데이터구조설계

DS_Project3 보고서

이름: 양서은

학번: 2022202061

학과: 컴퓨터정보공학부

교수님: 최상호 교수님

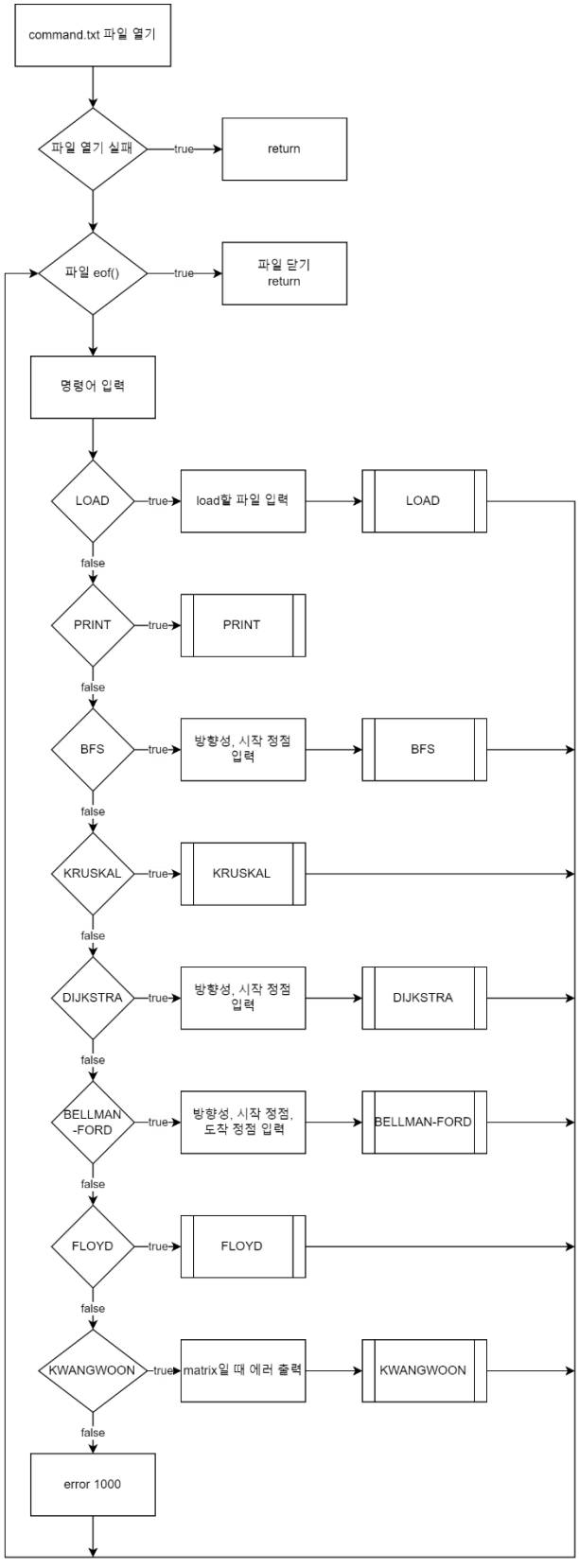
1. Introduction

그래프를 사용하여 그래프 연산 프로그램을 구현한다. 텍스트 파일로부터 프로그램 명령어와 그래프 정보를 입력 받고, command.txt 파일의 명령어를 순서대로 처리하며 결과를 출력한다. BFS, DFS, KRUSKAL, DIJKSTRA, BELLMAN-FORD, FLOYD, KWANGWOON, 7개의 알고리즘을 통해 그래프 연산을 수행한다. 프로그램으로부터 그래프 정보를 입력 받을 땐 list 형식과 matrix 형식으로 입력 받을 수 있다. 기본적으로 방향성과 가중치를 가지고 있지만 그래프 연산 형식에 따라 방향성 고려 여부와 가중치 고려 여부가 결정된다.
KRUSKAL 알고리즘의 경우 그래프의 모든 연결선을 가중치를 기준으로 정렬하는 과정이 필요하다. 이를 위해 Edge 클래스를 선언하여 연결선의 가중치, 연결선을 이루는 두 정점을 저장하고 quick sort, insertion sort 알고리즘을 활용해 정렬 알고리즘을 구현한다.

이 프로젝트는 텍스트 파일로부터 읽은 그래프 정보를 활용하여 다양한 그래프 연산을 수행하는 프로그램을 개발하는 것을 목표로 한다. 그래프 정보는 List 그래프 및 Matrix 그래프 두 가지 형식으로 저장된다. List 그래프는 edge의 출발 정점 및 도착 정점 및 가중치로 이루어진 데이터로 구성된다. Matrix 그래프는 행렬 형태로 저장되어 있으며, 각 행과 열은 각각 출발 정점과 도착 정점을 나타내고, 행렬 값은 해당 edge의 가중치를 나타낸다.

