데이터구조설계 보고서

Project 3

이　　름 : 김태관

학　　과 : 컴퓨터정보공학부

학　　번 : 2022202067

과　　목 : 데이터구조설계

담 당 교 수 : 이기훈 교수님

실 습 분 반 : 수 7, 8교시

A. Introduction

그래프를 이용해 그래프 연산 프로그램을 구현합니다. 이 프로그램은 그래프 정보가 저장된 텍스트 파일을 통해 그래프를 구현하고, 그래프의 특성에 따라 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford, FLOYD, Kwangwoon 총 7가지 알고리즘을 통해 그래프 연산을 수행합니다. 주어지는 데이터는 기본적으로 방향성과 가중치를 모두 가지고 있으며, 데이터 형태에 따라 List 그래프와 Matrix 그래프로 저장합니다. 그래프 탐색 알고리즘마다 방향성을 고려해야 하기도 하고 고려하지 않아도 된다는 조건이 존재합니다.

BFS & DFS (방향, 무방향 가능)

BFS와 DFS는 그래프의 방향성과 가중치를 고려하고, 그래프 순회 또는 탐색 방법을 수행합니다.

Kruskal (무방향만 가능)

Kruskal 알고리즘은 최소 비용 신장 트리(MST)를 만드는 방법으로 방향성이 없고, 가중치가 있는 그래프에서 수행합니다.

Dijkstra & Bellman-Ford (방향, 무방향 가능)

Dijkstra 알고리즘은 정점 하나를 출발점으로 두고 다른 모든 정점을 도착점으로 하는 최단경로 알고리즘으로 방향성과 가중치 모두 존재하는 그래프 환경에서 연산을 수행합니다. 만약 비용이 음수일 경우 Dijkstra는 에러를 출력하며, Bellman-Ford에서는 음수 사이클이 발생한 경우 에러를 출력하고, 발생하지 않았을 경우 최단 경로와 거리를 구합니다.

FLOYD (방향, 무방향 가능)

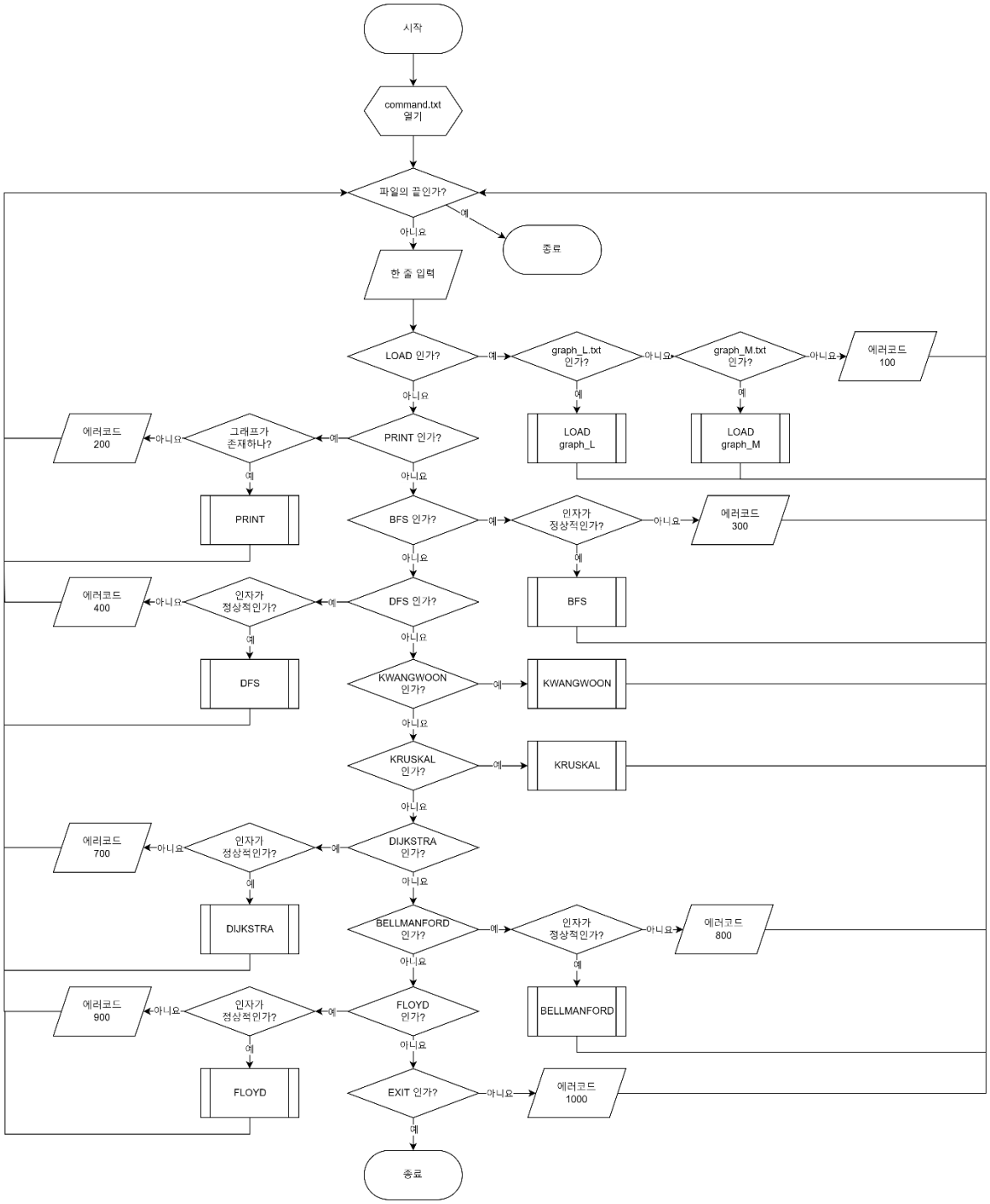
FLOYD에서는 음수 사이클이 발생한 경우 에러를 출력하고, 음수 사이클 발생하지 않았을 경우 최단 경로 행렬을 구합니다.

