데이터구조 및 설계 3차 프로젝트 보고서

2019202050 이강현

1. Introduction

이번 프로젝트에서는 그래프를 이용한 그래프 연산 프로그램을 구현한다. 텍스트파일에 저장된 그래프정보를 이용하여 그래프를 형성하고 그래프 속 정보를 토대로 BFS, DFS, KRUSKAL, DIJKSTRA, BELLMANFORD, FLOYD 연산을 수행한다.

BFS는 너비우선탐색으로 queue를 활용하여 구현하고 DFS는 깊이우선탐색으로 stack을 활용하여 구현한다. DFS_R은 stack이 아니라 재귀적으로 구현한다. 앞의 3개의 알고리즘은 그래프의 방향성이나 가중치를 고려하지 않고 vertex들을 탐색하는 방법이다.

KRUSKAL 알고리즘은 minimal spanning tree를 구현하는 알고리즘으로 방향성은 없으나 가중치는 고려하여 수행한다.

DIJKSTRA 알고리즘은 single source to all destination으로 하나의 정점에서 모든 정점까지의 최단경로를 구하는 알고리즘으로 방향성과 가중치 모두 고려한다.

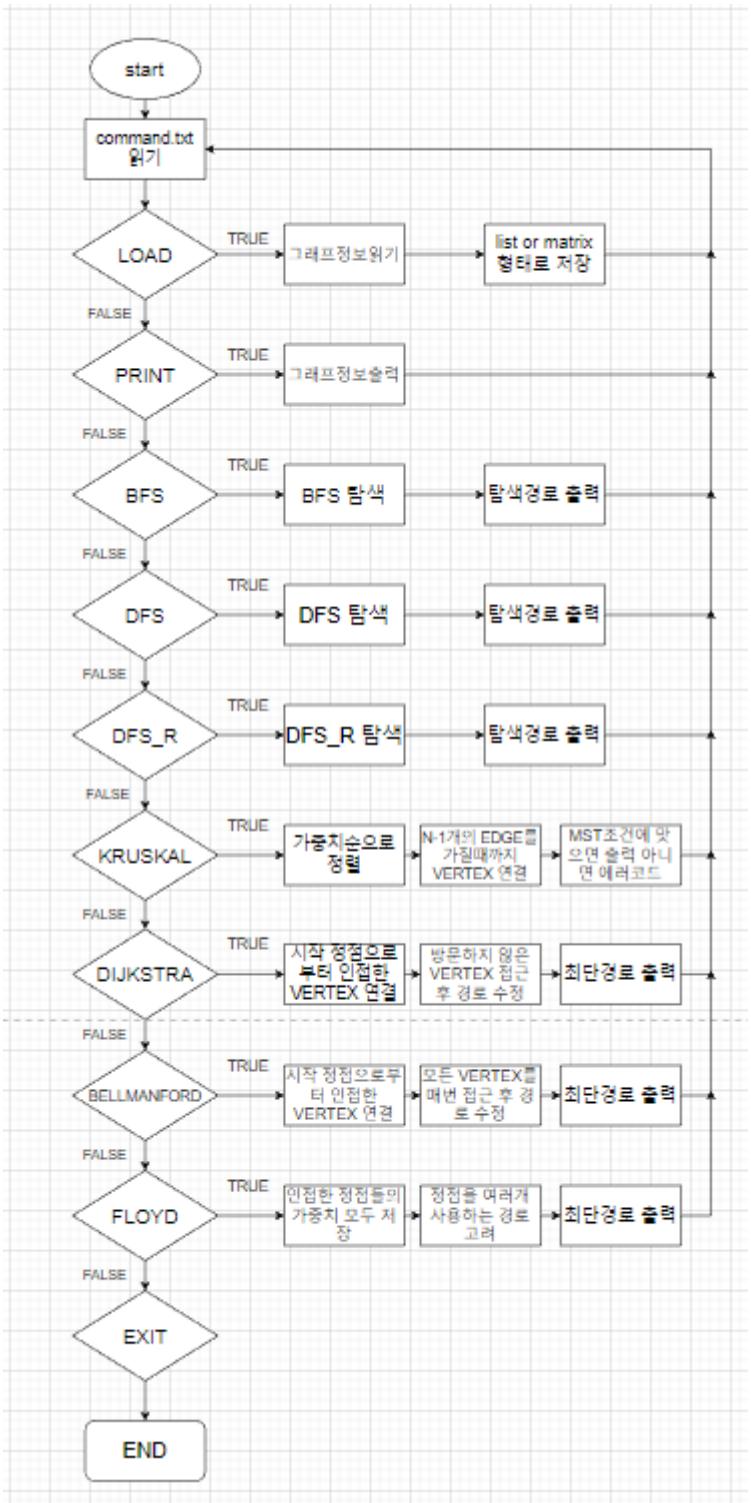
BELLMANFORD 알고리즘은 시작 정점으로부터 도착정점까지의 최단경로를 구하는 알고리즘으로 weight가 음수더라도 최단경로를 구할 수 있으나 음수 사이클이 발생한 경우에는 최단경로를 구할 수 없다.