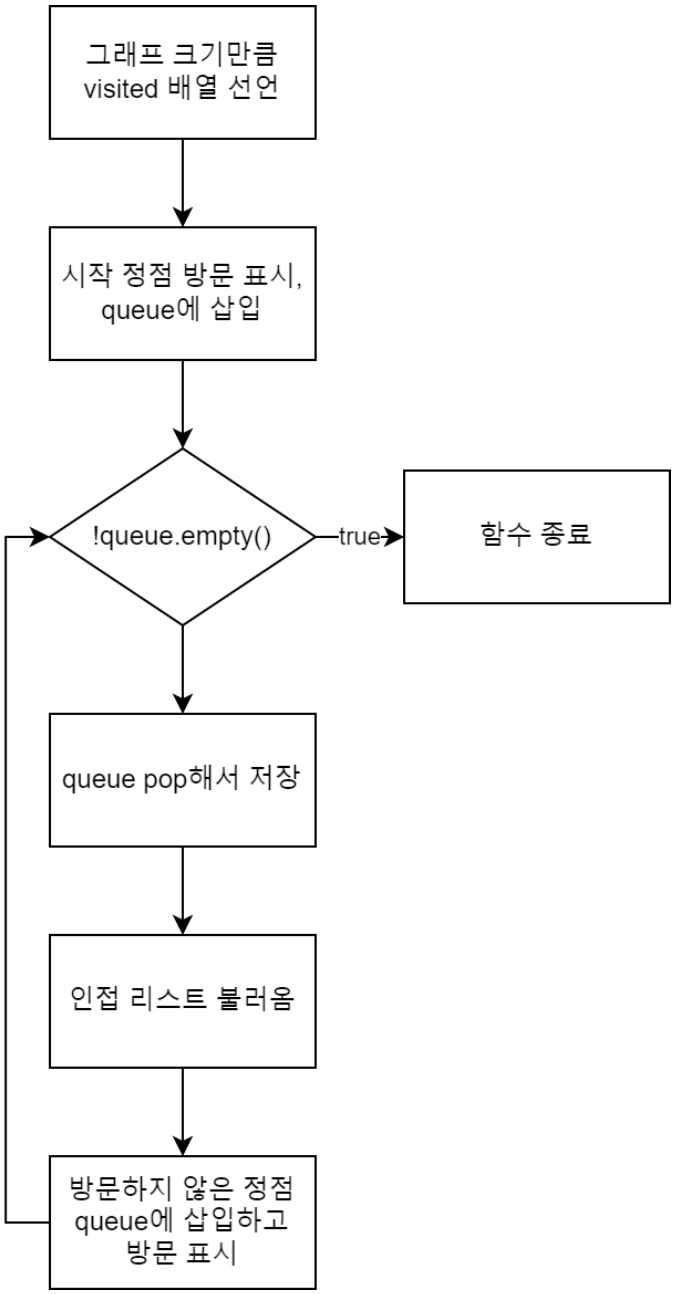
BFS와 DFS는 방향성을 고려한 그래프 순회 및 탐색 알고리즘이다. Kruskal은 무방향 그래프에서 최소 비용 신장 트리를 만드는 알고리즘이다. Dijkstra와 Bellman-Ford는 최단 경로를 찾는 알고리즘으로 방향성과 가중치를 고려한다. Floyd는 모든 정점 쌍에 대한 최단 경로를 찾는 알고리즘이다. KwangWoon은 무방향 List 그래프에서 특정 조건에 따라 정점을 탐색하는 알고리즘이다.

