CS124	Notes 12	Spring 2015

1 Optimal BST in quadratic time

In the last lecture note we gave an $O(n^3)$ algorithm for the optimal BST problem. Here we restate the problem (with a slight simplification that doesn't affect the core complexity of the problem; namely we ignore dummy nodes and assume users always queries for keys actually in the database), and then we describe an $O(n^2)$ time solution due to Knuth.

Recall we have n keys $k_1 < k_2 < \ldots < k_n$. We also have frequencies: K_i is the number of times key k_i is queried. We would like to organize the keys into a single binary search tree so that the cost of servicing all queries is minimized. The cost of a particular binary search tree T is the total number of key comparisons that must be done to service all $\sum_i K_i$ queries. This is equal to

$$\sum_{i=1}^{n} \text{level}_T(i) \cdot K_i$$

where the level of the root is considered to be 1.

Let us define $w(i,j)$ to be $\sum_{t=1}^{j} K_t$. Note in preprocessing we can create an $n \times n$ 2d-array containing all the $w(i,j)$ values, and the time taken is $O(n^2)$ (once we calculate $w(i,i) = K_i$, we can obtain loop over j i and set $w(i,j) = w(i,j-1) + K_j$).

Now, let $f(i,j)$ be the minimum cost of a tree to service all queries to keys $k_i, k_{i+1}, \ldots, k_j$. Then we have a recurrence relation:

$$f(i,j) = \begin{cases} K_i, & \text{if } i = j \\ w(i,j) + \min_{i \le r \le j} f(i,r-1) + f(r+1,j), & \text{otherwise} \end{cases}$$

In the recursive step, we can choose any key k_r for $i \le r \le j$ to be the root. Then the keys $k_i, k_{i+1}, \ldots, k_{r-1}$ go in the left subtree, and we should recursively try to build the best tree on them. The keys $k_{r+1}, \ldots, k_j$ go in the right subtree, and we should recursively try to build the best tree on them as well. The $w(i,j)$ is added in because any query on this set of keys must cause a comparison with the root, and there are $w(i,j)$ such queries.

Knuth's trick to speed up the DP solution above is as follows. Let $root[i][j]$ be the minimizer of the expression $f(i,r-1) + f(r+1,j)$; i.e. it is the r value which achieves the minimum in the recursive step. (Break ties arbitrarily;

also $root[i][i]$ is i.) Note we can record these values as we are building up the table of answers. Now, we change our recursion to be as follows. Define a new recurrence:

$$g(i,j) = \begin{cases} K_i, & \text{if } i = j \\ w(i,j) + \min_{root[i][j-1] \le r \le root[i+1][j]} g(i,r-1) + g(r+1,j), & \text{otherwise} \end{cases}$$

That is, the recurrence is exactly the same *except* that we loop over a different set of possibilities for r. We will show in these notes that in fact $f = g$; these recurrence relations define the same function! Also note that if we fill in the DP table in increasing order of $j-i$, then when we want to compute $g(i,j)$ we will already know $root[i][j-1]$ and $root[i+1][j]$ (see code below, where d represents $j-i$):

```
initialize g[i][i] to K_i for all i = 1..n
initialize g[i+1][i] to 0 for all i = 0..n
initialize root[i][i] to i for all i = 1..n
for d = 1 to n-1:
    for i = 1 to n-d:
        j = i+d
        g[i][j] = INFINITY
        for root[i][j-1] <= r <= root[i+1][j]:
            if w(i, j) + g[i][r-1] + g[r+1][j] < g[i][j]:
                g[i][j] = w(i, j) + g[i][r-1] + g[r+1][j]
                root[i][j] = r
```

Claim 12.1 *Given any set of keys $k_1 < \ldots < k_{n-1}$ and query frequencies $K_1, \ldots, K_{n-1}$, let $1 \le r \le n-1$ be optimal choice of root (break ties arbitrarily). Then, by adding one more key k_n larger than all other keys, and for any query frequency K_n, there will always be an optimal BST whose root does not shift to the left, i.e. a root no smaller than r.*

Given the symmetry of the problem, the above claim would also imply $root[i][j] \le root[i+1][j]$, i.e. adding one more key *smaller* than all other keys can never shift the root to the right.

