**Programming Assignment #4**

Lecturer: Prof. Jaesik Park

Teaching Assistants: Sangjun Lee, Suhyeon Jeong, Seunguk Do, Minjae Jeong

|  |
| --- |
| *\*\*\*\* PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT \*\*\*\**  Due date: 11:59PM June 2 , 2023  Evaluation policy:   * Late submission penalty   + 11:59PM Jun 2 ~ 11:59PM Jun 3     - Late submission penalty (30%) will be applied to the total score   + After 11:59PM Jun 3     - 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 * Please check if your program generates “submit.txt” same as “answer.txt”, after running ./pa4.exe 0 * We won’t accept any submission via email - it will be ignored * Please do not use the containers in C++ standard template library (STL)   + Such as:     - #include <queue>     - #include <vector>     - #include <stack>   + Any submission using the containers in STL will be disregarded   Any questions?   * Please use PLMS - Q&A board |

1. Basic instruction
   1. Please refer to the attached file named DataStructure\_PA4\_instructions.pptx
2. Bucket hashing (2 pts)
3. Implement **a closed hash table** with **bucket hashing**. This hash table is used with -bit integer keys, and hashing into a bucket table of size , and the size of each bucket is . So the total capacity of a hash table will be . Also, the hash table contains an overflow bucket, of which size is limited to , to store overflowed elements sequentially.

We use the **binary mid-square** hash function, and as a key is bits, your code should treat the square of the key as bits. You can assume that *r* is an even number, and keys are *non-negative* integers.

**Note that you cannot insert a new key into a deleted slot.** And you don’t need to consider an overflow of the overflow bucket.

You can modify hast\_function.cpp, hash\_function.h, hash\_table.cpp and hash\_table.h files for this problem.

1. Input & output

Input: A sequence of commands

* ('n',integer): the size of a key.  
   (The first command is always 'n')
* ('r',integer): the size of an index.  
   (The second command is always 'r')
* ('k',integer): the size of a bucket.  
   (The third command is always 'k')
* ('insert',integer): insert integer into the hash table.
* ('erase',integer): delete integer from the hash table.

Output: For each slot of the hash table and the overflow bucket, print out

* the value if the state of the slot is occupied.
* the state if the state of the slot is empty or deleted.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('n',4),('r',2),('k',2),('insert',0),('insert',1),('insert',3)] | Hash table  0: 0,1  1: 3,empty  2: empty,empty  3: empty,empty  ----------  Overflow  0: empty  1: empty  2: empty  3: empty |
| [('n',4),('r',2),('k',2),('insert',0),('insert',1),('insert',2),('erase',1),('erase',2)] | Hash table  0: 0,deleted  1: empty,empty  2: empty,empty  3: empty,empty  ----------  Overflow  0: deleted  1: empty  2: empty  3: empty |
| [('n',4),('r',2),('k',2),('insert',9),('insert',12),('insert',4),('erase',4),('insert',0),('insert',1),('insert',2)] | Hash table  0: 0,1  1: empty,empty  2: 9,12  3: empty,empty  ----------  Overflow  0: deleted  1: 2  2: empty  3: empty |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 1 "[('n',4),('r',2),('k',2),('insert',0),('insert',1),('insert',3)]"  [Task 1]  Hash table  0: 0,1  1: 3,empty  2: empty,empty  3: empty,empty  ----------  Overflow  0: empty  1: empty  2: empty  3: empty |

1. Double Hashing (2 pts)
   1. Implement **a closed hash table** with **double hashing**. This hash table is used with -bit integer keys and hashing into a table of size . The index of the key after -th collision, , is:

where is the **multiplication** hash function, and is the **digit folding** hash function. In the multiplication hash function, we first multiply the key by a constant real number in the range and extract the fractional part of . Then, we multiply this value by the size of a hash table and take the ﬂoor of the result. So will be denoted as

For simplicity, let’s use . Also, the digit folding hash function will sum all bits of a key, treating the key as bits. You can assume that keys are *non-negative* integers.

