CS/ECE 374 ✧ Spring 2017
Homework 6

Due Wednesday, March 15, 2017 at 10am

Groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

1. **Solution:** Let $F(i,l)$ denote the maximum profit we can make considering only the location $L_i, \cdots, L_n$ and opening at most l restaurants. Also, add an extra location L_{n+1} with $m_{n+1} = \infty$. For each $F(i,l)$, we can decide to: (1) select the current restaurant L_i, adding up its profit p_i and the maximum profit we can make from those restaurants starting at the next available location that is at least D distance away from L_i with opening at most $l-1$ restaurants, or (2) skip L_i and consider the maximum profit we can make from those restaurants starting at location L_{i+1} with opening at most l restaurants. Then we take the maximum of the two cases:

$$F(i,l) = \begin{cases} 0 & i > n \text{ or } l = 0 \\ \max\{p_i + F(next(i), l-1), F(i+1, l)\} & \text{otherwise} \end{cases}$$

where $next(i) = \min\{j : m_j \geq m_i + D\}$ is the next available location that is at least D distance away from the current location at L_i. As we have shown in HW5 Q3, all $next(i), \forall i = 1, \cdots, n$ can be calculated in $O(n)$ time. The initial call to the recurrence will be $F(1,k)$.

Naive way to memorize is to store into a two-dimensional array $F[1, \cdots, n+1; 0, \cdots, k]$. Space for the array is $O(nk)$. However, notice that a subproblem in the column indexed by l only depend on subproblems in the same column or the previous column indexed by $l-1$. Therefore, only two columns instead of k columns need to be memorized. Space needed is $O(n)$.

There are $O(nk)$ distinct subproblems. If $next(i)$ is preprocessed, each subproblem takes $O(1)$ time. There will be $O(n)$ extra space needed. Runtime is $O(nk)$ and preprocessing time is $O(n)$. Total runtime is $O(nk)$ and total space is $O(n)$.

Without the preprocessing, there will be $O(\log n)$ or $O(n)$ time cost for each $next(i)$ in each subproblem. The total running time will be $O(nk \log n)$ or $O(n^2 k)$ and the total space will be $O(n)$.

■

Rubric: Standard DP rubric. 10 points total. Furthermore:

- **0 point** for solving the problem without using constraint k.
- **5 points max** for no recurrence/pseudo code but correct description and idea presented.
- We do not penalize for not preprocessing $next(i)$.

2. **Solution:** Let $SCSS(i,j,k)$ be the length of the shortest common supersequence of $x_1 \ldots x_i$, $y_1 \ldots y_j$, and $z_1 \ldots z_k$. Also, define x_0, y_0, z_0 as three new symbols not used in X, Y, Z. The recurrence below satisfies this definition.

$$SCSS(i,j,k) = \begin{cases} & \textbf{Base cases:} \\ \infty & \text{if } i < 0 \text{ or } j < 0 \text{ or } k < 0 \\ 0 & \text{if } i = 0,\ j = 0,\ k = 0 \\ & \textbf{When all rightmost characters match:} \\ 1 + SCSS(i-1,\ j-1,\ k-1) & \text{if } x_i = y_j = z_k \\ & \textbf{When two rightmost characters match:} \\ 1 + \min \begin{cases} SCSS(i,\ j-1,\ k-1) \\ SCSS(i-1,\ j,\ k) \end{cases} & \text{if } y_j = z_k \neq x_i \\ 1 + \min \begin{cases} SCSS(i-1,\ j,\ k-1) \\ SCSS(i,\ j-1,\ k) \end{cases} & \text{if } x_i = z_k \neq y_j \\ 1 + \min \begin{cases} SCSS(i-1,\ j-1,\ k) \\ SCSS(i,\ j,\ k-1) \end{cases} & \text{if } x_i = y_j \neq z_k \\ & \textbf{When all rightmost characters are different:} \\ 1 + \min \begin{cases} SCSS(i-1,\ j,\ k) \\ SCSS(i,\ j-1,\ k) \\ SCSS(i,\ j,\ k-1) \end{cases} & \begin{array}{l} \text{(otherwise)} \\ \text{if } x_i \neq y_j,\ y_j \neq z_k,\ x_i \neq z_k \end{array} \end{cases}$$