2. Flowchart



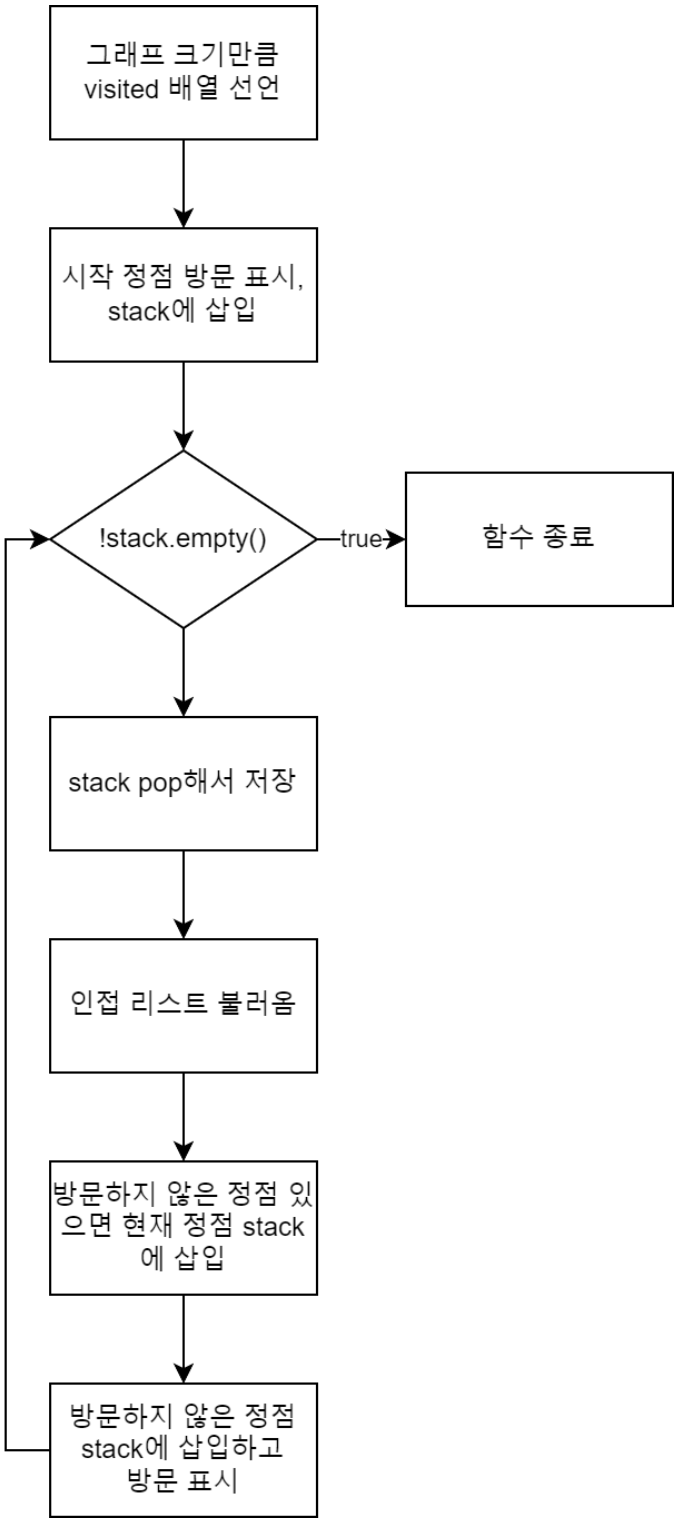
run

manager 멤버 함수 중 run 함수 명령어에 따라 선택하는 함수가 다름



BFS

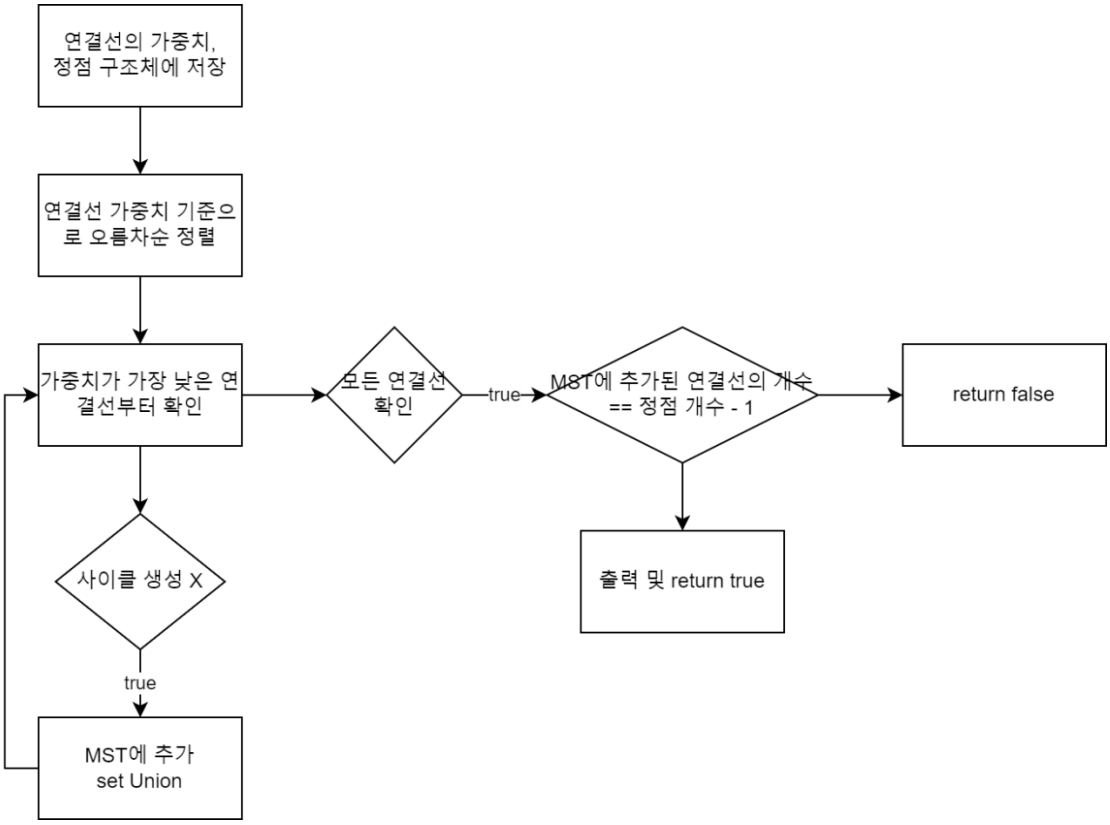
큐를 사용해 정점을 저장하고 방문 표시 배열을 선언해 정점 중복 방문을 방지함



DFS

stack을 통해 정점을 탐색하고 방문 표시 배열을 선언해 중복 탐색을 방지함

stack을 사용해 탐색할 경우 방문하지 않은 정점을 찾고 해당 정점을 stack에 삽입하기 전에 현재 정점을 stack에 먼저 삽입해야 함 (되돌아올 때 stack에서 pop하면서 다른 경로 탐색 가능)

