Programming Assignment #4

Lecturer: Prof. Jaesik Park
Teaching Assistants: Hyeonjun Lee, Hyunsoo Ahn, Sangwoo Ryu

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

Due date: 11:59 PM June 3, 2022

Evaluation policy:

- Late submission penalty.
    - 11:59 PM June 3 ~ 11:59 PM June 4.
        - Late submission penalty (30%) will be applied to the total score.
    - After 11:59 PM June 4.
        - 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.
    - There might be a penalty if the submission would not work in the "repl.it + C++11" environment.
- What you need to submit.
    - Your submission file should follow designated format as below.
    - Penalty (-1 point) will be applied for Inconsistency of submission format
    - A zip file named "pa4.zip" that contains
        - pa4.cpp
        - graph.cpp and graph.h

Any questions?

- Please use LMS - Q&A board.

1. Graph Traversal (2 pts)

a. Implement a function that finds DFS traverse from the given undirected graph. The graph may consist of several connected graphs. The search starts with the node coming first in **the lexicographical order of labels**. Also, the next node should be selected in lexicographical order among connected nodes. If there exist *n* connected graphs, then print *n* traverses separated with a newline in lexicographical 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.

b. Input & output

Input:
- Pairs of nodes with alphabet labels that indicate edges.
  `('A','B')`: an edge between node A and node B.

Output:
- Result of traverse separated with a white space
- Multiple traverses are separated with a new line, in lexicographical order.

c. Example Input & Output

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

`"[('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),('B','F'),('C','F'),('C','E'),('G','H')]"`	`A B C E F D`
	`G H`

`"[('A','B'),('A','C'),('A','D'),('B','F'),('C','F'),('H','G'),('E','J'),('I','J')]"`	`A B F C D`
	`E J I`
	`G H`

d. Example execution

```
>> ./pa4.exe 1
"[('A','C'),('A','B'),('A','D'),('B','C'),('B','D'),('B','E'),(
'B','F'),('C','F'),('C','E')]"
[Task 1]
A B C E F D
```

2. Tree in the Graph (2 pts)

a. Implement a function that find number of components satisfying tree property (Trees) in the given undirected graph. When a component is *connected and acyclic (or having n-1 edges with n nodes)*, we regarded the component as tree. You can modify `graph.cpp` and `graph.h` files for this problem.

b. Input & output

Input: Pairs of nodes with alphabet labels that indicate edges.

- If the input edge already exists in the graph, ignore the input.

Output: The number of trees in the graph generated from input.

- Return 0 if no component in the graph satisfies the tree property.

c. Example Input & Output

Input	Output
`[('A','B'),('B','C')]`	1
`[('A','B'),('C','D'),('E','F'),('F','G')]`	3
`[('A','B'),('C','D'),('E','F'),('F','G'),('B','E')]`	2
`[('A','B'),('C','D'),('E','F'),('F','G'),('B','E'),('B','G')]`	1

d. Example execution

```
>> ./pa4.exe 2 "[('A','B'),('C','D'),('E','F'),('F','G')]"
[Task 2]
3
```

3. Strongly Connected Component in a directed graph (2 pts)

a. Implement a function that retrieves the Strongly Connected Components (SCC) in the given **directed graph**. This function finds the **largest Strongly Connected Component**s that can be found in the given directed graph and prints out the node labels of it. There might be multiple SCC with the same size. If that is the case, retrieve the SCC that comes first in lexicographical order. For instance, if two connected components, `(A,B,C)` and `(D,E,F)` are found, select `(A,B,C)` because A comes first than B in lexicographical order.

   Unlike task 1, the edges of the graph have direction in this time. For instance, `(‘A’,’B’)` is different from `(‘B’,’A’)`. You can modify `graph.cpp` and `graph.h` files for this problem.

b. Input & output
   Input: Pairs of nodes with alphabet 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: A sequence of the node labels of the largest strongly connected component.
   - Output should be sorted in **lexicographical order** and separated with space.

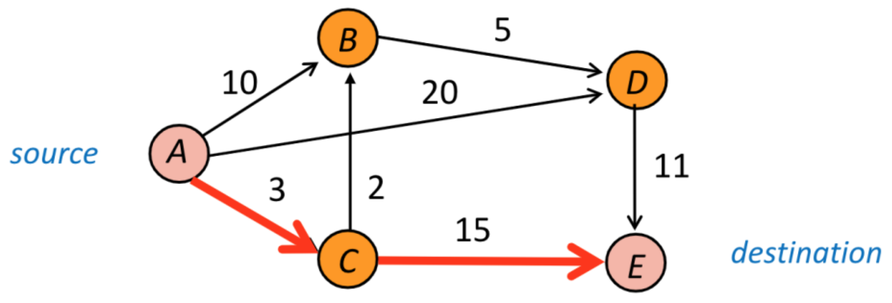
c. Example Input & Output

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

d. Example execution

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

4. Single-Source Shortest Path – Dijkstra's Algorithm (2 pts)



Shortest path length = 18

a. Implement a function that finds the **shortest path** from the source node to the destination node in the given 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.

b. Input & output

   Input: A sequence of commands

   - `('A-B',integer)`: an edge from 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