KwangWoon (무방향만 가능, List graph만 이용해 구현함)

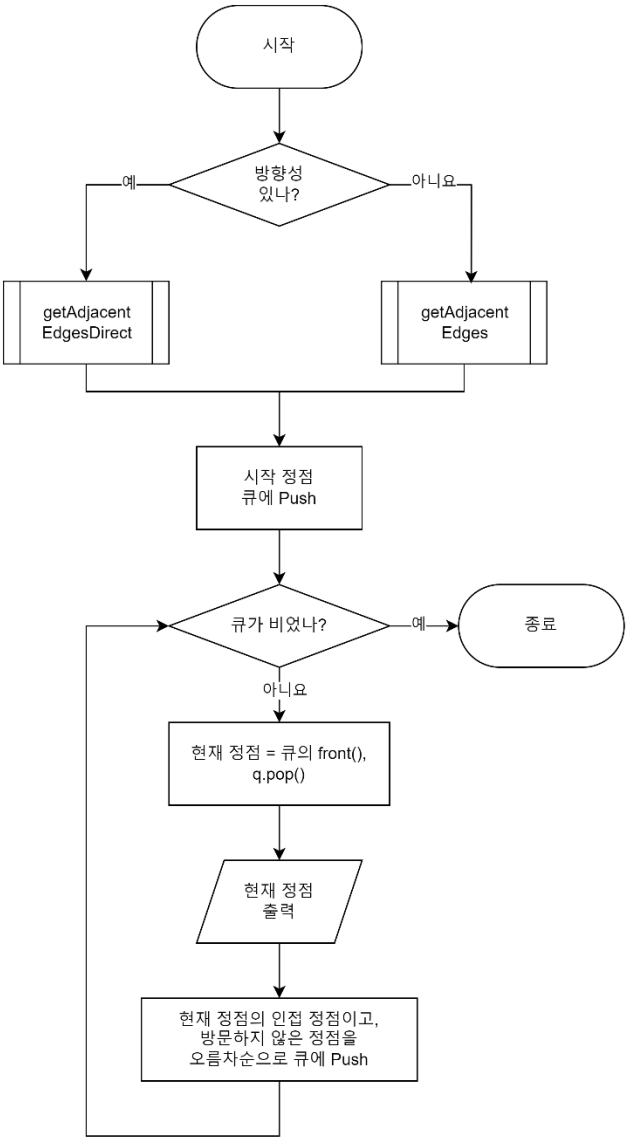
현재 정점에서 방문할 수 있는 정점(방문한 적이 없는 정점)들이 홀수개면 탐색할 수 있는 정점 중 가장 큰 정점 번호를 방문을 시작하고, 짝수개면 가장 작은 정점 번호로 방문을 시작합니다.

