데이터구조 설계

DS_project1

컴퓨터정보공학부

2020202031

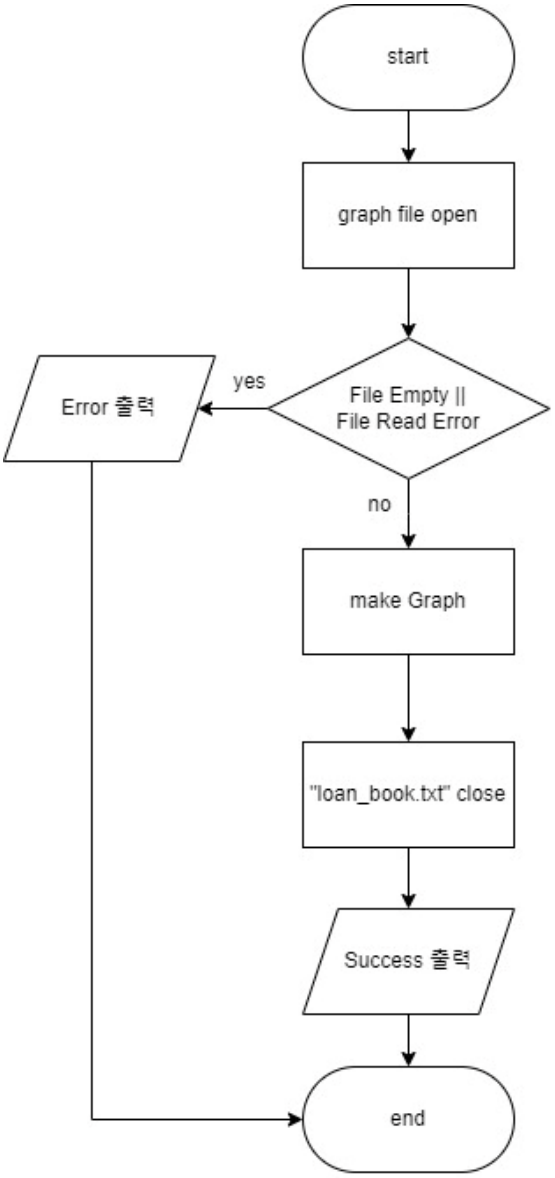
김재현

1. Introduction

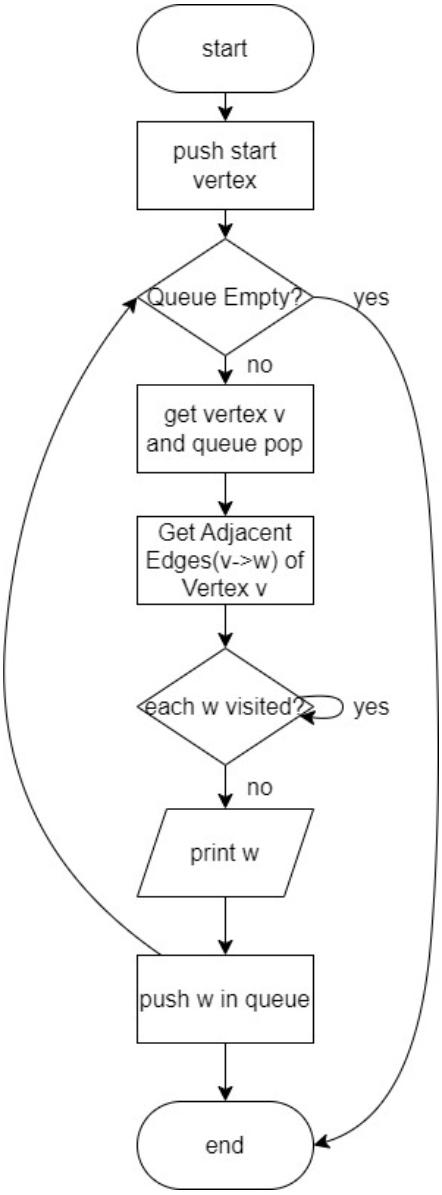
데이터구조설계 시간에 다양한 다양한 MST 찾기, 최소경로 결 찾기를 배웠습니다. 이번 프로젝트에서는 그래프를 이용해 그래프 연산 프로그램을 구현해 봄으로써 배운 내용들을 복습합니다. 이 프로그램은 List or Matrix로 구성된 그래프 정보가 저장된 텍스트 파일을 통해 그래프를 구현하고, 그래프의 특성에 따라 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford, FLOYD, KwangWoon 총 7 가지 알고리즘을 통해 그래프 연산을 수행합니다. 주어지는 그래프 데이터는 기본적으로 방향성(Direction)과 가중치(Weight)를 모두 가지고 있습니다. 하지만 그래프 탐색 알고리즘 마다 방향성을 고려해야 하기도 하고 고려하지 않아도 된다는 조건이 존재하므로, 무방향성일 때와 방향성일 때의 인접노드를 반환하는 함수를 각각 하나씩 작성합니다.

