Programming Assignment #5

Lecturer: Prof. Jaesik Park
Teaching Assistants: ByeongHoon So, Hyunmin Lee, Junha Lee

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59 PM June 10, 2020

Evaluation policy:

- Late submission penalty.
  - 11:59 PM June 10 ~ 11:59 PM June 11.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM June 11.
    - 100% penalty is applied for that submission.
- Your code will be automatically tested using an evaluation program.
  - Each problem has the maximum score.
  - A score will be assigned based on the behavior of the program.
- We won't accept any submission via email - it will be ignored.

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Coding:

- Please do not use the containers in C++ standard template library (STL).
  - Such as <hash_set>, <queue>, <vector>, and <stack>.
  - Any submission using the above headers will be disregarded.
  - Due to the many requests, <cstring> and <string> are fine to use.

Submission:

- Before submitting your work, compile and test your code using C++11 compiler in repl.it.
  - Please refer to the attached file named "PA_instructions_updated.pdf".
  - There might be a penalty if the submission would not work in the "repl.it + C++11" environment.
- What you need to submit.
  - A zip file named "pa5.zip" that contains
    - pa5.cpp
    - hash_function.cpp and hash_function.h
    - hash_table.cpp and hash_table.h
    - bloom_filter.cpp and bloom_filter.h
    - graph.cpp and graph.h

Any questions?

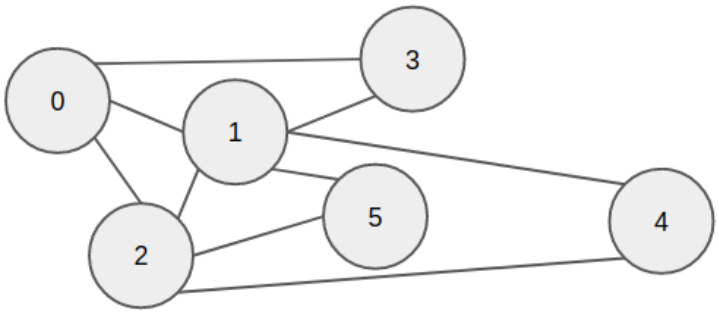
- Please use LMS - Q&A board.

1. Quiz (4 pts)

(1) What is the appropriate data structure for Depth First Search algorithm?

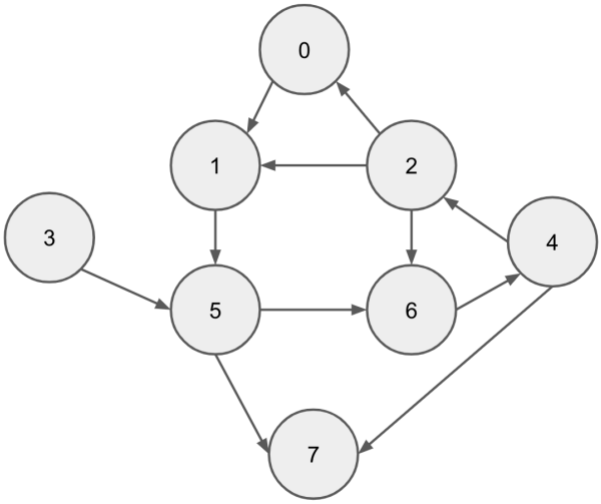
1. Stack
2. Queue
3. Priority queue
4. None of above

(2) Choose a sequence that can be obtained from Depth First Search traversal given the graph below. The search starts at vertex '0' and follows the ascending order.



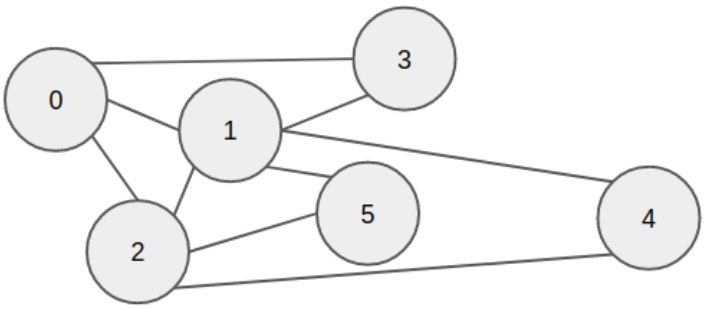
1. 0 1 2 4 3 5
2. 0 1 2 5 4 3
3. 0 1 2 4 5 3
4. 0 1 2 3 4 5

(3) Which vertices are strongly connected to the vertex '6' in the given graph?



1. 0, 1, 3, 4, 7
2. 0, 1, 2, 3, 4, 5
3. 0, 1, 2, 4, 5
4. 0, 1, 5, 7

(4) Choose a sequence that corresponds to the Breadth First Search traversal for a graph below. The search starts at vertex 0 and follows lexicographic ordering.



