Data Structure Assignment#4

German language and literature

2016130927 Park Jun Yeong

I. Development environment: Microsoft Visual Studio 2013 Professional

II. Explanation of the algorithm and the code

i. A complete overview of code:

1). Graph construction

At the first moment, this program accepts command-line arguments (input.txt and output.txt) and recognized the number of nodes and edges, from which this graph is connected (that is, whether this input graph has only one component is checked by implementing DFS algorithm) simultaneously. If there are any exceptions such as unconnected graph or unspecified argument text files, error message is printed out and program exits.

2). MST by Kruskal algorithm:

Edges of graph are sorted in increasing order by weight, implemented by min priority queue. After the disjoint sets are constructed by the number of nodes in graph, the smallest weighted edge is selected one by one and written in output text file in that order, if its two incident node do not belong to the same set at that moment and then a union set including them is updated: or that edge is just discarded so that a cycle would not be constructed in this spanning tree.

ii. A complete explanation of the algorithm

Below are the explanations of what I intended to implement in this program.

1). Graph implementation

A structure graph is composed of two parts: the information of graph (i.e. number of nodes and edges) and a detailed list of header nodes, where each node indicates all of its adjacent nodes so that how these nodes are connected could be checked from any kind of incident nodes.

*Since this input graph is regarded as undirected, the number of insertion is twice of the exact edges in input graph: a certain edge between A-B is regarded as A→B and B→A. This doubled insertion process is needed to confirm if there is no other component in this graph: in order to verify the number of components of this graph with DFS algorithm, the searching process is needed not only from [tail] dimension, but also [head] dimension.

2). Connectedness

Whether this input graph is connected or not is verified with DFS algorithm. If it is connected, all of nodes could be visited from one invocation of DFS. If not, DFS should be invoked several times in order to visit all of nodes. That is, the number of calling DFS is the exact number of components of that graph. That is, whether a certain graph is connected is verified by counting the number of components.

3). Kruskal algorithm

Before edges are sorted, redundant ones are eliminated so that not only doubly inserted edges (A→B and B→A: A→B) but also multiple edges (among several edges A→B, the smallest weighted edge is selected) for the sake of computational convenience.

From this new-made graph, which informs about how needed edges are constructed, edges are sorted in increasing order by weight through min-priority queue, where stored items are efficiently sorted and then popped.

*even if the exact heap-size of this priority queue is same as the number of edges, it is limited to MAX_SIZE, which is set irrelevant to the input number of edges for the sake of convenience: unintended error would occur if MAX_SIZE is smaller so that some edition is needed if any.

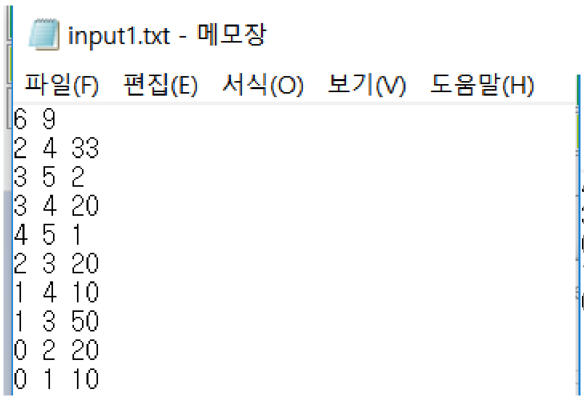
It is initialized that all of nodes are in each disjoint sets before edges are selected. In order to check whether a certain selected edge leads to a cycle in spanning tree or not, whether these two incident nodes of that edges belong to the same set is verified through Find and Union function: with the array Kruskal_Parent, Kruskal_Find function finds out a certain parent node, where value is identical to the index of Kruskal_Parent and they are unified in the same set if they are in different sets at this moment. This Find-set and Union function is the main logic of Kruskal algorithm to select which kinds of edges should be inserted.

*the size of Kruskal_Parent is also set as MAX_SIZE, which is irrelevant to the number of nodes in this case so that you may check whether it is smaller or not.