2. Flowchart

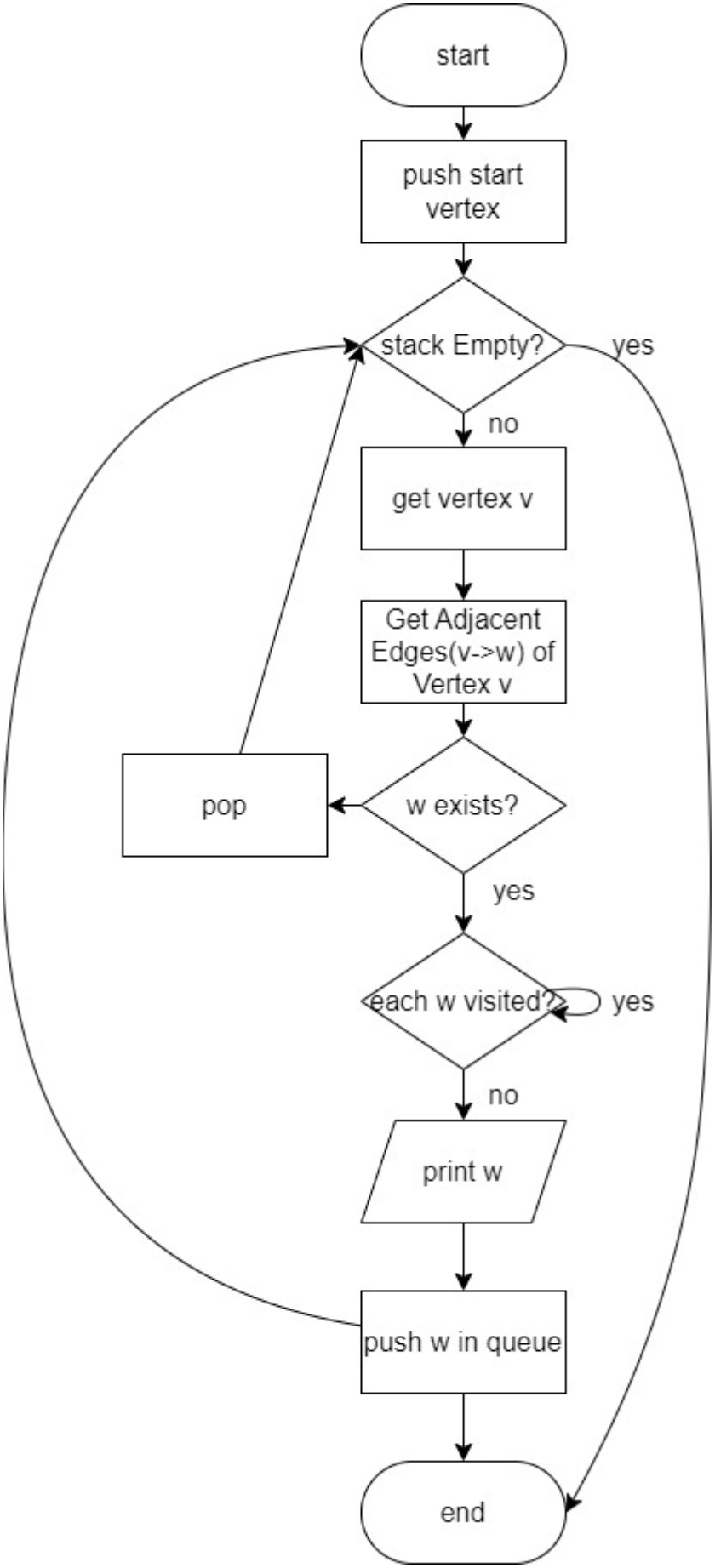
1) Load


start
graph file open
File Empty ||
File Read Error
yes
Error 출력
no
make Graph
"loan_book.txt" close
Success 출력
end

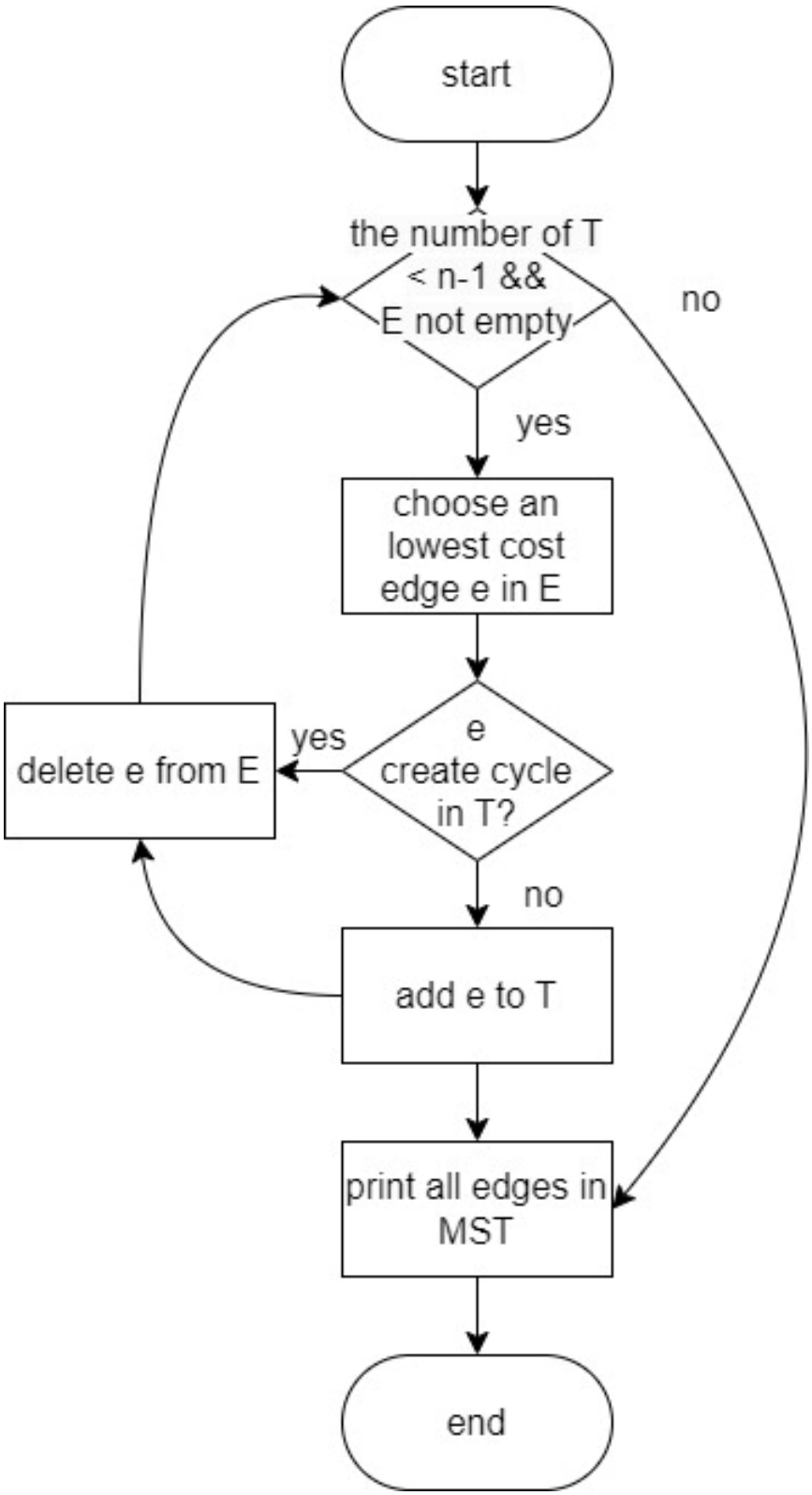
2) BFS


start
push start vertex
Queue Empty?
yes
no
get vertex v and queue pop
Get Adjacent Edges(v->w) of Vertex v
each w visited?
yes
no
print w
push w in queue
end

