
# 2019202005_DS_project3

수업 명 : 데이터구조실습
과제 이름 : DS project3
담당 교수님 : 이형근 교수님
학 번 : 2019202005
이 름 : 남종식

## Introduction

이번 과제에서는 다양한 그래프 연산 알고리즘들을 통하여 그래프 연산을 수행하는 과제입니다. 알고리즘은 총 7가지가 있으며 이중에는 방향성을 고려해야 하는 것과 고려하지 않아도 되는 것이 있습니다. 연산을 하기 위해서 그래프가 주어지는데 이는 list형태 또는 matrix형태로 주어질 수 있습니다. 모두 두 형태로 주어졌을 때 연산이 아무 문제없이 이루어져야 하지만 KwangWoon알고리즘은 list형태로 그래프가 주어졌을 때만 연산 가능합니다.

### LOAD

LOAD명령어를 통해 graph_L.txt파일과 graph_M.txt파일을 불러올 수 있으며 이는 LOAD명령어 뒤에 인자에 따라서 결정됩니다. 오류가 없을 시에는 불러온 data가 잘 저장되었다는 success를 출력합니다. 만약 텍스트 파일이 존재하지 않거나 비어 있는 경우에는 에러 코드 100을 출력합니다.
기존에 그래프 정보가 존재할 때 LOAD가 입력되면 기존 그래프 정보는 삭제하고 새로 그래프를 생성해야 합니다.

### PRINT

PRINT명령어를 통해 그래프의 상태를 출력할 수 있습니다. list형태의 그래프는 adjacency list로, matrix형태의 그래프는 adjacency matrix로 출력하며 이때 저장된 vertex들을 오름차순으로 출력하며 edge는 vertex를 기준으로 오름차순으로 출력합니다. 그래프에 저장된 데이터가 없으면 에러 코드 200을 출력합니다.

### BFS

BFS명령어를 통해 BFS그래프 연산을 진행할 수 있습니다. 이때, 인자로는 방향성과 시작 vertex를 받으며 BFS그래프 연산을 수행하면서 방문하는 vertex의 순서를 출력합니다. 인자로 받은 vertex가 그래프에 존재하지 않거나 vertex를 입력하지 않으면 에러 코드 300을 출력합니다.

### DFS

DFS명령어를 통해 DFS그래프 연산을 진행할 수 있습니다. 이때, 인자로는 방향성과 시작 vertex를 받으며 DFS그래프 연산을 수행하면서 방문하는 vertex의 순서를 출력합니다. 인자로 받은 vertex가 그래프에 존재하지 않거나 vertex를

입력하지 않으면 에러 코드 400을 출력합니다.

**KRUSKAL**

KRUSKAL명령어를 통해 먼저 MST를 구하고 이 MST를 구성하는 edge들의 vertex를 오름차순으로 출력하며 total cost를 출력합니다. 이때 MST를 구할 수 없거나 명령어를 수행할 수 없는 경우에는 에러 코드 600을 출력합니다.

**DIJKSTRA**

DIJKSTRA명령어를 통해 shortest path와 cost를 출력할 수 있습니다. 이때 인자로는 방향성과 start vertex를 받습니다. 만약, 그래프가 존재하지 않거나 vertex가 입력되지 않는 경우에는 에러 코드 700을 출력합니다.

**BELLMANFORD**

BELLMANFORD명령어를 통해 shortest path와 cost를 출력할 수 있습니다. 이때 인자로는 방향성과 start vertex 그리고 end vertex를 받습니다. 이때 음수 weight가 있는 경우에도 동작해야 하며 만약, 그래프가 존재하지 않거나 vertex가 입력되지 않는 경우에는 에러 코드 800을 출력합니다.

**FLOYD**

FLOYD명령어를 통해 모든 vertex쌍에 대하여 최소 cost를 matrix형태로 출력할 수 있습니다. 이때 인자로 방향성을 받으며 명령어를 수행할 수 없는 경우와 음수 사이클이 발생하는 경우에는 에러 코드 900을 출력합니다.

