CS4311 Design and Analysis of Algorithms

Homework 4 (Solution Sketch)

1. **Main Idea:** Use an array Best[1..n] such that Best[i] stores the maximum total costs of dishes you can select, under the condition that the ith dish must be taken, and the ith dish is the last dish.

   In general, we can argue that

   $$\text{Best}[i] = \text{cost of } i\text{th dish } + \max_{j<i} \{\ \text{Best}[j] \mid \text{cost of } j\text{th dish} < \text{cost of } i\text{th dish}\ \},$$

   so that Best[1..n] can be computed in a total of $O(n^2)$ time. Then, the maximum of Best[1..n] is equal to the maximum total costs you can select. Finally, to decide which dish to be selected, this can be done by the standard "backtracking" technique we discussed in the class (by maintaining a pointer to indicate how each Best[i] comes from).

2. **Main Idea:** Use of an $L \times n$ array Best such that Best[i, j] indicates the maximum total costs of coupon you can collect, under the condition that you have to reach Step j using exactly i steps, and ending at Step j.

   The entries Best[$1, 1$], Best[$1, 2$], and Best[$1, 3$] should be assigned with the values of the coupon in Steps 1, 2, and 3, respectively. The other Best[$1, j$] entries for $j \geq 4$ should be assigned with a value $-\infty$ to indicate it is impossible to reach these j's in exactly one step.

   In general, we can argue that

   $$\text{Best}[i, j] = \text{value of } j\text{th coupon} + \max\{\ \text{Best}[i-1, j-1], \text{Best}[i-1, j-2], \text{Best}[i-1, j-3]\ \}.$$

   so that all Best entries can be computed in $O(Ln) = O(n^2)$ time. Then the maximum value of coupon we can get, using at most L steps, is the maximum of Best[i, n] for all $i = 1, 2, \ldots, L$. In addition, to decide the corresponding steps that achieve the maximum value, this can be done by standard backtracking.

3. **Main Idea:** Make the tree a rooted tree with root r, and process the tree in a bottom-up manner. Each node v stores a value $M(v)$, which indicates the maximum customers we can obtain, only opening stores in the subtree rooted at v, under the condition that v opens a store. Each node v also stores a similar value $m(v)$, but assuming v does not open a store.

   In general, we can argue that

   $$M(v) = \#\text{customers at } v + \sum_{u \text{ is a child of } v} m(u),$$

   and

   $$m(v) = \sum_{u \text{ is a child of } v} \max\{M(u,), m(u)\}.$$

   Each $M(v)$ or $m(v)$ value can be computed in $O(n)$ time, so that in total $O(n^2)$ time. A closer observation reveals that the time for each node is proportional to the number of its children, so that the total time is proportional to the number of edges in the tree, which is $O(n)$. Then the maximum customer we can get is equal to the maximum of $M(r)$ and $m(r)$ at the root node r. In addition, deciding the location of the stores can be done by standard backtracking.

4. **Main Idea:** Use a $n \times V$ array Best such that Best$[i, j]$ indicates the maximum cost of food you can select, only with dishes from dish 1 to dish i, whose total volume is exactly j.

   In general, we can argue that

$$\text{Best}[i, j] = \max \{ \text{ Best}[i-1, j], \quad \text{cost of dish } i + \text{Best}[i-1, j - \text{volume}(i)] \} .$$

   so that all Best entries can be computed row by row in $O(nV)$ time. Then the maximum cost of dishes we can select is equal to Best$[n, V]$. In addition, deciding the corresponding dishes that achieve the maximum value can be done by standard backtracking.