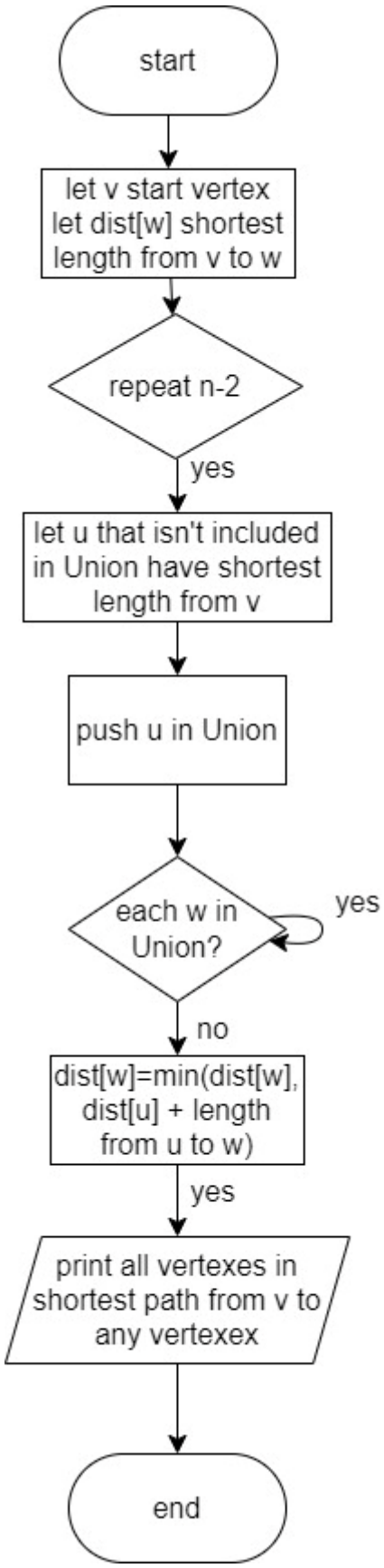
3) DFS


start
push start vertex
stack Empty?
yes
no
get vertex v
Get Adjacent Edges(v->w) of Vertex v
w exists?
pop
yes
each w visited?
yes
no
print w
push w in queue
end

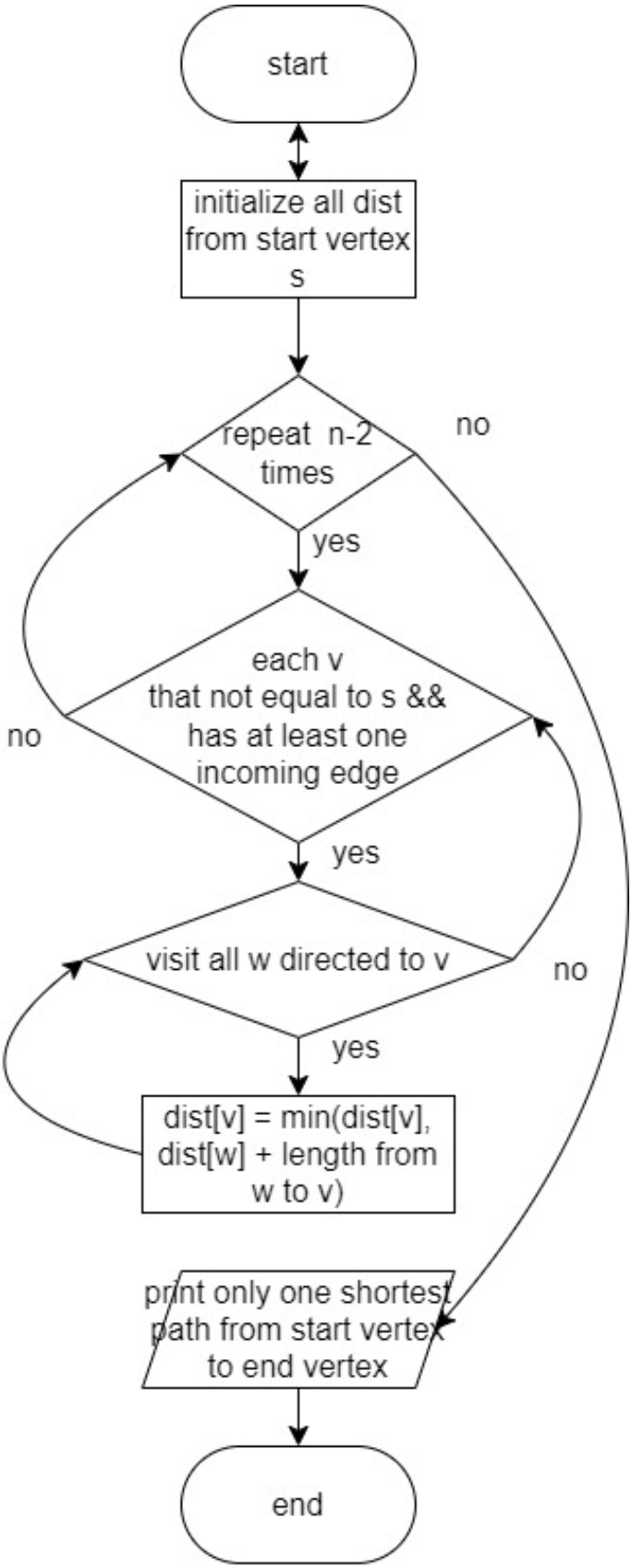
4) KRUSKAL


start
the number of T
< n-1 &&
E not empty
no
yes
choose an
lowest cost
edge e in E
e
create cycle
in T?
yes
delete e from E
no
add e to T
print all edges in
MST
end