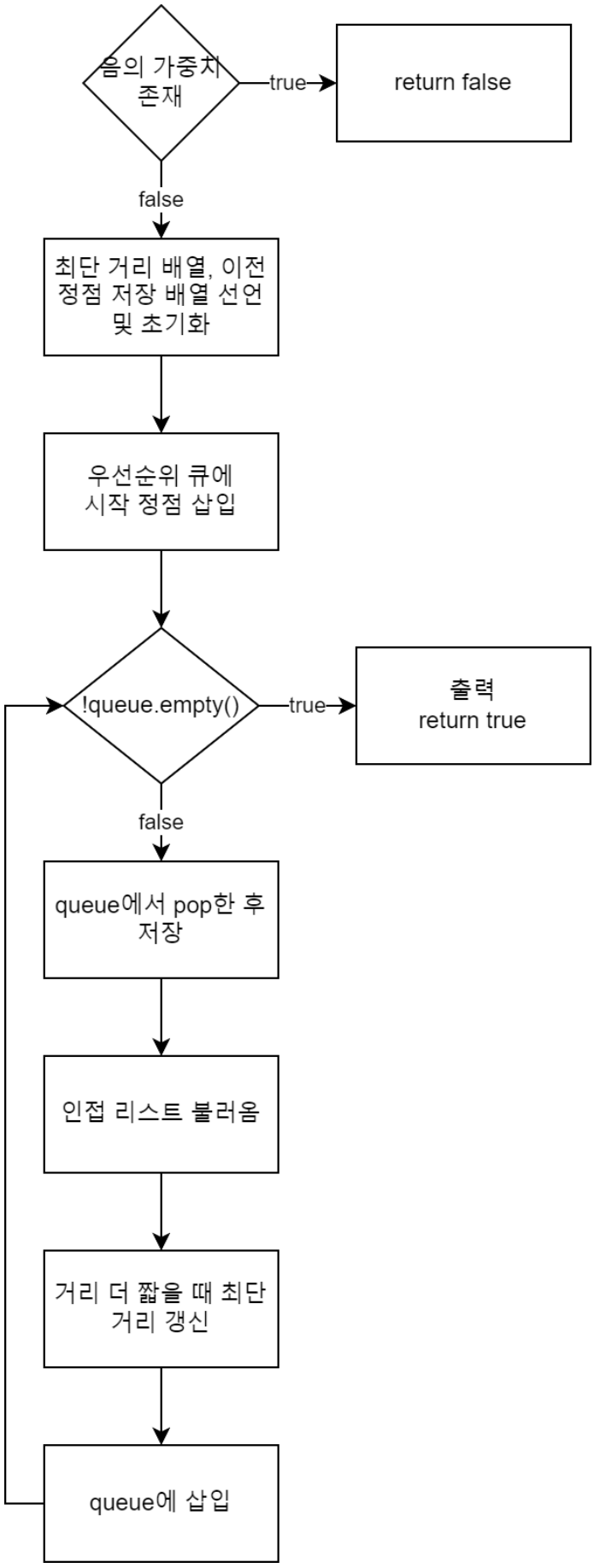


KRUSKAL

모든 연결선을 가중치를 기준으로 오름차순 정렬한 후 가장 낮은 연결선부터 확인

사이클을 생성하지 않을 경우 연결선을 MST에 추가한 후 연결선을 이루는 정점을 Union

모든 연결선을 확인한 후 MST에 추가된 연결선의 개수가 부족하면 MST를 만족하지 않기 때문에 false 반환

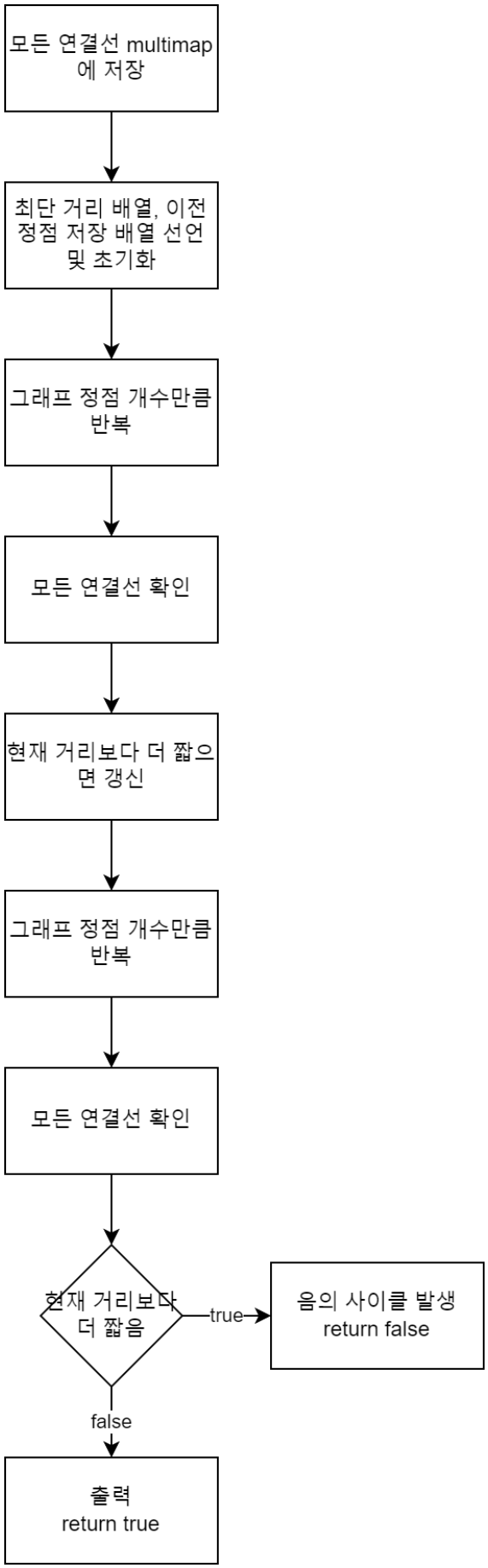


DIJKSTRA

음의 가중치를 가질 경우 동작하지 않음

우선 순위 큐에 최단거리를 key 값으로 넣어줌

큐에서 pop한 정점의 인접 리스트를 비교해 거리가 더 짧으면 최단 거리를 갱신

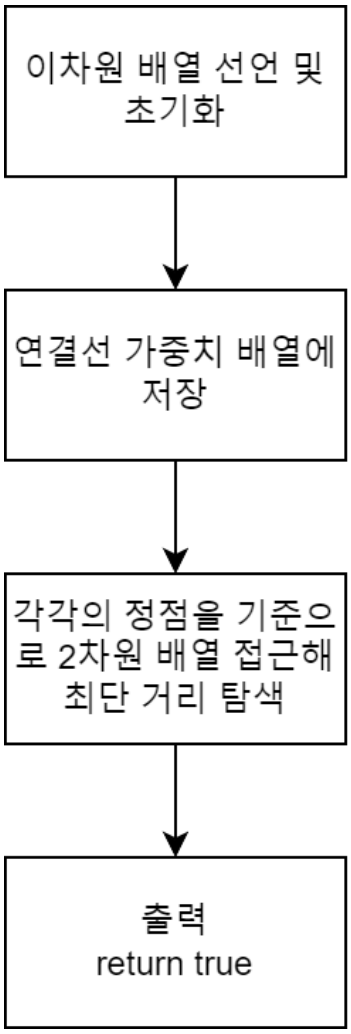


BELLMAN-FORD

모든 연결선을 그래프의 정점 개수만큼 계속 확인하며 최단거리를 탐색

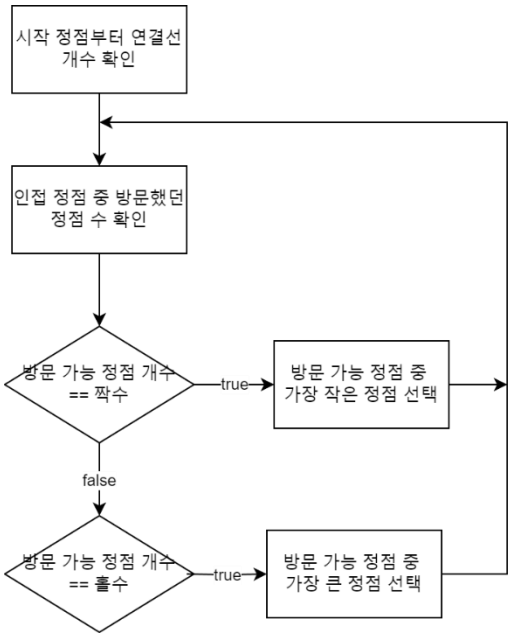
음의 가중치가 존재할 때도 탐색 가능

음의 사이클이 발생할 경우 탐색 종료



FLOYD

이차원 배열을 선언해 초기화해주고 3중 for문을 돌면서 각 정점을 기준으로 모든 정점 및 연결선을 확인하며 최단 거리를 갱신



KWANGWOON

정점에서 인접한 정점 중 방문 가능한 정점의 개수를 파악해 짝수일 때는 가장 작은 정점, 홀수일 때는 가장 큰 정점을 선택

3. Algorithm

1. BFS

   BFS는 Breadth-First Search의 약자로 너비 우선 탐색으로 불린다. 그래프나 트리의 탐색 알고리즘 중 하나이다. 시작 정점으로부터 차례대로 인접한 정점들을 모두 방문한 후 그 다음 단계의 인접 정점들을 차례대로 방문하는 방식이다. 한마디로 같은 레벨의 정점을 모두 방문한 후 다음 레벨로 넘어간다.
   Queue(큐)를 사용하여 구현하며 시작 정점을 큐에 넣고 방문 표시를 한다. 큐에서 정점을 하나씩 꺼내고 해당 정점과 인접해 있는 모든 정점을 큐에 넣고 방문 표시를 한다. 따라서 큐에서 나온 정점은 이미 방문한 것으로 간주한다. 큐가 비면 시작 정점과 연결되어 있는 모든 정점을 탐색한 것이다. 따라서 주어진 그래프 자료에서 2개 이상의 sub graph로 나누어진다면 시작하는 정점에 따라 탐색하지 못하는 정점이 발생할 수 있다.
   프로그램에서 BFS 탐색은 방향성은 고려하지만 가중치는 고려하지 않는다.