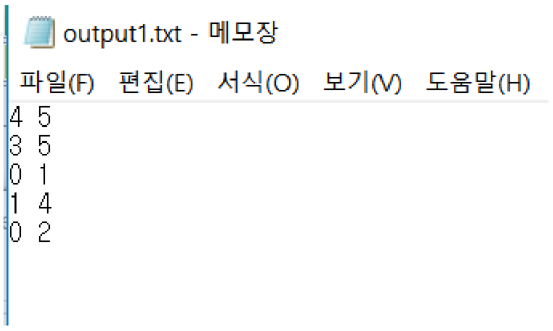
If selection occurs, that edge is also written in output text file so that the result of minimum spanning tree could be determined in this increasing order by weight of edges.

iii. Result of program with screenshot

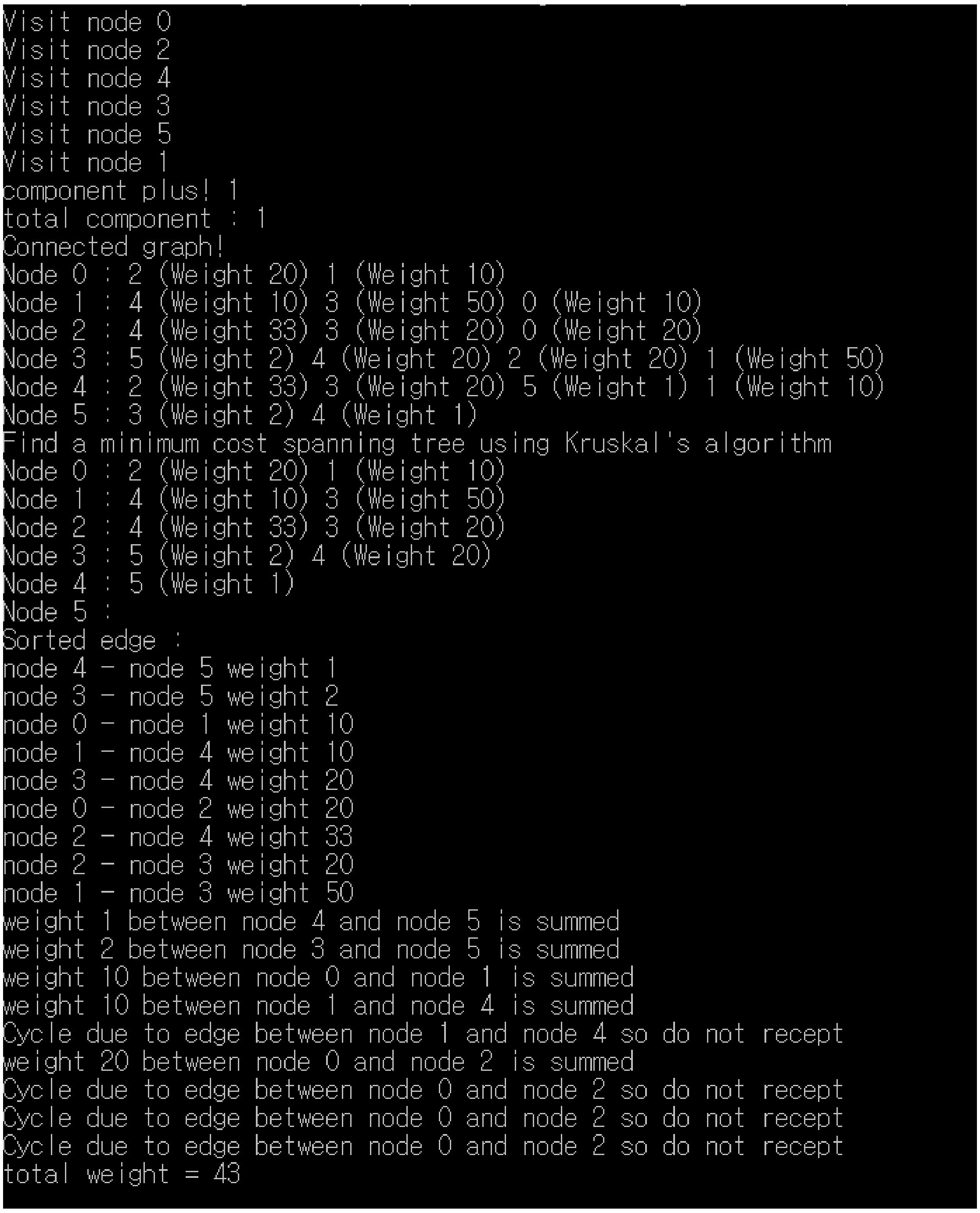
1). First test



```
6 9
2 4 33
3 5 2
3 4 20
4 5 1
2 3 20
1 4 10
1 3 50
0 2 20
0 1 10
```