**KWANGWOON**

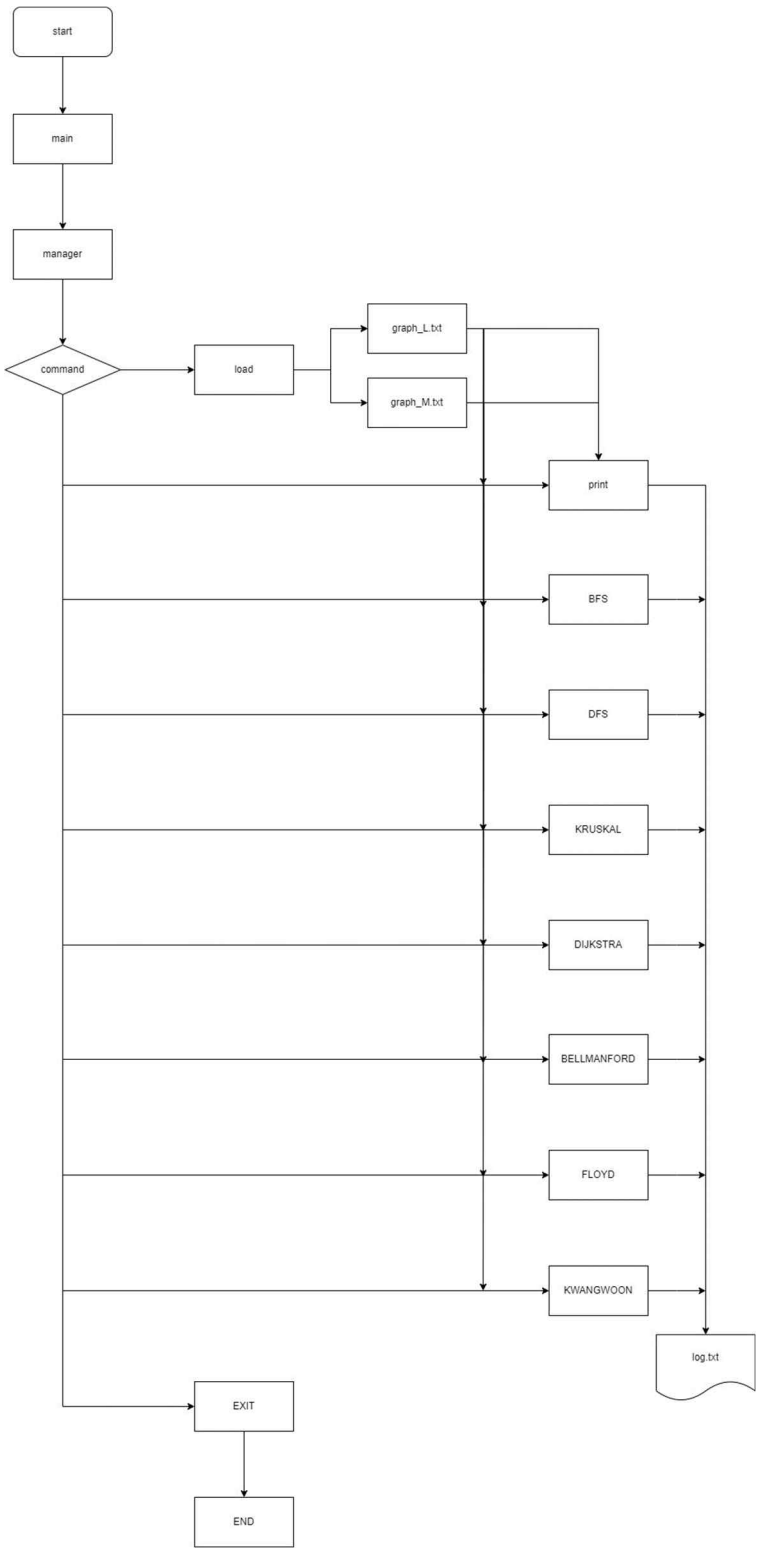
KWANGWOON명령어를 통해 세그먼트 트리를 이용하여 그래프를 구현할 수 있으며 이는 adjacency list형태의 데이터만 이용합니다. 현재 점에서 방문할 수 있는 정점(갈 수 있고 방문한적이 없는 정점)들이 홀수개면 그 정점 번호들의 가장 큰 정점 번호로 방문을 시작하고, 짝수개면 가장 작은 정점 번호로 방문을 시작하여 결과를 출력합니다. 만약, 그래프에 1번 정점이 없다면 에러 코드 500을 출력합니다.

