Programming Assignment #4

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Seokjun Choi, Yujin Jeon, Suhyun Shin, Eunsue Choi

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

Due date: 11:59 PM December 26, 2023

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 “DataStructure_PA_instructions.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 - Connected Component Count (2 pts)

a. Implement a function that returns the number of connected components in the given **undirected graph**. 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. The names of the nodes are restricted to uppercase alphabetic characters.
- If the input edge already exists in the graph, ignore the input.

Output:

- The number of connected components in the given undirected graph.

c. Example Input & Output

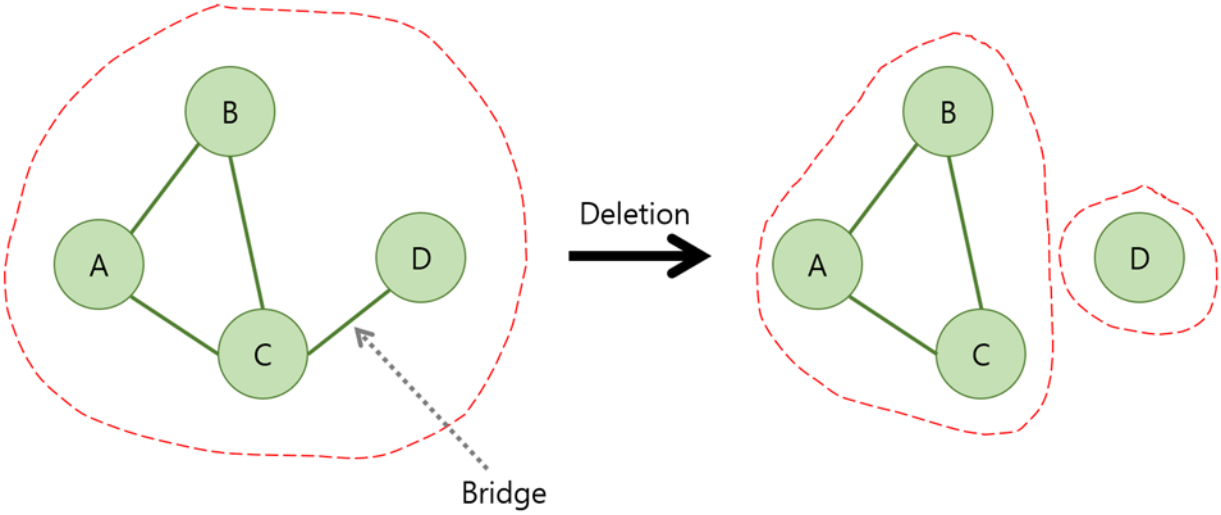
Input	Output
`"[('A','B'), ('B','C'), ('A','C')]"`	1
`"[('A','B'), ('A','C'), ('B','C'), ('D','E'), ('E','F'), ('C','G')]"`	2
`"[('A','B'), ('C','D'), ('E','F'), ('F','G')]"`	3

d. Example execution

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

2. Undirected Graph - Bridge Count (2 pts)

a. In graph theory, bridge is an edge of a graph whose deletion increases the graph's number of connected components. Implement a function that returns the number of bridges in the given **undirected graph**. 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. The names of the nodes are restricted to uppercase alphabetic characters.
- If the input edge already exists in the graph, ignore the input.

Output:

- The number of bridge.

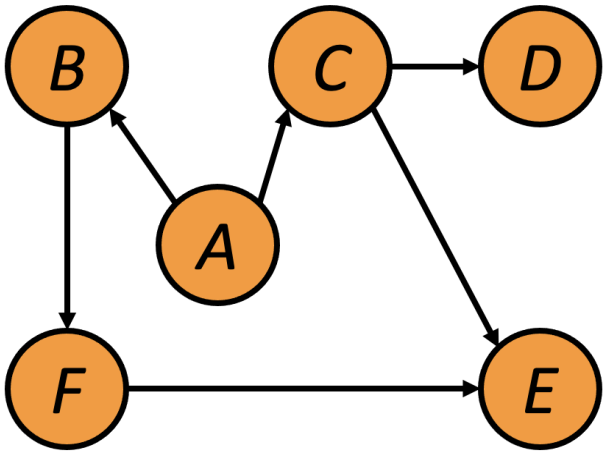
c. Example Input & Output

Input	Output
`"[('A','B'), ('B','C'), ('A','C')]"`	`0`
`"[('A','B'), ('A','C'), ('B','C'), ('D','E'), ('E','F'), ('C','G')]"`	`3`
`"[('A','B'), ('C','D'), ('C','E'), ('D','E'), ('C','F'), ('A','F')]"`	`3`

d. Example execution

```
>> ./pa4.exe 2 "[('A','B'), ('B','C'), ('A','C')]"
[Task 2]
0
```

3. Directed Graph - Topological Sort and Cycle Count (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, and the number of cycles. 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 with number of cycles.

