Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7205: Fundamentals of Computer Engineering

Spring 2022 - Homework 6

Instructions

- For programming problems:
  - Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
  - You are **not** allowed to use any advanced C++ library unless it is clearly allowed by the problem. For example, you cannot use a library function to sort a list of data if the problem is asking you to implement an algorithm to sort the list.
  - At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
  - Your code must compile and tested with a standard GCC compiler (available in the CoE Linux server). before submitting the source code file(s) (do not submit any compiled/executable files):
    a. If your program does not compile with a GCC compiler due to incompatible text encoding format, then make sure the program is saved with Encoding Unicode (UTF-8). In visual studio, Save As -> Click on the arrow next to Save -> Save with Encoding -> Yes -> Unicode (UTF-8) -> Ok
    b. Compile using g++ -std=c++11 <filename>

- Submit the following to the homework assignment page on Canvas:
  - Your homework report developed by a word processor and submitted as <u>one</u> PDF file. For answers that require drawing and if it is difficult on you to use a drawing application, which is preferred, you can neatly hand draw the answer, scan it, and include it into your report. The report includes the following (depending on the assignment contents):
    a. Answers to the non-programming problems that show all the details of the steps you follow to reach these answers.
    b. A summary of your approach to solve the programming problems.
    c. The screen shots of the sample run(s) of your program(s).
  - Your well-commented programs source code files (i.e., the .cc or .cpp files).

  Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

<u>Note</u>: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded. This means all required files must be included in this last submission attempt.

Problem 1 (40 Points)

In a Fibonacci sequence, each number is the sum of the two preceding ones, starting from 0 and 1.

The beginning of the sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Assume here that 0 is the 0^{th} number in the sequence and 89 is the 11^{th}.

Write a C++ program to find the n^{th} number in a Fibonacci sequence using two different approaches. Each approach is implemented by a function that takes *n* as its only parameter. Name the functions `FibonacciR` and `FibonacciD`. `FibonacciR` solves the problem recursively where it calls itself recursively exactly twice. `FibonacciD` solves the problem recursively but with the support of dynamic programming. Also, `FibonacciD` calls itself recursively exactly twice.

For `FibonacciD`, you are allowed to define the needed array as a global variable in your program. You are not allowed to use any loop in your functions. The only loop you will need in your whole program is the one in the `main()` function, which is shown below. You need this loop to initialize all elements in the array to an invalid Fibonacci value (e.g., -1) to keep track of the array elements that do not have a Fibonacci number calculated yet.

Test your code with *n* values 5, 10, 20, 30, 40

Submit your C++ program and in your report answer the following questions:

a) Comment on the running time of each function.
b) For each function, find the big *O* asymptotic notation of its running time growth rate.

Your program should be tested with the following main function:

```
int main() {
        int n; //The user inputs this number to calculate its Fibonacci sequence
        cout << "n = ";
        cin >> n;

        // initialize all elements in array A to an invalid Fibonacci value
        for (int i = 0; i <= n; i++)
                        A[i] = -1;

        //Find Fibonacci sequence n using the Dynamic programming function
        cout << " FibonacciD(n) = " << FibonacciD(n);

        //Find Fibonacci sequence n using the recursive only function
        cout << "\n FibonacciR(n) = " << FibonacciR(n);

 return 0;
}
```

Problem 2 (35 Points)

Write a C++ program to compare the *Recursive* implementation of the *Rod Cutting* problem with the *Dynamic Programming* implementation of the same problem. The comparison should be in terms of each implementation running time for rod sizes: 5, 10, 15, …… , 40, 45, and 50 inches.

- Use the C++ clock function to measure the time consumed by each implementation in **micro-seconds**. (Reference: http://www.cplusplus.com/reference/ctime/clock/)
- Use the following formula to calculate the price of each cut of length *L* inches:

$$\text{Price} = \begin{cases} 2 & \text{if } L = 1 \\ \textbf{floor}(L*2.5) & \text{if } 1 < L < \text{rod size} \\ (L*2.5)-1 & \text{if } L = \text{rod size} \end{cases}$$

  where **floor** is a C++ function that returns the largest integral value that is not greater than the value of its parameter.
  (Reference: http://www.cplusplus.com/reference/cmath/floor/)

Based on the results from your program, fill the following table:
(*Note*: if a run takes more than 2 minutes to finish, just terminate it and report in the table "no solution").

Rod Size	Recursive Time	Recursive Max Revenue	Dynamic Time	Dynamic Max Revenue
5				
10				
15				
20				
25				
30				
35				
40				
45				
50				

Problem 3 (25 Points)

Find the Huffman encoding tree and hence the Huffman code for each of the letter shown in the following letter-frequency table:

Letter	Z	K	M	C	U	D	L	E
Frequency	2	7	24	32	37	42	42	120

Note: No programming code is required for this problem.

Extra Credit Problem (20 Points)

Extend the shortest path program you implemented in the previous homework to include the following:

a. Add a vertex to represent the main entrance of Snell Library (in case you have not included it in your graph already).
b. Add in your program the necessary functions to implement the APPROX-TSP-TOUR algorithm presented in the "NP-Completeness and Approximation Algorithms" lecture. The goal is to display to the user the Traveling-Salesperson tour beginning at Snell Library and passing through every other vertex in your graph **at least once** (as your graph is not a complete graph) and terminates at Snell Library. You will need to modify the APPROX-TSP-TOUR algorithm to work with your graph as the algorithm is intended to work with a complete undirected graph.
c. Explore different options of traversing the resulted MST and comment on the total tour cost given by these options.
d. Show in your report the resulted TSP tour on the campus map.
e. As you will be using an approximation algorithm, try manually to find a better solution and compare it to the TSP tour given by the algorithm.