CMPS 1500 Introduction to Computer Science I – Fall 13

11/18/13

11. Homework

Programming portion (problem 1) due **11/26/13** at 11:55pm on Blackboard.
Written portion (problems 2,3,4) due Wednesday **12/4/13** at the beginning of class.

In order to receive any credit for the programming portions, you are required to thoroughly comment and test your code.

1. **Doubly-Linked List (16 points)**
   Write Python code that implements a doubly-linked list. Your code should contain the following:

   (a) (1 points) A class `List_Node_Double` that stores `data`, a reference `next` to the next node in the list, and a reference `prev` to the previous node in the list. Make sure to add an appropriate constructor.

   (b) (4 points) A **function** that adds a new element to the front of the list, as well as a **method** that adds a new element to the list. Make sure to properly update `next` and `prev` references. The code for the function and the method will look similar; the point is to practice the difference between functions and methods. Test both the function and the method.

   (c) (6 points) A **function** and a **method** that each print the doubly-linked list. In addition, implement the **__str__** method that returns a string representation of the list. The code for the function and the two methods will look similar; the point is to practice the difference between functions and methods, and between printing and returning a string. Make sure to test the function and the two methods.

   (d) (2 points) As comments in the code, provide the runtimes for all your functions and methods.

   (e) (3 points) Give test code that creates and prints a doubly-linked list $_{\leftarrow}\,1\,\substack{\rightarrow\\ \leftarrow}\,2\,\substack{\rightarrow\\ \leftarrow}\,4\,\substack{\rightarrow\\ \leftarrow}\,8\,^{\rightarrow}$. Then on paper, trace your code, providing detailed pictures of all the variables (and the linked list!) in memory.

2. **Truth Table (6 points)**
   Answer this question on paper.

   (a) (2 points) Use a truth table to show that $\neg x \wedge (y \vee z)$ and $(\neg x \wedge y) \vee (\neg x \wedge z)$ are equivalent.

   (b) (4 points) Let $x_1, x_2, \ldots, x_n$ be n binary inputs. Give a logical formula that determines whether at least one of the inputs is on. How many gates does a corresponding circuit require, in terms of n? What is the minimum required circuit depth, in terms of n? Draw a circuit with minimum circuit depth for $n = 8$.

FLIP OVER TO BACK PAGE $\Longrightarrow$

3. **Binary Search (4 points)**
   Answer this question on paper.

   (a) What are the input and output for binary search? How is the input stored?

   (b) What is its best-case runtime? Describe an example input (for arbitrary n) that yields this runtime.

   (c) What is its worst-case runtime? Describe an example input (for arbitrary n) that yields this runtime.

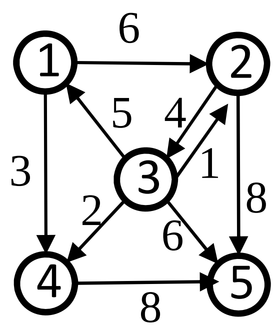
   (d) Now, assume that the input numbers are stored in a linked list. Can you give a formula that describes its worst-case runtime?

4. **Weighted Graphs (8 points)**
   Answer this question on paper.

   (a) (4 points) Show how the graph G below, which has directed edges with edge weights, can be stored (i) using adjacency lists and (ii) using an adjacency matrix. For (i) and (ii), give its abstract representation on paper, and in addition give valid Python code that produces such a representation. *(Hint: You could consider creating a Node class to use for the adjacency list representation.*

   (b) (2 points) For your adjacency lists representation from above, give Python code for a function that checks whether there is an edge from vertex i to vertex j, and if so, returns the weight of the edge. What is the runtime?

   (c) (2 points) For your adjacency matrix representation from above, give Python code for a function that checks whether there is an edge from vertex i to vertex j, and if so, returns the weight of the edge. What is the runtime?


6
1
2
5
4
3
1
3
2
8
6
4
5
8