Data Structure Project #3

2025년 11월 10일

Due date: 2025년 12월 7일 일요일 23:59:59까지

본 프로젝트에서는 그래프 관련 알고리즘을 수행하는 프로그램을 구현한다. 이 프로그램은 그래프 정보가 저장된 텍스트 파일을 읽어 그래프를 생성하고, 명령어가 저장된 텍스트 파일을 읽어 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford, FLOYD 알고리즘과 정점의 근접 중심성(closeness centrality)을 계산하는 연산을 수행한다. 주어지는 그래프 데이터는 기본적으로 방향성(direction)과 가중치(weight)를 모두 가지고 있으며, 그래프 데이터 형태에 따라 인접 리스트(adjacency list)와 인접 행렬(adjacency matrix)로 저장되어 있다. 하지만, 알고리즘의 특성에 따라 방향성 및 가중치의 적용 여부가 달라질 수 있다. Figure 1은 그래프 연산 프로그램의 구조이다. 그래프 저장 형태에 따른 그래프 구축 방법과 각 알고리즘에 대한 설명은 **program implementation**에서 설명한다.

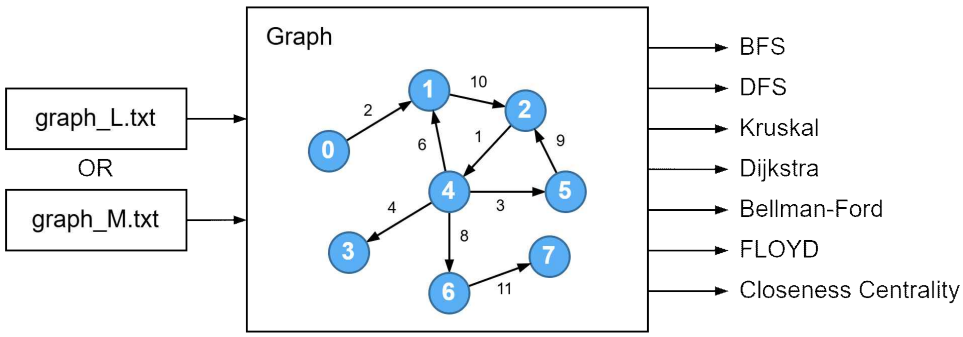

Figure 1. 그래프 연산 프로그램 구조

□ Program implementation

1) 그래프 정보 데이터

- 프로그램은 그래프 저장 유형, 정점 개수, 간선 연결 정보, 간선 가중치 정보가 저장된 파일 graph_L.txt 또는 graph_M.txt를 LOAD 명령어를 사용해 읽고 각 정보를 활용해 그래프를 생성한다.
- graph_L.txt와 graph_M.txt 모두 첫 번째 줄에는 그래프 저장 유형(L: List, M: Matrix)이 저장되어 있고, 두 번째 줄에는 정점의 개수가 저장되어 있다. 이후 데이터는 그래프 저장 유형에 따라 구분된다.
- 그래프의 모든 정점의 번호는 0 이상의 정수이며, 0부터 시작하여 0, 1, 2, 3... 순으로 정해지고, 적어도 5개의 정점이 입력된다고 가정한다.
- 간선의 가중치는 음의 정수 또는 양의 정수이며, 가중치가 0인 간선은 없다고 가정한다.
- 인접 리스트(adjacency list)는 Figure 2와 같이 간선의 출발 정점을 의미하는 줄과 간선의 도착 정점 및 가중치를 의미하는 줄로 구성되어 있으며, 만약 간선이 없는 정점의 경우 출발 정점을 의미하는 줄만 존재한다.
  - ✓ 각 줄의 개행 문자는 Line Feed('\n')이며, 간선의 도착 정점 및 가중치는 공백(' ')으로 구분한다.

