Homework No. 4
CSCI 4470/6470 Algorithms, CS@UGA, Fall 2017
Due Monday October 30, 2017

There are 100 points in total.

1. (15 points) Imagine you are planning a driving trip to Seattle in the coming fall break! Your car has a tank to hold $G$ gallons of gas when it is full. There are $n$ gas stations along the planned route. The distances between these gas stations $d_{i,i+1}$ are all known (Athens can be designated station 0 and Seattle station $n+1$). Assume your car drives $m$ miles per gallon. You would like to come up with an algorithm that allows you to make as fewer stops for gas as possible. Assume you will fill up the tank in Athens before setting off to the road. Show that the problem has a greedy choice property and prove it. (You do not need to write the algorithm.)

2. (15 points) Consider a slightly different version of the “coin change” problem. You are given $m$ types of coins with different denominations $d_1 = 1$, $d_1 < d_2 < \cdots < d_m$, with weights $w_1, w_2, \ldots, w_m$. Note that weights may not be proportional to values of coins. You want to make changes for a given $M$ cents of money so that the total weight of the coins is minimized.

Define an objective function and formulate a recursive solution that is suitable for a dynamic programming algorithm (you do not need to write the algorithm however).

3. (30 points) Consider the following problem to find the maximum number of left-right parentheses pairs. The input is a sequence of left and right parenthesis symbols, like

```
) ( ( ) ( )
1 2 3 4 5 6 7 8
```

(where the indexes are just for illustration purpose). There may be more than one way to pair left and right parentheses. For example, one pairing is $\{(2, 4), (5, 8)\}$, and
another is \{(2, 6), (3, 4), (7, 8)\}. However, we are only permitted to identify pairs that are combinations of nested pairs, e.g., like \((2, 8)\) and \((3, 4)\), and parallel pairs, e.g., \((2, 4)\) and \((5, 6), (7, 8)\). Crossing pairs, like \((2, 4)\) and \((3, 6)\) are not simultaneously allowed. **The goal** of the problem is to find the maximum number of non-crossing left-right parenthesis pairs.

The problem resembles the matrix chain multiplication problem as both problem require us to identify a best way to group data items on the input sequence and groupings can be in either parallel or nested but not crossing fashion. In light of the solution to the matrix chain multiplication problem, the following dynamic programming method has been proposed.

Assume input to be \(x_1 \ldots x_n\). Then a subproblem deals with a segment of the sequence \(x_i \ldots x_j\), where \(i \leq j\). There are a few situations to consider when it comes to further break this subproblem into smaller subproblems: (a) to pair \(x_i\) and \(x_j\) but only if \(x_i = '('\) and \(x_j = ')'\); (b) to abandon either \(x_i\) or \(x_j\); (c) to split the subsequence at the position of \(x_k\), for some \(k, i \leq k < j\).

**This question requires you:**

1. Define an objective function for the optimal solution;
2. Formulate recursively the objective function (don’t forget base cases);
3. Design an iterative algorithm to fill out DP table;

4. **(20 points)** Show how the algorithm **Recursive DFS** (along with To-Start-DFS) in the Lecture Note 4 works on the graph figure shown below. Assume for the graph, the adjacency list of every vertex is ordered alphabetically. Assume that the for loop of lines 4-5 of algorithm To-Start-DFS considers the vertices in the alphabetical order. Show the resulted depth-first-search tree (or forest). **Please show snapshot of the DFS search tree generated by the algorithm at every step when a new vertex is discovered.**
5. **(20 points)** Consider the following *Constrained Reachability* problem: the input consists of a directed graph $G(V, E)$, two vertices $s, t \in V$, and a subset $U \subseteq V$, where $U \cap \{s, t\} = \emptyset$. The output is “yes” if and only if there is a path from $s$ to $t$ in the graph **without** using any vertex in $U$. Apparently, the problem is the commonly known *Reachability* problem when $U = \emptyset$.

(1) Design a recursive DFS type of algorithm to *Constrained Reachability* solve problem. You may assume that the input graph is already stored in adjacency lists and $v, \mu$ indicates if vertex $v$ is in the set $U$ or not. The information about $\mu$ is also given along with the graph.

(2) Argue that your algorithm still runs in linear time (i.e., $O(|E| + |V|)$-time) though the set $U$ may contain a large number of vertices.
Optional Project:

This project is a coding practice for dynamic programming algorithms. Note that this project offers you an opportunity to understand better dynamic programming and how it may apply to real world problems. This project, however, is not designed to help you get extra credits for this course.

Consider an extension to problem Q3 as follow: given input sequence that is a string of symbols from alphabet \( \Sigma = \{A, C, G, U\} \), computes the maximum number of pairs between A and U and between C and G. For example,

\[
\text{AGCCUCGCUAGA} \\
(\.))O.(..)
\]

where the first line is the input sequence and the second is the output, where '(' and ')' indicate the pairings between A and U and between C and G.

This is a primitive abstraction of a practical yet challenging problem in bioinformatics called RNA secondary structure prediction. An RNA sequence consists of 4 nucleotides A, C, G, and U and they form Watson-Crick pairs through hydrogen bonding between A and U and between C and G. Since hydrogen and other chemical bondings reduce energy and increase structure stability, counting the maximum number of such pairs is an approximation (and a prediction) of how an RNA sequence may structurally fold back to itself. Secondary structure prediction is a critical step toward full atomic 3D structure prediction. Figure 1 gives a schematic illustration of transfer RNA (tRNA) folding.

More realistic objective functions can be designed to make the prediction by your program more practically useful. Typically,

1. nucleotides within a distance 4 (e.g., position 5 and 8) cannot form a Watson-Crick pairs due to steric constraints;
2. more sophisticated energy functions may be used to replace the simply counting of pairs;
3. more sophisticated dynamic programming algorithms may be employed to consider additional energetic contributions of two nested, neighboring pairs.

Contact me if you are interested in doing this project, even if just a part of it.
Figure 1: Schematic illustration of tRNA folding: (A) the secondary structure of a transfer RNA (tRNA), where short pink dash lines indicate hydrogen bondings, and (B) the tertiary (3D) structure of the tRNA, only the backbone folding is shown.