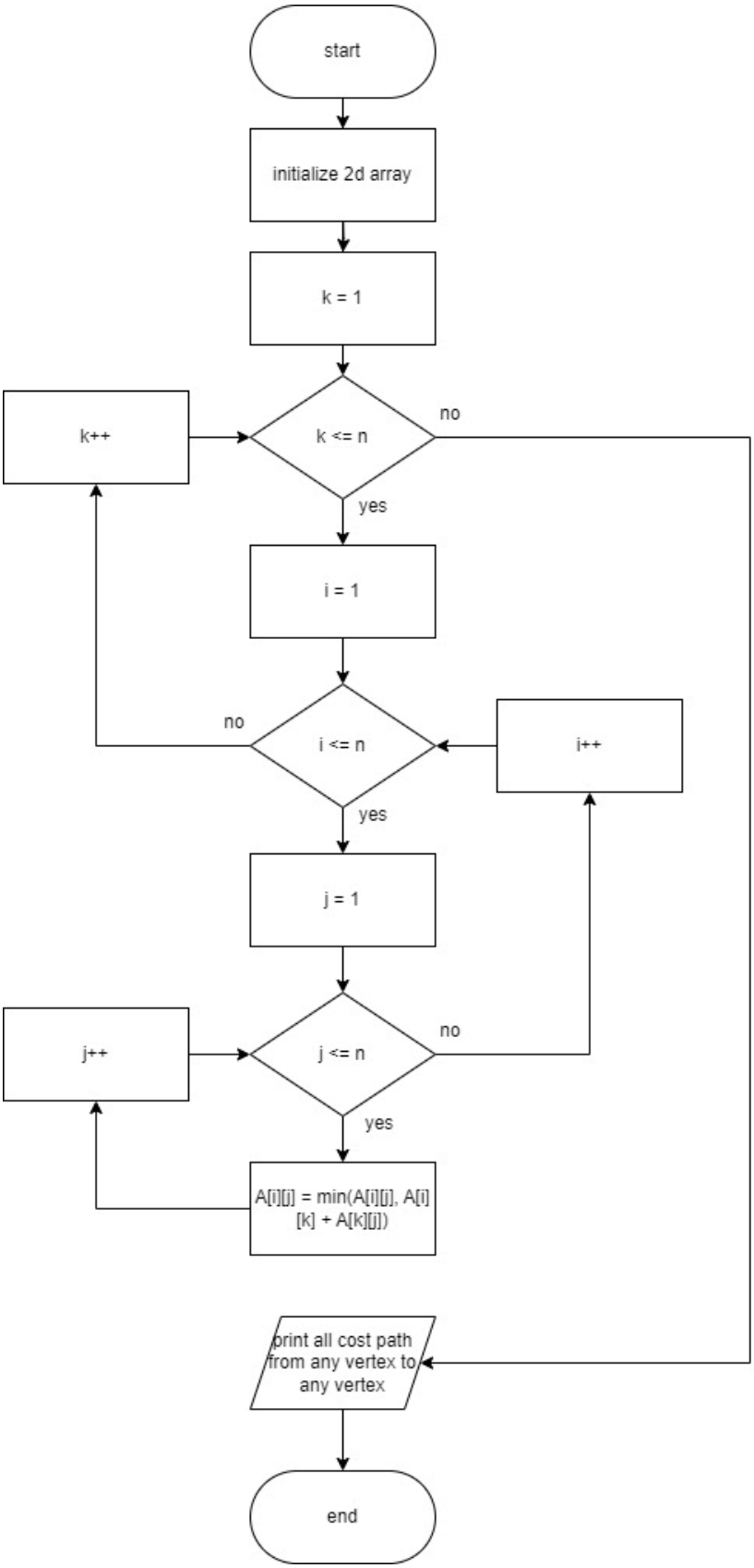
5) DIJKSTRA


start
let v start vertex
let dist[w] shortest
length from v to w
repeat n-2
yes
let u that isn't included
in Union have shortest
length from v
push u in Union
each w in
Union?
yes
no
dist[w]=min(dist[w],
dist[u] + length
from u to w)
yes
print all vertexes in
shortest path from v to
any vertexex
end

6) BELLMAN-FORD



7) FLOYD


start
initialize 2d array
k = 1
k <= n
no
yes
k++
i = 1
i <= n
no
i++
yes
j = 1
j <= n
no
yes
j++
A[i][j] = min(A[i][j], A[i][k] + A[k][j])
print all cost path from any vertex to any vertex
end

3. Algorithm

1) BFS

BFS는 목표로 하는 정점을 찾기 위한 graph search algorithm입니다. 하지만 이번 프로젝트에서 BFS는 어떠한 특정한 정점을 찾는 기능을 수행하기보다, BFS가 어떤 순서로 그래프를 탐색하는지를 확인하는 목표로 구현합니다.

BFS는 queue 자료구조를 통해 구현할 수 있습니다. 먼저 initialization을 위해 출발정점을 queue에 push해줍니다.

현재정점은 queue의 front에 존재하는 정점입니다. 현재정점을 queue에서 pop한 후, 현재정점의 인접정점들 중 방문하지 않은 정점들을 정점번호가 작은 순서대로 queue에 push해줍니다. push함과 동시에 출력해주면, BFS 알고리즘이 방문하는 정점 순서대로 출력할 수 있습니다. 만약 한 번도 방문하지 않은 인접정점이 없다면, queue의 front에 존재하는 다음 정점으로 넘어갑니다.

이 과정을 queue가 empty가 될 때까지 반복하면 BFS 방식으로 graph search를 완수할 수 있습니다.

2) DFS

DFS는 BFS와 같이 목표로 하는 정점을 찾기 위한 graph search algorithm입니다. 하지만 이번 프로젝트에서 BFS는 어떠한 특정한 정점을 찾는 기능을 수행하기보다, DFS가 어떤 순서로 그래프를 탐색하는지를 확인하는 목표로 구현합니다.

DFS는 재귀적으로 구현할 수 있습니다. 재귀적으로 구현하기 위해 이번 프로젝트에서는 stack 자료구조를 사용했습니다. 먼저 initialization을 위해 출발정점을 stack에 push해줍니다.

현재정점은 stack의 top에 존재하는 정점입니다. 현재정점의 인접정점들 중에서 한 번도 방문하지 않는 정점들 중에서도 가장 작은 번호를 가지는

정점을 stack에 push해줍니다 정점을 push함과 동시에 출력해주면, DFS알고리즘이 방문하는 정점 순서대로 출력할 수 있습니다.. 만약 한 번도 방문하지 않은 인접정점이 없다면, stack을 pop해줍니다.
이 과정을 stack이 empty가 될 때까지 반복하면 DFS 방식으로 graph search를 완수할 수 있습니다.

3) KRUSKAL

KRUSKAL은 minimum spanning tree(MST)를 구하는 알고리즘입니다.
KRUSKAL 알고리즘의 경우 weight가 적은 edge들을 이어나가는 알고리즘이기 때문에, edge에 대한 정보를 저장시키는 것이 관건이었습니다. 따라서 출발정점, 도착정점, weight 정보를 한 번에 저장하여 edge정보를 나타내기 위해, <pair<int, pair<int, int> > 구조로 weight, start vertex, end vertex의 정보를 E라는 벡터에 담았습니다. MST의 정보는 map으로 구성된 배열 T에 저장했습니다. T에 저장된 edges의 개수는 T에 edge가 저장될 때마다 count라는 변수를 1씩 더해줌으로써 파악할 수 있도록 설계했습니다. 또한 E에 포함된 edge가 T와 cycle을 구성하는지에 대한 여부는 Union이라는 class 객체의 isUnion이라는 메소드를 통해 판별할 수 있습니다
.E에서 weight가 가장 적은 edge e를 선택하고 E에서 삭제합니다. 만약 e가 T와 cycle을 이루지 않는다면 e를 T에 저장합니다. 만약 e가 T와 cycle을 그대로 반복문으로 돌아갑니다.
T의 edges 개수가 n-1보다 작고 E가 비어 있지 않다면 계속해서 반복합니다.
MST가 완성된 후 T 배열의 n번째 인덱스에는 n번 정점과 인접해있는 정점들이 번호순서대로 저장돼있으므로, T[1]~T[n]까지를 출력하면 MST가 어떤 식으로 구성이 됐는지 파악할 수 있다.

4) DIJKSTRA

DIJKSTRA는 양의 가중치를 가지는 그래프에서 특정 정점으로부터 다른 정점들까지의 최단경로를 구하는 알고리즘입니다.

DIJKSTRA를 구현하기 위해 먼저 출발정점에 인접한 정점들의 거리를 dist에 저장해주고, 인접하지 않은 정점들의 거리는 무한대라고 가정할 수 있을만큼 큰 값인 99999999를 저장해줍니다.
출발정점과 출발정점에서 가장 가까운 dist를 가진 정점을 제외하고 나머지 정점들의 dist만을 구해주면 되므로 다음의 과정을 n-2번 반복합니다. Union에 포함되지 않은 vertex중에서 출발정점으로부터의 거리 dist가 가장 가까운 vertex를 u라고 하자. w를 Union에 포함되지 않은 정점들이라고 하자. u를 Union에 포함시키고 w에 대해, w의 현재 거리인 dist[w]와 u를 거친 거리 dist[u]+length from u to w의 길이 중 더 작은 값으로 dist[w]를 업데이트시킵니다.