**EXIT**

EXIT명령어를 통해 프로그램 상의 메모리를 해제하며, 프로그램을 종료합니다.

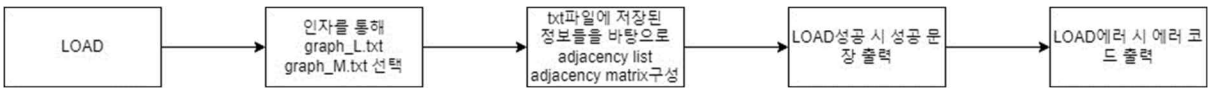
잘못된 명령어가 들어오면 에러 코드 1000을 출력합니다.

## Flowchart



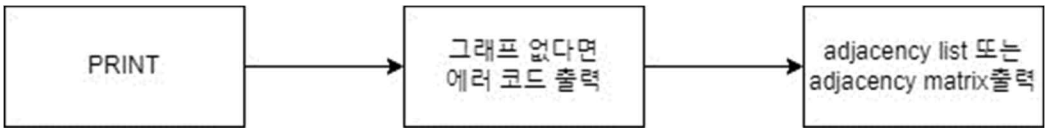
이번 프로젝트의 전체적인 흐름은 나타내는 manager의 순서도입니다. command.txt 파일에서 getline으로 한줄씩 읽어온 다음 이를 바탕으로 어떤 명령어가 실행될지 알 수 있습니다. 각 command가 들어오면 그에 해당하는 함수를 호출하게 됩니다. Command에 해당하는 동작을 정상 작동할 시에는 그 결과를 log.txt파일에 작성하게 됩니다. 마지막으로 command가 exit가 들어오면 프로그램은 종료하게 됩니다.

**LOAD**



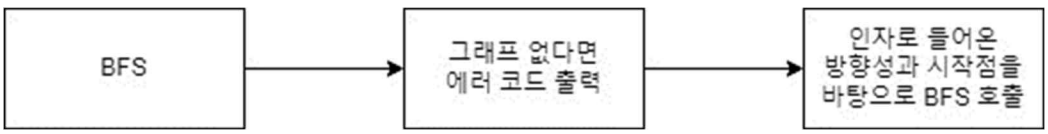

다음으로 LOAD command일 때 실행되는 순서도입니다. 기존의 그래프 정보가 존재한다면 이를 삭제하고 새로 그래프를 생성합니다. 인자로 들어온 그래프 형식이 list일때는 type을 0으로 하고 matrix일때는 type을 1로 하고 그래프의 사이즈를 바탕으로 그래프를 생성합니다. 그 후 방향성이 없는 그래프와 방향성이 있는 그래프에 edge와 weight를 바탕으로 그래프에 추가합니다. 에러 발생시 에러 코드를 성공시에는 성공 출력문을 출력합니다.

**PRINT**



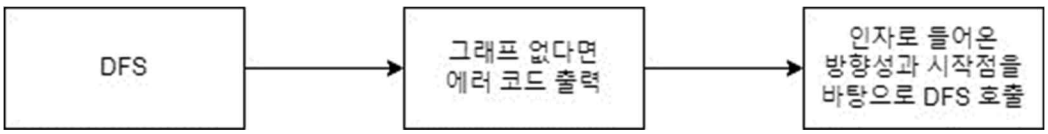

PRINT를 통해 그래프에 저장되어 있는 모든 vertex는 오름차순으로 출력하고 edge는 vertex를 기준으로 오름차순으로 출력할 수 있습니다. 그래프에 저장된 데이터가 없는 경우 에러 코드 200을 출력합니다. 데이터가 존재한다면 그래프의 상태를 출력합니다.

**BFS**



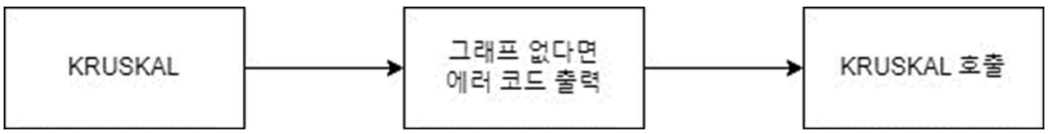
BFS명령어를 통해 들어온 인자 중 방향성과 시작 vertex를 분리하고 만약 NULL값이라면 에러 코드 300을 출력합니다. 방향성과 시작 vertex를 가지고 mBFS함수를 호출합니다. 그래프가 존재하지 않다면 에러 코드 300을 출력하며 방향성에 따라 그래프, 방향성, 시작 vertex를 가지고 BFS함수를 호출합니다.

