Assignment 3

Design and Analysis of Algorithms (CS 4071) – Spring 2022

Due Friday, March 4 by 11:59PM ET

The leader of your group should submit three files: (1) a scan of the written part; (2) source code for your program; and (3) output from a sample run of your program. For ease of grading, all the C++ code for your program should be included in a ***single*** *file and designed using the C++ Visual Studio Platform. It should be well-commented and the output user-friendly.*

REMINDER. Test 1 will be held on Wednesday, February 23.

.

Written Part (40 pts)

1. [10pts] Given any set of n distinct keys (identifiers) and an *arbitrary* binary tree T on n nodes, use mathematical induction to show that there is a unique assignment of the keys to the nodes in T such that T becomes a binary search tree for the keys.
   Hint. First show that there is a unique assignment of a key to the root

2. [15pts] In many practical applications such as Prim's algorithm for computing a minimum spanning tree and Dijkstra's algorithm for computing shortest paths, both of which we will see later in this course, the priority queue ADT needs to be expanded to include the operation of changing the priority of an element. Design and give pseudocode as well as analyze the algorithm *ChangePriority*(Q,x,v), which changes the priority value of an element x in the priority queue Q to v when Q is implemented as a **min-heap**. *ChangePriority* should have worst-case complexity $O(\log n)$, where n is the number of elements in the min-heap.

   Hint. Consider two cases, the priority is decreased and the priority is increased and correct a min-heap property violation in a similar way to what we did for insertion and deletion into a min-heap.

3. [15pts] Suppose we have the following parent implementation of a forest representing a partition of a set of 17 elements:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Parent[i]	7	–3	8	7	14	0	1	8	–13	3	14	4	–1	3	3	0	1

   a. Sketch the trees in F.
   b. Show the state of *Parent*[0:16] after a call to *Union*(*Parent*[0:16],1,8) and sketch the trees in F.
   c. Given the state of *Parent*[0:16] in part (b), show the state of *Parent*[0:16] after an invocation of *Find2*(*Parent*[0:16],4) and sketch the trees in F.

Programming Project (60pts)

Connected Components of a Network

The topology of a network is modeled with a graph. Write a C++ program that inputs a graph *G* by first inputting the number of vertices *n* followed by a sequence of pairs `i j` where `i` and `j` are integers between 0 and $n - 1$, inclusive, representing the edges of the graph, and ending with a negative integer sentinel to indicate the end of the input. For example,

```
5  0  1  1  4  2  3  1  3  3  4  -1
```

represents the graph $G = (V,E)$ given by:

$V = \{0,1,2,3,4\}$

$E = \{\{0,1\},\{1,4\},\{2,3\},\{1,3\},\{3,4\}\}$.

Your program will compute the (**connected) components** of G.

You can proceed as follows:

- Implement the graph *G* with using pointer-based **adjacency lists** (using an array of header nodes and a linked list for each header node). Your class `Graph` should include constructors and a destructor, operations of edge addition, edge deletion, etc.,
- Implement the function `DFS(G,v)` for performing a **depth-first search**.
- Implement a function `Components(G)` for computing the vertex sets of the **connected components** of *G*. `Components()` will employ a DFT (Depth-First Traversal), which involves calling `DFS(G,v)`, $v = 0,1, .. n - 1$.

For purposes of grading, store the entire source code, including the graph class and its implementation, in a **single** .cpp file. Your program should run using Visual C++.

Have your program output **the adjacency matrix** of the graph as well as the connected components. Output the components by **outputting the vertex set** of each component. Run your program for a disconnected graph (i.e., having at least two components) of your choosing and submit this output in additional to your source code.

Your program should be **user-friendly**, **well-commented**, with the output **well-documented**.