Programming Assignment #4

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

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

Due date: 11:59 PM May 30, 2024

Evaluation policy:

- Late submission penalty.
  - 11:59 PM May 30 ~ 11:59 PM May 31.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM May 31.
    - 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:

- Compile your file(s) using C++ 11 compiler on 'Replit' or 'Clion' and check your program before the submission
- All characters in submit.txt should be in uppercase letters, e.g., 'TRUE', 'FALSE', (except for [Task 1], [Task 2], …)
- 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. (Questions should be written in English and uploaded publicly)

1. Undirected Graph – Cycle (1 pts)

a. Implement a function that returns whether the given **undirected graph** forms a cycle or not. The given graph is a simple graph, which does not have more than one edge between any two vertices and self-loop. A cycle is a path of length 3 or more that starts and ends at the same vertices. 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. The names of the nodes are restricted to uppercase alphabetic characters.

Output:

- TRUE if the graph forms a cycle, FALSE otherwise. Note that they must be printed in uppercase.

c. Example Input & Output

Input	Output
`"[('A','B'), ('B','C'), ('A','C')]"`	TRUE
`"[('A','B'), ('A','C'), ('C','D')]"`	FALSE

d. Example execution

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

2. Undirected Graph – Furthest node (2 pts)

a. Implement a function that returns nodes which are the furthest distance from the given source node in the given **undirected graph**, and their distance. The distance is defined as the number of edges in the shortest path from the source node. Also, the given graph is a simple graph. You can modify `graph.cpp` and `graph.h` files for this problem.

b. Input & output

Input: Pairs of node labels that indicate edges, and the source node.

- `('A','B')`: an edge between node A and node B. The names of the nodes are restricted to uppercase alphabetic characters.
- `('A',NULL)`: the source node. You should find a node that is furthest from this source node.

Output:

- The furthest node. If there are some nodes that has the same distance with the found furthest node, print them all in lexicographic order.
- Distance

c. Example Input & Output

Input	Output
`"[('A','B'), ('B','C'), ('D','C'), ('A',NULL)]"`	`D` `3`

`"[('A','B'), ('B','C'), ('A','C'), ('A',NULL)]"`	`B`
	`C`
	`1`

`"[('A','B'), ('A','C'), ('C','D'), ('C','E'), ('B','D'), ('D','E'), ('A',NULL)]"`	`D`
	`E`
	`2`

`"[('A','B'), ('B','C'), ('D','C'), ('E','F'), ('A',NULL)]"`	`D` `3`

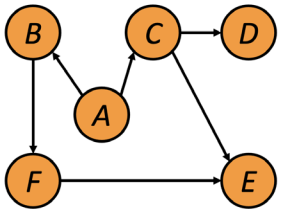
d. Example execution

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

```
D
3
```

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.
* You do not have to think about self-loop cases. We will not put such case in our test case.



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 and continue the program.