**DFS**



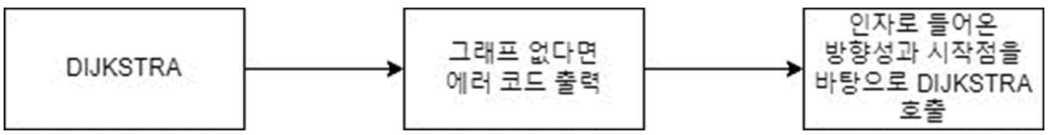

DFS명령어를 통해 들어온 인자 중 방향성과 시작 vertex를 분리하고 만약 NULL값이라면 에러 코드 400을 출력합니다. 방향성과 시작 vertex를 가지고 mDFS함수를 호출합니다. 그래프가 존재하지 않다면 에러 코드 400을 출력하며 방향성에 따라 그래프, 방향성, 시작 vertex를 가지고 DFS함수를 호출합니다.

**KRUSKAL**



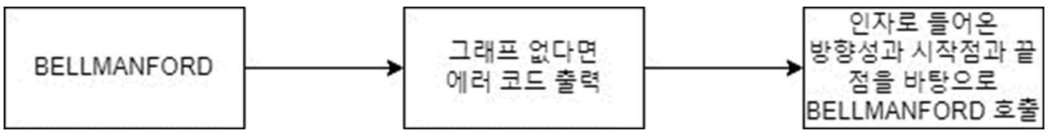

KRUSKAL명령어를 읽으면 mKRUSKAL함수를 호출하고 그래프가 존재하지 않다면 에러 코드 600을 출력하며 그래프가 존재한다면 KRUSKAL함수를 호출합니다.

**DIJKSTRA**



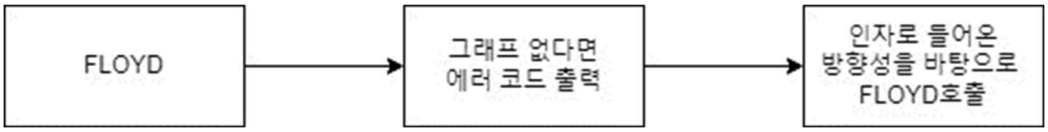

DIJKSTRA명령어를 읽으면 들어온 인자 중 방향성과 시작 vertex를 분리하고 만약 NULL값이라면 에러 코드 700을 출력합니다. 방향성과 시작 vertex를 가지고 mDIJKSTRA함수를 호출합니다. 그래프가 존재하지 않다면 에러 코드 700을 출력하며 방향성에 따라 그래프, 방향성, 시작 vertex를 가지고 Dijkstra함수를 호출합니다.

**BELLMANFORD**



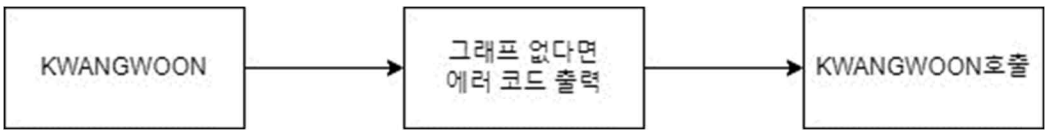

BELLMANFORD명령어를 읽으면 들어온 인자 중 방향성과 시작 vertex, 끝 vertex를 분리하고 만약 NULL값이라면 에러 코드 800을 출력합니다. 방향성과 시작 vertex, 끝 vertex를 가지고 mBELLMANFORD함수를 호출합니다. 그래프가 존재하지 않다면 에러 코드 800을 출력하며 방향성에 따라 그래프, 방향성, 시작 vertex, 끝 vertex를 가지고 Bellmaford함수를 호출합니다.