FLOYD 알고리즘은 모든 vertex쌍간의 최단경로를 구하는 것이다. Weight가 음수이더라도 최단경로를 구할 수 있으나 음수사이클이 발생한 경우에는 최단 경로를 구할 수 없다.

정렬연산은 KRUSKAL알고리즘에서 MST를 구할 때 edge를 weight에 따라 정렬하는데 사용한다. Quick sort를 사용하며 segment size가 재귀적으로 분할되어 크기가 6이하가 되면 selection sort를 사용한다.

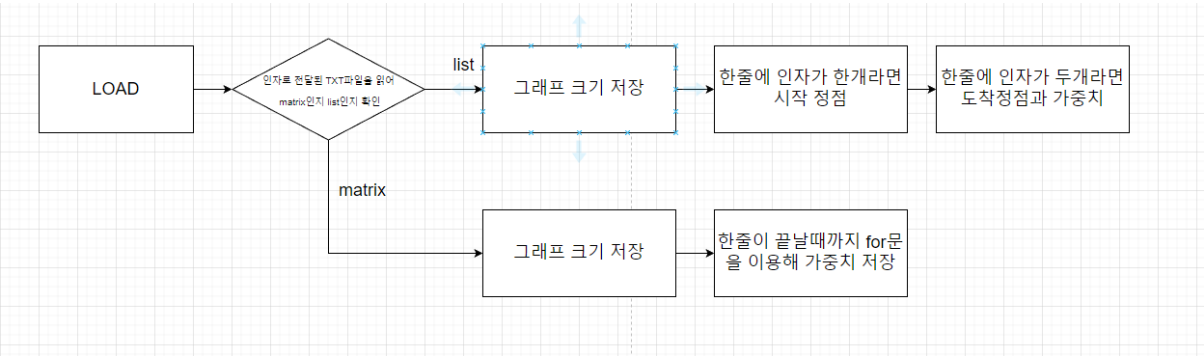
이러한 알고리즘들을 입력 받은 데이터에 따라 알맞게 출력되게 끔 하며 결과는 log.txt에 저장한다.

2. Flowchart



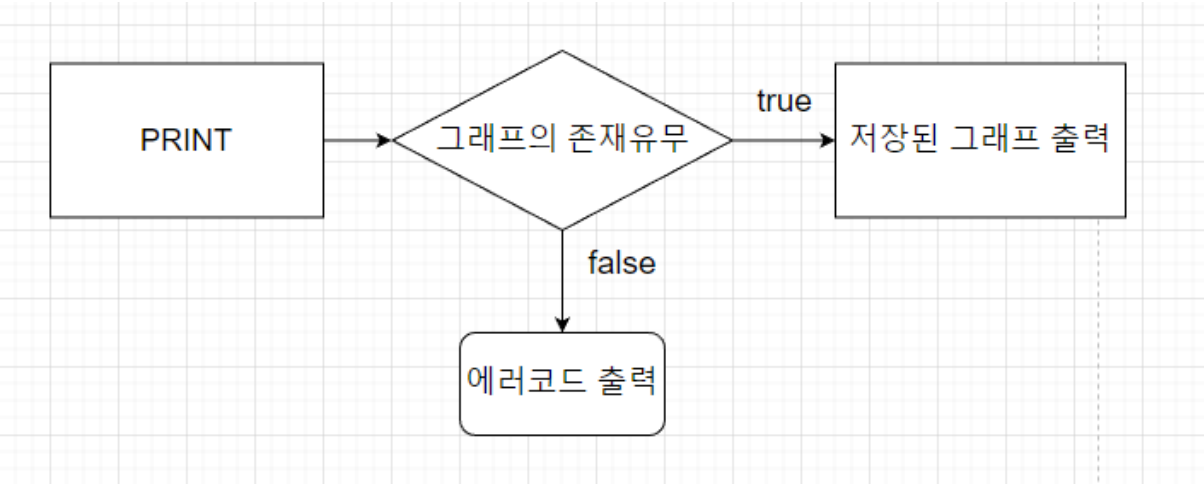
전체적인 프로그램의 흐름은 위와 같다. LOAD명령어를 통해 그래프를 구축하고 저장된 그래프 정보에 따라 아래 명령어들을 실행할 수 있다. 각각의 명령어는 아래에서 자세히 설명한다.

Load의 flowchart는 다음과 같다.

