Programming Assignment #4

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Chunghyun Park, Kanghee Lee, Seungjoo Shin, Dongmin You

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

Due date: 11:59 PM December 26, 2022

Evaluation policy:

- Late submission penalty.
  - 11:59 PM December 26 ~ 11:59 PM December 27.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM December 27.
    - 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.

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.
- Files you need to submit. (Do not change the filename.)
  - pa4.cpp
  - graph.cpp and graph.h

Any questions?

- Please use PLMS - Q&A board.

1. Undirected Graph - Graph Traversal (2 pts)

a. Implement a function that finds DFS, BFS traverse from the given **undirected graph**. The graph may consist of several connected graphs. The search starts with the node of the smallest label. If there exist *n* connected graphs, then print *n* traverses separated with a newline in the ascending 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 **integer labels** that indicate edges.

- `(A,B)`: an edge between node A and node B.
- `('End','DFS') or ('End','BFS')`:String representing traverse mode.

Output:

- Result of DFS or BFS traverse separated with a white space
- Multiple traverses separated with a new line, in the ascending 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'), ('End','DFS')]"	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)]	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')]"	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')]"	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'),
('End','DFS')]"
[Task 1]
A B C E F D
```

2. Undirected Graph - Cycle Count (1 pts)

a. Implement a function that returns the number of cycles in the given **undirected graph**. A cycle is a non-empty path in which the only repeated vertices are the first and last vertices. Note that any subset of vertices of a cycle **does not** form another cycle (that is, there is no **inner-cycle**). 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 between node A and node B.
- If the input edge already exists in the graph, ignore the input.

Output:

- The number of cycles in the given undirected graph

c. Example Input & Output

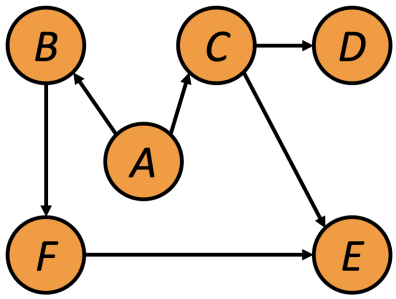
Input	Output
`"[('A','B'), ('B','C'), ('A','C')]"`	1
`"[('A','B'), ('A','C'), ('C','D'), ('C','E'), ('B','D'), ('D','E')]"`	2
`"[('A','B'), ('A','C'), ('C','D'), ('C','E'), ('D','B'), ('D','E'), ('C','F'), ('A','F')]"`	3

d. Example execution

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

3. Directed Graph - Topological Sort (2 pts)

a. Implement a function that performs a topological sort using the given directed graph. **If there exists more than one result, print the topological sort that comes first in the ascending order.** To take an example below, acceptable topological sorts are `'A B C D F E'`, `'A C B F E D'`, `'A C D B F E'`, etc. Among these, the desirable output is `'A B C D F E'`. Also, print `'error'` if the topological sort could not be performed. 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:

- Result of topological sort or `'error'` message

c. Example Input & Output

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

d. Example execution

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

4. Directed Graph - Cycle Count (2 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 an undirected graph, 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 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

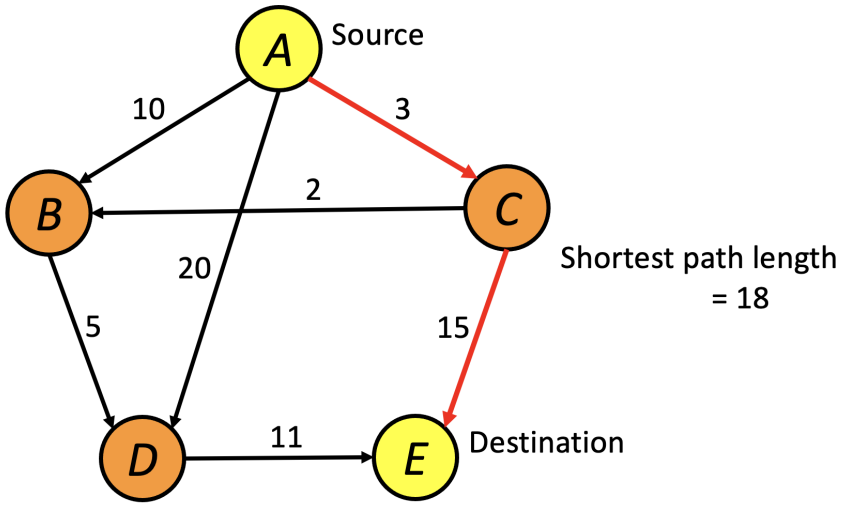
c. Example Input & Output

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

```
>> ./pa4.exe 4 "[('A','B'), ('A','C'), ('C','D'), ('C','E'),
('D','B'), ('D','E')]"
[Task 4]
0
```

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



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 node 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

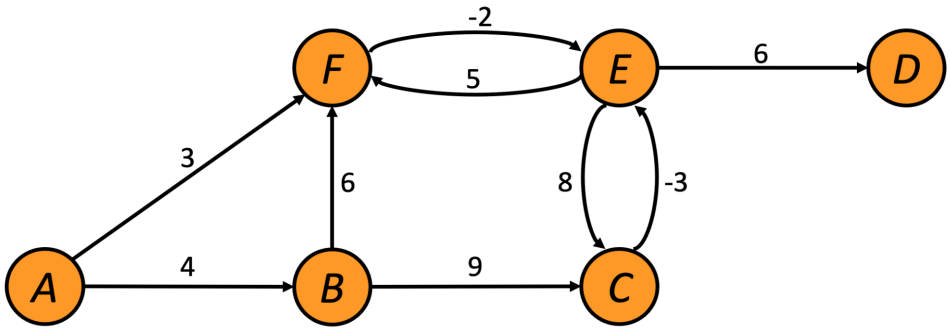
c. Example Input & Output

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

6. All Pairs Shortest Path of Graph – Floyd's Algorithm (2 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. a. Implement a function that finds the **shortest paths** of all pairs of nodes. 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 distances of all paths in the given graph like the above image. 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`}.

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)]"	0 4 13 5 -1 0 9 1 8 12 0 10 -2 2 8 0
"[('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)]"	0 4 9 7 1 3 INF 0 9 10 4 6 INF INF 0 3 -3 2 INF INF INF 0 INF INF INF INF 8 6 0 5 INF INF 6 4 -2 0
"[('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)]"	0 3 7 11 5 9 7 4 INF 0 4 8 2 6 8 1 INF INF 0 4 -2 3 INF INF INF INF INF 0 INF INF INF INF INF INF 8 6 0 5 INF INF INF INF 14 12 6 0 INF INF INF -4 0 4 -2 2 0 -3 INF INF 3 7 1 6 INF 0

d. Example execution

```
>> ./pa4.exe 4 "[('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)]"
[Task 4]
0 4 9 7 1 3
INF 0 9 10 4 6
INF INF 0 3 -3 2
INF INF INF 0 INF INF
INF INF 8 6 0 5
INF INF 6 4 -2 0
```

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

8. 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 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 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 6 "[('A-B', 3), ('A-C', 1), ('B-C', 4), ('B-D',
1), ('C-D', 2), ('D-E', 5), ('MST', NULL)]"
[Task 6]
A C 1
B D 1
C D 2
D E 5
9
```