다익스트라 알고리즘이 음수 가중치를 가지는 그래프에서는 사용할 수 없는 이유는, 다익스트라는 각 정점에 대해 최소경로를 구했다고 판단하면 더 이상 최소경로를 업데이트 시키지 않기 때문에, 음수 가중치로 인해 보다 더 최적인 최소경로를 발견해도 업데이트 시키지 않습니다. 따라서 음수 가중치를 가지는 그래프에서 다익스트라 알고리즘은 최적의 경로를 구해준다고 보기 어렵습니다.

5) BELLMAN-FORD

BELLMAN-FORD는 양, 음의 가중치에 상관없이, 그래프에서 특정 정점으로부터 다른 정점들까지의 최단경로를 구하는 알고리즘입니다 DIJKSTRA의 경우 각 정점들의 최적의 경로를 구했다고 판단하면 그 후에 보다 나은 최적의 경로가 발견돼도 업데이트 시켜주지 않기 때문에 음의 가중치를 가지는 그래프에서 유효하지 않았지만, BELLMAN-FORD의 경우 매 단계마다 모든 정점의 최적의 경로를 업데이트 해주기 때문에 음의 가중치를 가지는 그래프에서도 유효합니다. 하지만 음의 사이클을 가지게 될 경우, 최소경로가 무한히 업데이트 되므로 최소경로를 구할 수가 없습니다. 따라서 BELLMAN-FORD 알고리즘은 음의 가중치에서는 유효하지만 음의 사이클에서는 유효하지 않은 알고리즘이라고 할 수 있겠습니다.

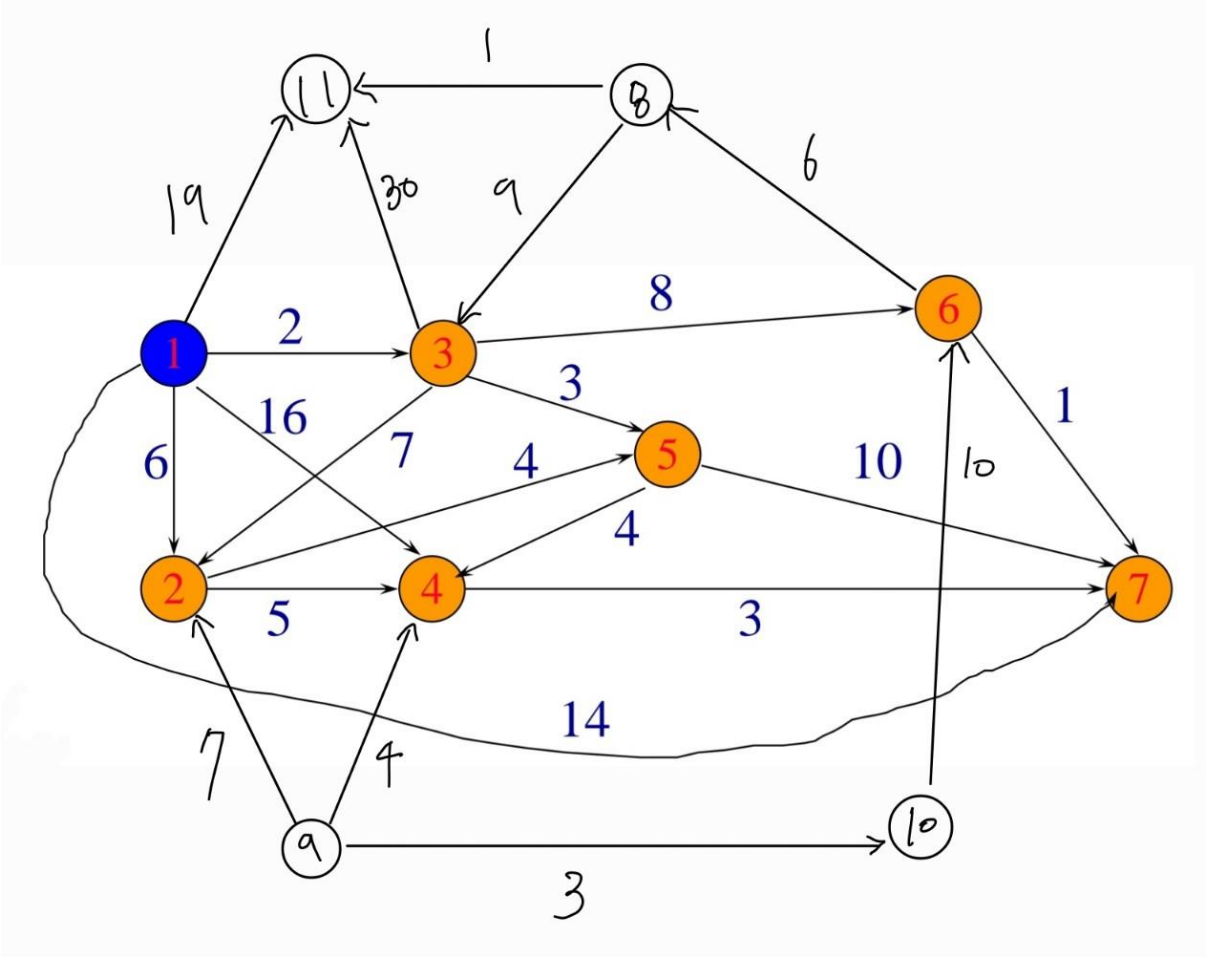
BELLMAN-FORD를 구현하기 위해 먼저 출발정점에 인접한 정점들의 거리를 dist에 저장해주고 prev에는 출발정점 번호를 저장해줍니다, 인접하지 않은 정점들의 거리는 무한대라고 가정할 수 있을만큼 큰 값인 99999999를 저장해줍니다. prev에는 존재하지 않는 정점 0을 저장해줍니다.
k는 edge사용개수의 제한을 의미하므로, initialization 단계에선 k가 1이었으므로 본격적인 알고리즘 시작단계에서는 k=2부터 시작하여 n-1까지 다음의 과정을 반복합니다.
출발정점이 아닌 모든 정점 v에 대해 v의 incoming edge ei=<w, v>에 대해 dist[v]와 dist[w] + length from w to v 중 더 짧은 거리로 dist[v]를 업데이트 시켜줍니다.
BELLMAN-FORD 알고리즘의 경우 위에서도 언급했듯이, 음의 사이클을 가질 수 없으므로, 음의 사이클을 판별해줘야 합니다. 음의 사이클을 판별하는 방법으로 알고리즘 연산이 끝난 후, 한 사이클을 추가로 더 실행시켜서, 최소경로가 업데이트되는 정점이 있다면 음의 사이클이 존재하는 것으로 판단하고 오류를 출력합니다.

