CS 202, Spring 2019

Homework #4 – Balanced Search Trees, Hashing and Graphs

Due Date: May 18, 2019

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, May 18, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_hw4.zip`.
- Your ZIP archive should contain the following files:
  - `hw4.pdf`, the file containing the answers to Questions 1, 2 and 3,
  - `graph.h`, `graph.cpp` and `main.cpp` files which contain the C++ source codes
  - `Makefile`
  - `readme.txt` the file containing anything important on the compilation and the execution of your program in Question 4.
  - Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as in Listing 1 to the beginning of each file:

Listing 1: Header style

```
/**
 * Title: Balanced Search Trees, Hashing and Graphs
 * Author: Name Surname
 * ID: 21000000
 * Section: 0
 * Assignment: 4
 * Description: description of your code
 */
```

  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities.

- **You should prepare the answers of Questions 1, 2 and 3 using a word processor (in other words, do not submit images of handwritten answers).**
- Please use the algorithms as exactly shown in lectures.
- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Thus, you may lose significant amount of points if your C++ code does not compile or execute on the **dijkstra** server.
- Please make sure that you are aware of the homework grading policy that is explained in the rubric for homeworks.
- This homework will be graded by your TA, Hasan Balcı. Thus, please contact him directly for any homework related questions.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL) except those that are given with provided code.

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

Question 1 – 10 points

(a) [*5 points*]

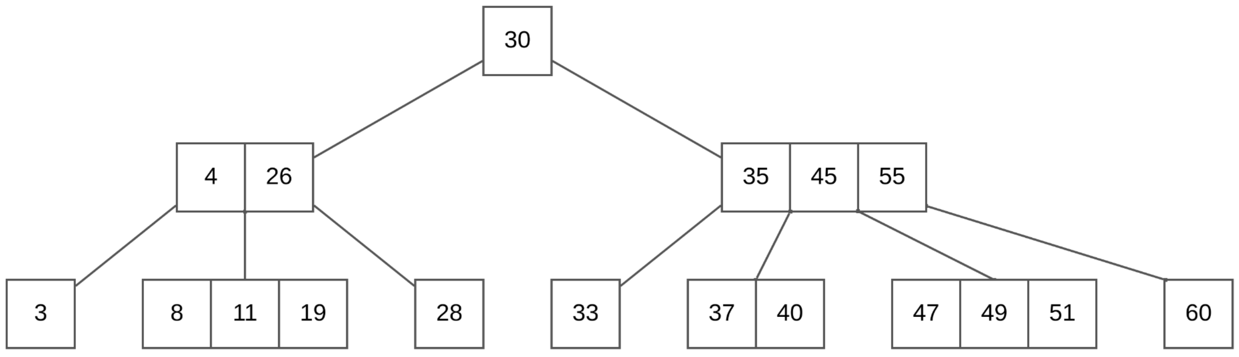


Figure 1: 2-3-4 tree

Draw an equivalent red-black tree of the 2-3-4 tree in Figure 1. Clearly indicate red and black nodes.

(b) [*5 points*] Draw only the final resulting 2-3-4 tree after inserting 48 into the original 2-3-4 tree in Figure 1.

Question 2 – 10 points

Fill in the below table appropriately by giving *expected* running times in big-O notation without providing any explanations for each operation. **insert** places a new item to the data structure and **extractMin** retrieves the item with minimum key and deletes it from the data structure. You may assume that hashing uses separate chaining.

Data Structure	**insert**	**extractMin**
unsorted array		
red-black tree		
hashing		
min-heap		
sorted linked list		

Question 3 – 10 points

(a) [*3 points*] What is the maximum number of keys that a 2-3 tree of height h can hold?

(b) [*2 points*] Is every subtree of a red-black tree also a red-black tree? Explain your answer with one sentence.

