Optimal Binary Search Trees

Vassos Hadzilacos

We are given keys $1, 2, \ldots, n$, and associated with each key i the probability $P(i)$ of searching for that key. If we store the keys in a binary search tree (BST) T, the expected number of comparisons made to find a key is

$$cost(T) = \sum_{i=1}^{n} P(i) \cdot \big(\mathbf{depth}(i, T) + 1\big) \qquad (\S)$$

where $\mathbf{depth}(i, T)$ is the depth of the node containing key i in tree T. (Recall that the depth of a node in a tree is the number of ***edges*** on the path from the root to the node, and so the number of nodes on that path is one more than the node's depth.) We will call this the ***cost*** of T. We wish to find a BST with minimum cost, i.e., a BST T that minimizes (§).

A brute-force approach is impractical, as the number of BSTs that contain n keys is huge: there are $\binom{2n}{n}/(n+1)$ such BSTs. The obvious greedy approach, of putting the most frequently sought key at the root and recursing on the remaining keys — those smaller than the key at the root forming the left subtree using the same procedure and those greater than the key at the root forming the right subtree also using the same procedure — does not necessarily produce an optimal BST: you can easily construct a counterexample.

The problem can be solved in $O(n^3)$ time using dynamic programming. As usual, we will first consider the problem of finding the ***value*** of the optimal object we are seeking, i.e., the cost of an optimal BST. It is then easy to "retrofit" the algorithm to actually find the optimal BST itself.

Subproblems to solve: For every $1 \le i \le n+1$ and $0 \le j \le n$ such that $i \le j+1$ compute

$$C(i, j) = \text{the cost of an optimal BST containing the set of keys } \{k : \ i \le k \le j\}. \qquad (*)$$

Note that for $i = j + 1$, this set is empty and we take the cost of an empty BST to be 0.

Solving the original problem: The answer to our original problem is simply $C(1, n)$.

Recursive formula to compute the subproblems:

$$C(i, j) = \begin{cases} 0, & \text{if } i = j+1 \\ \min\big\{C(i, k-1) + C(k+1, j) + \sum_{\ell=i}^{j} P(\ell) : \ i \le k \le j\big\}, & \text{if } i \le j \end{cases} \qquad (\dagger)$$

Justification why (†) correctly computes (*): Let T be any BST whose root contains the key k. Its left subtree T_L contains the keys less than k and its right subtree T_R contains the keys greater than k. By considering separately the terms of the cost of T contributed by the keys in the nodes on the left subtree, the right subtree, and the root k (whose depth in T is 0) we have

$$\begin{aligned} cost(T) &= \sum_{\ell \in T_L} P(\ell) \cdot \big(\mathbf{depth}(\ell, T) + 1\big) + \sum_{\ell \in T_R} P(\ell) \cdot \big(\mathbf{depth}(\ell, T) + 1\big) + P(k) \\ &= \sum_{\ell \in T_L} P(\ell) \cdot \mathbf{depth}(\ell, T) + \sum_{\ell \in T_R} P(\ell) \cdot \mathbf{depth}(\ell, T) + \sum_{\ell \in T} P(\ell) \\ &= \sum_{\ell \in T_L} P(\ell) \cdot \big(\mathbf{depth}(\ell, T_L) + 1\big) + \sum_{\ell \in T} P(\ell) \cdot \big(\mathbf{depth}(\ell, T_R) + 1\big) + \sum_{\ell \in T} P(\ell) \\ &= cost(T_L) + cost(T_R) + \sum_{\ell \in T} P(\ell). \end{aligned}$$

Hence, we have shown the following equation that relates the cost of a BST T to the cost of its left and right subtrees T_L and T_R:

$$cost(T) = cost(T_L) + cost(T_R) + \sum_{\ell \in T} P(\ell) \qquad (\ddagger)$$