Before proving the claim, let us see why this takes time $O(n^2)$. The running time of the above code to fill in the DP table is (since $j = i+d$):

$$O\left(\sum_{d=1}^{n-1}\sum_{i=1}^{n-d}(1 + root[i+1][i+d] - root[i][i+d-1])\right) = O\left(\sum_{d=1}^{n-1}\left[(n-d) + \sum_{i=1}^{n-d}(root[i+1][i+d] - root[i][i+d-1])\right]\right)$$

The key insight then is that the sum over i is a telescoping sum, and thus the above is

$$\sum_{d=1}^{n-1}[(n-d)+root[n-d+1][n]-root[1][d]] \leq \sum_{d=1}^{n-1}[(n-d)+root[n-d+1][n]] \leq \sum_{d=1}^{n-1}[(n-d)+n],$$

which is at most $2n^2 = O(n^2)$.

Now we just need to prove Claim 12.1 above.

Proof: The proof presented here follows the original of Knuth [Knuth71]. We prove the claim by induction on n. The claim is vacuously true for $n = 1$. For $n > 1$, let $T(\alpha)$ be the optimal tree on keys $k_1,\ldots,k_n$ with frequencies K_i but where $K_n = \alpha$. Let α be any value such that T is an optimum tree with the largest possible root when $K_n = \alpha - \varepsilon$, but it is not optimal when $K_n = \alpha + \varepsilon$ for all sufficiently small $\varepsilon > 0$. That is, the structure of the optimum tree changes at the value $K_n = \alpha$. Let T' be an optimal tree for $K_n = \alpha + \varepsilon$, and choose T' to have the largest root possible. We would like to argue that the root of T' cannot be smaller than that of T. Since the root in the case $\alpha = 0$ is the same as the root for the optimal tree on $k_1,\ldots,k_{n-1}$, this would prove the claim (gradually as we increase α, roots can only move to the right).

Now, for the sake of contradiction, suppose the root of T' is less than the root of T, i.e. the root moved to the left. Since the cost of a tree is a linear function in frequencies with coefficients being the levels, it must mean that $\ell_{T'}(n) < \ell_T(n)$, where $\ell_T(i)$ denotes the level of key i in T.

In both T and T', k_n must live in the rightmost path starting from the root. Suppose the indices of the keys traversed in T, starting from the root, to get to key n are $i_1 < i_2 < \ldots < i_{\ell_T(n)}$ (where $i_{\ell_T(n)} = n$), and in T' they are $j_1 < j_2 < \ldots < j_{\ell_{T'}(n)}$ (where $j_{\ell_{T'}(n)} = n, \ell_{T'}(n) < \ell_T(n)$). Since $j_1 < i_1$, by induction we know $j_2 \leq i_2$. If $j_2 < i_2$, then again by induction we have $j_3 \leq i_3$. Continuing in this way, since $\ell_{T'}(n) < \ell_T(n)$ we must at some point t have $j_t = i_t$. Then we can replace the right subtree of k_{i_t} in T with the right subtree of $k_{j_t} = k_{i_t}$ in T', obtaining a new tree whose cost is equal to that of T'. But this contradicts that T' is the optimal tree with the largest possible root. ■

One may wonder: more generally when can the optimization above be made for a dynamic programming problem? That is, when can a recurrence of the form f be replaced with one of the form g such that $f(i,j)$ still equals $g(i,j)$ for all i,j? As an exercise, you should show that if the recurrence is of the form for f above for an arbitrary weight function $w(i,j)$, and furthermore for any $a \leq b \leq c \leq d$ the following two inequalities hold:

- $w(a,c)+w(b,d) \leq w(a,d)+w(b,c)$
- $w(b,c) \leq w(a,d)$

then in fact $f = g$.

References

[1] Donald E. Knuth. Optimum Binary Search Trees. *Acta Inf.* 1: 14–25, 1971.