(c) [*5 points*] Assume that you have an array of length N that consists of unique integers. Describe an algorithm in plain English (no pseudo-code is needed) to find two integers in this array such that their sum is equal to a given target integer. Your algorithm must work in expected time *O(N)*. You can assume that exactly one pair of elements in the array sums to the given target integer.

Question 4 – 70 points

You are asked to develop a program that allows users to do some operations on the flight dataset given in the `flightDataset.txt` file. For this purpose, you will first construct a weighted, undirected graph from the dataset file and then develop some functionality for users to be able to analyze the dataset. Each line of the dataset file includes two airport names from the United Kingdom and the number of passengers transported between those airports in the year 2003. The format of each line is as follows:

```
airport1;airport2;numOfPassengers
```

You will use the given template in the `graph.h`, `graph.cpp` and `main.cpp` files to develop your program. In the graph structure you will implement, each vertex (node) will represent an airport and each edge will represent a reciprocal flight between airports. Each vertex will have an airport name and each edge will have the number of passengers transported between airports as value/weight. The adjacency list to represent this graph will actually be stored in a hash map as follows:

```
map< string, list<node> > adjList;
```

where `string` (key) keeps the airport name and `list<node>` (value) keeps the list of vertices adjacent to the corresponding airport together with the edge value.

Graph class in the template includes seven functions (excluding constructor), whose details are given below, waiting to be implemented in order to form the adjacency list and manipulate it with the required operations.

- **`void addAirport(const string& airportName);` (6 points)**

  `addAirport` function gets the name of an airport (`airportName`) as a parameter and then adds this airport to the map, where `airportName` is the key and an empty node list (`list<node>`) is the value.

  For example, when we add two new airports to an empty adjacency list:

  `addAirport("HEATHROW");`

  `addAirport("ABERDEEN");`

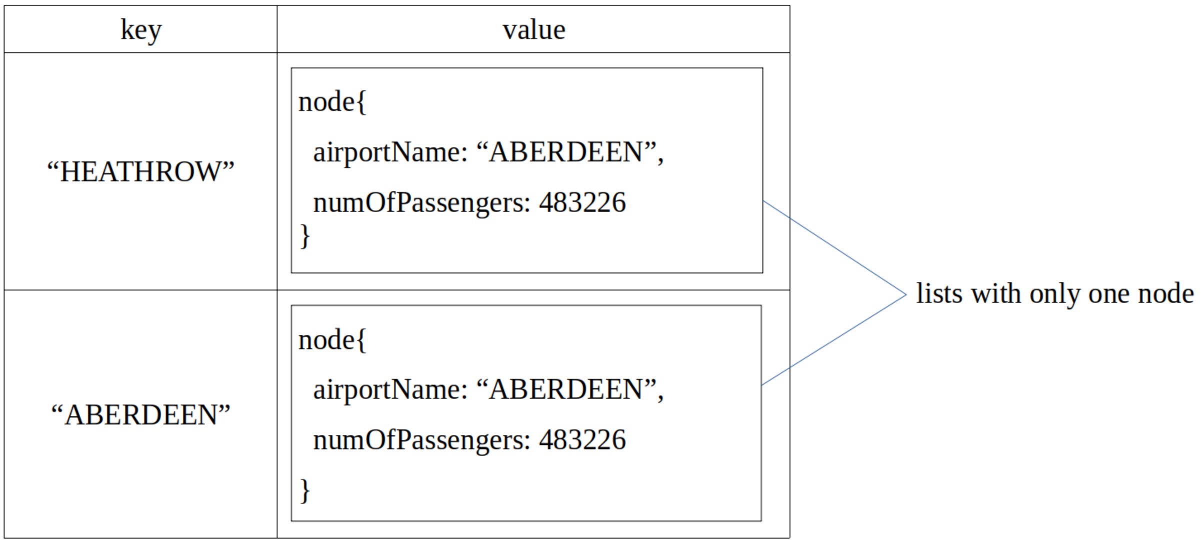
key	value
"HEATHROW"	empty list
"ABERDEEN"	empty list

- **`void addConnection(const string& airport1, const string& airport2, int numOfPassengers);` (6 points)**

  `addConnection` function adds a new node to the list of `airport1` using `airport2` and `numOfPassengers` data, and adds a new node to the list of `airport2` using `airport1` and `numOfPassengers` data.