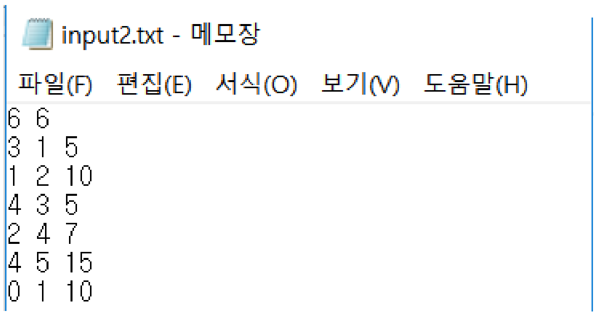


```
4 5
3 5
0 1
1 4
0 2
```



```
Visit node 0
Visit node 2
Visit node 4
Visit node 3
Visit node 5
Visit node 1
component plus! 1
total component : 1
Connected graph!
Node 0 : 2 (Weight 20) 1 (Weight 10)
Node 1 : 4 (Weight 10) 3 (Weight 50) 0 (Weight 10)
Node 2 : 4 (Weight 33) 3 (Weight 20) 0 (Weight 20)
Node 3 : 5 (Weight 2) 4 (Weight 20) 2 (Weight 20) 1 (Weight 50)
Node 4 : 2 (Weight 33) 3 (Weight 20) 5 (Weight 1) 1 (Weight 10)
Node 5 : 3 (Weight 2) 4 (Weight 1)
Find a minimum cost spanning tree using Kruskal's algorithm
Node 0 : 2 (Weight 20) 1 (Weight 10)
Node 1 : 4 (Weight 10) 3 (Weight 50)
Node 2 : 4 (Weight 33) 3 (Weight 20)
Node 3 : 5 (Weight 2) 4 (Weight 20)
Node 4 : 5 (Weight 1)
Node 5 :
Sorted edge :
node 4 - node 5 weight 1
node 3 - node 5 weight 2
node 0 - node 1 weight 10
node 1 - node 4 weight 10
node 3 - node 4 weight 20
node 0 - node 2 weight 20
node 2 - node 4 weight 33
node 2 - node 3 weight 20
node 1 - node 3 weight 50
weight 1 between node 4 and node 5 is summed
weight 2 between node 3 and node 5 is summed
weight 10 between node 0 and node 1 is summed
weight 10 between node 1 and node 4 is summed
Cycle due to edge between node 1 and node 4 so do not recept
weight 20 between node 0 and node 2 is summed
Cycle due to edge between node 0 and node 2 so do not recept
Cycle due to edge between node 0 and node 2 so do not recept
Cycle due to edge between node 0 and node 2 so do not recept
total weight = 43
```