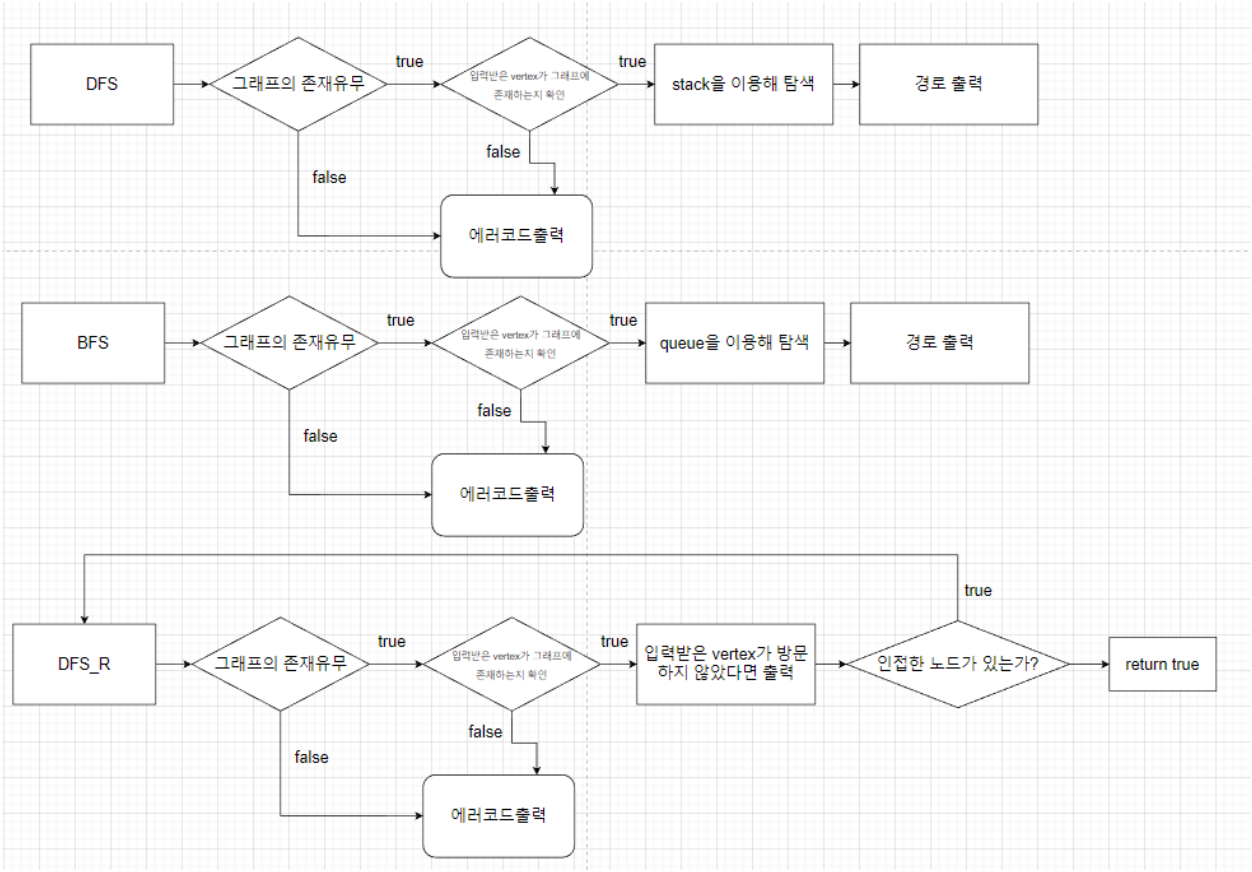


우선 그래프를 구축하는 정보의 소스는 txt파일이다. txt파일에서 그래프의 형식이 list인지 matrix인지 확인하고 이를 각각 다른 방법으로 저장한다. txt파일의 데이터는 한줄씩 불러와서 저장하므로 list형식일 경우 한줄에 하나의 숫자가 있다면 시작 정점을 가리키고 두개의 숫자가 있다면 도착 정점과 가중치를 나타내고, matrix형식일 경우 행렬의 형태를 나타내기 때문에 (start vertex, end vertex) = (0,0),(0,1),(0,2)...(0,n-1)까지를 한줄에서 저장하고 그 다음줄은 (1,0),(1,1),(1,2)....(1,n-1)이렇게 저장하여 마지막 (n-1,n-1)까지 저장하게 된다.행렬내의 데이터는 가중치를 의미한다.

Print의 flowchart는 다음과 같다.



Print는 형성된 그래프를 출력해주는 명령어이다. 현재 그래프가 존재하지 않는다면 에러코드를 출력하고 그래프가 존재한다면 그래프를 출력한다.