B. Flow chart

Manager

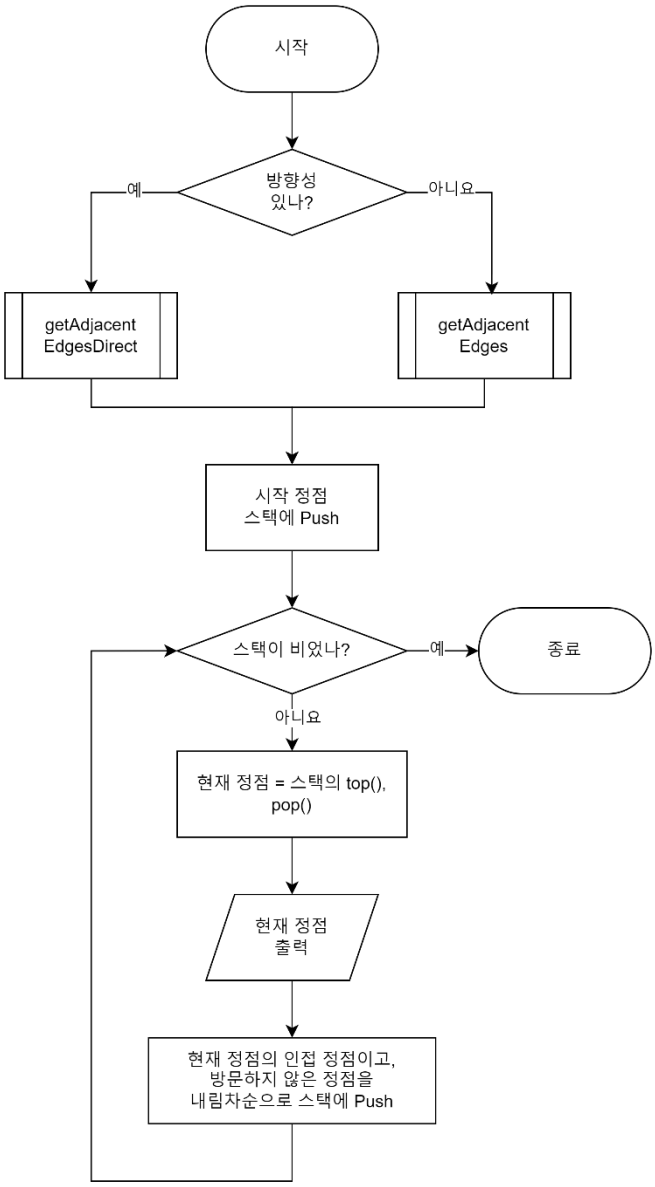


BFS



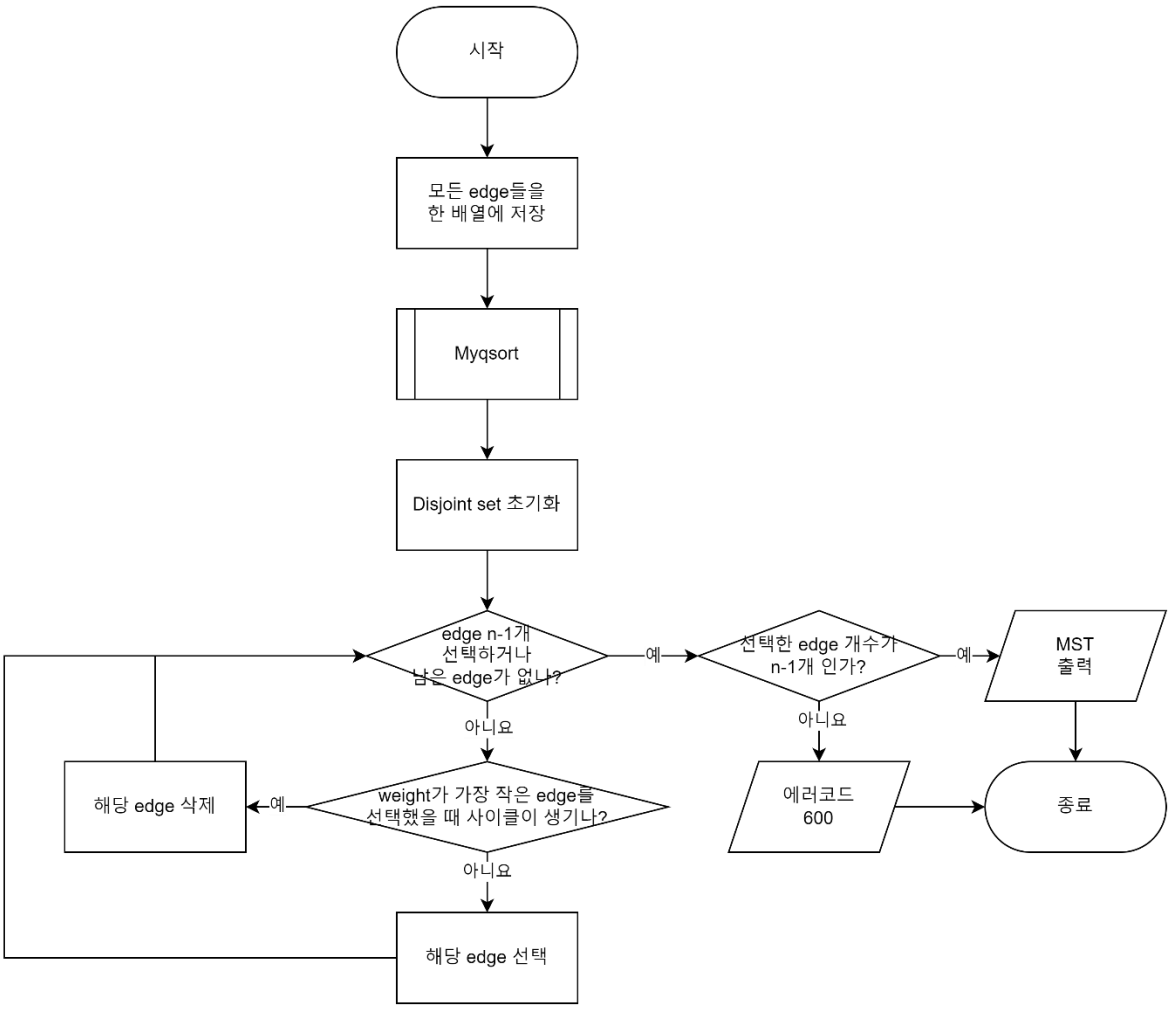
큐를 이용하여 그래프를 탐색합니다. 방문 가능한 정점이 여러 개일 때는 가장 작은 정점부터 방문합니다.

DFS



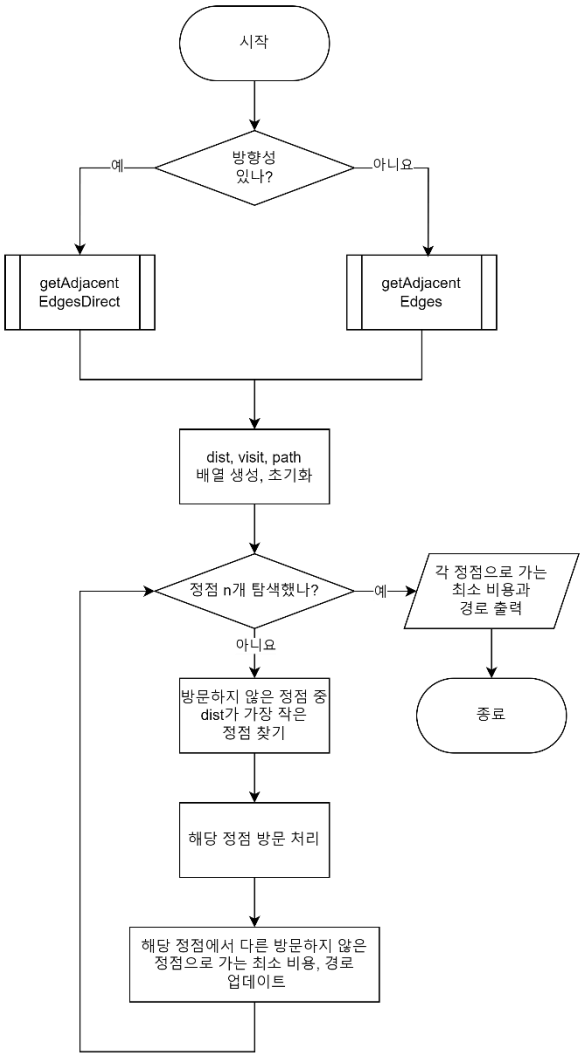
스택을 이용하여 그래프를 탐색합니다. 방문 가능한 정점이 여러 개일 때는 가장 작은 정점부터 방문합니다.

