CS 202, Fall 2018

Homework #4 – Hash Tables, Graphs

Due date: December 18, 2018

<u>Important Notes</u>

Please do not start the assignment before reading these notes.

- Before 23:55, December 18, upload your solutions in a single ZIP archive using the 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 and 2,
  - ✓ **`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 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 the following to the beginning of each file:

```
/*
 * Title : Hash Tables, 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 (for example to measure the time).

- **You should prepare the answers of Questions 1 and 2 using a word processor (in other words, do not submit images of handwritten answers).**

- Use the exact algorithms as shown in lecture slides

- 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. <u>Thus, you will lose significant amount of points if your C++ code does not compile or execute on the **dijkstra** server.</u>

- 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 – 18 points

Insert the keys 23, 15, 30, 16, 11 and 18 into a hash table with 7 slots in the given order by using the following hash function h1():

```
int h1 (int key) {
        int x = (key + 5) * (key + 5);
        x = x / 20;
        x = x + key;
        x = x (mod 7);
        return x;
}
```

Resolve the collisions with

a) (6 points) Linear Probing

b) (6 points) Quadratic Probing

c) (6 points) Double Hashing where the secondary hash function h2() is

(Reverse (key)) (mod 7).

Reverse (key) reverses the digits of the key and returns that value,

e.g. Reverse (123) = 321.

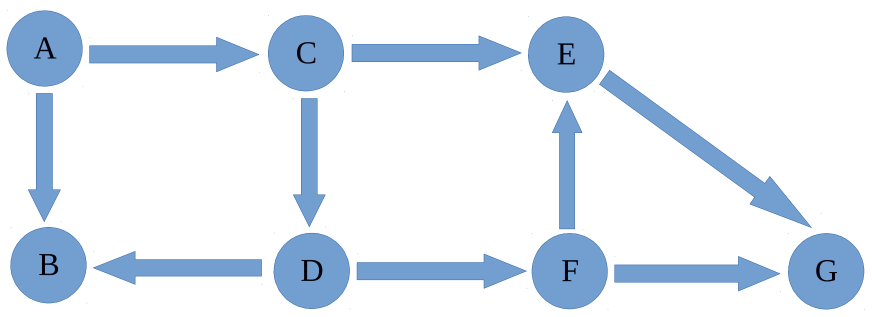
When necessary, use the conventions in the related lecture slides. Show the final content of the hash table in the following form:

Slot	0	1	2	3	4	5	6
Content							

Question 2 – 12 points

(a) [6 points] Change and rewrite the topSort1 algorithm in the lecture slides so that it removes vertices with no predecessors instead of successors and finds the topological sort.

(b) [6 points] By applying the new topological sorting algorithm from **part a)**, find 2 possible topological ordering of the vertices in the following graph:


$$\begin{tikzcd}
A \arrow[r] \arrow[d] & C \arrow[r] \arrow[d] & E \arrow[dr] & \\
B & D \arrow[l] \arrow[r] & F \arrow[u] \arrow[r] & G
\end{tikzcd}$$

Question 3 – 70 points

You are asked to develop a program that allows users to do some operations on the movie dataset given in the *movieDataset.txt* file. For this purpose, you will first construct a 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 a movie name with two performers who played in that movie and the director of the movie. The format of each line is as follows:

movieTitle;performer1;performer2;director

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 a performer and each edge will represent that two performers played in the same movie. Each vertex will have a performer name and each edge will have a movie title and the director of the movie as value. The adjacency list to represent this graph will actually be using a hash map as follows:

map<string, list<node>> adjList;

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

Graph class in the template includes eight 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 addPerformer(string performerName); (5 points)**

addPerformer function gets the name of a performer (performerName) as a parameter and then adds this performer to the map where performerName is the key and an empty node list (list<node>) is the value.
For example, when we add two new performers to an empty adjacency list:
addPerformer("Banderas, Antonio");
addPerformer("Abril, Victoria");

key	value
"Abril, Victoria"	empty list
"Banderas, Antonio"	empty list

- **void addEdge(string movieTitle, string performer1, string performer2, string director); (7.5 points)**

addEdge function adds a new node to the list of performer1 using performer2, movieTitle and director data, and adds a new node to the list of performer2 using performer1, movieTitle and director data.
For example, when we add a new edge to the adjacency list above:
addEdge("Tie Me Up! Tie Me Down!", "Banderas, Antonio", "Abril, Victoria", "Almodóvar, Pedro")

key	value

"Abril, Victoria"	node{
	name: *"Banderas, Antonio"*,
	movie: *"Tie Me Up! Tie Me Down!"*,
	director: *"Almodóvar, Pedro"}*

"Banderas, Antonio"	node{
	name: *"Abril, Victoria"*,
	movie: *"Tie Me Up! Tie Me Down!"*,
	director: *"Almodóvar, Pedro"*}

lists with only one node

- **list<string> getMovies(string performerName); (7.5 points)**

getMovies function gets name of a performer as a parameter and returns all movies in which the performer performed as a list of string.

- **list<string> deletePerformers(); (7.5 points)**

deletePerformers function deletes all performers who played in a movie directed by also themselves from the graph together with their incident edges and returns those performers as a list of string.

- **map<string, int> getMostActivePerformers(); (7.5 points)**

getMostActivePerformers function finds the performer(s) who played in *highest* number of movies and returns a map of performers where performer name is the key and the number of movies is the value.

- **map<string, int> getPerformerNumber(string performerName); (10 points)**

“performer number” of a performer is the number of degrees of separation s/he has from a given performer.

For example, suppose the performer given as parameter is “Connors, Chuck”. “Connors, Chuck”’s performer number is 0. “Adams, Maud” who played with “Connors, Chuck” in “Target Eagle” has performer number of 1. “Moore, Roger” who played with “Adams, Maud” in “Octopussy” has performer number of 2. “Bach, Barbara”, “Bouquet, Carole”, “Chiles, Lois”, “Ekland, Britt”, “Neil, Hildegard”, “York, Susannah” who played with “Moore, Roger” in various movies have the performer number of 3.

getPerformerNumber function finds all performers who have “performer number” up to 3 according to the performer given as parameter. It will return a map of all performers found where the performer name is the key and the corresponding performer number is the value.

- **map<string, int> getFrequentPartner(string performerName); (7.5 points)**

getFrequentPartner function gets name of a performer (performerName) as a parameter, finds performerName’s most frequent partner(s) in the movies s/he performed and returns a map of these partners where name of the partner is the key and the number of movies they performed together is the value.

- **map<string, int> getFrequentDirector(string performerName); (7.5 points)**

getFrequentDirector function gets name of a performer (performerName) as a parameter, finds performerName's most frequent director(s) in the movies s/he performed and returns a map of these directors where name of the director is the key and the number of movies they were together is the value.

- **main.cpp (10 points)**

In main.cpp, you are first asked to construct an adjacency list by reading and processing the movieDataset.txt file using *addPerformer* and *addEdge* functions. Then we want you to implement a menu which uses other functions in order for the user to do some operations with the graph. A sample menu can be as follows:

```
Welcome to Movie Dataset
Select an option:
1- See movies of a performer
2- Delete performers who directed the movie in which s/he played
3- Find performer(s) who played in highest number of movies
4- Find performer number of performers based on a given performer
5- Find performer(s) who played in highest number of movies with a given performer
6- Find director(s) who directed highest number of movies of a given performer
7- Exit
```

When the user enters a number, the program should ask the user the name of a performer if required and show the result. Then, it should show the menu again. If the user selects to exit, then it may show a farewell message.

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 to those places.
- ***map**, **list** and **queue*** data structures from the C++ standard template library (STL) are already included in graph.h. You can 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* in the implementation of *getPerformerNumber* function, if necessary.

- You can use the classes that are currently included; however, **do not include** any other class from the C++ standard library to graph.h and graph.cpp files.
- You can include necessary classes from the C++ standard library to the main.cpp file **only** for the purpose of **reading and processing** the dataset file. Please indicate added classes in readme.txt.
- You may need to use **C++11** standard to compile your files in Dijkstra server
  - g++ -std=c++11 your_file.cpp -o your_program
- Please ask your TA, **Hasan Balcı**, if anything is unclear for you.

IMPORTANT: 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:
http://mrbook.org/blog/tutorials/make/,
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/.
Please make sure that your **Makefile** works properly on the Dijkstra server, otherwise you will not get any points from Question 3.