**(p, q, n)

Let $e[1 \ldots n, 1 \ldots n], w[1 \ldots n, 1 \ldots n], \text{ and root}[1 \ldots n, 1 \ldots n]$ be new tables.

For $i = 1$ to $n + 1$

$e[i, i - 1] = 0$

$w[i, i - 1] = 0$

For $i = 1$ to $n$

For $j = i$ to $n + 1$

$e[i, j] = \infty$

$w[i, j] = w[i, j - 1] + p_j$

For $r = i$ to $j$

If $r < e[i, j]$

$e[i, j] = r$

root[i, j] = r

Return $e$ and root

Example

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$</td>
<td>.25</td>
<td>.2</td>
<td>.05</td>
<td>.2</td>
<td>.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$e$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.25</td>
<td>.5</td>
<td>1.25</td>
<td>2.1</td>
</tr>
<tr>
<td>2</td>
<td>.25</td>
<td>.5</td>
<td>1.25</td>
<td>2.1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>.5</td>
<td>1.25</td>
<td>2.1</td>
<td>3</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>1.25</td>
<td>2.1</td>
<td>3</td>
<td>4.5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2.1</td>
<td>3</td>
<td>4.5</td>
<td>5</td>
<td>7.3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4.5</td>
<td>5</td>
<td>7.3</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>25</td>
<td>45</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>45</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Step 4

CONSTRUCT-OPTIMAL-BST(root)
\[ r = \text{root}[1, n] \]
print “\(k\)” “is the root”
CONSTRUCT-OPT-SUBTREE(1, \(r - 1\), \(r\), “left”, root)
CONSTRUCT-OPT-SUBTREE(\(r + 1\), \(n\), \(r\), “right”, root)
CONSTRUCT-OPT-SUBTREE(\(i\), \(j\), \(r\), “dir”, root)
if \(i \leq j\)
\[ i = \text{root}[i, j] \]
print “\(k\)” “\(i\)” “dir” “\(i\)” “\(\text{child of} \ k\)”
CONSTRUCT-OPT-SUBTREE(\(i\), \(i - 1\), \(i\), “left”, root)
CONSTRUCT-OPT-SUBTREE(\(i + 1\), \(j\), \(i\), “right”, root)

Another Example

<table>
<thead>
<tr>
<th>i</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

![Diagram of a BST with nodes A, B, C, D and a root node labeled 'root'.]