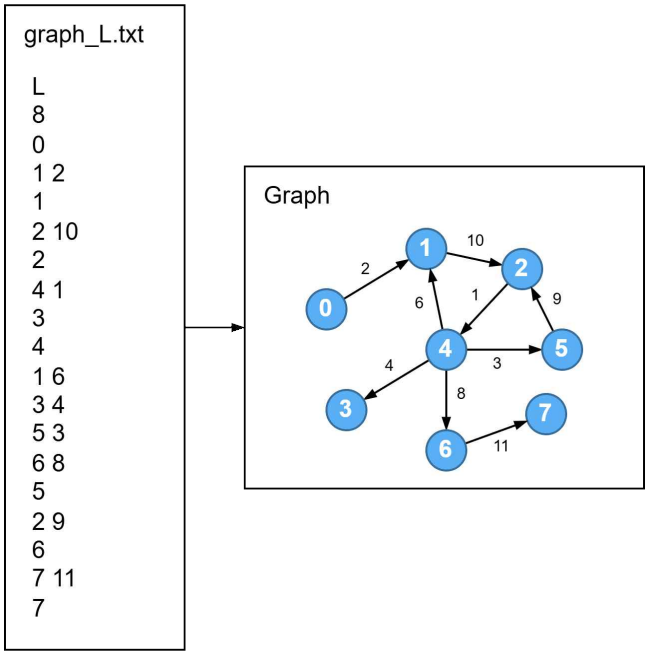

Figure 2. 인접 리스트 그래프 파일 구성

- 인접 행렬(adjacency matrix)는 Figure 3과 같이 간선 정보가 행렬 형태로 저장되어 있으며 행(row)은 출발 정점, 열(column)은 도착 정점, 행렬의 값(value)은 간선의 가중치를 의미한다. 만약 정점 사이 간선이 없는 경우 행렬의 값이 0으로 저장된다.
  ✓ 각 줄의 개행 문자는 Line Feed('\n')이며, 행렬의 값은 하나의 공백(' ')으로 구분한다.

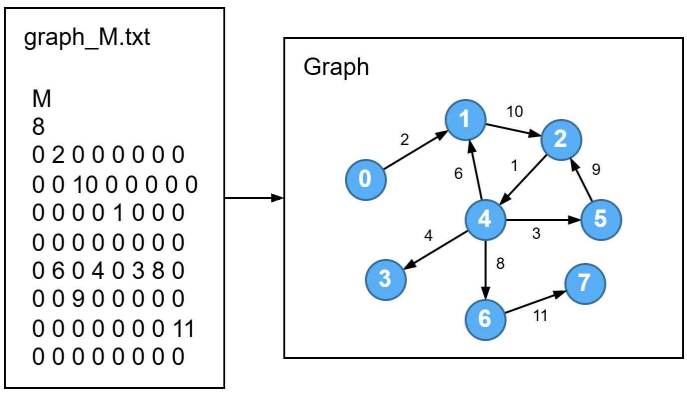

Figure 3. 인접 행렬 그래프 파일 구성

- 텍스트 파일에서 불러온 그래프 정보 중 그래프 저장 유형 및 크기(정점 개수)는 'Graph' 클래스에 저장하며, 그래프 연결 정보는 그래프 저장 유형에 따라 'ListGraph' 또는 'MatrixGraph'에 저장한다.

2) 그래프 연산

1. BFS & DFS (방향성 O/X, 가중치 X)
   - BFS 및 DFS 명령어는 방향성 적용 여부만 고려하여 그래프 탐색을 수행한다.
   - 프로그램은 각 명령어(BFS O 1, DFS X 2)에 포함된 시작 정점에서 탐색하기 시작하여 방문 가능한 모든 정점을 방문하고, 정점 방문 순서를 로그 파일에 출력한다.
   - 그래프와 명령어의 구성에 따라 방문 되지 않는 정점이 존재할 수 있다.
   - 한 정점에서 방문할 수 있는 정점이 여러 개일 경우, 번호가 가장 낮은 정점을 먼저 방문한다고 가정한다.

