CS 202, Spring 2021

Homework #4 – Balanced Search Trees, Hashing and Graphs

Due Date: May 10, 2021

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, May 10, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_secNo_hw4.zip`.
- Your ZIP archive should contain the following files:
  - `hw4.pdf`, the file containing the answers to Questions 1 and 2,
  - `WordNetwork.h` and `WordNetwork.cpp` (as well as any other files that contain any extra class that you wrote) as the C++ source codes for Question 3,
  - `readme.txt` the file containing anything important on the compilation of your program in Question 3.
  - 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 upload handwritten answers for Q1 and Q2 (in other words, do not submit answers prepared using a word processor).
- Use the exact algorithms 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.
- This homework will be graded by your TAs Kemal Büyükkaya and Mert Sayar. Thus, please contact them 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).

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

Question 1 – 15 points

Assume that we have the following balanced searched tree implementations:

1. 2-3 tree
2. 2-3-4 tree
3. Red-black tree

Starting with an empty balanced search tree, we would like to insert the following keys into the tree in the given order:

```
15 42 64 48 55 43 79 59 61
```

and then, delete the keys `48 79 59` (in the given order) from the tree. Note: while deleting an internal node, its inorder successor should be used as the substitute if needed.

Show the underlying tree for the 2-3 and 2-3-4 implementations after *each* insertion and deletion. Also show the underlying tree for the red-black implementation after *each* insertion (top-down). No deletion is required for the red-black tree.

Question 2 – 15 points

Assume that we have a hash table with size 17 (index range is $0 \ldots 16$), and we use the `mod` operator as a hash function to map a given numeric key into this hash table. Draw the hash tables after the insertion of the keys

```
22 23 24 39 40 26 41 43 26
```

in the given order for each of the following methods:

1. open addressing with linear probing,
2. open addressing with quadratic probing,
3. separate chaining.

Question 3 – 70 points

The goal of this question is to perform some graph-related operations on Prof. Donald Knuth's list of five-letter words in GraphBase (Donald E. Knuth, "The Stanford GraphBase: A Platform for Combinatorial Computing," ACM Press, New York, 1993). In this data set, each word is a vertex and two vertices are connected by an undirected edge if the corresponding words differ in only one letter. For example, *roger* and *rover* are connected but *roger* and *pager* are not.

You are asked to develop a program that allows users to do some operations on this word network. These operations should be implemented in a class named `WordNetwork` with all related code in the files named `WordNetwork.h` and `WordNetwork.cpp` (as well as any other files that contain any extra class that you wrote). The class definition and the descriptions of the required public functions are given below.

```
class WordNetwork {

  public:
    WordNetwork(const string vertexFile, const string edgeFile);
    ~WordNetwork();

    void listNeighbors(const string word);
    void listNeighbors(const string word, const int distance);
    void listConnectedComponents();
    void findShortestPath(const string word1, const string word2);

    ...
  private:
    // define your data members here
    // define private member functions here, if any
    // you MUST use the adjacency matrix representation
};
```

✓

`WordNetwork( const string vertexFile, const string edgeFile );` The constructor loads the word network from the given text files containing the vertices and the edges. You are given a text file named `vertexFile` that contains each word in a separate line. There are 5757 words in this file. You are also given a separate file named `edgeFile` that contains the edge information. Each line includes an edge with the words separated by a comma. There are 14135 edges in this data set.

The constructor must load the network data from these files and construct an **adjacency matrix** representation for the graph. You MUST use the adjacency matrix representation in this assignment. (Otherwise, you will get no points from this question).

You MUST also design a **hash table** to implement the mapping from the words to unique integer indices. These integer indices (in the range from 0 to 5756) are used to refer to the vertices (i) and edges (i,j) internally in the code whereas the input and output use the actual words as in the examples below. The hash table takes a word (string) as input and gives its index (integer) as the output. The goal of the hash table is to perform this lookup efficiently. You MUST use the hash function (Hash Function 3) given on slide 40 of the lecture notes on hashing. You are free to choose the table size as you wish. The hash table MUST use **separate chaining** to store the data. The integer indices will correspond to the line numbers for the words given in the vertex file. For example, roger will be indexed as 0, cages will be indexed as 1, brung will be indexed as 2, and so on. You must use this hash table with the words given as the keys and the line numbers as the item data that will be used as the vertex indices.

✓

`void listNeighbors( const string word );` This function lists all words that are immediate neighbors of the input word. Each neighbor should be displayed on a separate line as given in the following example:

```
Neighbors of roger:
roper
rover
rower
```

Display a warning message if the word does not exist in the graph.

`void listNeighbors( const string word, const int distance );` This function lists all words that can be reached from the input word with a distance given as the second input. The distance defines the number of letters that need to be changed to reach a word starting from the input word. For example, *order* is reachable from *aided* with a distance of 4 (aided → added → adder → odder → order). Display the resulting words on a separate line as in the previous function.

`void listConnectedComponents();` This function lists all connected components in the given graph. The connected components should be identified clearly in the list. For example:

```
Connected component 1:
piece
niece
Connected component 2:
ideal
ideas
Connected component 3:
setup
getup
letup
Connected component 4:
rumor
humor
tutor
tumor
Connected component 5:
...
```

`void findShortestPath( const string word1, const string word2 );` This function displays the words along the shortest path from the first word to the second word. For example:

```
Shortest path from nodes to graph:
nodes
lodes
lores
```

```
lords
loads
goads
grads
grade
grape
graph
```