GA2: Hashing & Binary Search Trees

Due date: 03/30/2021 11:59PM

Introduction

You recently completed an online coding assessment in hopes of landing an internship at a tech startup and are waiting for the result announcement. Two days later, you receive a cryptic email from the company containing information about your assessment result and, possibly, the next steps of your interview. In other words, if you pass the coding assessment, you will proceed to the final round of interview with an interviewer possessing a particular employee ID. You are required to find and meet with that interviewer on the site in order to complete the interview process.

To get that dream internship, your task is to write a program to extract essential interview information from that email. Fortunately, you are informed by the recruiter that you need to use ***Cuckoo Hashing*** to find out about your assessment result and ***Binary Search Tree*** to look for the said interviewer. The company thinks this is a great way to help you prepare for the final interview.

Rules and Operations

Part A: Cuckoo Hashing:

- The first line of the input contains two positive integers **m** and **n** (1≤**m**≤**n**≤100), with **m** telling the number of random values to be inserted in the hash table and **n** the size of the hash table. The next **m** lines are entries of indexes to insert the random elements into the hash table.
- Each entry consists of two alternative indexes to insert the element. Using the Cuckoo Hashing scheme, you will know that you passed the assessment if all elements are successfully inserted, and failed if there is a problem with the insertion; that is, when the collision results in a cycle.

Part B: Binary Search Tree:

- You may proceed to this step only if you passed the assessment. The next non-empty line in the input contains an integer **k** (1<**k**<100) that represents the number of people working at your interview site. The following **k** lines contain the IDs of those people. The work space is arranged in a binary search tree structure, and so their IDs are used to determine their desk location in the office.

- The next and last line in the input contains two IDs separated by space. The first ID belongs to the interviewer you are looking for and the second ID belongs to the employee who may or may not help you fast track your interviewer search. You must start your search from the latter. After meeting with your interviewer, you will be asked how many people you had walked past at the office before you could find the interviewer (with the interviewer included in the count) and list their IDs.

Note: STLs are not allowed!

Examples

Example 1:

Input21.txt

```
5 6                          // m n
2 3                         // Entry 1
3 1                         // Entry 2
1 2                         // Entry 3
5 1                         // Entry 4
2 5                         // Entry 5
2               // k    (Ignored because the result is 'Failed')
10768         // employee 1 (Ignored because the result is 'Failed')
10769        // employee 2 (Ignored because the result is 'Failed')
10768 10768  // interviewer first_seen_employee (Ignored because the result is 'Failed')
```

Output21.txt

```
Failed
```

Command line:

```
./graph input=input21.txt output=output21.txt
```

<u>**Example 2:**</u>

Input22.txt

```
3 3               // m n
0 1               // entry 1
1 2               // entry 2
2 0               // entry 3
                  // empty line
                  // empty line
2                // k
10768          // employee 1
10769         // employee 2
10768 10768  // interviewer first_seen_employee
```

Output22.txt

```
Passed
Looking for: 10768
Starting from: 10768
1
10768
```

Command line:

```
./graph input=input22.txt output=output22.txt
```

Example 3:

Input23.txt

```
5 12              // m n
2 0              // entry 1
3 5              // entry 2
1 1              // entry
5 3              // entry
0 2              // entry
                 // empty line
                 // empty line
9               // k
10405        // employee 1
10235        // employee 2
10709        // employee 3
10701        // employee 4
10718       // employee 5
10640      // employee 6
10714     // employee 7
10768    // employee 8
10770    // employee 9
10768 10709  // interviewer first_seen_employee
```

Output23.txt

```
Passed
Looking for: 10768
Starting from: 10709
3
10709 -> 10718 -> 10768
```

Command line:

```
./graph input=input23.txt output=output23.txt
```

Submission instructions for GA2:

We expect every student of each group will participate to solve the problem and discuss with each other. ***We will count the lowest grade of a group for every group member***, so make sure everyone in your team is submitting the same copy.

Every student of each group needs to submit ***the same copy*** of the solution. GA2 needs to be turned in to our Linux server. Make sure to create a folder under your root directory, name it ***"ga2"*** (name must be in lower case and exact), only copy your .cpp and .h files to this folder, no test case or other files needed. If you use ArgumentManager.h, don't forget to turn it in, too. ***GAs will be graded only once.*** You will not get the chance of resubmission after grading. So, make sure you are submitting the correct solution before the due date.