2. Kruskal (방향성 X, 가중치 O)
   - KRUSKAL 명령어는 방향성이 없고, 가중치만 고려하여 알고리즘을 수행한다.
   - 프로그램은 KRUSKAL 명령어 수행 시, 저장된 그래프 정보를 이용하여 그래프의 최소 신장 트리 (Minimum Spanning Tree; MST)를 구하고, 이를 로그 파일에 출력한다.
   - 가중치 기준 간선 오름차순 정렬 시, 가중치가 같은 케이스는 없다고 가정한다.

3. Dijkstra (방향성 O/X, 가중치 O)
   - DIJKSTRA 명령어는 방향성 적용 여부와 가중치를 모두 고려하여 알고리즘을 수행한다.
   - 프로그램은 DIJKSTRA 명령어(DIJKSTRA O 1)에 포함된 시작 정점에서 다른 모든 정점으로 가는 최단 경로(shortest path)와 그 비용(cost)을 로그 파일에 출력한다.
   - 저장된 그래프에 가중치가 음수인 간선이 존재하면 에러를 출력한다.

4. Bellman-Ford (방향성 O/X, 가중치 O)
   - BELLMANFORD 명령어는 방향성 적용 여부와 가중치를 모두 고려하여 수행한다.
   - 프로그램은 BELLMANFORD 명령어(BELLMANFORD O 4 2)에 포함된 출발 정점과 도착 정점 사이의 최단 경로와 비용을 로그 파일에 출력한다.
   - 저장된 그래프에 가중치가 음수인 간선이 존재해도 정상 동작하며, 알고리즘 수행 중 음수 사이클이 발생하면 에러를 출력한다.

5. Floyd (방향성 O/X, 가중치 O)
   - FLOYD 명령어는 방향성 적용 여부와 가중치를 모두 고려하여 알고리즘을 수행한다.
   - 프로그램은 FLOYD 명령어 수행 시, 모든 정점 쌍에 대한 최단 경로 행렬을 구하고 로그 파일에 출력한다.
   - 저장된 그래프에 가중치가 음수인 간선이 존재해도 정상 동작하며, 알고리즘 수행 중 음수 사이클이 발생하면 에러를 출력한다.

6. Closeness Centrality (방향성 X, 가중치 O)
   - 그래프 중심성(graph centrality)은 그래프에서 어떤 정점이 가장 중요한지 알 수 있는 척도이다. 그중 근접 중심성(closeness centrality)은 중요한 노드일수록 다른 노드까지 가는 경로가 짧을 것(비용이 최소)이라는 가정을 둔 계산법이다.
   - 각 정점의 근접 중심성(closeness centrality)은 (정점의 개수-1)을 다른 모든 정점에서 해당 정점까지 최단 경로 비용의 총합으로 나눈 값이다.
   - CENTRALITY 명령어는 방향성이 없다고 가정하고 가중치만 고려한다.
   - 프로그램은 CENTRALITY 명령어 수행 시, 모든 정점의 근접 중심성 값과 중심성이 가장 높은 정점을 로그 파일에 출력한다.
   - 최단 경로 연산 수행 중 음수 사이클이 발생하면 에러 코드를 출력한다.

□ Functional Requirements

- 명령어의 인자는 모두 공백(‘ ’)으로 구분한다.

표 1. 명령어 사용 예 및 기능 설명

명령어	명령어 사용 예 및 기능
LOAD	사용 예) LOAD graph_L.txt (or graph_M.txt) • 텍스트 파일의 그래프 정보를 불러오는 명령어로, 텍스트 파일에 그래프 데이터가 존재하는 경우 텍스트 파일을 읽고 그래프를 구성한다. • 여러 번 LOAD 하는 경우, 기존 그래프를 삭제하고 재구성한다. • 텍스트 파일이 존재하지 않으면 출력 파일에 에러 코드(100)를 출력한다. 텍스트 파일: graph_L.txt or graph_M.txt (이름 변경 시, 프로젝트 0점) 출력 포맷 예시) ========LOAD======== Success ======================== ========ERROR======== 100 ========================
PRINT	사용 예) PRINT • 그래프 정보를 출력하는 명령어로, List 형태로 저장된 경우, 인접 리스트(adjacency list)를, Matrix 형태로 저장된 경우, 인접 행렬(adjacency matrix)을 출력 파일에 출력한다. • List 형태로 출력 시, 출발 정점은 오름차순으로 출력하고, 간선은 도착 정점 기준 오름차순으로 출력한다. • 저장된 그래프가 없는 경우, 출력 파일에 에러 코드(200)를 출력한다. 출력 포맷 예시) 1) List 형태로 저장된 경우 ========PRINT======== [0] -> (1,2) [1] -> (2,10) [2] -> (4,1) [3] -> [4] -> (1,6) -> (3,4) -> (5,3) -> (6,8)