  For example, when we add a new connection (edge) to the adjacency list above:

```
addConnection("HEATHROW", "ABERDEEN", 483226)
```



key	value
"HEATHROW"	node{ airportName: "ABERDEEN", numOfPassengers: 483226 }
"ABERDEEN"	node{ airportName: "ABERDEEN", numOfPassengers: 483226 }

- **`list<string> getAdjacentAirports(const string& airportName);` (6 points)**

  `getAdjacentAirports` function gets name of an airport as a parameter and returns all adjacent airports of the given airport as a list of string.

  For example, for the airport "NORWICH", it should return "ABERDEEN", "EDINBURGH", "HUMBERSIDE", "MANCHESTER".

- **`int getTotalPassengers(const string& airportName);` (6 points)**

  `getTotalPassengers` function gets name of an airport as a parameter and returns the total number of passengers transported from/to the given airport. In other words, it returns the sum of the values of the incident edges of the given vertex.

  For example, for the airport "KIRKWALL", it should return 80264.

- **`list<string> findShortestPath(const string& airport1, const string& airport2);` (15 points)**

  `findShortestPath` function gets two airport names as parameter, calculates the shortest path between the airports and returns the calculated path as a list of string. Assume that all distances among the airports are equal. If there are more than one shortest path between two airports, it is enough to return only one of them.

  For example, for the airports "KIRKWALL" and "JERSEY", the shortest paths are "KIRKWALL" →"EDINBURGH" →"JERSEY" and "KIRKWALL" →"GLASGOW" →"JERSEY". Your function should return one of these paths.

- **`list< pair<string, string> > findMST();` (25 points)**

  `findMST` function simply calculates the minimum spanning tree(MST) of the flight network by using the edge values which are the number of passengers. This function returns a list of string pairs where each string pair represents an edge included in the MST by its incident vertices.

  For example, some edges in the MST of the flight network are:

  "DUNDEE"-"HUMBERSIDE", "ABERDEEN"-"PLYMOUTH", "MANCHESTER"-"LIVERPOOL" etc.

- **`void deleteAirport(const string& airportName);` (6 points)**

  `deleteAirport` function deletes the given airport from the graph together with its incident connections (edges).

- **`main.cpp`**

  In `main.cpp`, construct an adjacency list by reading and processing the `flight-Dataset.txt` file using `addAirport` and `addConnection` functions. Then, test the other functions with different inputs to make sure that they work correctly. Although you will write your own main function to test your functions, we will use our own main function to test whether or not your functions work.

Some important notes:

- The places that require implementation in the `graph.cpp` and `main.cpp` files are indicated with the comment /*YOUR IMPLEMENTATION*/. Please do your implementation in those places.

- `map`, `list` and `queue` data structures from the C++ standard template library (STL) are already included in `graph.h`. Use these data structures in your implementation directly. `map` and `list` are used by the adjacency list and as return types of many functions. You may use `queue` (and also `priority_queue`) in the implementation of `findShortestPath` and `findMST` functions, if necessary.

- Use the classes that are currently included; however, **do not include** any class (other than those specified) from the C++ standard library to `graph.h` and `graph.cpp` files.

- Additionaly you may include necessary classes from the C++ standard library to the `main.cpp` file for the purpose of **reading and processing** the dataset file. Please indicate added classes in `readme.txt`.

- Please ask your TA, Hasan Balcı, if anything is unclear for you.

At the end, write a basic `Makefile` which compiles all your code and creates an executable file named `hw4`. Check out these tutorials for writing a simple make file: tutorial 1, tutorial 2. Please make sure that your `Makefile` works properly, otherwise you may lose points from Question 4.