For a dynamic programming algorithm using this recurrence, we memoize $SCSS(i,j,k)$ in a 3D array with dimensions $(r+1) \times (s+1) \times (t+1)$, noting that there are extra cells for when $i,j,k = 0$. All cells depend on cells with lower coordinates, so we fill the array like so: fill the array in 2D slices from lowest k to highest k, where for each 2D slice, we fill the i dimension in increasing order, and for each i, we fill the j dimension in increasing order. The goal is to compute $SCSS(r,s,t)$, which gives the shortest common subsequence for all of X, Y, and Z.

Using the filling order, each individual array cell can be filled in constant time; filling a single array cell requires at most three constant-time memoized lookups in a min function, which is still constant time overall for each cell. To fill the whole array, $O(r \cdot s \cdot t)$ cells must be filled. This gives $O(rst)$ as the running time and $O(rst)$ as the space.[1] ■

Rubric: Standard DP rubric. Furthermore:
- **5 points max** for giving super-sequence solution for 2 strings instead of 3.
- **5 points max** for no recurrence/pseudo code but correct description and idea presented.

[1]You don't need to specify the space requirement. Also, technically, it takes $O(\log(r+s+t))$ bits to write down the value in each cell, since $r+s+t$ is the longest that the shortest common supersequence might be if you concatenate the three sequences, so the space and time bounds might be better stated as $O(rst\log(r+s+t))$.

3. **Solution:** Let T be the binary input tree. For each node v in T, let $\mathit{MinRounds}(v)$ denote the minimum number of rounds required, after v learns the message, to inform every descendant of v. We need to compute $\mathit{MinRounds}(\mathit{root}(T))$. This function obeys the following recurrence:

$$\mathit{MinRounds}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + \mathit{MinRounds}(v.\mathit{left}) & \text{if } v \text{ has no right child} \\ 1 + \mathit{MinRounds}(v.\mathit{right}) & \text{if } v \text{ has no left child} \\ \min \left\{ \begin{array}{l} \max \left\{ \begin{array}{c} 1 + \mathit{MinRounds}(v.\mathit{left}) \\ 2 + \mathit{MinRounds}(v.\mathit{right}) \end{array} \right\} \\ \max \left\{ \begin{array}{c} 1 + \mathit{MinRounds}(v.\mathit{right}) \\ 2 + \mathit{MinRounds}(v.\mathit{left}) \end{array} \right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

This recurrence utilizes the fact that if a node v has both a left and a right child, the minimum number of rounds required for v to inform all of its descendants depends on which of v's children takes the longest to inform *its* descendants (along with which child is informed first). We can memoize this function in the tree itself, by adding a new field $v.\mathit{MinRounds}$ to each node record; because the result at each node depends only on the result at each of its children, we can calculate the value of $\mathit{MinRounds}(\mathit{root}(T))$ by filling in the records of each node in T using a post-order traversal, starting at the leaves of T and moving up towards the root.

But in fact, the following recursive algorithm evaluates this function purely recursively, implicitly memoizing $\mathit{MinRounds}(v)$ at the *parent* of v, in the temporary variables ℓ and r.

```
MinRounds($v$):
  if $v$ is a leaf
      return 0
  else if $v.right = \text{Null}$
      return $1 + \text{MinRounds}(v.left)$
  else if $v.left = \text{Null}$
      return $1 + \text{MinRounds}(v.right)$
  else
      $\ell \leftarrow \text{MinRounds}(v.left)$
      $r \leftarrow \text{MinRounds}(v.right)$
      if $\ell < r$
          return $r + 1$
      else if $\ell > r$
          return $\ell + 1$
      else
          return $\ell + 2$
```

The algorithm runs in $\boldsymbol{O(n)}$ ***time***. ■

Rubric: 10 points: standard DP rubric.