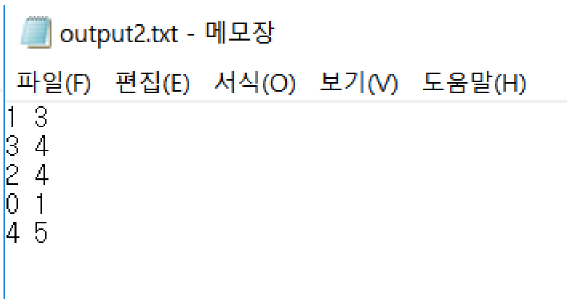
2). Second test



input2.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

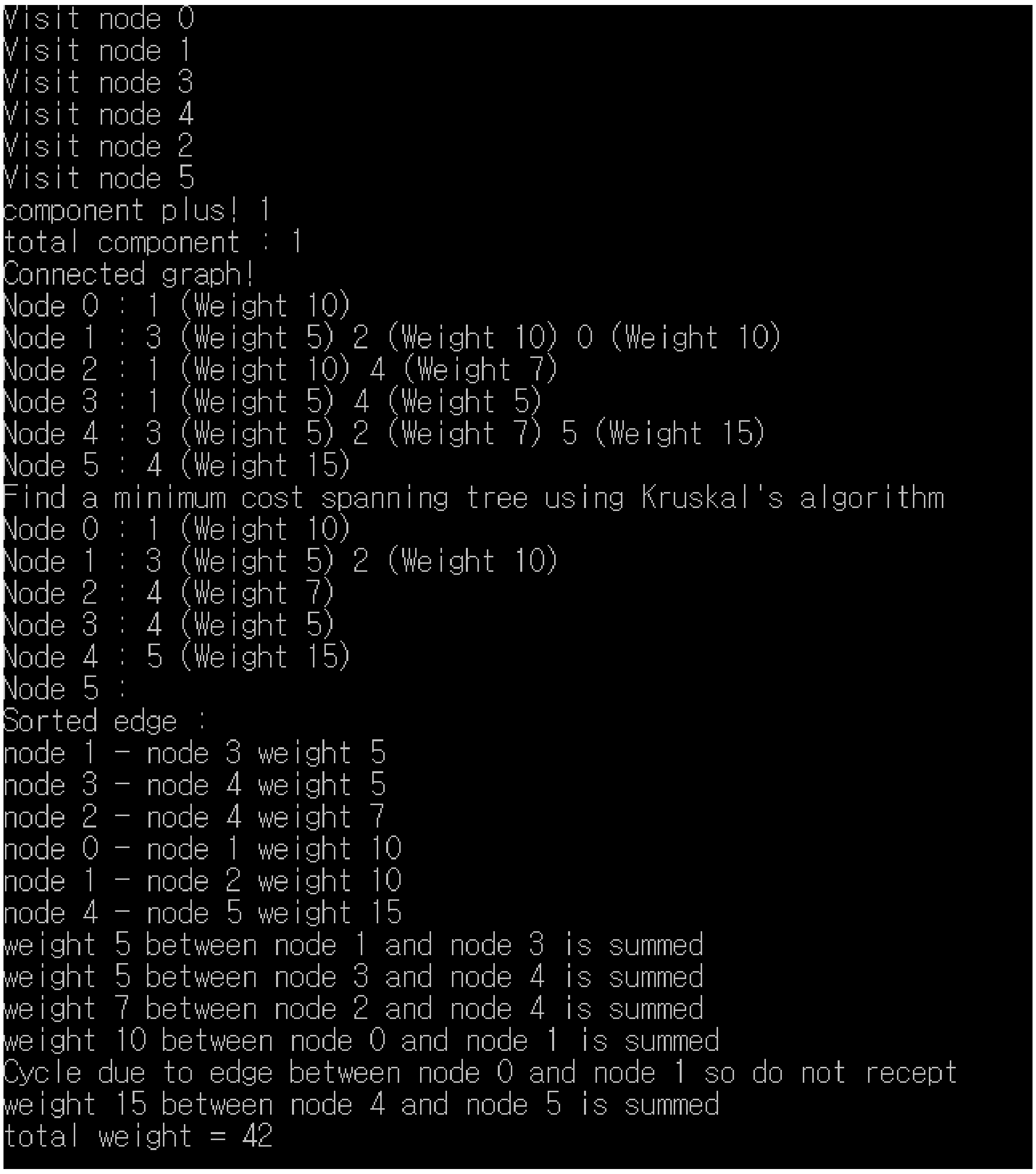
```
6 6
3 1 5
1 2 10
4 3 5
2 4 7
4 5 15
0 1 10
```



output2.txt - 메모장

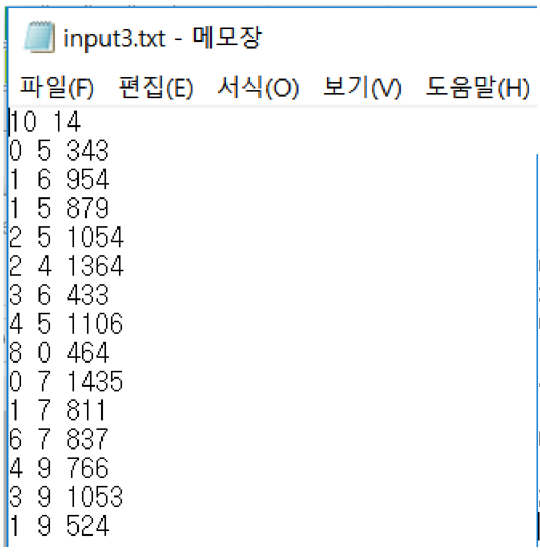
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
1 3
3 4
2 4
0 1
4 5
```

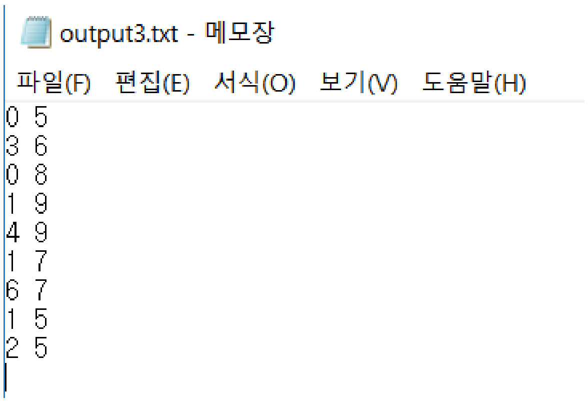


