Optimal Binary Search Trees

April 28, 2011

1 Definition of the Problem

You will find quite a number of web sites that deal with the problem of constructing an optimal binary search tree. I did not find one that I think is easy enough to understand.

We are given a list L of non-negative *frequencies*, $P_1, P_2, \ldots P_n$. (We will use $L = 8, 3, 5, 7, 2$ as an example.) The problem is to construct a binary tree T with n nodes which minimizes a certain sum which represents expected search time.

We will number of the levels of T starting from 1 (instead of 0, which is more typical). Let H_i be the level of the i^{th} node of T in inorder. For example, if T is given in the figure below

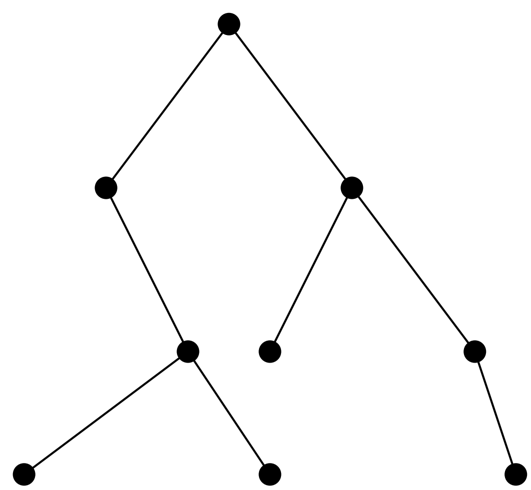

Figure 1

then the list of levels is 2,4,3,4,1,3,2,3,4.

We define the *weighted path length* of T to be $\sum_{i=1}^{n} H_i P_i$. For example, if the list of frequencies is 2,3,8,5,4,9,6,2,1, then the expected path length of the tree T given in Figure 1 is

$$2 \cdot 2 + 4 \cdot 3 + 3 \cdot 8 + 4 \cdot 5 + 1 \cdot 4 + 3 \cdot 9 + 2 \cdot 6 + 3 \cdot 2 + 4 \cdot 1 = 113$$

as we show in Figure 2.

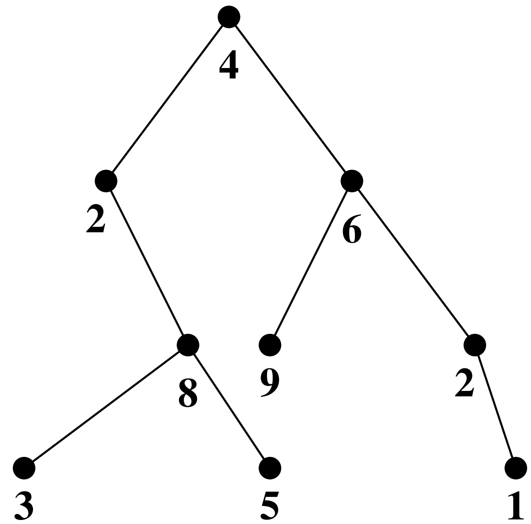

Figure 2

We say that a binary tree T is *optimal* for a given frequency list L of length n, if the weighted path length is minimum over all binary trees with n nodes. (The example tree shown in the figures is quite obviously not optimal.

2 Dynamic Programming

Every list L of length n has a total of $\binom{n+1}{2} = \Theta(n^2)$ contiguous sublists. For any $1 \leq i \leq j \leq n$, let $L_{i,j}$ be the sublist of L consisting of the i^{th} through the j^{th} terms. For example, if L is the list we used above, then $L_{2,5} = 3, 8, 5, 4$. We define $W_{i,j} = P_i + \cdots P_j$, the sum of the terms of $L_{i,j}$.

Let $T_{i,j}$ be the binary tree which is optimal for the list $L_{i,j}$. When we attach frequencies to $T_{i,j}$, then the root of $T_{i,j}$ will have frequency P_k for some $i \leq k \leq j$, and by the principle of optimality, the left and right subtrees of $T_{i,j}$ will be $T_{i,k-1}$ and $T_{k+1,j}$, respectively.[1] This gives us an obvious $O(n^3)$ time algorithm to construct an optimal binary search tree.

Let $C_{i,j}$ be the weighted path length of $T_{i,j}$. We can compute all $C_{i,j}$ in a bottom-up fashion, using the following dynamic program. The weighted path length of $T_{1,n}$ will then be $C_{1,n}$.

[1]We will assume that $T_{i,i-1}$ is the empty binary tree.

Compute $W_{i,j}$ for all i and j.
for $1 \leq i \leq n$ **do**
  $C_{i,i} = P_i$
end for
for $1 \leq i < n$, in reverse order **do**
  for $i < j \leq n$ **do**
    $C_{i,j} = \infty$
    for $i \leq k \leq j$ **do**
      if $C_{i,k-1} + C_{k+1,j} + W_{i,j} < C_{i,j}$ **then**
        $C_{i,j} = C_{i,k-1} + C_{k+1,j} + W_{i,j}$
      end if
    end for
  end for
end for

3 Knuth's Quadratic Time Algorithm

Let $R_{i,j}$ be the index of the root of $T_{i,j}$, that is, the best choice of k in the range $i \leq k \leq j$. Knuth observed that $R_{i,j-1} \leq R_{i,j} \leq R_{i+1,j}$ for all $1 \leq i < j \leq n$. This allows us to speed up the algorithm by elimination most of the searching done in the third (interior) loop of the algorithm.

Compute $W_{i,j}$ for all i and j.
for $1 \leq i \leq n$ **do**
  $C_{i,i} = P_i$
  $R_{i,i} = i$
end for
for $1 \leq i < n$, in reverse order **do**
  for $i < j \leq n$ **do**
    $R_{i,j} = R_{i,j-1}$
    $k = R_{i,j-1}$
    $C_{i,j} = C_{i,k-1} + C_{k+1,j} + W_{i,j}$
    while $k < R_{i+1,j}$ **do**
      $k++$
      if $C_{i,k-1} + C_{k+1,j} + W_{i,j} < C_{i,j}$ **then**
        $C_{i,j} = C_{i,k-1} + C_{k+1,j} + W_{i,j}$
        $R_{i,j} = k$
      end if
    end while
  end for
end for

Compute an optimal binary search tree on the list $2, 3, 8, 5, 4, 9, 6, 2, 1$. Show the matrices W, R, and C.