	[5] -> (2,9) [6] -> (7,11) [7] -> ======================= 2) Matrix 형태로 저장된 경우 ========PRINT======== [0] [1] [2] [3] [4] [5] [6] [7] [0] 0 2 0 0 0 0 0 0 [1] 0 0 10 0 0 0 0 0 [2] 0 0 0 0 1 0 0 0 [3] 0 0 0 0 0 0 0 0 [4] 0 6 0 4 0 3 8 0 [5] 0 0 9 0 0 0 0 0 [6] 0 0 0 0 0 0 0 11 [7] 0 0 0 0 0 0 0 0 ======================= ========ERROR======== 200 =======================
BFS	사용 예 1) BFS O 1 사용 예 2) BFS X 0 • O는 방향성이 있는 그래프를 나타내고, X는 방향성이 없는 무방향 그래프를 나타낸다. • 마지막 인자는 시작 정점 번호이며, 시작 정점부터 BFS를 수행하고 방문 순서를 출력 파일에 출력한다. • 그래프가 없거나, 저장된 그래프에 입력한 시작 정점이 존재하지 않는 경우, 출력 파일에 에러 코드(300)를 출력한다. 출력 포맷 예시) 1) BFS O 1 ========BFS======== Directed Graph BFS Start: 1 1 -> 2 -> 4 -> 3 -> 5 -> 6 -> 7 =======================

	2) BFS X 1 ========BFS======== Undirected Graph BFS Start: 1 1 -> 0 -> 2 -> 4 -> 5 -> 3 -> 6 -> 7 ======================= ========ERROR======== 300 =======================
DFS	사용 예 1) DFS O 1 사용 예 2) DFS X 0 • O는 방향성이 있는 그래프를 나타내고, X는 방향성이 없는 무방향 그래프를 나타낸다. • 마지막 인자는 시작 정점 번호이며, 시작 정점부터 DFS를 수행하고 방문 순서를 출력 파일에 출력한다. • 그래프가 없거나, 저장된 그래프에 입력한 시작 정점이 존재하지 않는 경우, 출력 파일에 에러 코드(400)를 출력한다. 출력 포맷 예시) 1) DFS O 1 ========DFS======== Directed Graph DFS Start: 1 1 -> 2 -> 4 -> 3 -> 5 -> 6 -> 7 ======================= 2) DFS X 1 ========DFS======== Undirected Graph DFS Start: 1 1 -> 0 -> 2 -> 4 -> 3 -> 5 -> 6 -> 7 ======================= ========ERROR======== 400 =======================

<table>
<tr><td>KRUSKAL</td><td>사용 예) KRUSKAL

• 저장된 그래프의 최소 신장 트리(MST)를 구하며, MST를 구성하는 간선들의 정점 번호 및 가중치를 정점 번호 오름차순으로 출력하고, 가중치의 총합을 출력 파일에 출력한다.
• 그래프가 없거나, 접근 불가능한 정점이 있는 경우 출력 파일에 에러 코드(500)를 출력한다.