2. DFS

   DFS는 Depth-First Search의 약자로 깊이 우선 탐색으로 불린다. BFS와 마찬가지로 그래프나 트리의 탐색 알고리즘이고 프로그램에서 가중치를 고려하지 않는다. DFS는 시작 정점으로부터 임의의 연결선을 따라 경로 끝까지 탐색한 후, 되돌아와 다른 경로를 탐색한다. 즉, 가능한 깊이 들어가 더 이상 탐색할 수 없을 때까지 진행한 후, 다시 돌아와 다른 경로를 탐색한다.
   Stack(스택)이나 재귀함수를 사용해 구현한다. 본인은 이번 프로젝트에서 스택을 사용하여 구현했다.
   시작 정점을 스택에 넣고 방문 표시를 한다. 스택에서 정점을 꺼내고 인접한 정점 중 방문하지 않은 정점을 선택한다. 스택은 FIFO로 마지막에 삽입된 요소가 먼저 나오기 때문에 경로 끝까지 탐색하고 되돌아오기 위해 현재 정점을 스택에 다시 넣고 새 정점 또한 스택에 삽입한다. 그 후 스택이 빌 때까지 반복하면 시작 정점과 연결되어 있는 모든 정점을 탐색할 수 있다. BFS와 마찬가지로 인접 노드를 기준으로 그래프를 탐색하기 때문에 그래프 내에 sub graph가 2개 이상 존재할 경우 특정 정점만 탐색할 가능성이 존재한다.

3. KRUSKAL

   이번 프로젝트에서 제일 까다로웠던 알고리즘이었다.
   Kruskal 알고리즘은 최소 신장 트리(Minimum Spanning Tree), 즉 MST를 찾기 위한 그래프 알고리즘이다. MST는 그래프의 모든 정점을 포함하면서 연결선의 가중치 합이 최소인 트리이다. 가중치의 합을 최소한으로 줄이기 위해 그래프의 모든 연결선을

가중치를 기준으로 오름차순 정렬해야 한다. 이 과정에서 quick sort, insertion sort 알고리즘을 사용해 정렬한다. 가장 작은 가중치를 가지는 연결선부터 차례대로 선택해 나가면서 사이클이 형성되지 않도록 MST를 만든다.

사이클 형성 여부는 추가하고자 하는 연결선의 두 정점의 부모 정점 비교를 통해 확인할 수 있다. 부모 정점이 같다면 같은 요소로 이미 연결되어 있다고 판단해 사이클을 형성한다. 이처럼 비교를 위해 연결선을 추가할 때마다 두 정점의 부모 정점을 저장한다. 즉 Union끼리 하나의 트리를 구성하여 최종적으론 모두 한 개의 정점을 부모 정점으로 반환하게 된다. 부모 정점이 같지 않다면 사이클을 이루지 않으며 부모 정점을 합침으로써 한 요소로 통합된다. 모든 연결선을 확인하며 MST를 구성했을 때 연결선의 개수가 (그래프의 정점 개수)-1개가 아니라면 정상적인 MST를 구성하지 않았다고 판단한다. 따라서 주어진 그래프의 sub graph가 2개 이상이라면 Kruskal 알고리즘을 사용할 수 없다. 또한 해당 알고리즘에서 방향성은 고려하지 않는다.

4. DIJKSTRA

다익스트라 알고리즘은 그래프에서 최단 경로를 찾는 알고리즘 중 하나이다. BFS와 유사한 방식으로 동작하지만, 각 노드까지으 최단 경로를 찾는 데에 특화되어 있다. 음의 가중치가 없는 그래프에서만 사용되며 음의 가중치가 있을 경우 밸만 포드 알고리즘이나 다른 알고리즘을 사용해야 한다.

출발 정점을 선택하고 출발 정점으로부터 각 정점까지의 거리를 무한대로 초기화한다. 출발 정점 자체의 거리는 0으로 설정한다. 우선 순위 큐를 사용해 현재까지 알려진 최단 거리가 가장 작은 정점을 선택한다. 선택한 정점을 기반으로 인접한 정점들의 최단 거리를 갱신한다. 이 때 선택한 노드를 거쳐가는 경로가 더 짧다면 해당 정점의 최단 거리를 업데이트한다. 이 과정을 반복하여 모든 정점을 탐색한다. 서로 연결되어 있지 않은 정점이라면 탐색하지 못할 수 있다. 이 알고리즘은 연결선의 가중치가 음수라면 정확하지 않은 결과를 출력할 수 있다.

5. BELLMAN-FORD

밸만 포드 알고리즘은 음의 가중치를 포함하는 최단 경로를 찾는 알고리즘이다. 밸만 포드 알고리즘은 다익스트라 알고리즘보다 시간 복잡도가 높기 때문에 음의 가중치가 없는 경우에는 다익스트라 알고리즘을 선호한다. 밸만 포드는 모든 간선을 순회하며 최단 거리를 갱신하는 방식으로 동작한다. 출발 정점으로부터 모든 정점의 거리를 무한대로 초기화하고 출발 정점 자체의 거리를 0으로 설정한다. 모든 간선을 순회하며 현재까지 저장된 최단 거리를 이용하여 정점들의 최단 경로를 갱신한다. 이 때 갱신이 이루어지는 경우 해당 간선은 현재까지의 최단 경로를 포함하고 있다. 그래프의 정점 개수만큼 반복하여 최단 거리를 탐색한다.