6) FLOYD

FLOYD는 거쳐갈 수 있는 정점을 제한함으로써 최적의 경로를 차례로 찾아가는 알고리즘입니다. k가 정점의 제한을 나타내는 번호인데, k=0 일 땐, 단순히 인접정점들과의 거리로 A를 초기화합니다. 인접하지 않은 정점들 사이의 거리는 무한으로 가정할 수 있을 만큼 큰 값인 99999999으로 초기화합니다.
k가 정점을 제한하는 번호라면, i는 출발정점의 번호, j는 도착정점의 번호를 나타냅니다. 따라서 k의 모든 반복에서 모든 length from i to j를 계속해서 업데이트해줍니다. 그러므로 k, I, j에 대한 삼중 반복문을 사용하여 모든 k의 값에 대해 모든 경로의 최적의 경로를 업데이트 시켜줍니다.
초기화 단계에서 자기자신으로 향하는 경로는 0으로 초기화하고 시작합니다. 하지만 음수 사이클이 존재한다면 자기자신으로 향하는 경로가 음수가 될 수 있으므로 음수 사이클이 존재하는 그래프에서 FLOYD 알고리즘은 유효하지 않습니다. 따라서 음수 사이클이 존재하는지를 판별해주는 작업

이 필요합니다. 음수 사이클을 판별하는 방법으로는, FLOYD연산이 끝난 후 모든 대각행렬을 탐방하며 0이 아닌 정점이 존재하는지를 확인하는 것입니다. 음수 사이클이 존재한다면 적어도 하나의 정점은 자기자신으로 돌아가는 경로가 음수일 것이므로 이를 기준으로 음수 사이클을 판별하여 음수 사이클이 존재한다면 오류를 출력합니다.

4. Result Screen



첫번째로 다음과 같은 양의 가중치만을 가지는 그래프로 실험을 해봤습니다.

Outgoing edges만을 가지는 정점은 1, 9

Incoming edges만을 가지는 정점은 7, 11입니다. 무방향 그래프일 때는 의미가 없지만, 방향 그래프에서는 특별한 특징이라고 할 수 있기에 command에

넣어서 실험했습니다. graph_L, graph_M 텍스트파일은 위의 그래프의 정보를 전달하도록 구성했습니다.

1) BFS

```
========LOAD=======
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
====================

======== BFS ========
Directed Graph BFS result
startvertex: 1
1->2->3->4->7->11->5->6->8
====================

======== BFS ========
Undirected Graph BFS result
startvertex: 1
1->2->3->4->7->11->5->9->6->8->10
====================

======== BFS ========
Directed Graph BFS result
startvertex: 7
7
====================

======== BFS ========
Undirected Graph BFS result
startvertex: 7
7->1->4->5->6->2->3->11->9->8->10
====================
```

```
========LOAD=======
Success
====================

======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
=====================

======== BFS ========
Directed Graph BFS result
startvertex: 1
1->2->3->4->7->11->5->6->8
=====================

======== BFS ========
Undirected Graph BFS result
startvertex: 1
1->2->3->4->7->11->5->9->6->8->10
=====================

======== BFS ========
Directed Graph BFS result
startvertex: 7
7
=====================

======== BFS ========
Undirected Graph BFS result
startvertex: 7
7->1->4->5->6->2->3->11->9->8->10
=====================
```

1에서 출발하는 경우와 7에서 출발하는 경우 다 올바른 결과가 출력됩니다. 7번 정점의 경우 Incoming edge만이 존재하므로 Directed Graph에서

인접정점으로의 이동이 불가능하므로 오직 자기자신만을 출력하는 것을 확인할 수 있습니다.

2) DFS

```
========LOAD========
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)-> (11,30)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
====================

======== DFS ========
Directed Graph DFS result
startvertex: 1
1->2->4->7->5->3->6->8->11
====================

======== DFS ========
Undirected Graph DFS result
startvertex: 1
1->2->3->5->4->7->6->8->11->10->9
====================

======== DFS ========
Directed Graph DFS result
startvertex: 7
7
====================

======== DFS ========
Undirected Graph DFS result
startvertex: 7
7->1->2->3->5->4->9->10->6->8->11
====================
```

```
========LOAD=======
Success
====================

======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
======================

======== DFS ========
Directed Graph DFS result
startvertex: 1
1->2->4->7->5->3->6->8->11
======================

======== DFS ========
Undirected Graph DFS result
startvertex: 1
1->2->3->5->4->7->6->8->11->10->9
======================

======== DFS ========
Directed Graph DFS result
startvertex: 7
7
======================

======== DFS ========
Undirected Graph DFS result
startvertex: 7
7->1->2->3->5->4->9->10->6->8->11
======================
```

DFS 역시 올바른 결과가 잘 출력되며, 7의 경우 Incoming edges만이 존재하므로 자기자신만을 출력하는 것을 확인할 수 있습니다.