    c. Example Input & Output

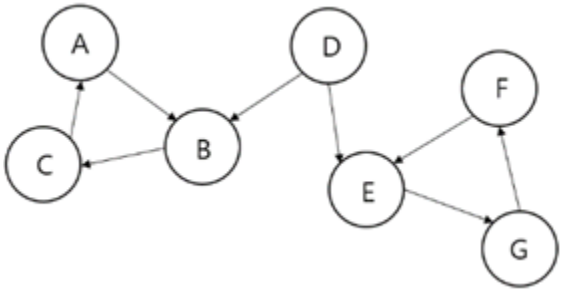
Input	Output
`"[('A','B'),('A','C'),('B','F'),('F','E'),('C','E'),('C','D')]"`	`A B C D F E`
`"[('A','B'), ('A','D'), ('B','C'), ('C','E'), ('D','E'), ('E','F')]"`	`A B C D E F`
`"[('B','C'), ('C','D'), ('D','B')]"`	`Error` `1`

d. Example execution

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

4. Directed Graph - Strongly Connected Components (2 pts)
    a. Implement a function that returns the strongly connected components in the given **directed graph**. Print the strongly connected components in the ascending order. We show an example below. 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:
        - Strongly connected components in ascending order. Each strongly connected components are separated with Enter as shown in Output examples.
    c. Example Input & Output



The input and output of such graph is written in the first row.

Input	Output

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

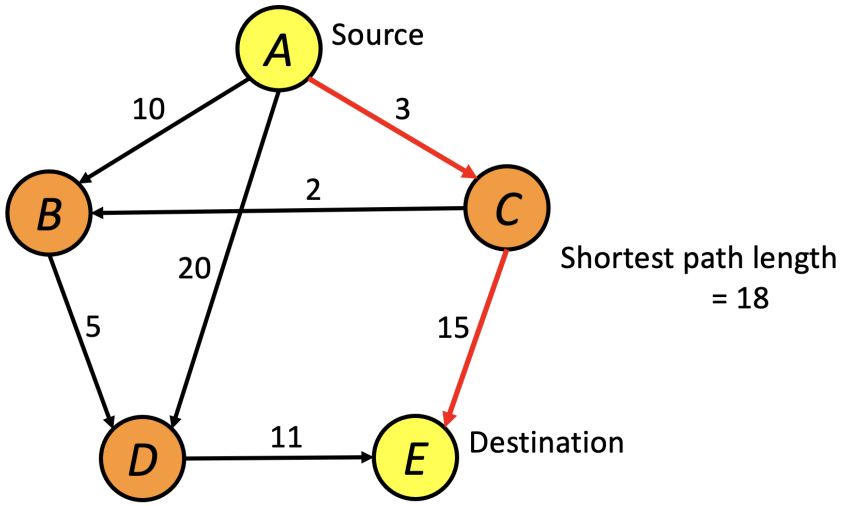
`"[('A','B'), ('B','C'), ('A','C')]"`	`A`
	`B`
	`C`

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

d. Example execution

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

5. Single Source Shortest Path – Dijkstra's Algorithm (3 pts)



a. Implement a function that finds paths from the source node to all other nodes that the total cost of path is lower that the given budget. 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 pair of the reachable destination node and the total cost (sum of the weights of the edges) of the path.If the path from the source node to the destination node doesn't exist, return a empty line. 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', integer)`: The first element indicates the source node and the second integer indicates the allowed budget.

Output:

- Pairs of the destination node and the total cost of the path. (the sequence of pairs should be in lexicographical order.) (node and cost separated with space.)
- An empty line if the path does not exist. (It should include '\n', a newline character.)

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',11)]"	B 5 C 3 D 10
"[('A-B',10),('A-C',3),('B-D',5),('C-B',2), ('C-E',15),('A-D',20),('D-E',11),('A',1)]"	
"[('A-B',10),('A-C',3),('B-D',5),('C-B',2), ('C-E',15),('A-D',20),('D-E',11),('A',20)]"	B 5 C 3 D 10 E 18
"[('A-B',10),('A-C',3),('B-D',5),('C-B',2), ('C-E',15),('A-D',20),('D-E',11),('D',12)]"	E 11

d. Example execution

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

6. 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 a weight value of {`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
```

7. Finding the Path of MST (2 pts)

a. Implement a function that determines the path between two specified nodes in the Minimum Spanning Tree (MST). Once the MST is obtained using Kruskal's algorithm, this function identifies the minimum-cost path between the given nodes. Assume that the given graph is a connected graph. You may modify the graph.cpp and graph.h files for this problem. (Note: This problem relies on Task 7; ensure that Task 7 is successfully implemented for this problem.)

b. Input & Output

Input: A sequence of commands

- `('A-B', integer)`: an edge between node A and node B with a weight value of {`integer`}.
- `('PATH-A-B', NULL)`: find the minimum-cost path of MST after performing the Kruskal's algorithm.

Output:

- Print the calculated path ([source node] [destination node] [weight])
- Print the cost of the path

c. Example Input & Output

Input	Output
"[('A-B', 3), ('A-C', 1), ('B-C', 4), ('B-D', 1), ('C-D', 2), ('D-E', 5), ('PATH-A-E', NULL)]"	A C 1 C D 2 D E 5 8
"[('A-B', 1), ('B-C', 1), ('C-D', 1), ('D-A', 1), ('A-C', 1), ('B-D', 1), ('PATH-D-B', NULL)]"	D A 1 A B 1 2

d. Example execution

```
>> ./pa4.exe 7 "[('A-B', 3), ('A-C', 1), ('B-C', 4), ('B-D',
1), ('C-D', 2), ('D-E', 5), ('PATH-A-E', NULL)]"
[Task 7]
A C 1
C D 2
D E 5
8
```