For $1 \le i \le k \le j$, let T_{ij}^k be a BST of ***minimum cost*** that contains the keys in the range $i..j$ ***and has the key*** k ***in its root***. Thus the left subtree of T_{ij}^k is a BST that contains the keys in the range $i..k-1$, and the right subtree is a BST that contains the keys in the range $k+1..j$. Since T_{ij}^k has minimum cost among all BSTs that contain the keys $i..j$ and have key k in their root, by ($\ddagger$) and a straightforward cut-and-paste argument, the left and right subtrees of T_{ij}^k must be ***optimal*** trees containing the keys $i..k-1$ (left subtree) and keys $k+1..j$ (right subtree). By definition (*), the cost of these are $C(i, k-1)$ and $C(k+1, j)$ respectively. Therefore, by ($\ddagger$), $cost(T_{ij}^k) = C(i, k-1) + C(k+1, j) + \sum_{\ell=i}^{j} P(\ell)$.

A minimum cost BST that contains the keys $i..j$ is the tree among all the trees T_{ij}^k, $i \le k \le j$, that has minimum cost. The cost of this tree is, by the definition (*), $C(i, j)$. Therefore,

$$C(i,j) = \min\{cost(T_{ij}^k) : \ i \le k \le j\} = \min\{C(i,k-1) + C(k+1,j) + \sum_{\ell=i}^{j} P(\ell) : \ i \le k \le j\},$$

as wanted.

PSEUDOCODE: Our algorithm computes $C(i, j)$ in order of increasing $j - i$. That is, we first compute the cost of the empty subtrees $C(i, i-1)$ as the base case, and the the cost of all subtrees that contain one key, then the cost of all subtrees that contain two consecutive keys, three consecutive keys, and so on, based on (†). Before doing so we pre-compute $S(i) = \sum_{\ell=1}^{i} P(\ell)$ for $0 \le i \le n$; this allows us to compute in constant time the quantity $\sum_{\ell=i}^{j} P(\ell)$, for any $1 \le i \le j \le n$, simply as $S(j) - S(i-1)$.

```
    OPTBST(n, P)
1   S(0) := 0
2   for i := 1 to n do S(i) := S(i − 1) + P(i)
3   for i := 1 to n + 1 do C(i, i − 1) := 0
4   for d := 0 to n − 1 do
5       for i := 1 to n − d do
6           j := i + d
7           C(i, j) := +∞
8           For k := i to j do
9               if C(i, k − 1) + C(k + 1, j) + (S(j) − S(i − 1)) < C(i, j) then
10                  C(i, j) := C(i, k − 1) + C(k + 1, j) + (S(j) − S(i − 1))
11  return C(1, n)
```

RUNNING TIME ANALYSIS: The running time is dominated by the triply-nested loop in lines 4-10. Each loop runs for at most n iterations, so the total running time is $O(n^3)$. With a little more effort you can show that this is a tight bound: The running time of the algorithm is $\Theta(n^3)$.

Exercise 1: Retrofit the above algorithm so that it returns an optimal BST, as opposed to its cost.

Exercise 2: Assume that, in addition to the probabilities $P(i)$ for $1 \le i \le n$, we are given probabilities $Q(i)$ for $0 \le i \le n$, where $Q(0)$ is the probability of searching for keys less than key 1, $Q(n)$ is the probability of searching for keys greater than key n, and $Q(i)$, for $1 \le i < n$, is the probability of searching for keys between (but not including) keys i and $i+1$. In other words, Q specifies the probability of ***unsuccessful*** searches — i.e., searches for keys that are not in the tree. Define the cost of a binary search tree as the expected number of comparisons for a (successful or unsuccessful) search, and give a dynamic programming algorithm to construct a minimum cost BST under this cost criterion. (Hint: Add nodes to the tree to represent the between-keys ranges, and note that the number of comparisons made by a search for a key that is not in the tree is the depth of the node that represents the relevant between-keys range, and not the depth of that node plus 1.)