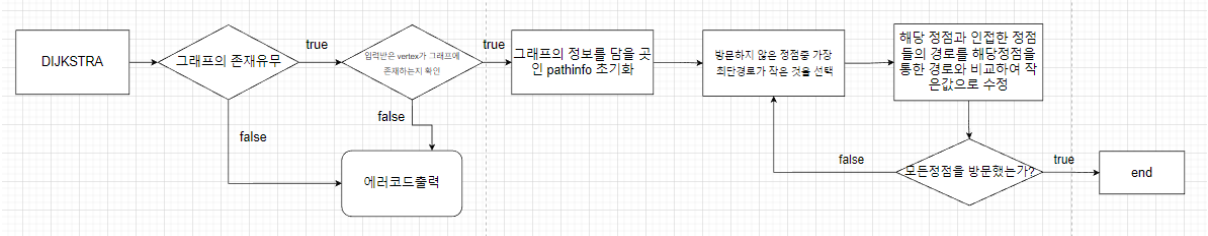


위는 DFS, BFS, DFS_R의 흐름도이다.

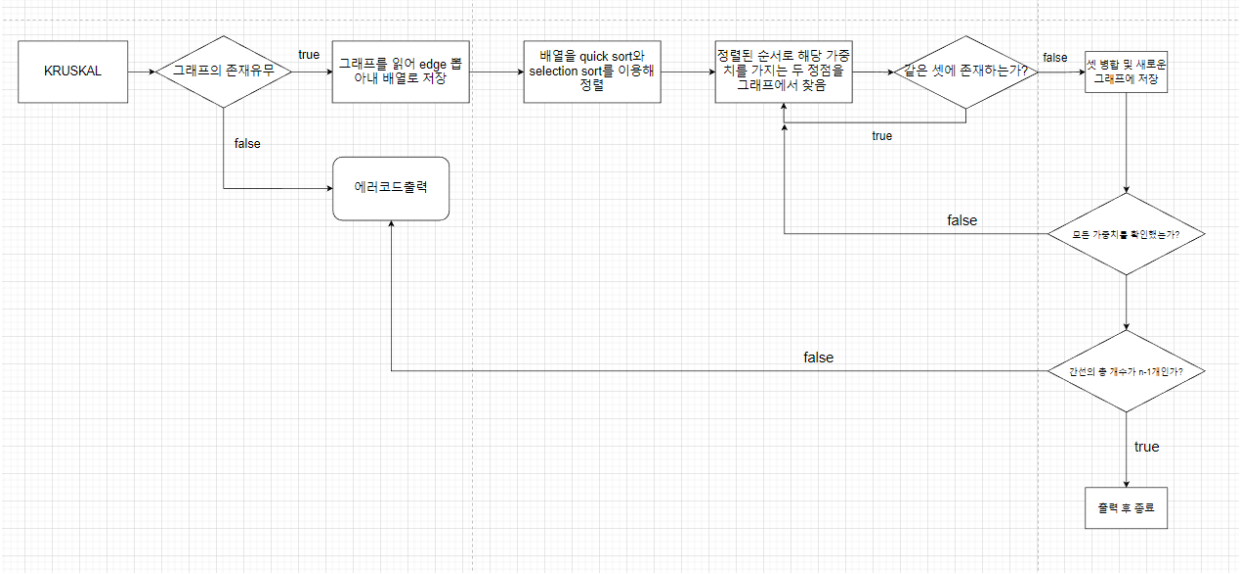
DFS는 stack을 이용한 깊이탐색방법으로 인접한 노드들을 순차적으로 스택에 삽입한 후 방문 표시를 한 후 다시 스택에서 꺼낸 노드들을 접근하는 방법으로 진행된다.

BFS는 queue를 이용한 너비탐색방법으로 인접한 노드들을 순차적으로 큐에 삽입한 후 방문 표시를 한 수 다시 큐에서 꺼낸 노드들을 접근하는 방법으로 진행된다.

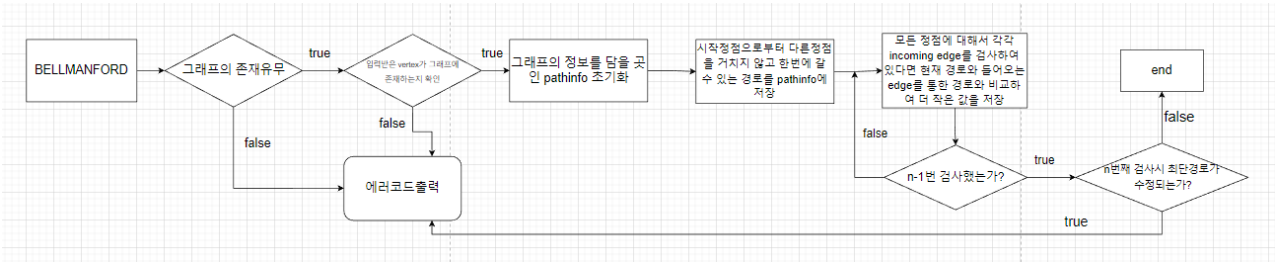
DFS_R는 DFS_R함수를 재귀적으로 방문하여 인접한 노드들을 모두 방문한다.



