CSCI3160: Regular Exercise Set 6

Prepared by Yufei Tao

Problem 1*. Let A be an array of n integers. Define a function $f(x)$ — where $x \geq 0$ is an integer — as follows:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \max_{i=1}^{x}(A[i] + f(x-i)) & \text{otherwise} \end{cases}$$

Consider the following algorithm for calculating $f(x)$:

algorithm $f(x)$
if $x = 0$ **then return** 0
$max = -\infty$
for $i = 1$ **to** x
    $v = A[i] + f(x - i)$
    if $v > max$ **then** $max = v$
return max

Prove: the above algorithm takes $\Omega(2^n)$ time to calculate $f(n)$.

Solution. Let $g(x)$ denote the time of the algorithm in calculating $f(x)$. We know:

$$\begin{aligned} g(0) &\geq 1 \\ g(1) &\geq 1 \\ g(n) &\geq \sum_{i=0}^{n-1} g(i) \end{aligned}$$

We will show by induction that $g(n) \geq 2^{n-1}$ for $n \geq 1$. First, this is obviously correct when $n = 1$. Next, we will prove the claim on $n = k$ for any $k \geq 2$, assuming that it is correct for all $n \leq k - 1$.

$$\begin{aligned} g(n) &\geq \sum_{i=0}^{n-1} g(i) \\ &\geq 1 + \sum_{i=1}^{n-1} g(i) \\ &\geq 1 + \sum_{i=1}^{n-1} 2^{i-1} \\ &\geq 2^{n-1}. \end{aligned}$$

Problem 2. Consider once again Problem 1. Design an algorithm to calculate $f(n)$ in $O(n^2)$ time.

Solution. Calculate $f(x)$ in ascending order of $x = 0, 1, ..., n$. After $f(0), ..., f(x-1)$ are ready, $f(x)$ can be obtained in $O(1 + x)$ time. The total running time is therefore $\sum_{x=0}^{n} O(1 + x) = O(n^2)$.

Problem 3. Recall that, on the optimal BST problem, we have explained in the class how to calculate $optavg(1, n)$ using dynamic programming in $O(n^3)$ time where function $optavg(a, b)$ is recursively defined as

$$optavg(a,b) \quad = \quad \begin{cases} 0 & \text{if } a > b \\ \sum_{i=a}^{b} W[i] + \min_{r=a}^{b}\{optavg(a,r-1) + optavg(r+1,b)\} & \text{otherwise} \end{cases}$$

However, we have not yet explained how to build in an optimal BST. Describe an algorithm to do so in $O(n^3)$ time (in fact, you can build the tree in $O(n)$ time after having computed $optavg(1, n)$, but you will need to modify what we did in dynamic programming slightly).

Solution. Recall that using dynamic programming we can obtain $optavg(a, b)$ for all $1 \le a \le b \le n$. For any such a, b define $bestroot(a, b)$ to be the $r \in [a, b]$ that minimizes

$$optavg(a, r-1) + optavg(r+1, b)$$

It is straightforward to slightly extend the algorithm to compute also $bestroot(a, b)$, for all a, b satisfying $1 \le a \le b \le n$, also within the same time complexity $O(n^3)$.

Then we can construct an optimal BST as follows. First, create a root node u with the key $r = bestroot(1, n)$. Recursively create an optimal BST T_1 on the set $\{1, 2, ..., r-1\}$ and an optimal BST T_2 on the set $\{r+1, r+2, ..., n\}$. Make the root of T_1 the left child of u, and then the root of T_2 the right child of u.

Problem 4 (Rod-Cutting; Section 15.1 of the Textbook). Let A be an array of n integers. Let us define an *n-sum sequence* as a sequence of integers $x_1, x_2, ..., x_t$ (where t can be any integer at least 1) satisfying both conditions below:

- $1 \le x_i \le n$ for all $i \in [1, t]$
- $\sum_{i=1}^{t} x_i = n$.

Define the *cost* of the above n-sum sequence as $\sum_{i=1}^{t} A[x_i]$. Give an algorithm to produce an n-sum sequence with the largest cost in $O(n^2)$ time.

Solution. Define function $opt(x)$ as the largest cost of all x-sum sets; specially, if $x = 0$, define $opt(x) = 0$. This function satisfies:

$$opt(x) \quad = \quad \max_{i=1}^{x}\{A[i] + opt(x-i)\}.$$

In Problem 2, we have given an algorithm to calculate $opt(n)$ in $O(n^2)$ time.

An n-sum sequence with the greatest cost can be produced in another $O(n)$ time as follows. For any $x \ge 1$, define $bestChoice(x)$ to be the $i \in [1, x]$ that maximizes

$$A[i] + opt(x-i).$$

The values $bestChoice(x)$ of all $x \in [1, n]$ can be computed by slightly modifying the algorithm in Problem 2 without increasing its time complexity. To produce an optimal sequence, first set $x_1 = bestChoice(n)$. If $x_1 = n$, we are done; otherwise, append to x_1 an optimal $(n - x_1)$-sum sequence.