출력 포맷 예시)
========KRUSKAL========
[0] 1(2)
[1] 0(2) 4(6)
[2] 4(1)
[3] 4(4)
[4] 1(6) 2(1) 3(4) 5(3) 6(8)
[5] 4(3)
[6] 4(8) 7(11)
[7] 6(11)
Cost: 35
========================

========ERROR========
500
========================</td></tr>
<tr><td>DIJKSTRA</td><td>사용 예) DIJKSTRA O 1

• O는 방향성이 있는 그래프를 나타내고, X는 방향성이 없는 무방향 그래프를 나타낸다.
• 마지막 인자는 시작 정점 번호이며, 시작 정점부터 Dijkstra 알고리즘을 수행하여 다른 모든 정점까지의 최단 경로(shortest path)와 그 비용(cost)을 출력 파일에 출력한다.
• 경로는 시작 정점부터 도착 정점까지 순서대로 출력한다.
• 도착 정점에 접근할 수 없는 경우 최단 경로 대신 x를 출력한다.
• 그래프가 없거나, 음수 가중치가 존재하는 경우, 출력 파일에 에러 코드(600)를 출력한다.

출력 포맷 예시)
========DIJKSTRA========
Directed(Undirected) Graph Dijkstra <- 방향성에 따라 다르게 출력
Start: 1
[0] x</td></tr>
</table>

	[1] 1 (0) [2] 1 -> 2 (10) [3] 1 -> 2 -> 4 -> 3 (15) [4] 1 -> 2 -> 4 (11) [5] 1 -> 2 -> 4 -> 5 (14) [6] 1 -> 2 -> 4 -> 6 (19) [7] 1 -> 2 -> 4 -> 6 -> 7 (30) ========== ========ERROR======== 600 ==========
BELLMANFORD	사용 예) BELLMANFORD O 4 2 • O는 방향성이 있는 그래프를 나타내고, X는 방향성이 없는 무방향 그래프를 나타낸다. • 두 번째 인자는 시작 정점 번호이며, 마지막 인자는 도착 정점 번호이다. • 시작 정점부터 도착 정점까지의 최단 경로와 비용을 Bellman-Ford 알고리즘을 수행하여 계산하고, 최단 경로와 그 비용을 출력 파일에 출력한다. • 시작 정점에서 도착 정점에 도달할 수 없는 경우 경로와 비용 대신 x를 출력한다. • 그래프가 없거나, 음수 사이클이 발생한 경우, 에러 코드(700)를 출력한다. 출력 포맷 예시) ========BELLMANFORD======== Directed(Undirected) Graph Bellman-Ford <- 방향성에 따라 다르게 출력 4 -> 5 -> 2 Cost: 12 ========== ========ERROR======== 700 ==========

<table>
<tr><td>FLOYD</td><td>사용 예) FLOYD X

• O는 방향성이 있는 그래프를 나타내고, X는 방향성이 없는 무방향 그래프를 나타낸다.
• 저장된 그래프의 모든 정점 쌍에 대한 최단 경로의 비용을 행렬 형태로 출력한다.
• 행렬의 행은 시작 정점이고, 열은 도착 정점이다.
• 시작 정점에서 도착 정점에 도달할 수 없는 경우 비용 대신 x를 출력한다.
• 그래프가 없거나, 음수 사이클이 발생한 경우, 에러 코드(800)를 출력한다.

출력 포맷 예시)
========FLOYD========
Directed(Undirected) Graph Floyd <- 방향성에 따라 다르게 출력
[0] [1] [2] [3] [4] [5] [6] [7]
[0] 0 2 12 17 13 16 21 32
[1] x 0 10 15 11 14 19 30
[2] x 7 0 5 1 4 9 20
[3] x x x 0 x x x x
[4] x 6 12 4 0 3 8 19
[5] x 16 9 14 10 0 18 29
[6] x x x x x x 0 11
[7] x x x x x x x 0
========================

========ERROR========
800
========================</td></tr>
<tr><td>CENTRALITY</td><td>사용 예) CENTRALITY