1. 0 1 2 4 3 5
2. 0 1 2 5 4 3
3. 0 1 2 4 5 3
4. 0 1 2 3 4 5

Print out a sequence of answers of each sub-quiz with the string separated with the spacebar. Each sub-quiz will have 1 correct answer. If you think the answers of the quiz 1-(1), 1-(2), 1-(3) and 1-(4) are 1, 1, 1, 1 respectively, then print 1 1 1 1. You can modify `task_1` function in `pa5.cpp`.

- Example execution

```
>> ./pa5.exe 1
[Task 1]
1 1 1 1
```

2. Connected Component of Graph (3 pts)

a. Implement a function that retrieves **the largest** connected components from the given undirected graph. This function finds **the largest** connected component that can be found in the given graph and prints out the node labels of it. There might be multiple connected components with the same size. **If that is the case, retrieve the component that comes first in lexicographical order.** For instance, if two connected components, `(A, B, C)` and `(A, B, D)` are found, select `(A, B, C)` because C comes first than D in lexicographical order. You can modify `graph.cpp` and `graph.h` files for this problem.

b. Input & output
Input: Pairs of node labels that indicate edges.
Output: A sequence of the node labels of the largest connected component. It should be sorted in **lexicographical** order and separated with space.

c. Example Input & Output

Input	Output
`[('A','B'), ('B','C'), ('C','A')]`	`A B C`
`[('A','B'), ('C','D'), ('C','E')]`	`C D E`
`[('A','B'), ('A','C'), ('C','D'), ('C','E'), ('D','B'), ('D','E')]`	`A B C D E`

d. Example execution

```
>> ./pa5.exe 2 "[('A','B'), ('C','D'), ('C','E')]"
[Task 2]
C D E
```

3. Cyclic / Acyclic Graph (4 pts)

a. Implement a function that returns the number of cycles in the given **directed graph**. A cycle is a non-empty path in which the only repeated vertices are the first and last vertices. Unlike task 2, the edges of the graph have direction in this time. For instance, `('A','B')` is different from `('B','A')`. This function should return `1` if the given directed graph has at least one cycle, or `0` otherwise. You can modify `graph.cpp` and `graph.h` files for this problem.

b. Input & output

Input: Pairs of node labels that indicate edges.

- (A, B): an edge from node A to node B.
- If the input edge already exists in the graph, ignore the input.

Output:

- The number of cycles in the given directed graph.

c. Example Input & Output

Input	Output
`[('A','B'), ('B','C'), ('C','A')]`	`1`
`[('A','B'), ('B','C'), ('A','C')]`	`0`
`[('A','B'), ('A','C'), ('C','D'), ('C','E'), ('D','B'), ('D','E')]`	`0`
`[('A','B'), ('A','C'), ('C','D'), ('E','C'), ('D','B'), ('D','E')]`	`1`
`[('A','B'), ('A','C'), ('C','D'), ('E','C'), ('D','B'), ('D','E'), ('C','F'), ('F','A')]`	`2`

d. Example execution

```
>> ./pa5.exe 3 "[('A','B'), ('A','C'), ('C','D'), ('C','E'),
('D','B'), ('D','E')]"
[Task 3]
0
```

4. Open hashing (4 pts)

a. Implement **a hash table** with **open hashing** to handle collision (a.k.a separate chaining). This hash table uses a key obtained from the string folding method (please check page 6 in Week10_02_Dictionary_Hashing.pdf). The size of the hash table is `M`. You can modify `hash_function.cpp`, `hash_function.h`, `hash_table.cpp`, and `hash_table.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- `('M', integer)`: the size of the hash table.
  (The first command is always `'M'`)
- `('insert', 'string')`: insert `string` into the hash table.
- `('search', 'string')`: search `string` in the hash table.

Output:

- Print out "hit" or "miss" for each search command
- Print out the list of the values for each slot of the hash table

c. Example Input & Output

Input	Output
[('M', 4), ('insert', 'apple'), ('insert', 'book'), ('insert', 'car'), ('search', 'apple')]	hit 0: [] 1: [] 2: ['apple' 'car'] 3: ['book']
[('M', 4), ('insert', 'book'), ('insert', 'car'), ('insert', 'car'), ('search', 'apple'), ('search', 'car'), ('search', 'dog')]	miss hit miss 0: [] 1: [] 2: ['car' 'car'] 3: ['book']
[('M', 4), ('insert', 'car'), ('search', 'dog'), ('search', 'car'), ('insert', 'dog'), ('insert', 'car'), ('search', 'dog'), ('search', 'car')]	miss hit hit hit 0: [] 1: [] 2: ['car' 'dog' 'car'] 3: []

d. Example execution

```
>> ./pa5.exe 4 "[('M', 4), ('insert', 'apple'),
('insert', 'book'), ('insert', 'car'), ('search', 'apple')]"
[Task 4]
hit
0: []
1: []
2: ['apple' 'car']
3: ['book']
```