```
Visit node 0
Visit node 1
Visit node 3
Visit node 4
Visit node 2
Visit node 5
component plus! 1
total component : 1
Connected graph!
Node 0 : 1 (Weight 10)
Node 1 : 3 (Weight 5) 2 (Weight 10) 0 (Weight 10)
Node 2 : 1 (Weight 10) 4 (Weight 7)
Node 3 : 1 (Weight 5) 4 (Weight 5)
Node 4 : 3 (Weight 5) 2 (Weight 7) 5 (Weight 15)
Node 5 : 4 (Weight 15)
Find a minimum cost spanning tree using Kruskal's algorithm
Node 0 : 1 (Weight 10)
Node 1 : 3 (Weight 5) 2 (Weight 10)
Node 2 : 4 (Weight 7)
Node 3 : 4 (Weight 5)
Node 4 : 5 (Weight 15)
Node 5 :
Sorted edge :
node 1 - node 3 weight 5
node 3 - node 4 weight 5
node 2 - node 4 weight 7
node 0 - node 1 weight 10
node 1 - node 2 weight 10
node 4 - node 5 weight 15
weight 5 between node 1 and node 3 is summed
weight 5 between node 3 and node 4 is summed
weight 7 between node 2 and node 4 is summed
weight 10 between node 0 and node 1 is summed
Cycle due to edge between node 0 and node 1 so do not recept
weight 15 between node 4 and node 5 is summed
total weight = 42
```

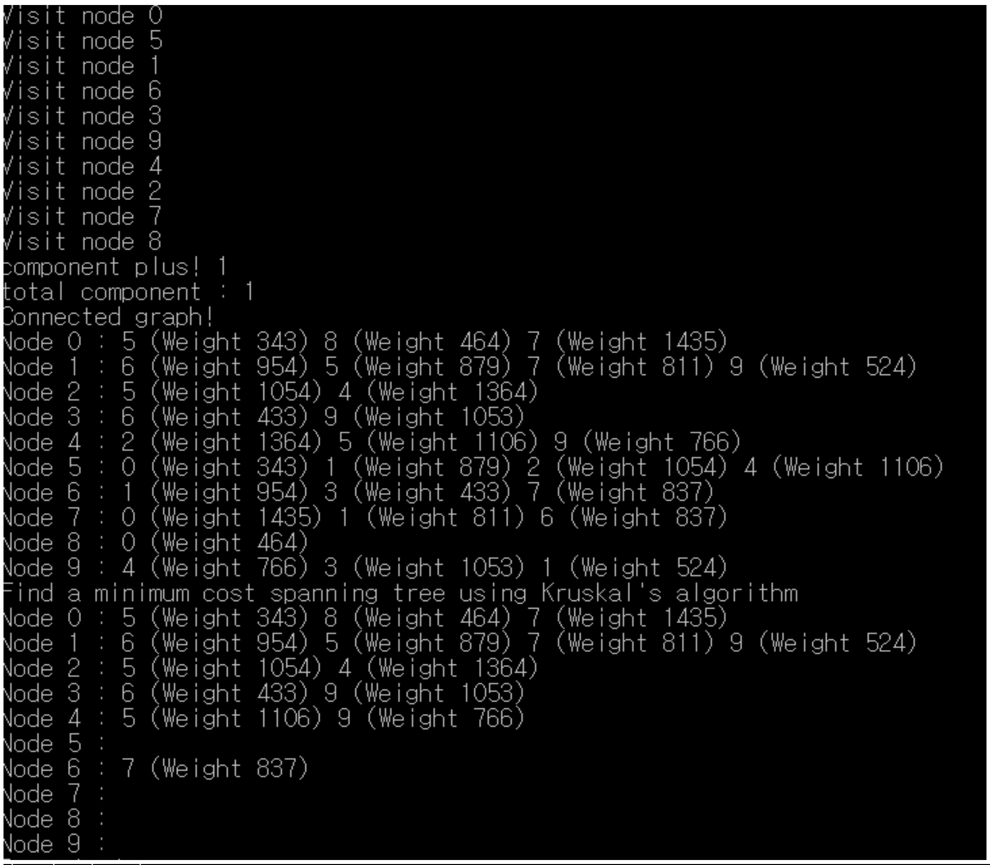
3). Third test



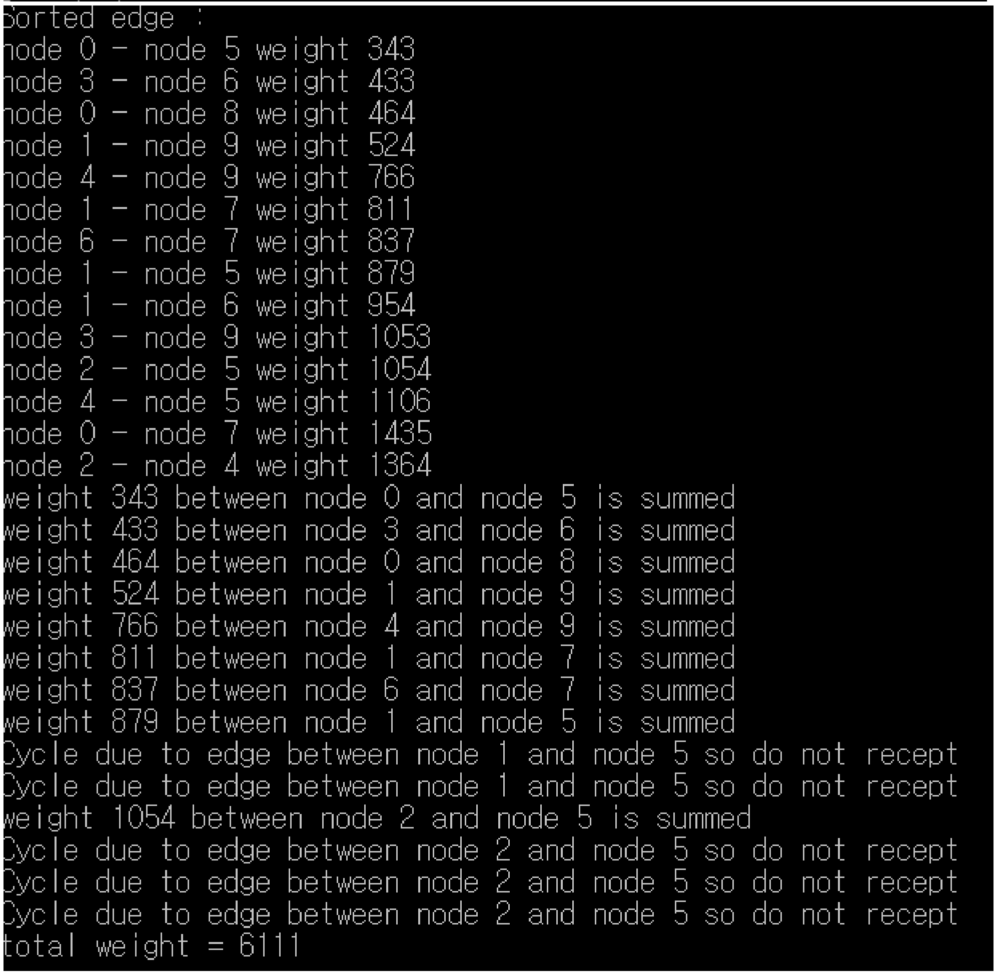
```
10 14
0 5 343
1 6 954
1 5 879
2 5 1054
2 4 1364
3 6 433
4 5 1106
8 0 464
0 7 1435
1 7 811
6 7 837
4 9 766
3 9 1053
1 9 524
```