밸만 포드 알고리즘은 음의 가중치와 음의 순환을 감지한다. 음의 순환이 이루어질 경우 최단 거리가 무한하게 감소할 수 있으며 이 때 알고리즘은 종료된다.

6. FLOYD

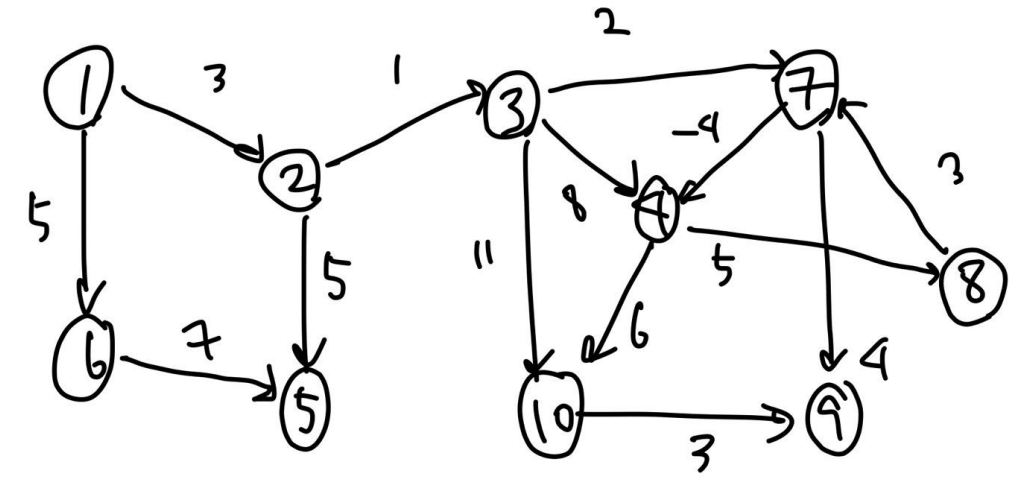
   Floyd-Warshall 알고리즘은 그래프에서 모든 쌍의 최단 경로를 찾는 동적 프로그래밍 기반의 알고리즘이다. 음의 가중치와 음의 순환도 처리할 수 있기 때문에 유용하게 사용할 수 있다. 모든 정점 쌍 간의 최단 거리를 저장하는 2차원 배열을 선언해 초기화해준다. 두 정점 사이에 직접적인 연결선이 있다면 해당 거리를, 없다면 무한대로 초기화한다. 이 때 자기 자신과의 연결은 0으로 초기화해준다. 모든 정점 쌍을 순회하면서 현재까지 알려진 최단 거리를 이용하여 거리를 갱신한다. k번째 정점을 경유하며 i에서 j까지 가는 거리와 현재까지 알려진 i에서 j까지의 거리를 비교해 더 짧은 거리로 업데이트한다.

7. KWANGWOON

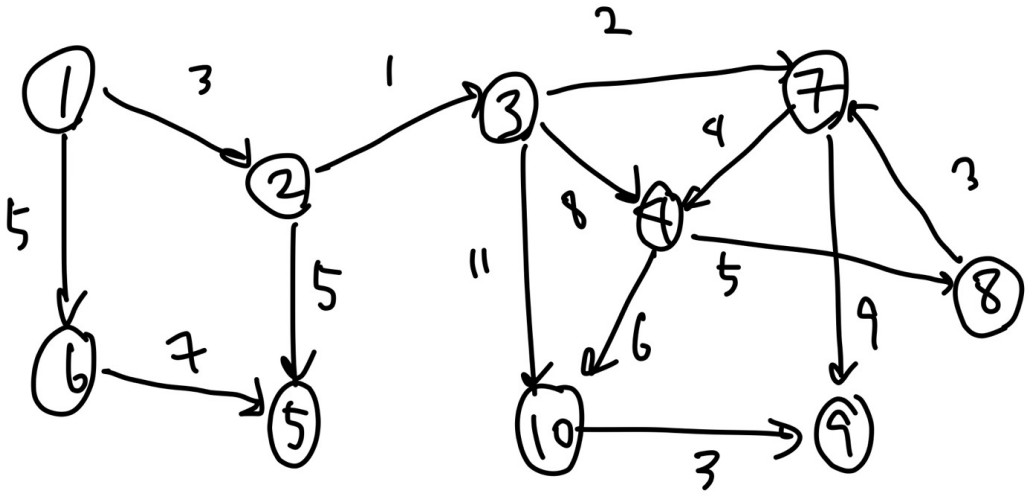
   그래프를 탐색할 때 무방향으로 가정하고 가중치는 고려하지 않는다. 시작 정점부터 방문할 수 있는 정점의 수가 짝수면 가장 작은 정점을 다음으로 방문하고 홀수면 가장 큰 정점을 다음으로 방문한다. 해당 알고리즘도 결국 정점으로부터 인접한 정점에 부여된 수를 기준으로 판단하기 때문에 정렬이 필요했고 insertion sort를 사용해 간단하게 정렬했다.
   세그먼트 트리를 사용해 다른 정점으로 이동할 수 있는지 확인한다. 이동 가능한 정점들에 한해서 트리를 구축한다. 따라서 탐색을 수행하면서 세그먼트 트리를 수정해야 한다. 정점과 연결선의 개수가 클 때 효율적으로 탐색할 수 있다.