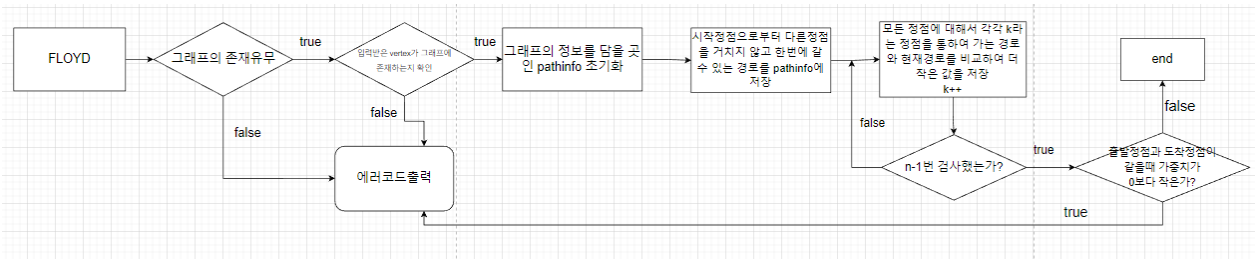
DIJKSTRA는 그래프를 읽고 방문하지 않았고 시작정점으로부터 최단경로가 가장 작은 정점을 선택하고 이를 방문했음을 표시한다. 선택한 정점과 인접한 정점들의 경로들을 기존의 최단경로와 선택한 정점을 통한 경로와 비교하여 더 작은 값을 선택한다.



KRUSKAL은 방향성이 없는 그래프를 읽어 가중치를 배열에 저장하고 오름차순으로 정렬한다. 그 후 정렬된 순서로 간선들을 연결하는데 연결할 시 같은 셋에 존재하는지 모든 가중치들을 모두 확인했는지를 판단하고 반복문을 종료한 후 간선이 전체 vertex수-1개인지 검사하고 맞다면 MST를 생성하였기에 그래프를 출력하고 아니라면 에러코드를 출력한다.



BELLMANFORD는 시작 정점으로부터 다른 정점을 거치지 않고 한번에 갈 수 있는 경로를 저장한 상태에서 시작한다. 그 후 모든 정점을 검사하는데 현재 정점으로 들어오는 경로를 가진 정점까지의 경로와 들어오는 경로의 가중치를 더한 값과 현재정점의 기존 경로중 더 작은 것을 선택하여 최단경로를 갱신한다. 정점들을 n-1보다 더 많이 검사하게 되었을 때 최단경로가 수정된다면 음수사이클이 발생하는 경우이므로 에러코드를 출력한다.



FLOYD는 BELLMANFORD와 마찬가지로 다른정점을 거치지않고 한번에 갈 수 있는 경로

를 저장해 둔 뒤 모든 정점에 대해서 k라는 정점을 통하여 가는 경로와 기존경로를 비교하여 더 작은 값을 저장한다. K는 값을 갱신한 후 k++을 통해 값을 올려준다. N-1번 비교했다면 자기자신으로 가는 경로들이 0이 아닌지를 검사한 후 0보다 작은값이 나온다면 음수사이클이 생기는 경우이므로 에러코드를 출력한다.

3. Algorithm

BFS는 너비 우선 탐색(breath-first search)로 임의의 노드에서 시작해 인접한 노드를 먼저 탐색하는 방법이다.

방향성이 없고 가중치가 없다는 가정하에 탐색목적으로 사용된다.

노드의 방문여부를 확인해야 한다는 조건이 있다. 그렇지 않으면 방문했던 노드를 다시 방문하여 루프를 탈출하지 못할 수 있다.

방문 순서를 인접한 노드부터 탐색하게끔 구현되므로 queue 자료구조를 사용하여 구현한다.

DFS는 깊이 우선 탐색(depth first search)로 임의의 노드에서 시작해 다음 분기로 넘어가기 전 해당 분기를 완벽하게 탐색하는 방법이다.

BFS와 마찬가지로 방문여부를 확인해야 한다는 조건이 있다.

깊이를 우선적으로 탐색해야 하는 특성 때문에 stack이라는 자료구조를 이용하거나 재귀적으로 구현한다.

Kruskal은 greedy method를 이용하여 모든 경로를 그때 그때 최소비용으로 연결하는 알고리즘이다.

