CS41 Homework 7

This homework is due at 11:59PM on Sunday, November 10. Write your solution using LaTeX. Submit this homework in files named `hw7.tex` and either `rnasubstructure.cpp` or `rnasubstructure.py` using **github**. This is a **10 point assignment**. This is a partnered homework. You should primarily be discussing this homework with your partner.

It's ok to discuss approaches at a high level. In fact, I encourage you to discuss general strategies. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab partner/group *while in lab.* In this case, note (in your post-homework survey) who you've worked with and what parts were solved during lab.

The main **learning goals** of this homework are to get more practice with dynamic program algorithm design and implementation.

1. **Implementing RNA Substructure Dynamic Program**

   For this problem, your task is to write a dynamic program that takes as input an RNA string and outputs the size of the largest matching, according to the rules of the RNA Substructure problem described in class.

   - You may write your program in C++ or in Python.
   - If you write in C++, we encourage you to use the limited starting code we've provided.
   - Your solution must be a dynamic program.
   - Your solution must be efficient—graders will test your program against large RNA substrings, and there will be nontrivial penalties for programs that take too long.
   - The file format consists of a single string of letters from $\{A, C, G, U\}$. Your program should read in the file, compute the largest RNA Subsequence matching, and output the result.

   The `/home/brody/public/cs41/rna-test-data` contains some test data you can use to test your implementation. Additionally, there are two executables in the `/home/brody/public/cs41` directory: `rna-A` and `rna-B`. Both are correct solutions, but one runs in exponential time and one runs in polynomial time. Feel free to run these programs and compare the output and time to your program.

2. **Longest Palindrome.** Let Σ be a finite set called an *alphabet.*[1] A *palindrome* is a string which reads the same backwards and forwards. Let s be a string of characters from Σ and let $x \in \Sigma$ be some character. The reversal of s is denoted s^R. Then the strings ss^R (that is, s concatenated with s^R) and sxs^R are both palindromes.

   In the **Longest Palindrome Problem**, you're given a string x of n characters from Σ and must output the length of the longest palindrome that is a substring of x.

   (a) *Briefly* describe a simple $\Theta(n^3)$ algorithm that solves the longest palindrome problem. Why is your algorithm $\Theta(n^3)$?

[1] For example, Σ might be $\{0, 1\}$ or $\{a, b, c, \ldots, z\}$.

(b) Design an algorithm that uses dynamic programming to solve the longest palindrome problem in less than n^3 time.

3. **Moving on a Checkerboard.** Suppose you are given an $n \times n$ checkerboard and a single checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule: at each step, you may move the checker to one of the following three squares:

   (a) the square immediately above,

   (b) the square that is one up and one to the left (but only if the checker is not already in the leftmost column)

   (c) the square that is one up and one to the right (but only if the checker is not already in the rightmost column)

   Each time you move from square x to square y, you receive $P(x, y)$ dollars. You are given the values $P(x, y)$ for all pairs (x, y) for which a move from x to y is legal. $P(x, y)$ may be negative.

   Give a polynomial-time algorithm that computes the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point, and any square along the top edge as an ending point, in order to maximize the number of dollars gathered along the way.

   What is the runtime of your algorithm?