3) KRUSKAL

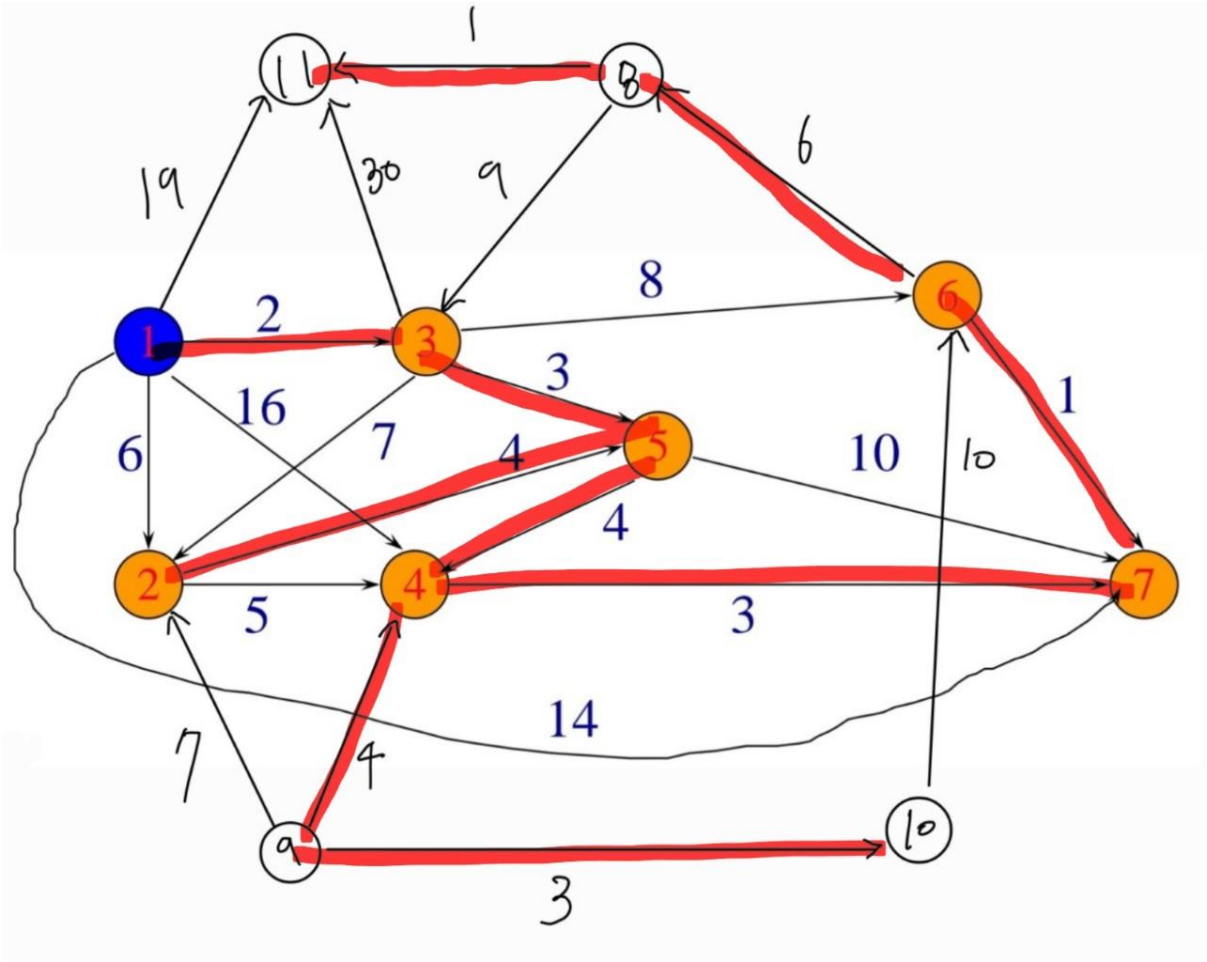
```
========LOAD=======
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)-> (11,30)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
=====================

====== Kruskal =======
[1]     3(2)
[2]     5(4)
[3]     1(2)5(3)
[4]     5(4)7(3)9(4)
[5]     2(4)3(3)4(4)
[6]     7(1)8(6)
[7]     4(3)6(1)
[8]     6(6)11(1)
[9]     4(4)10(3)
[10]    9(3)
[11]    8(1)
cost: 31
=====================
========LOAD=======
Success
====================

======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
=====================

====== Kruskal =======
[1]     3(2)
[2]     5(4)
[3]     1(2)5(3)
[4]     5(4)7(3)9(4)
[5]     2(4)3(3)4(4)
[6]     7(1)8(6)
[7]     4(3)6(1)
[8]     6(6)11(1)
[9]     4(4)10(3)
[10]    9(3)
[11]    8(1)
cost: 31
=====================
```



이와 같은 올바른 결과가 출력됨을 확인할 수 있다.

4) DIJKSTRA

```
========LOAD=======
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)-> (11,30)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
====================

====== Dijkstra =======
Directed Graph Dijkstra result
startvertex: 1
[2] 1 -> 2(6)
[3] 1 -> 3(2)
[4] 1 -> 3 -> 5 -> 4(9)
[5] 1 -> 3 -> 5(5)
[6] 1 -> 3 -> 6(10)
[7] 1 -> 3 -> 6 -> 7(11)
[8] 1 -> 3 -> 6 -> 8(16)
[9] x
[10] x
[11] 1 -> 3 -> 6 -> 8 -> 11(17)
====================

====== Dijkstra =======
Undirected Graph Dijkstra result
startvertex: 1
[2] 1 -> 2(6)
[3] 1 -> 3(2)
[4] 1 -> 3 -> 5 -> 4(9)
[5] 1 -> 3 -> 5(5)
[6] 1 -> 3 -> 6(10)
[7] 1 -> 3 -> 6 -> 7(11)
[8] 1 -> 3 -> 8(11)
[9] 1 -> 2 -> 9(13)
[10] 1 -> 2 -> 9 -> 10(16)
[11] 1 -> 3 -> 8 -> 11(12)
====================

====== Dijkstra =======
Directed Graph Dijkstra result
startvertex: 11
[1] x
[2] x
[3] x
[4] x
[5] x
[6] x
[7] x
[8] x
[9] x
[10] x
====================

====== Dijkstra =======
Undirected Graph Dijkstra result
startvertex: 11
[1] 11 -> 8 -> 3 -> 1(12)
[2] 11 -> 8 -> 6 -> 7 -> 4 -> 2(16)
[3] 11 -> 8 -> 3(10)
[4] 11 -> 8 -> 6 -> 7 -> 4(11)
[5] 11 -> 8 -> 3 -> 5(13)
[6] 11 -> 8 -> 6(7)
[7] 11 -> 8 -> 6 -> 7(8)
[8] 11 -> 8(1)
[9] 11 -> 8 -> 6 -> 7 -> 4 -> 9(15)
[10] 11 -> 8 -> 6 -> 10(17)
====================
```

```
========LOAD=======
Success
====================

======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
====================

====== Dijkstra =======
Directed Graph Dijkstra result
startvertex: 1
[2] 1 -> 2(6)
[3] 1 -> 3(2)
[4] 1 -> 3 -> 5 -> 4(9)
[5] 1 -> 3 -> 5(5)
[6] 1 -> 3 -> 6(10)
[7] 1 -> 3 -> 6 -> 7(11)
[8] 1 -> 3 -> 6 -> 8(16)
[9] x
[10] x
[11] 1 -> 3 -> 6 -> 8 -> 11(17)
====================

====== Dijkstra =======
Undirected Graph Dijkstra result
startvertex: 1
[2] 1 -> 2(6)
[3] 1 -> 3(2)
[4] 1 -> 3 -> 5 -> 4(9)
[5] 1 -> 3 -> 5(5)
[6] 1 -> 3 -> 6(10)
[7] 1 -> 3 -> 6 -> 7(11)
[8] 1 -> 3 -> 8(11)
[9] 1 -> 2 -> 9(13)
[10] 1 -> 2 -> 9 -> 10(16)
[11] 1 -> 3 -> 8 -> 11(12)
====================

====== Dijkstra =======
Directed Graph Dijkstra result
startvertex: 11
[1] x
[2] x
[3] x
[4] x
[5] x
[6] x
[7] x
[8] x
[9] x
[10] x
====================

====== Dijkstra =======
Undirected Graph Dijkstra result
startvertex: 11
[1] 11 -> 8 -> 3 -> 1(12)
[2] 11 -> 8 -> 6 -> 7 -> 4 -> 2(16)
[3] 11 -> 8 -> 3(10)
[4] 11 -> 8 -> 6 -> 7 -> 4(11)
[5] 11 -> 8 -> 3 -> 5(13)
[6] 11 -> 8 -> 6(7)
[7] 11 -> 8 -> 6 -> 7(8)
[8] 11 -> 8(1)
[9] 11 -> 8 -> 6 -> 7 -> 4 -> 9(15)
[10] 11 -> 8 -> 6 -> 10(17)
====================
```

1번에서 출발하는 다익스트라 Directed graph의 경우 9번 정점은

outgoing edge만이 존재하고 10번 정점은 9번으로부터 들어오는 edge만이 존재하므로 9, 10 정점에는 접근할 수 없다.