You don’t need to consider an insertion when the table is full, a deletion of a key which does not exist, and multiple insertions of the same key. And also you don’t need to consider the case when the hash function cannot find an available slot. **Note that you cannot insert a new key into the deleted slot.**

You can modify hast\_function.cpp, hash\_function.h, hash\_table.cpp and hash\_table.h files for this problem.

Hint: Try to use fmod function in math library to extract the fractional part after division.

* 1. Input & Output

Input: A sequence of commands

* ('n',integer): the size of a key.

(The first command is always 'n')

* ('r',integer): the size of an index.

(The second command is always 'r')

* ('insert',integer): insert integer into the hash table.
* ('erase',integer): delete integer from the hash table.

Output: For each slot of the hash table, print out

* the value, if the state of the slot is occupied.
* the state if the state of the slot is empty or deleted.
  1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('n',4),('r',2),('insert',15),('insert',13),('insert',3)] | 0: empty  1: 13  2: 15  3: 3 |
| [('n',4),('r',2),('insert',15),('insert',13),('insert',3),('erase',13),('erase',3),('insert',4)] | 0: 4  1: deleted  2: 15  3: deleted |
| [('n',4),('r',3),('insert',2),('insert',3),('insert',12),('insert',13),('insert',7),('erase',3),('insert',10)] | 0: 7  1: 2  2: deleted  3: 12  4: 10  5: 13  6: empty  7: empty |

* 1. Example execution

|  |
| --- |
| >> ./pa4.exe 2 "[('n',4),('r',2),('insert',15),('insert',13),('insert',3)]"  [Task 2]  0: empty  1: 13  2: 15  3: 3 |

1. Undirected Graph - Graph Traversal (2 pts)
2. Implement a function that perform DFS and BFS traversal from the given **undirected graph**. The graph may consist of several connected components. The search starts with the node of the smallest label in **lexicographical** order.   
   If there exist *n* connected component, then 1) print the total number of connected component and 2) print *n* traverses separated with a newline in the ascending lexiographical order.

e.g.) A D B C E

C E (correct) A D B (incorrect)

You can modify graph.cpp and graph.h files for this problem.

1. Input & Output

Input: Pairs of node labels that indicate edges.

* + - ('A','B'): an edge between node A and B.
    - ('End','DFS') or ('End','BFS'):String representing traverse mode.
    - If the input edge already exists in the graph, ignore the input.

Output:

* The number of connected component in the given undirected graph
* The result of DFS or BFS traverse separated with white space. Multiple traverses should be separated with a new line, in an ascending lexiographical order.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),('B','F'),('C','F'),('C','E'),('End','DFS')] | 1  A B C E F D |
| [('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),('B','F'),('C','F'),('C','E'),('End','BFS')] | 1  A B C D E F |
| [('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),('B','F'),('C','F'),('C','E'),('G','H'),('End','DFS')] | 2  A B C E F D  G H |
| [('A','B'),('A','C'),('A','D'),('B','F'),('C','F'),('H','G'),('E','J'),('J','I'),('End','DFS')] | 3  A B F C D  E J I  G H |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 3 "[('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),('B','F'),('C','F'),('C','E'),('End','DFS')]"  [Task 3]  1  A B C E F D |

1. Directed Graph - Strongly Connected Component (1 pts)
2. Implement a function that finds every strongly connected components in the given directed graph. A strongly connected component is a **maximal strongly connected** subgraph which every vertex is reachable from every other vertex within the subgraph.  
   If there exist *n* strongly connected component, then 1) print the total number of strongly connected component and 2) print the strongly connected components in an **ascending lexiographical** order of it’s **smallest** label of nodes. Print each strongly connected component by printing an **ascending lexiographical order of node labels** separated by white space.

e.g.) A D B C E D A

B C E (correct) A D (incorrect) B C E (incorrect)

You can modify graph.cpp and graph.h files for this problem.