KRUSKAL



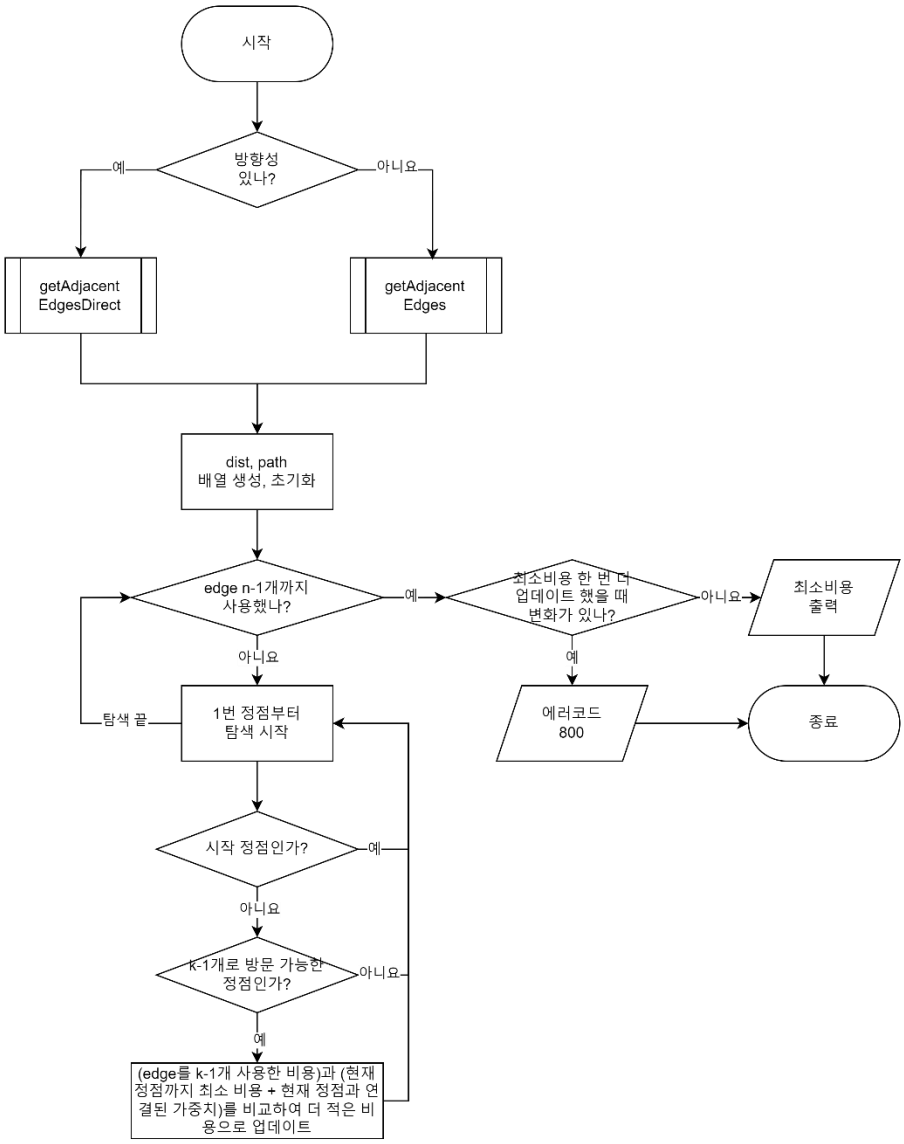
그래프의 모든 edge를 배열에 저장하고 weight 오름차순으로 정렬합니다. 그리고 weight가 가장 작은 edge를 선택했을 때 기존에 선택한 edge들과 cycle을 형성하는지 확인한 뒤 선택합니다. cycle을 확인하는 방법은 edge의 양 쪽 정점이 같은 disjoint set에 존재하면 cycle이 생기는 것이고, 다른 disjoint set에 존재하면 cycle이 생기지 않는 것입니다. edge를 선택할 때는 두 disjoint set을 합칩니다.