11번에서 출발하는 Directed graph의 경우 Incoming edge만이 존재하므로 그 어떠한 인접노드도 존재하지 않아 모든 경로가 x로 표시된다. 나머지는 올바른 결과가 출력됨을 확인할 수 있습니다.

5) BELLMAN-FORD

```
========LOAD=======
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)-> (11,30)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
1 -> 3 -> 6 -> 7
cost: 11
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
x
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
x
====================

====== Bellman-Ford =======
Undirected Graph Bellman-Ford result
7 -> 6 -> 8 -> 11
cost: 8
====================
```

```
========LOAD=======
Success
====================

======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
1 -> 3 -> 6 -> 7
cost: 11
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
x
====================

====== Bellman-Ford =======
Directed Graph Bellman-Ford result
x
====================

====== Bellman-Ford =======
Undirected Graph Bellman-Ford result
7 -> 6 -> 8 -> 11
cost: 8
====================
```

7 -> 11, 5 -> 9 의 경우, 출발정점이 incoming edge만을 가지고 있거나, 도착정점이 outgoing edge만을 가지고 있는 상황이므로 Directed graph에서는 경로를 찾지 못하고 x를 출력함을 확인할 수 있다.

6) FLOYD

```
========LOAD=======
Success
====================

======== PRINT========
[1]-> (2,6)-> (3,2)-> (4,16)-> (7,14)-> (11,19)
[2]-> (4,5)-> (5,4)
[3]-> (2,7)-> (5,3)-> (6,8)-> (11,30)
[4]-> (7,3)
[5]-> (4,4)-> (7,10)
[6]-> (7,1)-> (8,6)
[7]
[8]-> (3,9)-> (11,1)
[9]-> (2,7)-> (4,4)-> (10,3)
[10]-> (6,10)
[11]
====================

======== FLOYD ========
Directed Graph FLOYD result
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2   9   5  10  11  16   x   x  17
[2]  x   0   x   5   4   x   8   x   x   x   x
[3]  x   7   0   7   3   8   9  14   x   x  15
[4]  x   x   x   0   x   x   3   x   x   x   x
[5]  x   x   x   4   0   x   7   x   x   x   x
[6]  x  22  15  22  18   0   1   6   x   x   7
[7]  x   x   x   x   x   x   0   x   x   x   x
[8]  x  16   9  16  12  17  18   0   x   x   1
[9]  x   7  28   4  11  13   7  19   0   3  20
[10]  x  32  25  32  28  10  11  16   x   0  17
[11]  x   x   x   x   x   x   x   x   x   x   0
==========================

======== FLOYD ========
Undirected Graph FLOYD result
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2   9   5  10  11  11  13  16  12
[2]  6   0   7   5   4   9   8  15   7  10  16
[3]  2   7   0   7   3   8   9   9  11  14  10
[4]  9   5   7   0   4   4   3  10   4   7  11
[5]  5   4   3   4   0   8   7  12   8  11  13
[6] 10   9   8   4   8   0   1   6   8  10   7
[7] 11   8   9   3   7   1   0   7   7  10   8
[8] 11  15   9  10  12   6   7   0  14  16   1
[9] 13   7  11   4   8   8   7  14   0   3  15
[10] 16  10  14   7  11  10  10  16   3   0  17
[11] 12  16  10  11  13   7   8   1  15  17   0
==========================
```

```
======== PRINT========
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2  16   0   0  14   0   0   0  19
[2]  0   0   0   5   4   0   0   0   0   0   0
[3]  0   7   0   0   3   8   0   0   0   0   0
[4]  0   0   0   0   0   0   3   0   0   0   0
[5]  0   0   0   4   0   0  10   0   0   0   0
[6]  0   0   0   0   0   0   1   6   0   0   0
[7]  0   0   0   0   0   0   0   0   0   0   0
[8]  0   0   9   0   0   0   0   0   0   0   1
[9]  0   7   0   4   0   0   0   0   0   3   0
[10]  0   0   0   0   0  10   0   0   0   0   0
[11]  0   0   0   0   0   0   0   0   0   0   0
====================

======== FLOYD ========
Directed Graph FLOYD result
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2   9   5  10  11  16   x   x  17
[2]  x   0   x   5   4   x   8   x   x   x   x
[3]  x   7   0   7   3   8   9  14   x   x  15
[4]  x   x   x   0   x   x   3   x   x   x   x
[5]  x   x   x   4   0   x   7   x   x   x   x
[6]  x  22  15  22  18   0   1   6   x   x   7
[7]  x   x   x   x   x   x   0   x   x   x   x
[8]  x  16   9  16  12  17  18   0   x   x   1
[9]  x   7  28   4  11  13   7  19   0   3  20
[10]  x  32  25  32  28  10  11  16   x   0  17
[11]  x   x   x   x   x   x   x   x   x   x   0
==========================

======== FLOYD ========
Undirected Graph FLOYD result
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
[1]  0   6   2   9   5  10  11  11  13  16  12
[2]  6   0   7   5   4   9   8  15   7  10  16
[3]  2   7   0   7   3   8   9   9  11  14  10
[4]  9   5   7   0   4   4   3  10   4   7  11
[5]  5   4   3   4   0   8   7  12   8  11  13
[6] 10   9   8   4   8   0   1   6   8  10   7
[7] 11   8   9   3   7   1   0   7   7  10   8
[8] 11  15   9  10  12   6   7   0  14  16   1
[9] 13   7  11   4   8   8   7  14   0   3  15
[10] 16  10  14   7  11  10  10  16   3   0  17
[11] 12  16  10  11  13   7   8   1  15  17   0
==========================
```

올바른 결과가 출력됨을 확인할 수 있다.

5. Consideration

BELLMANFORD와 FLOYD 알고리즘의 경우 음의 가중치를 처리할 수 있기에 음의 사이클을 가지는 예외사항에 대해 고려해줘야 하는데, 음의 사이클이 존재하는지 판별하는 아이디어를 떠올리는데 많은 시간과 고통이 함께 했던 것 같습니다. 하지만 BELLMANFORD의 경우 음의 사이클을 가지면 최소경로가 무한히 업데이트된다는 사실을 통해 알고리즘이 끝난 후 한번 더 사이클을 실행시킴으로써 음의 사이클을 판별했고, FLOYD의 경우 음의 사이클을 가지

면 자기자신으로의 최소경로가 음수가 된다는 사실을 통해 알고리즘이 끝난 후 대각성분만을 탐방하며 음의 가중치가 있는지를 통해 음의 사이클을 판별할 수 있었습니다.