1. 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 strongly connected component in the given directed graph.
* Strongly connected components separated with new line. It should be printed so that the label of the smallest node in each component is in ascending lexiographical order.
* The label of nodes in strongly connected component should be printed in an ascending lexiographical order and separated with white space.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A','B'),('B','C'),('A','C')] | 3  A  B  C |
| [('A','B'),('B','C'),('C','A')] | 1  A B C |
| [('A','B'),('A','C'),('D','C'),('C','E'),('B','D'),('E','D')] | 3  A  B  C D E |
| [('A','B'),('A','C'),('C','D'),('C','E'),('D','B'),('D','E'),('E','D'),('C','F'),('F','A')] | 3  A C F  B  D E |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 4 "[('A','B'),('A','C'),('D','C'),('C','E'),('B','D'),('E','D')]"  [Task 4]  3  A  B  C D E |

1. Single-Source Shortest Path – Dijkstra’s Algorithm (2 pts)

라인, 도표, 스크린샷, 원이(가) 표시된 사진

자동 생성된 설명

1. Implement a function that finds the shortest **path** from the source node to the destination node in the given undirected graph using **Dijkstra algorithm**.   
   We assume that the given graph is a directed, weighted, and weakly-connected graph. All weights of edges are **positive (i.e. larger than 0)**.   
   This function should return the sequence of node labels along the path and also the length (sum of the weights of the edges) of the path. If there exists multiple shortest path, print out all of them with ascending lexicographic order. (e.g. if both of A->B->E and A->C->E are shortest paths with cost 5, prints out ‘A B E 5’ then ‘A C E 5’). If the path from the source node to the destination node doesn’t exist, return ‘error’.

You can modify the graph.cpp and graph.h files for this problem.

1. Input & output

Input: A sequence of commands

* ('A-B',integer): an edge between node A to node B with a weight value {integer}.
* ('A','B'): a pair of nodes with alphabet labels that indicates the source and the destination node. The first element indicates the source node and the second one indicates the destination node.

Output:

* A sequence of the node labels of the shortest path(s) followed by length of the path. If there exists multiple paths, you should print out all of them sorted by ascending lexicographic order, separated with newline(\n).
* error if the shortest path could not be determined

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A-B',10),('A-C',3),('B-D',5),('C-B',2), ('C-E',5),('A-D',20),('D-E',11),('A','E')] | A C E 8 |
| [('A-B',10),('A-C',3),('B-D',1),('C-B',2), ('C-E',15),('A-D',20),('D-E',11),('A','D')] | A C B D 6 |
| [('A-B',10),('A-C',3),('C-B',2),('D-E',11), ('A','D')] | error |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 5 "[('A-B',10),('A-C',3),('B-D',5),('C-B',2),('C-E',5),('A-D',20),('D-E',11),('A','E')]"  [Task 5]  A C E 8 |

1. All Pairs Shortest Path of Graph – Floyd’s Algorithm (3 pts)

원, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. Implement a function that finds the shortest **path** from the source node to the destination node in the given undirected graph using **Floyd’s Algorithm**. Unlike problem 5, there will be an edge(s) with negative weight value(s). There will be no negative cycle in the given undirected graph. This function should return the sequence of node labels along the path and also the length (sum of the weights of the edges) of the path. You can modify the graph.cpp and graph.h files for this problem. You don’t need to consider the multiple path case.
2. Input & Output

Input: A sequence of commands

* ('A-B',integer): an edge between node A to node B with weight value {integer}.
* ('A','B'): a pair of nodes with alphabet labels that indicates the source and the destination node. The first element indicates the source node and the second one indicates the destination node.