**FLOYD**




FLOYD명령어를 읽으면 들어온 인자 중 방향성을 분리하고 만약 NULL값이라면 에러 코드 900을 출력합니다. 방향성을 가지고 mFLOYD함수를 호출합니다. 그래프가 존재하지 않다면 에러 코드 900을 출력하며 방향성에 따라 그래프, 방향성을 가지고 FLOYD함수를 호출합니다.

**KWANGWOON**




KWANGWOON명령어를 읽으면 mKwoonWoon함수를 호출하고 그래프가 존재하지 않다면 에러 코드 500을 출력하며 그래프가 존재한다면 KWANGWOON함수를 호출합니다.

## Algorithm

### BFS

BFS를 구현하기 위해서는 큐를 사용하였으며 STL을 활용하여 쉽게 구현할 수 있었습니다. vertex가 그래프 내에 존재하지 않는 경우 에코 코드 300을 출력하도록 했습니다. 다음으로 큐에 vertex를 push한 다음에 큐가 빌 때까지

방문한 노드를 false에서 true로 바꿔주며 출력을 진행함으로써 BFS알고리즘을 구현했습니다.

**DFS**

DFS는 BFS와 거의 비슷한 형태로 구현할 수 있었으며 큐 대신 스택을 사용하여 구현하였습니다. 이때 또한 STL을 활용하여 쉽게 구현할 수 있었습니다.

**KRUSKAL**

Disjoint Set 초기화를 하기 위해 각 정점을 독립적인 집합으로 초기화하고, 각 집합의 대표 정점을 설정합니다. 그리고 모든 간선을 가중치의 오름차순으로 정렬합니다. 선택한 간선의 양 끝 정점이 같은 집합에 속해 있는지 확인하기 위해, Find 연산을 사용합니다. 두 정점이 다른 집합에 속해 있다면 Union 연산을 통해 두 정점을 하나의 집합으로 합칩니다. 간선을 선택하면서 사이클을 형성하지 않으면 해당 간선을 최소 신장 트리에 추가하고, 간선의 가중치를 누적하여 최종 MST의 가중치를 계산합니다. 이런 방식으로 KRUSKAL 알고리즘은 그래프에서 최소 신장 트리를 효과적으로 찾을 수 있습니다.

**DIJKSTRA**

각 정점에서 다른 정점까지의 최단 경로를 저장하기 위해 별도의 벡터를 선언하여 사용했습니다. 우선수위 큐를 사용하여 현재까지 발견된 최단 경로 중에서 가장 짧은 경로를 먼저 선택합니다. 리스트 자료구조를 사용하여 각 정점에서 도달 가능한 이웃 정점들의 정보를 저장하였습니다. dist[v] > dist[u] + weight일 때, 현재까지 알고있던 v로의 최단 경로보다 u를 거쳐서 가는 것이 더 짧은 경우에 dist[v]를 업데이트하고, 우선 순위 큐에 해당 정점을 다시 추가합니다. 시작 정점은 출력하지 않고, 초기화 값과 같은 무한대인 경우 도달할 수 없는 정점으로 간주하여 "x"를 출력하였습니다. 이와 같은 방식으로 DIJKSTRA 알고리즘을 구현하여 최단 경로와 그에 해당하는 거리를 구할 수 있습니다.

**BELLMANFORD**

각 정점에서 도달 가능한 이웃 정점들의 정보를 리스트 자료구조를 사용하여 저장합니다. BELLMANFORD알고리즘은 각 정점을 순회하면서 모든 간선에 대해 최단 거리를 갱신합니다. 만약 현재까지의 dist[v]보다 dist[i] + weight가 더

작다면 dist[v]를 업데이트합니다.

**FLOYD**

length 배열을 생성하고 모든 요소를 무한대(INF)로 초기화합니다. 각 정점에서 자기 자신으로 가는 경우에는 0으로 초기화합니다. 그래프의 간선 정보를 length 배열에 입력합니다. dist 배열을 length 배열로 복사하여 초기화합니다. dist[i][j]는 정점 i에서 정점 j로의 현재까지의 최단 거리를 나타냅니다. 세 개의 중첩된 반복문을 사용하여 모든 정점 쌍 (i, j)에 대해 최단 경로를 갱신합니다. k를 거쳐가는 경로(dist[i][k] + dist[k][j])가 현재까지의 최단 경로보다 짧으면 dist[i][j]를 갱신합니다.