그래프에서 서로 다른 두 정점간 정점을 사용하지 않은 경로의 가중치를 오름차순으로 정렬한 후 정렬된 경로들을 이용하여 사이클이 만들어지지 않게끔 만들고 모든 가중치를 검사하고 난 후 n-1개의 간선이 만들어진다면 MST가 생성된 것이다.

Dijkstra는 그래프에서 어떠한 vertex를 기준으로 다른 vertex까지의 최단경로를 구하는 알고리즘이다.

출발노드부터 자신을 제외한 나머지 vertex까지의 최단경로를 구하는데 현재 vertex에서 인접한 vertex중 방문하지 않은 vertex를 구분하고 이 중 가장 최단경로를 가진

vertex부터 방문하여 최단경로를 갱신한다.

가중치가 음수를 가진 그래프에서는 사용할 수 없다.

Bellmanford는 Dijkstra와 마찬가지로 시작 정점으로부터 모든 정점까지의 경로를 구할 수 있는 알고리즘이다. Dijkstra와 다른점은 음수 가중치를 가지더라도 적용할 수 있다는 점과 최단경로 갱신시 모든 정점을 방문하므로 방문여부를 파악하지 않아도 된다는 장점이 있다.

시작 정점으로부터 인접한 정점들을 저장해둔 후가 알고리즘을 적용할 첫 시점이다. 그 후는 모든 정점들을 방문하여 인접한 정점들 중 들어오는 경로가 있을 시 시작 정점부터 기존경로와 들어오는 경로중 더 작은 가중치를 가지는 경로로 최단경로를 갱신한다.

이러한 작업을 반복하게 될 때 최단경로를 찾게 되면 더 이상 경로의 가중치가 갱신되지 않는데 전체 정점의 수 이상만큼 검사하게 되었을 때 경로가 갱신된다면 음수 사이클이 있음을 알 수 있다.

Floyd는 한 정점부터 모든 정점까지의 최단거리가 아닌 모든 vertex쌍에 대하여 최단거리를 구할 수 있는 알고리즘이다. 음수 가중치를 가지더라도 적용할 수 있으며

k이하의 정점을 사용한 i부터 j까지의 최단거리는 k-1이하의 정점을 사용한 i부터 k-1까지의 거리와 k-1부터 j까지의 거리의 합과 k-1이하의 정점을 사용한 i부터 j까지의 거리 중 더 작은 값을 선택하여 구할 수 있다.

비교를 마친 후 자기자신에서부터 자기자신으로 오는 것의 가중치가 음수가 되면 음수 사이클이 발생함을 알 수 있다.

4. Result screen

```
LOAD graph_M_krus.txt
LOAD graph_empty.txt
LOAD graph_L.txt
PRINT
```

```
===========LOAD==========
Success
=========================
Can only be made from lists or arrays
======== ERROR ========
100
=======================
===========LOAD==========
Success
=========================
=============PRINT==============
[0] -> (1,6) -> (2,2)
[1] -> (3,5)
[2] -> (1,7) -> (4,3) -> (5,8)
[3] -> (6,3)
[4] -> (3,4)
[5] -> (6,1)
[6] -> (4,10)
================================
```

위와 같이 4개의 명령을 실행하였을 때 그래프가 온전한 것은 성공적으로 load 하였고 그래프가 비어있는 경우에는 오류를 출력하였다. 여러 번의 LOAD끝에는 최종적으로 LOAD한 그래프의 정보만 남아 PRINT시 그래프정보는 하나만 남은 것을 확인할 수 있다.

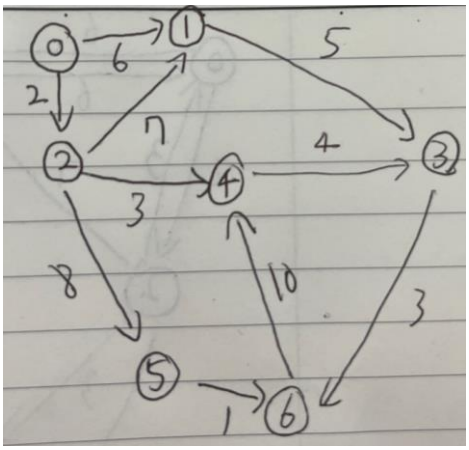
command.txt

```
1   LOAD graph_L.txt
2   PRINT
3   BFS 0
4   DFS 0
5   DFS_R 0
6   KRUSKAL
7   DIJKSTRA 1
8   DIJKSTRA 2
9   BELLMANFORD 0 6
10  BELLMANFORD 1 6
11  FLOYD
12  EXIT
```

이와 같은 명령 예시를 만들었고 해당 명령 순서대로 결과를 확인해보려고 한다.

```
==========LOAD==========
Success
========================
=============PRINT==============
[0] -> (1,6) -> (2,2)
[1] -> (3,5)
[2] -> (1,7) -> (4,3) -> (5,8)
[3] -> (6,3)
[4] -> (3,4)
[5] -> (6,1)
[6] -> (4,10)
================================
```