Output:

* A sequence of the node labels of the shortest path followed by length of the path. If the source and the destination node are the same, the distance should be 0. If the path doesn’t exist, the distance is INF.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A-B',4),('B-C',9),('B-D',1), ('C-D',10),('D-A',-2),('C','B')] | C B 9 |
| [('A-B',4),('A-F',3),('B-F',6), ('B-C',9),('F-E',-2),('C-E',-3),  ('E-D',6),('A','D')] | A F E D 7 |
| [('A-B',4),('C-E',-2),('H-C',3),  ('B-F',6),('A-G',7),('G-B',-4),  ('G-F',7),('E-D',6),('G','C')] | INF |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 6 "[('A-B',4),('A-F',3),('B-F',6),('B-C',9),('F-E',-2),('C-E',-3),('E-D',6), ('A','D')]"  [Task 6]  A F E D 7 |

1. Prim’s Algorithm (3 pts)
2. Implement a function that finds the Minimum-cost Spanning Tree (MST) of the given weighted undirected graph **using Prim’s algorithm**. Given a start node, this function starts with the single-vertex tree of the given node. Then, the function prints the path from the start node to the newly added node. If there are multiple edges with the same weight, this function checks **the label of the added node** (i.e., the node which is not included in the tree) and selects the node with the first label in **lexicographical order**. Finally, the function returns the cost of the MST (i.e., the sum of edge weights). You can assume that the given graph is a connected graph.

You can modify graph.cpp and graph.h files for this problem.

1. Input & Output

Input: A sequence of commands

* ('A-B',integer): an edge between node A to node B with a weight value {integer}.
* ('MST','A'): find MST using the Prim’s algorithm which starts with node A.

Output:

* For each time the tree grows, print **the path from the start node to the added node** as a string separated with a white space.
* Print the **cost of MST**.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','A')] | A C  A C D  A C D B  A C D E  9 |
| [('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','E')] | E D  E D B  E D C  E D C A  9 |
| [('A-B',3),('A-C',1),('B-C',1),('B-D',4),('C-D',2),('D-E',5),('MST','E')] | E D  E D C  E D C A  E D C B  9 |

1. Example execution

|  |
| --- |
| >> ./pa4.exe 7 "[('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','A')]"  [Task 7]  A C  A C D  A C D B  A C D E  9 |

1. Kruskal’s Algorithm (3 pts)
   1. Implement a function that finds the Minimum-cost Spanning Tree (MST) of the given weighted undirected graph **using Kruskal's algorithm**. The function prints the added edge and the weight of the edge each time the tree grows. When printing an edge, you must print the label in **lexicographical order.** If there are multiple edges with the same weight, this function also selects the edge in lexicographical order. That means it compares the first node of edges, and if the first node is the same, it compares the second node of edges. The function returns the path between two nodes and the cost of the MST (i.e., the sum of edge weights).

You can assume that the given graph is a connected graph. You can modify graph.cpp and graph.h files for this problem.

* 1. Input & Output

Input: A sequence of commands

* ('A-B',integer): an edge between node A to node B with a weight value {integer}.
* ('MST',char): find MST using Kruskal's algorithm and find the path starting from the node with the smallest label in lexicographical order to the node with label {char}.

Output:

* For each time the tree grows, print the labels of the nodes indicating the added edges in lexicographical order and the weight of the edge as a string separated with a white space.
* Print **the path** from the node with the **smallest label in lexicographical order** to the node with **label {char}** and **the cost of MST**. You can assume the given **char** is different with the smallest label of the MST.
  1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','B')] | A C 1  B D 1  C D 2  D E 5  A C D B 9 |
| [('D-B',1),('D-C',2),('E-D',5), 'B-A',3),('C-A',1),('C-B',4),('MST','D')] | A C 1  B D 1  C D 2  D E 5  A C D 9 |
| [('A-B',1),('B-C',1),('C-D',1),('D-A',1),('A-C',1),('B-D',1),('MST','D')] | A B 1  A C 1  A D 1  A D 3 |

* 1. Example execution

|  |
| --- |
| >> ./pa4.exe 8 "[('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','B')]"  [Task 8]  A C 1  B D 1  C D 2  D E 5  A C D B 9 |