DIJKSTRA



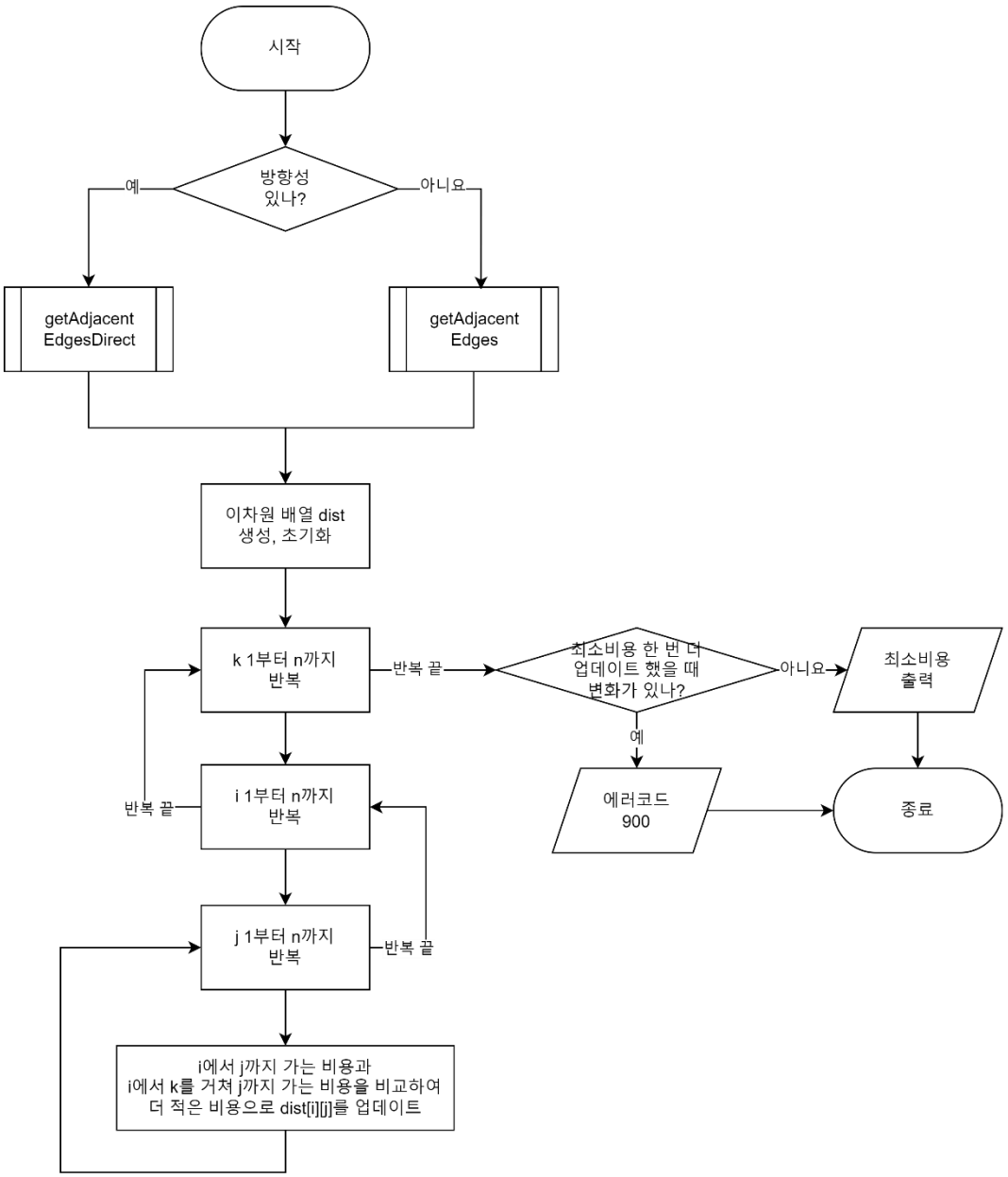
방향성 유무를 입력 받아 그래프를 복사합니다. 시작 정점을 기준으로 dist, visit, path 배열을 생성, 초기화합니다. 모든 정점을 방문할 때까지 탐색을 진행합니다. 방문하지 않은 정점 중 dist가 가장 작은 정점을 선택하여 방문 처리하고, 해당 정점에서 방문 가능한 다른 방문하지 않은 정점으로 가는 최소 비용을 dist에 업데이트합니다.

Bellman-Ford



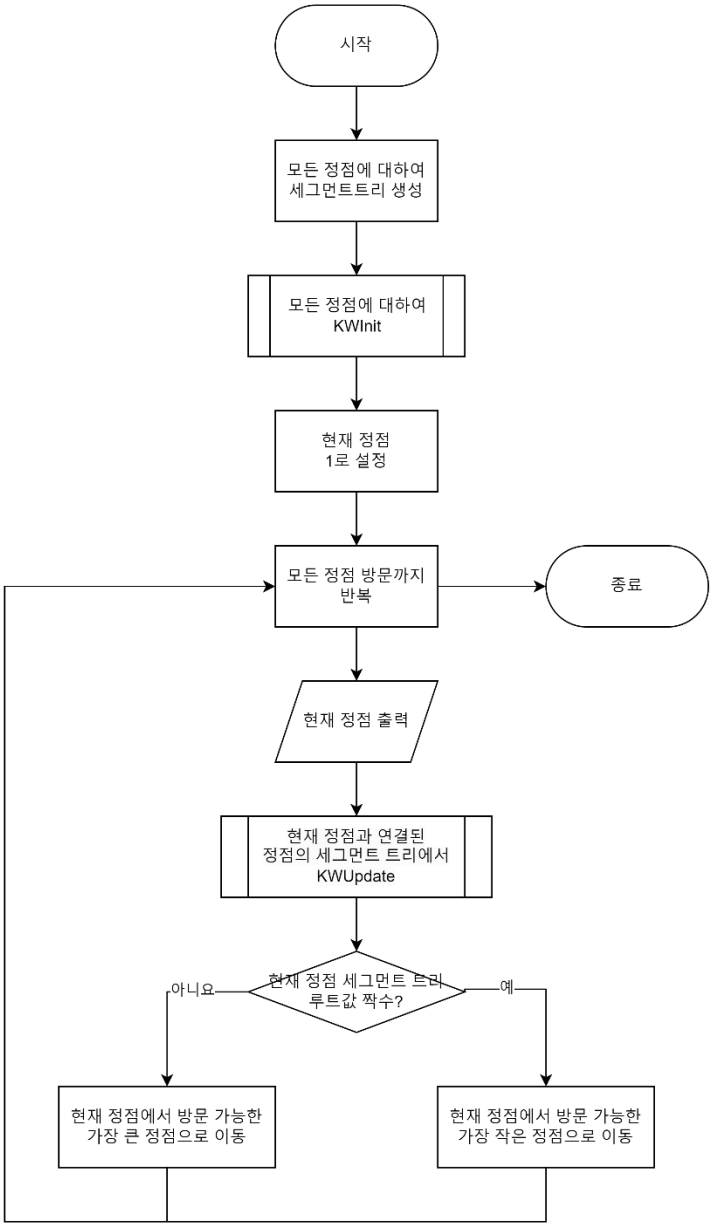
방향성 유무를 입력 받아 그래프를 복사합니다. 시작 정점을 기준으로 dist, path 배열을 생성, 초기화합니다. edge를 2개 사용할 때부터 n-1개 사용할 때까지 다음 작업을 반복합니다. 시작 정점을 제외한 모든 정점을 탐색하며 k-1개로 방문 가능한 정점일 때, 해당 정점에서 다른 정점으로 가는 비용이 기존 비용보다 더 적으면 최소 비용을 업데이트합니다. 반복이 끝난 뒤, 같은 작업을 한 번 더 진행하였을 때 최소 비용에 변화가 있으면 음수 사이클이 발생했다는 의미이므로 에러 코드를 출력합니다.

