CS 202, Spring 2018
Homework #4 – Hashing and Graphs
Due Date: May 13, 2018

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, May 13, 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 and 3,
  - `FlightNetwork.h`, `FlightNetwork.cpp`, `main.cpp` files which contain the C++ source codes, and the `Makefile`.
  - 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: 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 (for example to measure the time).
  - You should prepare the answers of Questions 1 and 3 using a word processor (in other words, do not submit images of handwritten answers).
- 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 a significant amount of points if your C++ code does not compile or execute on the dijkstra server.

- This homework will be graded by your TA, Ilkin Safarli. 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).

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

Question 1 – 15 points

(a) [*6 points*] Insert $\{12, 15, 20, 30, 41, 29, 17, 25, 22\}$ into an empty hash table which:

- uses linear probing for collision resolution.
- uses quadratic probing for collision resolution.
- uses separate chaining.

The size of each hash table is 13 and the hash function is: $h(x) = x \ mod \ 13$

(b) [*9 points*] Find the average number of probes for a successful search and an unsuccessful search for the hash tables which you created in part a. Use the following numbers for unsuccessful searches: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 38\}$

Table 1: Average number of probes

	Successful Search		Unsuccessful Search	
	Calculated	Theoretical	Calculated	Theoretical
Linear Probing				
Quadratic Probing				
Separate Chaining				

Question 2 – 70 points

In this part you will design and implement a flight database network system. You will use the dataset, `routes.csv`, provided with the homework.[1] The dataset has the following format:

Dataset format

```
Airline, Source airport, Destination airport

Airline:  2 or 3-letter code of the airline.
Source airport:  3 or 4-letter code of the source airport.
Destination airport:  3 or 4-letter code of the destination airport.

In total, there are 67652 routes between 3425 airports in the dateset.
```

```
Sample entry:
TK,ESB,IST

The example above means that there is a direct flight from ESB (Ankara Esenboga Airport) to IST
(Istanbul Ataturk Airport) operated by TK (Turkish Airlines).
```

```
Note:  routes are directional:  if an airline operates services from A to B and from B to A, both
A-B and B-A are listed separately.
```

First create a class named `FlightNetwork`, put all related code into `FlightNetwork.h` and `FlightNetwork.cpp` files. In the constructor of the `FlightNetwork` class, you will read the provided dataset and store it in appropriate data structure (airports in the dataset will be vertices and flights will be directed edges in the graph). You should decide whether *adjacency matrix* or *adjacency list* fits best for the purpose of this assignment. You need to implement the following functionalities in `FlightNetwork` class.

(a) [*10 points*] Checks if there are any direct flights from airport A to airport B.

`input:  source airport, destination airport`

`output:  boolean`

(b) [*10 points*] Finds and prints all airports which are directly accessible from airport A.

`input:  source airport`

(c) [*10 points*] Checks if there are any flight paths from airport A to airport B.

`input:  source airport, destination airport`

`output:  boolean`

(d) [*15 points*] Finds and prints the flight path from airport A to airport B with the minimum number of stops. If there are multiple such paths, printing one of them is sufficient.

`input:  source airport, destination airport`

[1] Retrieved from https://openflights.org/data.html#route and simplified.

(e) [*15 points*] Finds and prints all of the flight paths with n or lower number of stops from airport A to airport B. Go to the hint!

```
input:  source airport, destination airport, n
```

(f) [*10 points*] In this part of the assignment, you will write a program which takes arguments from the command line and runs appropriate functions. The first argument defines which function will be run. If the first argument is `a`, then you will run `part a`. If it is `b`, then you will run `part b` and etc. The remaining arguments will be passed to the corresponding function. For example: if your program is called as

```
./hw4 d IST ESB
```

then in your main function, you should call the function in `part d` and provide `IST` and `ESB` parameters to it.

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 on the **dijkstra** server.

Question 3 – 15 points

After implementing `FlightNetwork` class, you are expected to prepare a single page report.[2] In your report, state which data structure (adjacency matrix or adjacency list) you used to store the graph and explain the reasoning behind your decision. And answer the following questions:

- What would the time complexity (in terms of D - degree of source airport) of part a be if:
  - adjacency matrix is used?
  - adjacency list is used?
- What is the worst case time complexity of part c in terms of $|E|$ and $|V|$?
- What is the worst case time complexity of part d in terms of $|E|$ and $|V|$?

For time complexity calculations, show all your work clearly. Answers without explanations will not get any credits.

[2]Please make sure that your report does not exceed the specified page limit, otherwise you may lose points

Hint for Question 2 Part (e)

You can solve this problem by modifying the BFS algorithm. Instead of vertices, keep the paths in the queue.

- Initial path is source airport, add this to the queue.
- Then until queue is empty repeat the following:
  - dequeue an element from the queue and assign it to `currentPath`
  - set v to be the last element of `currentPath`
  - if v == *destination airport*, then print `currentPath`
  - visit each vertex u adjacent to v:
    - if u is not in the `currentPath`, create a new path by combining `currentPath` and u, and add it to the queue