Introduction to Algorithms, Fall 2012
Homework #2 Sample Solution

November 2, 2012

15.5-4 In this exercise, we assume the inequality $root[i, j-1] \leq root[i, j] \leq root[i+1, j]$ holds for all $1 \leq i < j \leq n$. For the reason why this inequality is true, one can refer to Knuth's paper [212] or *The Art of Computer Programming, Vol. 3.* The key idea of reducing running time is that once the values of $root[i, j-1]$ and $root[i+1, j]$ have been found then $root[i, j] = r$ can be searched in that interval. That is to say, we do not need to try all values between i and j. Hence a modification for OPTIMAL-BST(p, q, n) is to change Line 10 from "**for** $r = i$ **to** j" to "**for** $r = root[i, j-1]$ **to** $root[i+1, j]$". After that, the running time is

$$
\begin{aligned}
T_{\text{new}} =& \overbrace{2n}^{\text{Line 2-4}} \quad + \overbrace{\sum_{l=1}^{n} \sum_{i=1}^{n-l+1} \{root[i+1, i+l-1] - root[i, i+l-2] + 1\}}^{\text{Line 5-14}} \\
=& 2n \quad + \sum_{l=1}^{n} \left\{ \begin{array}{l} root[(n+l+1)+1, (n+l+1)+l-1] \\ \qquad\qquad - root[(n+l+1), (n+l+1)+l-2] + 1 \\ \qquad\qquad \ldots \\ +root[3+1, 3+l-1] - root[3, 3+l-2] + 1 \\ +root[2+1, 2+l-1] - root[2, 2+l-2] + 1 \end{array} \right\} \\
=& \ldots \qquad \text{(details omitted)} \\
=& \Theta(n^2).
\end{aligned}
$$

Note that the above result is based on the fact $1 \leq root[i, j] \leq n$ for all i and j.

P.S. Here is an *unreasonable* recurrence:

$$e[i, j] = min \{root[i, j-1] \leq r \leq root[i+1, j], e[i, r-1] + e[r+1, j] + w[i, j]\}.$$

15-1 Denote $f(x)$ as the weight of longest weighted path from x to t, and $p(x)$ as the very y that makes $f(x)$ be the maximum.
We define a recursive function $f(x)$ as follows:

$f(t) = 0$ (Base case);
$f(x) = max\{f(y) + edgeWeight(y, x)\}$ for every y that is adjacent to x,
and record the very y that produces the maximum value into $p(x)$.
(We could find the path using this information.)

Solve the f function in the topological order.

Because any answer of the vertex depends on its neighbors only,
the subprogram graph looks exactly the same as $G(V, E)$.

The algorithm runs in $O(|V| + |E|)$.

15-2 Define:
(1) $c[i]$ as the *character* at the position i.
(2) $+$ as the string *concatenation operator* between strings.
(3) $max(\cdot)$ be the *function* comparing the *length* of two strings and return the longer one.

We define a recursive function $f(x, y)$ as follows:

$f(x+1, x) =$ Empty string (Base cases).
$f(x, x) = c[i]$ (Base cases).
If $c[x] == c[y]$
$f(x, y) = c[x] + f(x+1, y-1) + c[y]$;
else
$f(x, y) = max\{f(x+1, y), f(x, y-1)\}$.

Denote the length of the input string as L.
The algorithm runs in $O(L^3)$.
In fact, we can improve the running time to be $O(L^2)$ if it is implemented well.

15-6 The company hierarchical structure can be explained as a tree whose root is r. For each node x in this tree, if x is invited, we denote $M(x)$ as the maximum conviviality rating of the subtree with the root x. On the other hand, if x is not invited, we denote $M'(x)$ as the maximum conviviality rating of the subtree with the root x. Note that only one of an employee and her/his immediate supervisor can be invited. We can construct a recursive function as follows:

$$\begin{cases} M(x) = \sum_{y \in x's\ child} M'(y) + x's\ conviviality\ ratinig \\ M'(x) = \sum_{y \in x's\ child} max\{M'(y), M(y)\} \end{cases}$$

We can compute this problem by $max\{M(r), M'(r)\}$. The above recursion shows that for each fixed x, we refer $M(x)$ and x both once and $M'(x)$ twice. So every node is traced at most four times, which means the time complexity is $O(N)$.

15-9 By observing optimal substructures for every substring in S, we can construct a recursive function as follows:

$$c[i,j] = \begin{cases} 0 & \text{if } i = j-1 \ or \ i = j \\ L[j] - L[i] & \text{if } i = j-2 \\ Min_{i<k<j}\{c[i,k] + c[k,j]\} + L[j] - L[i] & otherwise \end{cases}$$

where $c[i, j]$ is the minimum cost for breaking substring between $L[i]$ and $L[j]$. We can construct an algorithm by this recursion:

input: a string S, breaking-point array with increasing order $L[0...m+1]$, $L[0] = 0,\ L[m+1] = N$
output: the minimum cost c and a sequence of break $b[1...m]$.

Let $c[0...m+1, 0...m+1]$, $index[0...m+1, 0...m+1]$ be new tables,
    $b[1...m]$ be a new array.
for $i = 0$ **to** $m + 2$
    $c[i, i] = 0$
for $i = 0$ **to** $m + 1$
    $c[i, i + 1] = 0$
for $i = 0$ **to** m
    $c[i, i + 2] = L[i + 2] - L[i]$
    $index[i, i + 2] = i + 1$
for $len = 3$ **to** $m + 2$
    $min = \infty$
    for $i = 1$ **to** $m - len + 2$
        for $k = 1$ **to** $len - 1$
            if $c[i, i + k] + c[i + k, i + len] < min$
                $min = c[i, i + k] + c[i + k, i + len]$
                $index[i][i + len] = k + i$
$FIND\_SEQ(b, index, 0, m + 1)$
return $c[0, m + 1]$ and b

$FIND\_SEQ(b, index, start, end)$
    if $end - start \leq 1$
    return
$cut = index[start, end]$
    $b.append(cut)$
    $FIND\_SEQ(b, index, start, cut)$
    $FIND\_SEQ(b, index, cut, end)$

Time complexity: there are 3 nested for loops in our main function, and the subroutine $FIND\_SEQ$ puts m elements to the array b, so its time complexity is $O(m^3+m) = O(m^3)$.