**KWANGWOON**

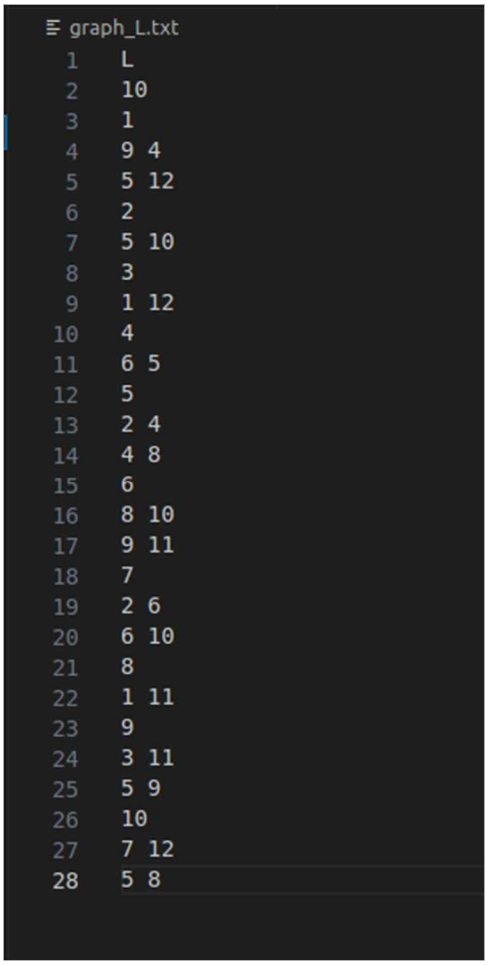
항상 정점 1에서 시작합니다. 현재 정점에서 연결된 간선의 수가 짝수인지 홀수인지 확인합니다. 짝수 간선이면 가장 작은 정점 번호로 방문을 시작하고 홀수 간선이면 가장 큰 정점 번호로 방문을 시작합니다. 세그먼트 트리는 현재 정점에서 다른 정점으로의 이동 여부를 효율적으로 저장하고 리프 노드는 다른 정점으로의 이동 여부를 나타냅니다. 그래프에 1번 정점이 없으면 에러를 출력합니다.

## Result Screen

command.txt

```
1   LOAD graph_L.txt
2   PRINT
3   BFS Y 1
4   DFS Y 1
5   DIJKSTRA Y 5
6   BELLMANFORD Y 1 3
7   FLOYD Y
8   KRUSKAL
9   KWANGWOON 1
10  EXIT
```

먼저 위 command.txt에 따라서 진행하겠습니다.




Graph_L.txt는 다음과 같습니다.

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

먼저 LOAD가 잘 되어 success를 출력하는 모습을 확인할 수 있습니다.

```
========PRINT========
[1] -> (5, 12) -> (9, 4)
[2] -> (5, 10)
[3] -> (1, 12)
[4] -> (6, 5)
[5] -> (2, 4) -> (4, 8)
[6] -> (8, 10) -> (9, 11)
[7] -> (2, 6) -> (6, 10)
[8] -> (1, 11)
[9] -> (3, 11) -> (5, 9)
[10] -> (5, 8) -> (7, 12)
====================
```

PRINT명령어 수행결과입니다. Graph_L.txt에 저장된 데이터들이 adjacency list로 잘 출력되는 모습을 확인할 수 있습니다.

```
======== BFS ========
Directed Graph BFS result
startvertex: 1
1 -> 5 -> 9 -> 2 -> 4 -> 3 -> 6 -> 8
=====================
```

시작점을 1로 하는 방향성 있는 그래프 BFS의 출력 결과입니다.

```
======== DFS ========
Directed Graph DFS result
startvertex: 1
1 -> 5 -> 2 -> 4 -> 6 -> 8 -> 9 -> 3
=====================
```

시작점을 1로 하는 방향성 있는 그래프 DFS의 출력 결과입니다.

```
========= Dijkstra =========
Directed Graph Dijkstra result
startvertex: 5
[1] 5 -> 4 -> 6 -> 8 -> 1 (34)
[2] 5 -> 2 (4)
[3] 5 -> 4 -> 6 -> 9 -> 3 (35)
[4] 5 -> 4 (8)
[6] 5 -> 4 -> 6 (13)
[7] x
[8] 5 -> 4 -> 6 -> 8 (23)
[9] 5 -> 4 -> 6 -> 9 (24)
[10] x
==========================
```