```
0 5
3 6
0 8
1 9
4 9
1 7
6 7
1 5
2 5
```



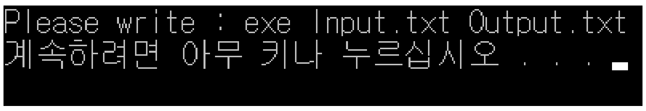
```
Visit node 0
Visit node 5
Visit node 1
Visit node 6
Visit node 3
Visit node 9
Visit node 4
Visit node 2
Visit node 7
Visit node 8
component plus! 1
total component : 1
Connected graph!
Node 0 : 5 (Weight 343) 8 (Weight 464) 7 (Weight 1435)
Node 1 : 6 (Weight 954) 5 (Weight 879) 7 (Weight 811) 9 (Weight 524)
Node 2 : 5 (Weight 1054) 4 (Weight 1364)
Node 3 : 6 (Weight 433) 9 (Weight 1053)
Node 4 : 2 (Weight 1364) 5 (Weight 1106) 9 (Weight 766)
Node 5 : 0 (Weight 343) 1 (Weight 879) 2 (Weight 1054) 4 (Weight 1106)
Node 6 : 1 (Weight 954) 3 (Weight 433) 7 (Weight 837)
Node 7 : 0 (Weight 1435) 1 (Weight 811) 6 (Weight 837)
Node 8 : 0 (Weight 464)
Node 9 : 4 (Weight 766) 3 (Weight 1053) 1 (Weight 524)
Find a minimum cost spanning tree using Kruskal's algorithm
Node 0 : 5 (Weight 343) 8 (Weight 464) 7 (Weight 1435)
Node 1 : 6 (Weight 954) 5 (Weight 879) 7 (Weight 811) 9 (Weight 524)
Node 2 : 5 (Weight 1054) 4 (Weight 1364)
Node 3 : 6 (Weight 433) 9 (Weight 1053)
Node 4 : 5 (Weight 1106) 9 (Weight 766)
Node 5 :
Node 6 : 7 (Weight 837)
Node 7 :
Node 8 :
Node 9 :
```