4. Result Screen

graph_L.txt



graph_M.txt



LOAD graph_L.txt PRINT graph_L.txt 파일을 불러와 그래프 정보를 저장하고 양식에 맞게 출력	========LOAD======= Success ==================== ========PRINT======== [1] -> (2,3) -> (6,5) [2] -> (3,1) -> (5,5) [3] -> (4,8) -> (7,2) -> (10,11) [4] -> (8,5) -> (10,6) [5] [6] -> (5,7) [7] -> (4,-4) -> (9,4) [8] -> (7,3) [9] [10] -> (9,3) ====================
BFS Y 1 BFS N 1 방향성, 무방향성을 고려하여 그래프를 탐색 가중치는 고려하지 않음	========BFS======== Directed Graph BFS result startvertex: 1 1 -> 2 -> 6 -> 3 -> 5 -> 4 -> 7 -> 10 -> 8 -> 9 ==================== ========BFS======== Undirected Graph BFS result startvertex: 1 1 -> 2 -> 6 -> 3 -> 5 -> 4 -> 7 -> 10 -> 8 -> 9 ====================
DFS Y 1 DFS N 1 방향성, 무방향성을 고려하여 그래프를 탐색 가중치는 고려하지 않음	========DFS======== Directed Graph DFS result startvertex: 1 1 -> 2 -> 3 -> 4 -> 8 -> 7 -> 9 -> 10 -> 5 -> 6 ==================== ========DFS======== Undirected Graph DFS result startvertex: 1 1 -> 2 -> 3 -> 4 -> 7 -> 8 -> 9 -> 10 -> 5 -> 6 ====================

KRUSKAL MST를 구성하며 최소 cost를 계산 방향성을 고려하지 않음 각 정점을 기준으로 연결되어 있는 정점과 해당하는 가중치를 표시함	====== Kruskal ======= [1] 2(3) 6(5) [2] 1(3) 3(1) 5(5) [3] 2(1) 7(2) [4] 7(-4) [5] 2(5) [6] 1(5) [7] 3(2) 4(-4) 8(3) 9(4) [8] 7(3) [9] 7(4) 10(3) [10] 9(3) cost: 22 ======================
DIJKSTRA Y 1 음의 가중치를 가지기 때문에 탐색 불가	========ERROR======= 700 ======================
BELLMANFORD Y 1 10 BELLMANFORD N 1 8 방향성과 가중치를 고려해 시작 정점으로부터 도착 정점까지의 최단 거리 출력 무방향성일 경우 음의 가중치가 존재할 때 동작하지 않기 때문에 에러 코드 출력	====== Bellman-Ford ======= Directed Graph Bellman-Ford result 1 -> 2 -> 3 -> 7 -> 4 -> 10 cost: 8 ====================== ========ERROR======= 800 ======================

FLOYD Y
FLOYD N
 	
방향성과 무방향성을 고려해 최단거리를 탐색하고 출력
방향성의 경우 도달하지 못할 때 x로 출력
	``
	====== FLOYD =======
	Directed Graph FLOYD result
	    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
	[1] 0 3 4 2 8 5 6 7 10 8
	[2] x 0 1 -1 5 x 3 4 7 5
	[3] x x 0 -2 x x 2 3 6 4
	[4] x x x 0 x x 8 5 9 6
	[5] x x x x 0 x x x x x
	[6] x x x x 7 0 x x x x
	[7] x x x -4 x x 0 1 4 2
	[8] x x x -1 x x 3 0 7 5
	[9] x x x x x x x x 0 x
	[10] x x x x x x x x 3 0
	====================
	
	====== FLOYD =======
	Undirected Graph FLOYD result
	    [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
	[1] 0 3 4 2 8 5 6 7 10 8
	[2] 3 0 1 -1 5 8 3 4 7 5
	[3] 4 1 0 -2 6 9 2 3 6 4
	[4] 2 -1 -2 0 4 7 -4 -3 0 -2
	[5] 8 5 6 4 0 7 8 9 12 10
	[6] 5 8 9 7 7 0 11 12 15 13
	[7] 6 3 2 -4 8 11 0 1 4 2
	[8] 7 4 3 -3 9 12 1 0 5 3
	[9] 10 7 6 0 12 15 4 5 0 3
	[10] 8 5 4 -2 10 13 2 3 3 0
	====================
	``

KWANGWOON KWANGWOON 알고리즘에 맞게 정점 1부터 차례대로 탐색	``` ======== KWANGWOON ======== startvertex: 1 1 -> 2 -> 3 -> 10 -> 4 -> 7 -> 8 ============================== ```
LOAD graph_M.txt PRINT 저장되어 있던 그래프 정보를 삭제하고 graph_M.txt 파일에 저장되어 있는 정보 새로 저장 및 출력	``` ========LOAD======= Success ==================== ========PRINT======== [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [1] 0 3 0 0 0 5 0 0 0 0 [2] 0 0 1 0 5 0 0 0 0 0 [3] 0 0 0 8 0 0 2 0 0 11 [4] 0 0 0 0 0 0 0 5 0 6 [5] 0 0 0 0 0 0 0 0 0 0 [6] 0 0 0 0 7 0 0 0 0 0 [7] 0 0 0 4 0 0 0 0 9 0 [8] 0 0 0 0 0 0 3 0 0 0 [9] 0 0 0 0 0 0 0 0 0 0 [10] 0 0 0 0 0 0 0 0 3 0 ==================== ```

BFS Y 1 BFS N 1 방향성, 무방향성을 고려하여 그래프를 탐색 가중치는 고려하지 않음	========BFS======== Directed Graph BFS result startvertex: 1 1 -> 2 -> 6 -> 3 -> 5 -> 4 -> 7 -> 10 -> 8 -> 9 ===================== ========BFS======== Undirected Graph BFS result startvertex: 1 1 -> 2 -> 6 -> 3 -> 5 -> 4 -> 7 -> 10 -> 8 -> 9 =====================
DFS Y 1 DFS N 1 방향성, 무방향성을 고려하여 그래프를 탐색 가중치는 고려하지 않음	========DFS======== Directed Graph DFS result startvertex: 1 1 -> 2 -> 3 -> 4 -> 8 -> 7 -> 9 -> 10 -> 5 -> 6 ===================== ========DFS======== Undirected Graph DFS result startvertex: 1 1 -> 2 -> 3 -> 4 -> 7 -> 8 -> 9 -> 10 -> 5 -> 6 =====================
KRUSKAL MST를 구성하며 최소 cost를 계산 방향성을 고려하지 않음 각 정점을 기준으로 연결되어 있는 정점과 해당하는 가중치를 표시함	====== Kruskal ======= [1] 2(3) 6(5) [2] 1(3) 3(1) 5(5) [3] 2(1) 7(2) [4] 7(4) 10(6) [5] 2(5) [6] 1(5) [7] 3(2) 4(4) 8(3) [8] 7(3) [9] 10(3) [10] 4(6) 9(3) cost: 32 ======================