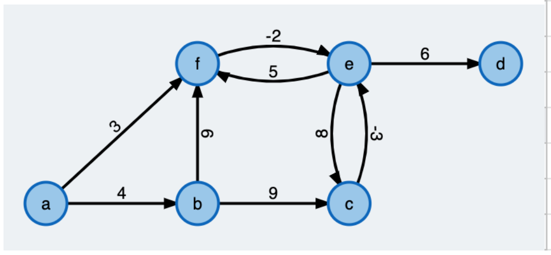
c. Example Input & Output

Input	Output
`[('A-B',10),('A-C',3),('B-D',5),('C-B',2),('C-E',15),('A-D',20),('D-E',11),('A','E')]`	`A C E 18`
`[('A-B',10),('A-C',3),('B-D',5),('C-B',2),('C-E',15),('A-D',20),('D-E',11),('A','D')]`	`A C B D 10`
`[('A-B',10),('A-C',3),('B-D',5),('C-B',2),('C-E',15),('A-D',20),('D-E',11),('D','A')]`	`error`

d. Example execution

```
>> ./pa4.exe 4 "[('A-B',10),('A-C',3),('B-D',5),('C-B',2),('C-
E',15),('A-D',20),('D-E',11),('A','E')]"
[Task 4]
A C E 18
```

5. All Pairs Shortest Path of Graph – Floyd's Algorithm (3 pts)



	a	b	c	d	e	f
a	0	4	9	7	1	3
b	INF	0	9	10	4	6
c	INF	INF	0	3	-3	2
d	INF	INF	INF	0	INF	INF
e	INF	INF	8	6	0	5
f	INF	INF	6	4	-2	0

a. Implement a function that finds the **shortest paths** of all pairs of nodes using **Floyd's Algorithm**. Unlike problem 3, there will be an edge(s) with negative weight value(s). There will be no negative cycle in the given graph. This function should return the sequence of node labels along the path and also the distance of the path from the source to the destination. You can modify the `graph.cpp` and `graph.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- `('A-B',integer)`: an edge from 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 distance matrix in a string format. The nodes in the row and column of the matrix are sorted in lexicographic order. 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`.

c. Example Input & Output

Input	Output
`[('A-B',4),('B-C',9),('B-D',1),('C-D',10),('D-C',8),('D-A',-2),('C','B')]`	`C D A B 12`
`[('A-B',4),('A-F',3),('B-F',6),('B-C',9),('F-E',-2),('E-F',5),('C-E',-3),('E-C',8),('E-D',6),('A','D')]`	`A F E D 7`
`[('A-B',4),('B-C',9),('C-E',-2),('E-C',8),('B-H',1),('A-H',4),('H-C',3),('B-F',6),('A-G',7),('G-B',-4),('B-G',8),('G-F',7),('F-E',6),('E-F',5),('E-D',6),('G','C')]`	`G B H C 0`

d. Example execution

```
>> ./pa4.exe 5 "[('A-B',4),('A-F',3),('B-F',6),('B-C',9),('F-
E',-2),('E-F',5),('C-E',-3),('E-C',8),('E-D',6), ('A','D')]"
[Task 5]
A F E D 7
```

6. Prim's Algorithm (2 pts)

a. 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 added edge and the weight of the edge each time the tree grows. When printing an edge, you first have to print the label of the node that already exists in the tree, then print the label of the node that was recently inserted into the tree. 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.

b. Input & Output

   Input: A sequence of commands
   - `('A-B',integer)`: an edge between node A and node B with 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 labels of the nodes indicating the added edges and the weight of the edge as a string separated with a white space.
   - Print the cost of MST.

c. 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 1 C D 2 D B 1 D E 5 9
`[('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST','E')]`	E D 5 D B 1 D C 2 C A 1 9
`[('A-B',3),('A-C',1),('B-C',1),('B-D',4),('C-D',2),('D-E',5),('MST','E')]`	E D 5 D C 2 C A 1 C B 1 9

d. Example execution

```
>> ./pa4.exe 6 "[('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-
D',2),('D-E',5),('MST','A')]"
[Task 6]
A C 1
C D 2
D B 1
D E 5
9
```

7. Kruskal's Algorithm (2 pts)

a. 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 have to 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 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.

b. Input & Output
   Input: A sequence of commands
   - `('A-B',integer)`: an edge between node A and node B with weight value {`integer`}.
   - (`'MST',NULL`): find MST using Kruskal's algorithm.

   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 cost of MST.

c. Example Input & Output

Input	Output

`[('A-B',3),('A-C',1),('B-C',4),('B-D',1),('C-D',2),('D-E',5),('MST',NULL)]`	`A C 1`
	`B D 1`
	`C D 2`
	`D E 5`
	`9`

`[('D-B',1),('D-C',2),('E-D',5),('B-A',3),('C-A',1),('C-B',4),('MST',NULL)]`	`A C 1`
	`B D 1`
	`C D 2`
	`D E 5`
	`9`

`[('A-B',1),('B-C',1),('C-D',1),('D-A',1),('A-C',1),('B-D',1),('MST',NULL)]`	`A B 1`
	`A C 1`
	`A D 1`
	`3`

d. Example execution

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

This is the last page of the programming assignments.
We hope you have enjoyed the course.

We do appreciate your participation to make this class better.

Warm wishes,

Jaesik Park Hyeonjun Lee Hyunsoo Ahn Sangwoo Ryu