```
Sorted edge :
node 0 - node 5 weight 343
node 3 - node 6 weight 433
node 0 - node 8 weight 464
node 1 - node 9 weight 524
node 4 - node 9 weight 766
node 1 - node 7 weight 811
node 6 - node 7 weight 837
node 1 - node 5 weight 879
node 1 - node 6 weight 954
node 3 - node 9 weight 1053
node 2 - node 5 weight 1054
node 4 - node 5 weight 1106
node 0 - node 7 weight 1435
node 2 - node 4 weight 1364
weight 343 between node 0 and node 5 is summed
weight 433 between node 3 and node 6 is summed
weight 464 between node 0 and node 8 is summed
weight 524 between node 1 and node 9 is summed
weight 766 between node 4 and node 9 is summed
weight 811 between node 1 and node 7 is summed
weight 837 between node 6 and node 7 is summed
weight 879 between node 1 and node 5 is summed
Cycle due to edge between node 1 and node 5 so do not recept
Cycle due to edge between node 1 and node 5 so do not recept
weight 1054 between node 2 and node 5 is summed
Cycle due to edge between node 2 and node 5 so do not recept
Cycle due to edge between node 2 and node 5 so do not recept
Cycle due to edge between node 2 and node 5 so do not recept
total weight = 6111
```

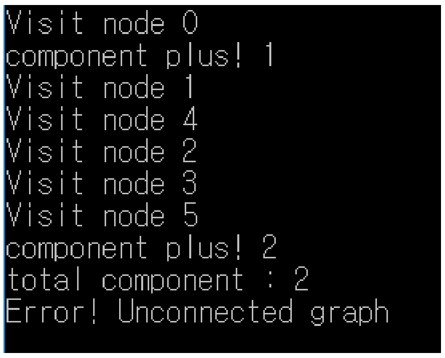
4). Exception cases

(1). Input.txt or Output.txt fopen failed



```
Please write : exe Input.txt Output.txt
계속하려면 아무 키나 누르십시오 . . .
```

(2). Input graph is unconnected



```
Visit node 0
component plus! 1
Visit node 1
Visit node 4
Visit node 2
Visit node 3
Visit node 5
component plus! 2
total component : 2
Error! Unconnected graph
```