Optimal Binary Search Tree ....

input: A sequence $p[1..n]$ of frequencies, where $p[j]$ is the frequency of key j.

output: The minimum cost of any *binary search tree* T for the n keys. A binary search tree is a rooted binary tree with the n keys $\{1, 2, \ldots, n\}$, as nodes, *in search-tree order.* (So, at each node, the key is larger than the keys in the node's left subtree, and smaller than the keys in the node's right subtree.) The cost of T is the weighted average depth of the keys:

$$\mathsf{cost}(T) = \sum_{j=1}^{n} p[j]\,\mathsf{depth}_T(j).$$

For any such tree T and key j, define T_j to be the subtree whose root node has key j.

Observation 1. *For any tree T and key j, the set of keys in the subtree T_j is some contiguous sequence $\{i, i+1, \ldots, k\}$, where $i \le j \le k$. So any such subtree T_j is a binary search tree for the* Optimal Binary Search Tree *instance $p[i..k]$.*

The cost of T_j for its instance $p[i..k]$ is the weighted average depth of keys in T_j, that is, $\mathsf{cost}(T_j) = \sum_{h=i}^{k} p[h]\,\mathsf{depth}_{T_j}(h)$. For any such subproblem $p[i..k]$, define $\mathsf{wt}(p[i..k]) = \sum_{h=i}^{k} p[i]$ to be the sum of the frequencies in the instance $p[i..k]$.

Observation 2. *Consider any subtree T_j. Let $p[i..k]$ be the subproblem it solves. Let T_ℓ and T_r be its left and right subtrees. Then $\mathsf{cost}(T_j) = \mathsf{wt}(p[i..k]) + \mathsf{cost}(T_\ell) + \mathsf{cost}(T_r)$.*

Also, the subproblems solved by T_ℓ and T_r are $p[i..j-1]$ and $p[j+1..k]$.

Here is the algorithm. Fix any input instance $p[1..n]$.

subproblems. For each $i, k \in \{1, 2, \ldots, n\}$ define $M[i, k]$ to be the minimum cost of any binary search tree for the subproblem $p[i..k]$. (We allow $i > k$, in which case $p[i..k]$ is empty and $M[i, k] = 0$.) The given instance $p[1..n]$ has solution $M[1, n]$.

recurrence. For any i and k with $i > k$ (so that $p[i..k]$ is empty), $M[i, k] = 0$. Otherwise

$$M[i, k] = \min\big\{\,\mathsf{wt}(p[i..k]) + M[i, j-1] + M[j+1, k] : j \in \{i, i+1, \ldots, k\}\big\}.$$

The intuition is that an optimal tree for $p[i..k]$ can be obtained by choosing some key $j \in \{i, i+1, \ldots, k\}$ for the root, then choosing any optimal tree T_ℓ for subproblem $p[i..j-1]$ as the left subtree, and any optimal tree T_r for subproblem $p[j+1..k]$ as the right subtree. The subtree for $p[i..k]$ is then a subtree T_j, and, by Observation 2, its cost is $\mathsf{cost}(T_j) = \mathsf{wt}(p[i..k]) + \mathsf{cost}(T_\ell) + \mathsf{cost}(T_r) = \mathsf{wt}(p[i..k]) + M[i, j-1] + M[j+1, k]$.

time. There are $O(n^2)$ subproblems. For each, the right-hand side of the recurrence can be evaluated in time $O(n)$. So the total time to compute $M[1, n]$ is $O(n^3)$.

correctness. To prove correctness, we would give a more careful proof that the recurrence relation is correct. Correctness of the algorithm would follow by induction (on the size of the subproblem).

Is this algorithm equivalent to running the Shortest Path algorithm on an appropriate graph?

External resources on Optimal Binary Search Tree

- CLRS Chapter 15
- Jeff Edmond's Lecture 5
  `http://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-dynprog.pdf`