FLOYD



방향성 유무를 입력 받아 그래프를 복사합니다. dist 이차원 배열을 생성, 초기화합니다. k와 i, j를 각각 1부터 n까지 반복하며 모든 정점 쌍에 대한 최소비용을 찾습니다. 반복이 끝난 뒤 같은 반복문을 한 번 더 실행하여 변화가 감지되면 음수 사이클이 있다는 의미이므로 에러 코드를 출력합니다.

KwangWoon



각 정점에서 방문 가능한 정점의 개수를 세그먼트 트리로 구현합니다. 반복문의 시작부에서 현재 정점과 연결된 다른 정점의 세그먼트 트리에서 세그먼트 트리 업데이트를 진행하며 방문 가능한 정점의 수를 업데이트합니다. 현재 정점에서 방문 가능한 정점의 개수가 짝수이면 가장 작은 정점으로 이동하고, 홀수이면 가장 큰 정점으로 이동합니다.

C. Algorithm

BFS & DFS

각각 큐와 스택을 이용하여 그래프를 탐색합니다. 정점이 여러 개일 경우 작은 정점을 우선 탐색합니다.

Kruskal

최소 신장 트리를 구하는 알고리즘입니다. 현재 그래프에서 edge 를 한 배열에 저장하고 weight 오름차순으로 정렬합니다. 모든 edge 에 대하여 weight 가 작은 edge 부터 차례대로 탐색합니다. edge 에 대하여 사이클 형성 여부를 판단하고 사이클이 형성되지 않으면 MST 에 추가합니다. 사이클 형성 여부는 disjoint set 을 이용해 판단합니다.

Dijkstra

시작 정점에서 다른 정점으로 가는 최소 비용을 구하는 방법입니다. 현재 상황에서 최선의 선택을 하는 그리디 알고리즘입니다. 현재 방문한 정점들의 인접 정점으로 가는 비용이 가장 작은 정점을 추가하는 것을 반복합니다. 추가한 뒤에는 해당 정점의 인접 정점으로 가는 비용을 업데이트합니다.

Bellman-Ford

시작 정점에서 다른 정점으로 가는 최소 비용을 구하는 방법입니다. 사용 가능한 edge 의 개수를 점점 늘려가면서 최소 비용을 구하는 알고리즘입니다. 이전 edge 개수의 최소 비용과 새로 만들어지는 비용을 비교하여 더 적은 비용을 최소 비용으로 업데이트하는 것을 반복합니다.

FLOYD

모든 정점의 쌍들에 대해 최소 비용을 구할 수 있는 방법입니다. 시작 정점, 중간 정점, 도착 정점에 대하여 모두 반복합니다. 시작 정점에서 바로 도착 정점으로 가는 최소 비용과 시작 정점에서 중간 정점을 거쳐 도착 정점으로 가는 최소 비용을 비교하여 더 적은 비용으로 업데이트하는 과정을 반복합니다.

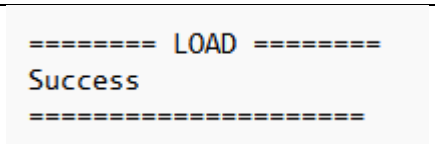
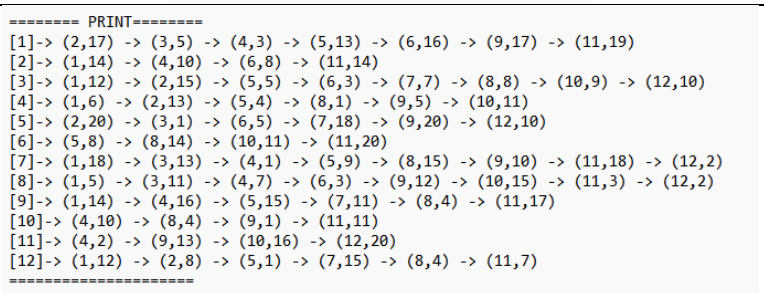
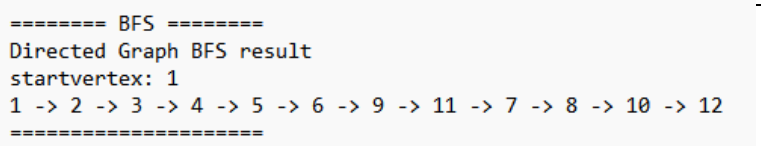
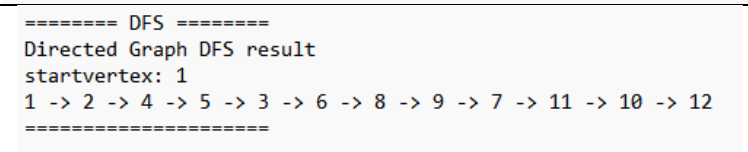
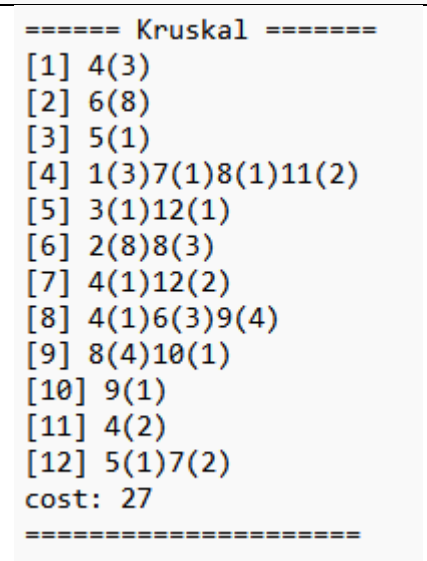
KwangWoon