다음은 시작점이 5인 방향성 있는 다익스트라 그래프의 출력 결과입니다.
모든 정점의 path와 cost가 출력되는 모습을 볼 수 있고 가능한 path가 없는 경우에는 x를 출력하는 모습입니다.

```
====== Bellman-Ford ======
Directed Graph Bellman-Ford result
1 -> 9 -> 3
cost: 15
==========================
```

시작점이 1이고 끝점이 3인 방향성 있는 bellman ford그래프입니다. Path가 출력되고 cost가 아래에 출력되는 모습을 확인할 수 있습니다.

```
======== FLOYD ========
Directed Graph FLOYD result
    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
[1] 0   16  15  20  12  25  x   35  4   x
[2] 44  0   45  18  10  23  x   33  34  x
[3] 12  28  0   32  24  37  x   47  16  x
[4] 26  29  27  0   25  5   x   15  16  x
[5] 34  4   35  8   0   13  x   23  24  x
[6] 21  24  22  28  20  0   x   10  11  x
[7] 31  6   32  24  16  10  0   20  21  x
[8] 11  27  26  31  23  36  x   0   15  x
[9] 23  13  11  17  9   22  x   32  0   x
[10]    42  12  43  16  8   21  12  31  32  0
=====================
```

다음은 방향성 있는 FLOYD의 그래프입니다. 정점이 10부터는 조금 밀려서 출력이 되어 10에서 1로 가는 cost를 42라고 생각하면 되겠습니다. 모두 잘 출력되는 모습을 확인할 수 있으며 path가 없는 경우에는 x를 출력하고 자기 자신에게 가는 경우에는 0을 출력하는 모습을 확인할 수 있습니다.

```
====== Kruskal =======
[1] 9(4)
[2] 5(4) 7(6)
[3] 9(11)
[4] 5(8) 6(5)
[5] 2(4) 4(8) 9(9) 10(8)
[6] 4(5) 8(10)
[7] 2(6)
[8] 6(10)
[9] 1(4) 3(11) 5(9)
[10] 5(8)
cost: 65
=====================
```

다음은 크루스칼의 출력 결과입니다. 각 vertex와 cost가 모두 잘 출력되는 모습을 확인할 수 있습니다.

```
======== KWANGWOON ========
startvertex: 1
1->5->2
=====================
```

시작점을 1로 하는 광운 그래프 또한 잘 출력되는 모습을 확인할 수 있습니다.

## Consideration

데이터 구조 설계 기말고사의 주요 내용인 다양한 그래프들을 구현하며 과제와 시험 공부가 별개가 아닌 병행하면 진행할 수 있었습니다. 난이도 측면에서는 이전 과제들에 비해 많이 쉽다고 까지는 느껴지지 않았는데 그렇게 어렵다고도

느껴지지 않았습니다. 이전 과제들은 과제를 진행하면서 시험 공부와 별개로 진행된다는 느낌을 많이 받았지만 이번 과제는 시험 공부에 정말 많은 도움이 되어 개인적으로는 의미 있는 과제라고 생각합니다. 일단 대체적으로 강의자료에서 슈도코드와 개념적으로 접한 내용을 바탕으로 대부분의 코드를 작성할 수 있어서 이해도 빨랐고 구현도 빠르게 할 수 있었습니다. 그리고 이번에 처음 접하는 그래프들도 있어서 이들의 개념적인 내용들도 기말고사를 통해 공부하고 과제를 통해 구현도 할 수 있어서 빠르게 이해할 수 있었습니다. 그리고 이름만 알고 있고 확실하게는 몰랐던 알고리즘도 있었는데 그 중 BFS, DFS가 있었습니다. 지금 생각해보면 그래프 알고리즘이라서 어려울 거라고 생각만 하고 제대로 공부하려고 하지 않았던 것 같다는 생각이 들었고 이번에 확실하고 개념적인 부분이나 이들의 차이점에 대해 알게 되어 나중에 알고리즘 관련 문제를 접했을 때에도 큰 도움이 될 것 같습니다.