DIJKSTRA Y 1 DIJKSTRA N 1 음의 가중치가 존재하지 않으므로 그래프 탐색 가능 시작 정점으로부터 각 정점까지의 최단 거리를 탐색해 경로와 함께 출력 방향성을 고려하지 않을 경우 방향성을 고려할 때보다 더 짧은 최단거리를 탐색할 수 있음	====== Dijkstra ======= Directed Graph Dijkstra result startvertex: 1 [2] 1 -> 2 (3) [3] 1 -> 2 -> 3 (4) [4] 1 -> 2 -> 3 -> 4 (12) [5] 1 -> 2 -> 5 (8) [6] 1 -> 6 (5) [7] 1 -> 2 -> 3 -> 7 (6) [8] 1 -> 2 -> 3 -> 4 -> 8 (17) [9] 1 -> 2 -> 3 -> 7 -> 9 (15) [10] 1 -> 2 -> 3 -> 10 (15) ========== ====== Dijkstra ======= Undirected Graph Dijkstra result startvertex: 1 [2] 1 -> 2 (3) [3] 1 -> 2 -> 3 (4) [4] 1 -> 2 -> 3 -> 4 (12) [5] 1 -> 2 -> 5 (8) [6] 1 -> 6 (5) [7] 1 -> 2 -> 3 -> 7 (6) [8] 1 -> 2 -> 3 -> 7 -> 8 (9) [9] 1 -> 2 -> 3 -> 7 -> 9 (15) [10] 1 -> 2 -> 3 -> 10 (15) ==========

<table>
<tr>
<td>BELLMANFORD Y 1 7
BELLMANFORD N 1 10

시작 정점으로부터 도착 정점까지의 최단거리를 탐색하고 경로와 함께 출력</td>
<td>====== Bellman-Ford =======
Directed Graph Bellman-Ford result
1 -> 2 -> 3 -> 7
cost: 6
======================

====== Bellman-Ford =======
Undirected Graph Bellman-Ford result
1 -> 2 -> 3 -> 10
cost: 15
======================</td>
</tr>
<tr>
<td>FLOYD Y
FLOYD N

방향성과 무방향성을 고려해 최단거리를 탐색하고 출력
방향성의 경우 도달하지 못할 때 x로 출력</td>
<td>====== FLOYD =======
Directed Graph FLOYD result
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
[1] 0 3 4 10 8 5 6 15 15 15
[2] x 0 1 7 5 x 3 12 12 12
[3] x x 0 6 x x 2 11 11 11
[4] x x x 0 x x 8 5 9 6
[5] x x x x 0 x x x x x
[6] x x x x 7 0 x x x x
[7] x x x 4 x x 0 9 9 10
[8] x x x 7 x x 3 0 12 13
[9] x x x x x x x x 0 x
[10] x x x x x x x x 3 0
====================

====== FLOYD =======
Undirected Graph FLOYD result
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
[1] 0 3 4 10 8 5 6 9 15 15
[2] 3 0 1 7 5 8 3 6 12 12
[3] 4 1 0 6 6 9 2 5 11 11
[4] 10 7 6 0 12 15 4 5 9 6
[5] 8 5 6 12 0 7 8 11 17 17
[6] 5 8 9 15 7 0 11 14 20 20
[7] 6 3 2 4 8 11 0 3 9 10
[8] 9 6 5 5 11 14 3 0 12 11
[9] 15 12 11 9 17 20 9 12 0 3
[10] 15 12 11 6 17 20 10 11 3 0
====================</td>
</tr>
<tr>
<td>KWANGWOON

matrix 형태로 저장되어 있기 때문에 탐색하지 않음</td>
<td>========ERROR=======
500
======================</td>
</tr>
<tr>
<td>EXIT</td>
<td>===== EXIT =====
Success
================</td>
</tr>
</table>

5. Consideration

1차, 2차 프로젝트에 비해 어렵지 않게 구현할 수 있었다. 그래프 탐색 알고리즘을 사용해 그래프를 탐색하는 것이 주 기능이었기 때문에 수업 자료 및 여러 오픈소스를 참고할 수 있었다.

그래프 탐색 알고리즘보다 정렬 함수를 구현하는 것이 더 어려웠다. KRUSKAL 알고리즘을 사용해 그래프를 탐색할 때 모든 연결선을 가중치를 기준으로 오름차순 정렬을 해야 한다. 프로젝트 제안서를 제대로 이해하지 않았을 땐 multimap을 사용해 가중치를 key 값으로 연결된 정점을 element로 저장하는 방식을 생각했다. 실제로 구현했을 때 그래프 탐색이 정상적으로 동작함을 확인했다. 하지만 제안서를 다시 읽어봤을 때 sorting algorithm을 사용해 연결선을 정렬해야 됨을 깨달았고 다시 구현했다. 지금까지 정렬 알고리즘을 구현했을 땐 한 가지 방법만 사용했기 때문에 Insertion sort와 quick sort를 같이 사용한다는 개념을 이해하기가 어려웠다. 하지만 주어진 의사 코드를 토대로 고민했을 때 quick sort를 사용해 분할하다 배열의 크기 7보다 작아지면 insertion sort를 사용해 정렬한다. 따라서 배열의 크기가 1,2가 될 때까지 분할하지 않아도 된다는 장점이 생긴다. 데이터구조 기말고사를 준비할 때 STL sort 함수가 이와 같이 동작한다는 것을 알았고 마침내 출제자의 의도를 알아챌 수 있었다. 정렬 알고리즘을 직접 구현하고 multimap과 동일한 결과를 출력하는 것을 확인해 올바르게 구현함을 확인했다.

Visual Studio에서 코드를 짜고 리눅스로 옮겨 프로그램을 실행했는데 FLOYD 기능만 동작하지 않는 것을 확인했다. 올바르지 않은 입력으로 처리되어 900 error가 출력되어 다시 코드를 살펴보았고 getline 함수를 사용하면서 생긴 윈도우와 리눅스의 차이인 것으로 확인되었다. >> 연산자를 사용해 텍스트 파일로부터 직접 받아오도록 하여 정상적으로 동작하게 수정했다.

이번 학기 다른 전공 과목과 같이 공부하면서 과제의 양이나 난이도가 상당해 정말 힘들었는데 그래도 잘 마무리하게 된 거 같아 뿌듯하다. 이 경험을 바탕으로 앞으로도 열심히 공부하고 성장해 나가고 싶다.