이동 가능한 정점의 개수를 이용하여 그래프를 탐색하는 방법입니다. 현재 정점에서 이동 가능한 정점의 개수는 세그먼트 트리를 이용해 계산합니다. 이동 가능한 정점의 개수가 짝수이면 갈 수 있는 가장 작은 정점으로 이동하고, 홀수이면 가장 큰 정점으로 이동합니다. 이동한 뒤에는 인접 정점의 세그먼트 트리를 업데이트하는 것을 반복합니다.

My Quick Sort

배열의 크기가 6 이하이면 삽입 정렬을 진행하고, 6 보다 크면 피봇을 기준으로 분할하여 왼쪽 부분과 오른쪽 부분에 대하여 정렬 함수를 재귀적으로 실행합니다.

D. Result Screen

결과 화면	설명
	LOAD graph_L.txt 의 결과입니다.
	PRINT 의 결과입니다.
	BFS Y 1 의 결과입니다.
	DFS Y 1 의 결과입니다.
	KRUSKAL 알고리즘으로 구한 최소 비용 신장 트리 (MST)입니다.

`` ====== Kruskal ======= [1] 8(5) [2] 6(8) [3] 5(1)6(3) [4] 7(1)8(1)11(2) [5] 3(1)12(1) [6] 2(8)3(3) [7] 4(1)12(2) [8] 1(5)4(1)9(4) [9] 8(4)10(1) [10] 9(1) [11] 4(2) [12] 5(1)7(2) cost: 29 ==================== ```	위 KRUSKAL 그래프에서 1->4 간선의 가중치를 50 으로 입력했을 때 결과가 달라지는 것을 확인할 수 있습니다.
`` ====== Dijkstra ======= Directed Graph Dijkstra result startvertex: 1 [2] 1 -> 4 -> 8 -> 12 -> 2(14) [3] 1 -> 3(5) [4] 1 -> 4(3) [5] 1 -> 4 -> 5(7) [6] 1 -> 4 -> 8 -> 6(7) [7] 1 -> 3 -> 7(12) [8] 1 -> 4 -> 8(4) [9] 1 -> 4 -> 9(8) [10] 1 -> 4 -> 10(14) [11] 1 -> 4 -> 8 -> 11(7) [12] 1 -> 4 -> 8 -> 12(6) ==================== ```	DIJKSTRA 알고리즘으로 구한 방향이 있는 그래프의 각 정점까지의 최단 경로입니다.
`` ====== Dijkstra ======= Directed Graph Dijkstra result startvertex: 1 [2] 1 -> 2(17) [3] 1 -> 3(5) [4] 1 -> 3 -> 7 -> 4(13) [5] 1 -> 3 -> 5(10) [6] 1 -> 3 -> 6(8) [7] 1 -> 3 -> 7(12) [8] 1 -> 3 -> 8(13) [9] 1 -> 3 -> 10 -> 9(15) [10] 1 -> 3 -> 10(14) [11] 1 -> 3 -> 8 -> 11(16) [12] 1 -> 3 -> 7 -> 12(14) ==================== ```	비교 1 위 DIJKSTRA 그래프에서 1->4 간선의 가중치를 50 으로 입력했을 때 결과가 달라지는 것을 확인할 수 있습니다.

<table>
<tr>
<td>

```
====== Dijkstra =======
Undirected Graph Dijkstra result
startvertex: 1
[2] 1 -> 4 -> 7 -> 12 -> 2(14)
[3] 1 -> 3(5)
[4] 1 -> 4(3)
[5] 1 -> 4 -> 5(7)
[6] 1 -> 4 -> 8 -> 6(7)
[7] 1 -> 4 -> 7(4)
[8] 1 -> 4 -> 8(4)
[9] 1 -> 4 -> 9(8)
[10] 1 -> 4 -> 9 -> 10(9)
[11] 1 -> 4 -> 11(5)
[12] 1 -> 4 -> 7 -> 12(6)
======================
```

</td>
<td>비교 2
Dijkstra 알고리즘으로 위 그래프와 같은 데이터의 무방향 그래프의 각 정점까지의 최단 경로입니다.
몇몇 정점에 대해 경로가 짧아진 것을 확인할 수 있습니다.</td>
</tr>
<tr>
<td>

```
======= FLOYD =======
Directed Graph FLOYD result
       [1]    [2]    [3]    [4]    [5]    [6]    [7]    [8]    [9]    [10]   [11]   [12]
[1]    0      14     5      3      7      7      12     4      8      14     7      6
[2]    14     0      15     10     14     8      22     11     15     19     14     13
[3]    12     15     0      8      5      3      7      8      10     9      11     9
[4]    6      11     5      0      4      4      12     1      5      11     4      3
[5]    13     16     1      9      0      4      8      9      11     10     12     10
[6]    19     24     9      17     8      0      16     14     12     11     17     16
[7]    7      10     4      1      3      5      0      2      6      12     5      2
[8]    5      10     4      5      3      3      11     0      10     13     3      2
[9]    9      14     8      9      7      7      11     4      0      17     7      6
[10]   9      14     8      9      7      7      12     4      1      0      7      6
[11]   8      13     7      2      6      6      14     3      7      13     0      5
[12]   9      8      2      9      1      5      9      4      12     11     7      0
==========================
```