5. Bloom filter (5 pts)

a. Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element (key) is a member of a set. The filter tells us that an element is either **possibly in the set** (hit) or **definitely not in the set** (miss).

b. Add **Bloom filter** into the hash table in task 4
Before you search for a key in the hash table, you have to first check if the key is in the hash table using the Bloom filter. If the filter returns hit, search the key in the hash table. If the filter returns miss, you can skip the searching in the hash table. You can modify `hash_table.cpp`, `hash_table.h`, `bloom_filter.cpp` and `bloom_filter.h` files for this problem.

c. Implementation
Bloom filter uses an b-bit vector B and k hash functions $h_1(\cdot), \ldots, h_k(\cdot)$. We fix k to 2 and provide the two hash functions:
   - A division hash function $h_1(\cdot)$
   - A mid-square hash function $h_2(\cdot)$

   Given a value as a string, the Bloom filter calculates the key of a string using the string folding method. Unlike task 4, the filter gets the key before dividing it by the size of the hash table, M.

d. Algorithm of Bloom filter
   - **Initialize**: All bits in B are set to 0
   - **Insert** a key x:
     1. Get the indices of the k hash functions, $h_1(x), \ldots, h_k(x)$.
     2. Set the bits of the indices to 1.
   - **Search** a key x:
     1. Get the indices of the k hash functions, $h_1(x), \ldots, h_k(x)$.
     2. If all the bits of the indices are 1, x is possibly in the set (hit).
        If any bit of the indices is 0, x is not in the set (miss).
   - **Delete**: Bloom filter does not support the deletion.

e. Input & Output

Input: A sequence of commands

- `('M', integer)`: the size of the hash table.
  (The first command is always `'M'`)
- `('b', integer)`: the size of a bloom filter.
  (The second command is always `'b'`)
- `('insert', 'string')`: insert `string` into the hash table.
- `('search', 'string')`: search `string` in the hash table.

Output:

- For each search command, print out "hit" or "miss" of the Bloom filter. If the filter is "miss", print out "hit" or "miss" of the hash table.
- Print out the bits of the Bloom filter.
- Print out the list of values for each slot of the hash table.

f. Example Input & Output

Input	Output

`[('M', 4), ('b', 16),`
`('insert', 'apple'), ('insert', 'book'),`
`('insert', 'car'), ('search', 'apple')]`
	`Bloom: hit, HashTable: hit`
	`0110011000010000`
	`0: []`
	`1: []`
	`2: ['apple' 'car']`
	`3: ['book']`

`[('M', 4), ('b', 16), ('insert', 'book'),`
`('insert', 'car'), ('insert', 'car'),`
`('search', 'apple'), ('search', 'car'),`
`('search', 'dog')]`
	`Bloom: miss`
	`Bloom: hit, HashTable: hit`
	`Bloom: miss`
	`0000011000010000`
	`0: []`
	`1: []`
	`2: ['car' 'car']`
	`3: ['book']`

`[('M', 4), ('b', 16), ('insert', 'car'),`
`('search', 'dog'), ('search', 'car'),`
`('insert', 'dog'), ('insert', 'car'),`
`('search', 'dog'), ('search', 'car')]`
	`Bloom: miss`
	`Bloom: hit, HashTable: hit`
	`Bloom: hit, HashTable: hit`
	`Bloom: hit, HashTable: hit`
	`0000011000100000`
	`0: []`
	`1: []`
	`2: ['car' 'dog' 'car']`
	`3: []`

Explanation

String	Key	Division $h_1(\cdot)$	Mid-Square $h_2(\cdot)$
apple	530	530 % 16 = 2	(530*530) // (2^14) % 16 = 1
book	427	427 % 16 = 11	(427*427) // (2^14) % 16 = 11
car	310	310 % 16 = 6	(310*310) // (2^14) % 16 = 5
dog	314	314 % 16 = 10	(314*314) // (2^14) % 16 = 6

g. Example execution

```
>> ./pa5.exe 5 "[('M', 4), ('b', 16), ('insert', 'apple'),
('insert', 'book'), ('insert', 'car'), ('search', 'apple')]"
[Task 5]
Bloom: hit, HashTable: hit
0110011000010000
0: []
1: []
2: ['apple' 'car']
3: ['book']
```