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 3 “[('A','B'),('A','C'),('B','F'),('F','
E'),('C','E'),('C','D')]”
[Task 3]
A B C D F E
```

4. Directed Graph – Strongly & Weakly Connected Components (2 pts)

a. Implement a function that returns the strongly connected components and weakly connected components in the given directed graph. Print the strongly and weakly connected components in the ascending order. We show an example below. You can modify **graph.cpp and graph.h and pa4.cpp** files for this problem.

b. Input & output

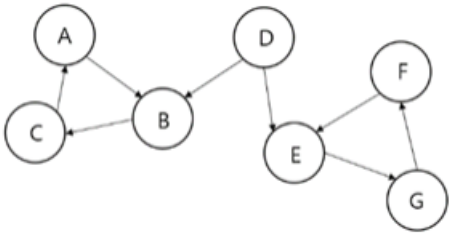
Input: Pairs of node labels that indicate edges.

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

Output:

- Strongly and Weakly connected components in ascending order. Each strongly and Weakly connected components are separated with Enter as shown in Output examples. Please first print all the strongly connected components then start to print the weakly connected components.

c. Example Input & Output



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

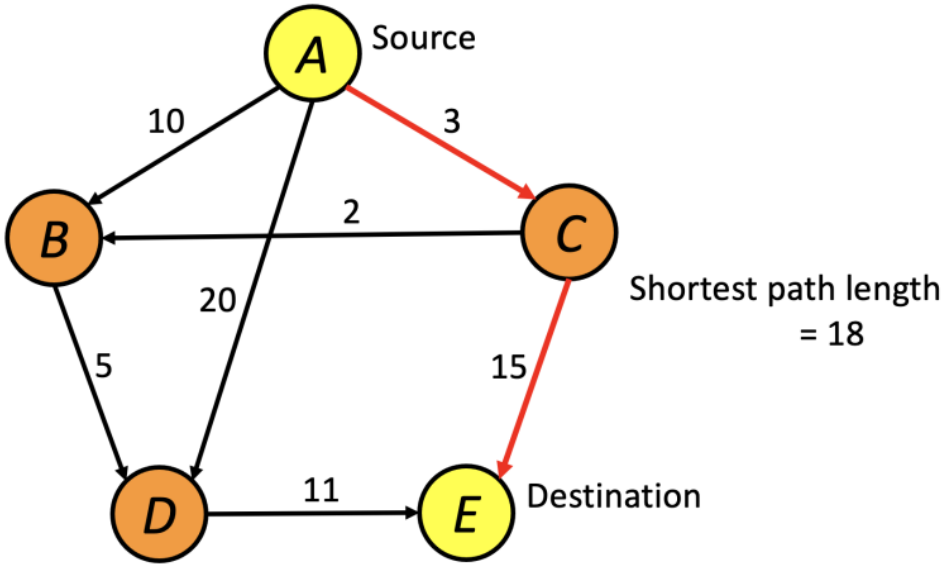
Input	Output
"[('A','B'),('C','A'),('B','C'),('D','B'),('D	A B C

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

d. Example execution

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

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



a. Implement a function that performs the following steps:
   i. Find all paths from the source node to other nodes such that the total cost of the path is lower than the given budget. Return pairs of reachable destinations nodes and the total cost (sum of the weights of the edges) of the paths.
   ii. From the found paths, identify and print the path that includes the most nodes and its total cost within the budget.

b. 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). If the path from the source node to the destination node doesn't exist, return an empty line.

c. The maximum number of nodes in the graph is 26 (A to Z), without considering duplicate nodes.

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

e. 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.
- Print the path that includes the most nodes within the budget at the end.
- An empty line if the path does not exist. It should include '\n', a newline character.

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 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),('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 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',12)]"	E 11 D E 11

f. Example input & output

g. 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',11)]"
[Task 5]
B 5
C 3
```

```
D 10
A C B D 10
```

6. Advanced Kruskal's Algorithm (4 pts)

a. Implement a function that finds the Minimum-cost Spanning Tree (MST) of the given weighted undirected graph **using Kruskal's algorithm**, with the option to **include or exclude specific edges**. The MST should include/exclude specific edges as indicated by input values of 0 or -1 (please refer to the input & output section for clarity). The function should print each added edge and its weight as the MST grows. The edge weights will be within the range of 1 to 100. When printing an edge, the labels must be in **lexicographical order**. If multiple edges have the same weight, the function should select edges in lexicographical order by comparing the first node of each edge, and if they are the same, then comparing the second node. The function returns the total cost of the MST (i.e., the sum of the edge weights). You can assume that the given graph is connected. You are allowed to 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`}.
- `('A-B', -1)`: an edge that should be excluded in MST
- `('A-B', 0)`: an edge that should be included in MST
- `('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), ('C-A', 1), ('C-B', 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), ('B-D', -1), ('MST', NULL)]"	A C 1 C D 2 A B 3 D E 5 11
"[('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-B', 1), ('D-C', 2), ('E-D', 5), ('B-A', 3), ('C-A', 1), ('C-B', 4), ('A-B', 0), ('MST', NULL)]"	A C 1 B D 1 A B 3 D E 5 10

d. Example execution

```
>> ./pa4.exe 6 "[('D-B', 1), ('D-C', 2), ('E-D', 5), ('B-A',
3), ('C-A', 1), ('C-B', 4), (‘B-D’, -1), ('MST', NULL)]"
[Task 6]
A C 1
C D 2
A B 3
D E 5
11
```