</td>
<td>FLOYD Y 의 결과입니다.
모든 정점의 쌍에 대하여 최단 경로를 알 수 있습니다.</td>
</tr>
<tr>
<td>

```
======= FLOYD =======
Directed Graph FLOYD result
       [1]    [2]    [3]    [4]    [5]    [6]    [7]    [8]    [9]    [10]   [11]   [12]
[1]    0      17     5      13     10     8      12     13     15     14     16     14
[2]    14     0      15     10     14     8      22     11     15     19     14     13
[3]    12     15     0      8      5      3      7      8      10     9      11     9
[4]    6      11     5      0      4      4      12     1      5      11     4      3
[5]    13     16     1      9      0      4      8      9      11     10     12     10
[6]    19     24     9      17     8      0      16     14     12     11     17     16
[7]    7      10     4      1      3      5      0      2      6      12     5      2
[8]    5      10     4      5      3      3      11     0      10     13     3      2
[9]    9      14     8      9      7      7      11     4      0      17     7      6
[10]   9      14     8      9      7      7      12     4      1      0      7      6
[11]   8      13     7      2      6      6      14     3      7      13     0      5
[12]   9      8      2      9      1      5      9      4      12     11     7      0
==========================
```

</td>
<td>비교 1
위 FLOYD 그래프에서 1->4 간선의 가중치를 50 으로 입력했을 때 결과가 달라지는 것을 확인할 수 있습니다.</td>
</tr>
<tr>
<td>

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

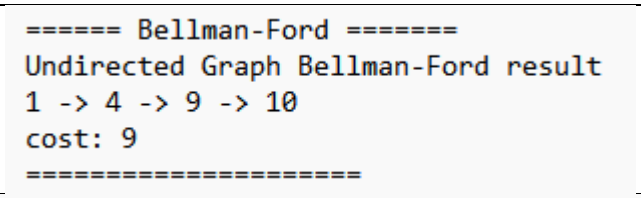
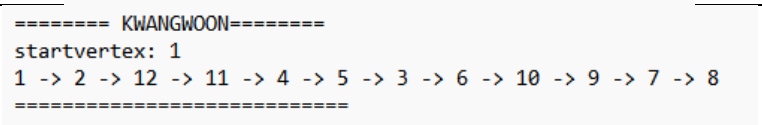
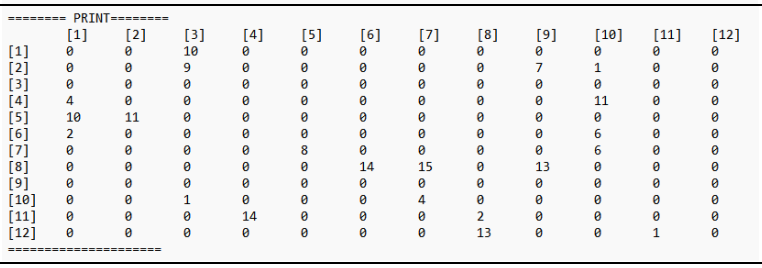
</td>
<td>비교 2
FLOYD 알고리즘으로 위 그래프와 같은 데이터의 무방향 그래프의 최단 경로를 구한 결과입니다.</td>
</tr>
<tr>
<td>

```
====== Bellman-Ford =======
Directed Graph Bellman-Ford result
1 -> 4 -> 8
cost: 4
======================
```

</td>
<td>BELLMANFORD Y 1 8 의 결과입니다</td>
</tr>
<tr>
<td>

```
====== Bellman-Ford =======
Directed Graph Bellman-Ford result
1 -> 3 -> 8
cost: 13
======================
```

</td>
<td>위 Bellman-ford 그래프에서 1->4 간선의 가중치를 50 으로 입력했을 때 결과가 달라지는 것을 확인할 수 있습니다.</td>
</tr>
</table>

	Bellman-Ford 알고리즘으로 구한 무방향 그래프로 1 에서 10 까지 가는 최단 경로의 결과입니다
	KWANGWOON 의 결과입니다.
	Matrix 형식의 데이터를 LOAD 한 뒤 PRINT 한 결과입니다.

E. Consideration

처음 과제 스펙을 보고 방향, 무방향을 구분하는 알고리즘에 대해서는 방향과 무방향 2개의 알고리즘을 만들어야 된다고 생각했습니다. 하지만 Graph의 함수를 구현하는 중 무방향 그래프에 대해서는 같은 가중치를 가지는 뒤집힌 방향의 edge를 추가하는 것으로 무방향 그래프를 표현할 수 있다는 것을 알 수 있었습니다. 이후 방향을 고려한 알고리즘만 구현하여 궁금증을 해결하였습니다.

광운 알고리즘을 구현하며 처음에 방향성을 완전 잘못 잡아버려 문제가 생겼었습니다. kw_graph에 세그먼트 트리를 구현하는 것으로 이해하고 구현하던 도중 실습 자료에 각 정점마다 세그먼트 트리를 구성한다 라는 내용이 있는 것을 확인했습니다. 구현한 내용을 완전 갈아엎고 실습 자료와 과제 스펙을 한 번 더 본 뒤 구현하였습니다.

마지막으로 결과를 동기들과 맞춰보는 도중 크루스칼의 결과가 조금씩 달랐습니다. 생각을 해보니 정렬 알고리즘이 stable한지 아닌지에 따라서 edge를 선택하는 순서가 조금씩 다르고 그것이 결과에 영향을 주는 것으로 판단하였습니다. 이를 통해 Local optimum이 발생하는 이유를 알 수 있었습니다.