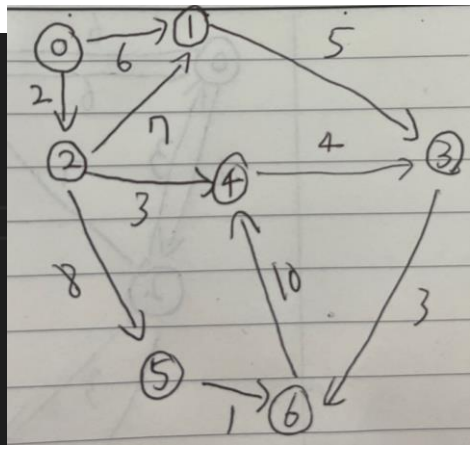
우선 올바르게 그래프가 LOAD되었고 PRINT또한 list정보가 담긴 그래프형식에 맞게 list형식으로 출력하였다. 그래프를 그림으로 나타내면 아래와 같다.



BFS,DFS,DFS_R명령어는 방향성이 없고 가중치도 고려하지 않기에

아래와 같은 결과가 출력되어야 한다.

```
======== BFS ========
startvertex: 0
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6
====================
======== DFS ========
startvertex: 0
0 -> 1 -> 3 -> 4 -> 6 -> 5 -> 2
====================
======== DFS_R ========
startvertex: 0
0 -> 1 -> 2 -> 4 -> 3 -> 6 -> 5
======================
```

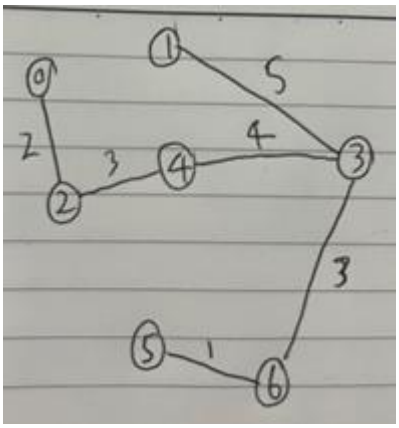


BFS는 너비탐색이므로 올바르게 나온 것을 확인할 수 있고 DFS와 DFS_R 또한 깊이 탐색이므로 위와 같이 나오는 것을 확인할 수 있다.

하지만 DFS는 스택을 사용하여 구현하였고 DFS_R는 재귀적으로 구현하였다.

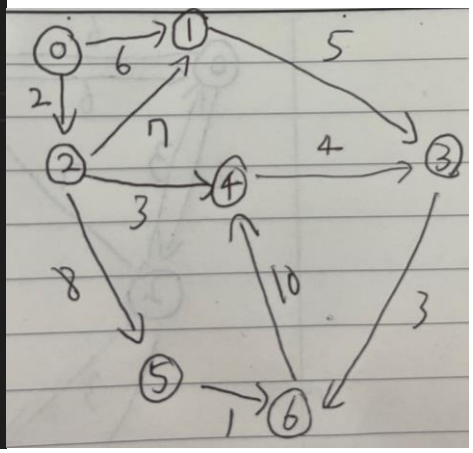
따라서 DFS는 1이 먼저 스택에서 빠져나와 3이라는 vertex를 스택에 넣기에 3이 먼저 빠져나가는 것이고 DFS_R은 1이 실행된 후 2와 3중 더 적은 번호부터 선택하여 내려가게끔 구현하였기에 순서는 다소 다를 수 있으나 동일한 깊이탐색방법임을 알 수 있다.

```
======== KRUSKAL ========
[0] 2(2)
[1] 3(5)
[2] 0(2) 4(3)
[3] 1(5) 4(4) 6(3)
[4] 2(3) 3(4)
[5] 6(1)
[6] 3(3) 5(1)
cost: 18

=======================
```



Krustal은 위의 그래프를 토대로 만든다면 오른쪽과 같이 생성되므로 올바르게 형성된 것을 확인할 수 있다.

```
======== DIJKSTRA ========
startvertex: 1
[0] x
[1] x
[2] x
[3] 1->3(5)
[4] 1->3->6->4(18)
[5] x
[6] 1->3->6(8)

=======================
======== DIJKSTRA ========
startvertex: 2
[0] x
[1] 2->1(7)
[2] x
[3] 2->4->3(7)
[4] 2->4(3)
[5] 2->5(8)
[6] 2->5->6(9)

=======================
```

