Optimal Binary Search Trees

Subhash Suri

November 2, 2017

1 Optimal Binary Search Trees

- Binary search trees are used to organize a set of keys for fast access: *the tree maintains the keys in-order so that comparison with the query at any node either results in a match, or directs us to continue the search in left or right subtree.*
- A balanced search tree achieves a worst-case time $O(\log n)$ for each key search, but fails to take advantage of the structure in data.
- For instance, in a search tree for English words, a frequently appearing word such as "the" may be placed deep in the tree while a rare word such as "machiocolation" may appear at the root because it is a median word.
- In practice, key searches occur with different frequencies, and an *Optimal Binary Search Tree* tries to exploit this non-uniformity of access patterns, and has the following formalization.
- The input is a list of keys (words) $w_1, w_2, \ldots, w_n$, along with their access probabilities $p_1, p_2, \ldots, p_n$. The prob. are known at the start and do not change.
- The interpretation is that word w_i will be accessed with *relative frequency* (fraction of all searches) p_i. The problem is to arrange the keys in a binary search tree that **minimizes the (expected) total access cost.**
- In a binary search tree, accessing a key at depth d incurs search cost $d + 1$. Therefore, if the word w_i is placed at depth d_i in the tree, the total search cost (the quantity we want to minimize) is:

$$\sum_{i=1}^{n} p_i \times (d_i + 1)$$

- An example:

Word	Probability
a	0.22
am	0.18
and	0.20
egg	0.05
if	0.25
the	0.02
two	0.08

- Notice that the access probabilities of these 7 words sum to 1.
- Now look at the following 3 search trees:

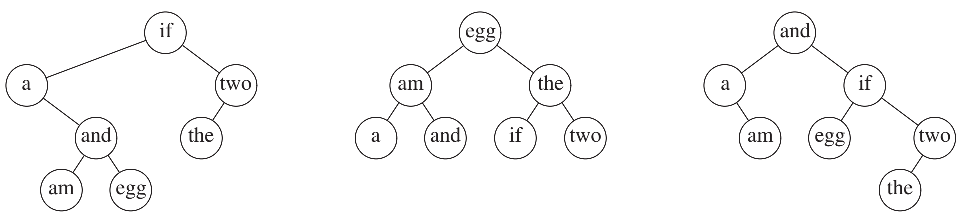

Figure 1: Greedy, Balanced, and Optimal search trees.

- The three trees are constructed by a Greedy method, balanced tree, and optimal tree.
  - The greedy puts the most frequent word at the root, and then recursively builds the left and right subtrees.
  - The balanced makes the height the smallest.
  - The third is created by the optimal algorithm, about to be discussed.
- The costs of these trees are: 2.43 (Greedy), 2.70 (Balanced), and 2.15 (Optimal).

  For instance, the Greedy tree's search cost is calculated as

$$0.22 \times 2 + 0.18 \times 4 + 0.20 \times 3 + 0.05 \times 4 + 0.25 \times 1 + 0.02 \times 3 + 0.08 \times 2 \quad = \quad 2.43.$$

- Neither greedy nor balanced is optimal.
- The problem is also different in two crucial ways from the Huffman coding problem.: First, the keys are not restricted to be in leaves only (no prefix problem), as was the case in Huffman. Second, the in-order of the keys is fixed—dictated by the ordering of the keys.

- The Dynamic Program for the optimal search tree follows the same pattern we have seen multiple times now.
  - We consider a sub-problem $[i, j]$, namely, the subset of words $w_i, \ldots, w_j$.
  - Let $S(i, j)$ be the total search cost for the optimal tree for this subproblem.
  - Suppose the opt tree for this subproblem has w_r as root, where $i \leq r \leq j$, with depth 0, then the picture looks like the following:

```
                    wr



wi, .., w{r-1}              w{r+1}, ..., wj
```

- We can therefore write the following recurrence for the total cost of this tree:

$$S(i, j) \;=\; p_r + S(i, r-1) + S(r+1, j) + \sum_{k=i}^{r-1} p_k + \sum_{k=r+1}^{j} p_k,$$

  which has the following explanation.

- The root w_r has depth 0, and search cost 1, so it contributes $p_r \times 1$ to the overall cost.

- $S(i, r-1)$ and $S(r+1, j)$ are the search trees for their subproblems *assuming we count the search from their respective roots.*

- Making them children of w_r increases the path length of each of their nodes by 1, and so the two remaining terms are simply adding those additional costs.

- We can simplify this calculation as follows:

$$S(i, j) \;=\; S(i, r-1) + S(r+1, j) + \sum_{k=i}^{j} p_k$$

- This shows that the problem satisfies the principle of optimality: since the last term is fixed regardless of how the two subtrees are built, the optimal solution for $[i, j]$ must use optimal solutions for the subproblems $[i, r-1]$ and $[r+1, j]$.

- Finally, as usual, since we don't the r, we optimize this expression over all choices of r, giving us the final recurrence, for $i < j$.

$$S(i,j) \;=\; \min_{i \le r \le j} \{S(i, r-1) + S(r+1, j) + \sum_{k=i}^{j} p_k\}$$

  If $i \geq j$, then $S[i,j] = 0$ clearly.

- The running time is $O(n^3)$ time.