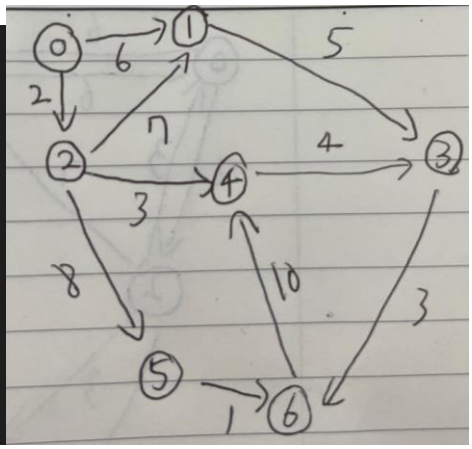


Dijkstra는 1과 2에서 각각의 vertex까지의 경로와 비용을 출력한다. 갈 수 없는 곳이라면 x를 출력하였다.

```
======== BELLMAN-FORD ========
0->2->5->6
cost: 11

=======================
======== BELLMAN-FORD ========
1->3->6
cost: 8

=======================
```

Bellmanford는 0에서 6까지 1에서 6까지의 최단경로를 출력하였고

```
======== FLOYD ========
     [0] [1] [2] [3] [4] [5] [6]
[0]  0   6   2   9   5   10  11
[1]  x   0   x   5   18  x   8
[2]  x   7   0   7   3   8   9
[3]  x   x   x   0   13  x   3
[4]  x   x   x   4   0   x   7
[5]  x   x   x   15  11  0   1
[6]  x   x   x   14  10  x   0

=======================
```



Floyd는 모든 vertex쌍에 대해서 최단경로의 cost를 출력한다.

그래프와 비교하였을 때 올바르게 잘 나온 것을 확인할 수 있다.

x인경우는 도달할 수 없는 곳이라는 뜻이다.

5. Consideration

우선 LOAD명령어부터 구현할 때 LOAD는 비교적 쉽게 구현하였다. 이전 프로젝트때 여러 번 명령어에 대한 분기를 잘 나눠 왔고 특히 fpgrowth를 생성했던 지난 프로젝트때 그래프나 자료구조를 형성하고 그에 저장하는 경우를 많이 접했었기 때문에 수월했던 것 같다. DFS,BFS,DFS_R과 같은 알고리즘은 방향성이 없고 가중치도 고려하지 않게 만들어야 했다. 하지만 방향성이 있는 그래프를 입력받더라도 정상 작동 해야했기 때문에 인접노드를 구하고 중복성을 제거하는 것을 어떻게 해야할지에 대한 고민을 많이 했던 것 같다. 현재노드에서 인접한 노드를 구하는 것은 쉽지만 그렇게 하면서 들어오는 vertex를 어떻게 구할지를 생각을 많이 했던 것 같다. 들어오는 vertex에 대한 생각은 bellmanford를 구현할 때도 사용되었기에 유용하였다.

그리고 가장 어려웠던 알고리즘이 나왔는데 바로 Kruskal이었다. 가중치별로 퀵정렬과 선택정렬을 조건에 맞게 적절히 사용하여 정렬하고 이를 하나하나 선택하여 MST를 구현하는 것이 목적이었다. 정렬을 수업시간에 배운 개념을 토대로 구현하였더니 쉽게 구현할 수 있는 개념이었으나 정렬한 이러한 가중치를 어떻게 정점과 정점사이 관계를 나타내게 끔 연결지을 지가 고민이었다. 가중치를 정렬하더라도 가중치가 그래프내에서 중복된다면 해당 가중치가 어떤 정점들끼리의 가중치인지 알 수 없기 때문이었다. 이에 대한 해결을 가중치가 검사되면 저장된 그래프에서 지우는 것으로 해결하였다. 매 명령어마다 그래프에서 해당알고리즘이 사용할 정보를 정제하여 저장해주는 배열을 선언해두었는데 이러한 배열에서 가중치와 짝지어진 것들은 0으로 바꾸어 다음 중복되는 가중치는 다른 정점들과 짝지어지게끔 구현하였더니 수월하게 구현할 수 있었다.

그 이후의 알고리즘들은 shortpath강의를 듣고 의사코드를 기반으로 작성하니 초기조건만 잘 맞춰준다면 알고리즘을 사용하기만 하면 구현이 쉬웠기 때문에 큰 어려움은 없었던 것 같다.

음수사이클을 발견해내는 것에 대한 개념이 다소 신기했는데 bellmanford는 최단거리를 구하는 알고리즘을 적용할수록 계속 최단경로가 갱신된다면 음수사이클 Floyd은 자기자신으로부터 자기자신까지의 edge가 음수가 되면 음수사이클을 발견할 수 있는데 알고리즘마다 이렇게 음수사이클을 다르게 발견할 수 있는 것이 좀 신기했던 것 같다.

데이터 구조설계를 수강하면서 여러 알고리즘과 자료구조를 학습하였는데 마지

막 프로젝트인 만큼 구현하고 나서의 성취감이 더 컸던 것 같다.