• 저장된 그래프의 모든 정점의 근접 중심성(closeness centrality)을 계산하고, 정점 번호 오름차순으로 중심성을 분수 형태로 출력하며, 중심성이 가장 높은 정점을 표시한다.
• 중심성이 가장 높은 정점이 여러 개인 경우, 정점 번호 오름차순으로 모두 출력한다.
• 최단 경로의 비용은 프로젝트에서 구현한 Floyd 알고리즘을 활용한다.
• 어떤 정점에서 출발해도 도달할 수 없는 정점의 경우, 근접 중심성 값 대신 문자 ‘x’를 출력한다.
• 그래프가 없거나, 최단 경로 계산 중 음수 사이클이 발생한 경우, 에러 코드(900)를 출력한다.</td></tr>
</table>

	출력 포맷 예시) ========CENTRALITY======== [0] 7/85 [1] 7/73 [2] 7/55 [3] 7/73 [4] 7/49 <- Most Central [5] 7/67 [6] 7/81 [7] 7/147 ======================= ========ERROR======== 900 =======================
EXIT	사용 예) EXIT 프로그램상의 모든 메모리를 해제하며, 프로그램을 종료한다. 오류가 발생하는 상황은 없다. 출력 포맷 예시) ========EXIT======== Success =======================

□ 동작 별 에러 코드

동작	에러 코드
LOAD	100
PRINT	200
BFS	300
DFS	400
KRUSKAL	500
DIJKSTRA	600
BELLMANFORD	700
FLOYD	800
CENTRALITY	900

□ Requirements in implementation

- ✓ 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
- ✓ 모든 명령어는 반드시 대문자로 입력한다.
- ✓ 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력받으면 입력한 명령어에 알맞은 에러 코드를 출력한다.
- ✓ 예외처리에 대해 반드시 에러 코드를 출력한다.
- ✓ 출력은 "출력 포맷"을 반드시 따라 한다. "="의 개수와 행렬 출력의 구분에 사용되는 공백(' ') 개수는 고려하지 않아도 된다.
- ✓ log.txt 파일에 출력 결과를 반드시 저장한다.
  - log.txt가 이미 존재할 경우 이전 로그는 삭제 후 새로 저장한다.
- ✓ 프로그램 종료 시, 할당된 모든 메모리를 해제하여 메모리 누수를 방지한다.
  - 프로그램 종료 시 메모리 누수가 1byte라도 발생할 시 감점 -10%

□ 구현 시 반드시 정의해야하는 Class 및 파일

- ✓ Graph - Graph 클래스
- ✓ ListGraph - ListGraph 클래스
- ✓ MatrixGraph - MatrixGraph 클래스
- ✓ GraphMethod - 그래프 연산을 수행하는 함수를 모아둔 파일
- ✓ Manager - Manager 클래스
  - 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

□ Files

- ✓ graph_L.txt : 그래프가 인접 리스트 형태로 저장된 파일
- ✓ graph_M.txt : 그래프가 인접 행렬 형태로 저장된 파일
- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ log.txt : 프로그램 출력 결과를 모두 저장하고 있는 파일

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 코드(github 주소 참고)를 이용하여 구현하며 작성된 소스 파일의 이름과 클래스와 함수 이름 및 형태를 임의로 변경하지 않는다.
- ✓ 클래스의 함수 및 함수 인자, 변수는 자유롭게 추가 구현이 가능하다.
- ✓ 제시된 Class를 각 기능에 알맞게 모두 사용한다.
- ✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- ✓ 채점 시 코드를 수정해야 하는 일이 없도록 한다.
- ✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점 - 10%)
- ✓ 프로그램은 반드시 리눅스(Ubuntu 22.04)에서 동작해야 한다. (컴파일 에러 발생 시 감점 - 50%)
  - 제공되는 Makefile을 사용하여 테스트하도록 한다.
- ✓ **인터넷에서 공유된 코드나 다른 학생이 작성한 코드를 절대 카피하지 않도록 하며, 적발시 전체 프로젝트 0점 처리됨을 명시한다.**

□ 채점 기준

✓ 코드

채점 기준	점수
LOAD 명령어가 정상 동작하는가?	1
PRINT 명령어가 정상 동작하는가?	1
BFS 명령어가 정상 동작하는가?	1
DFS 명령어가 정상 동작하는가?	1
KRUSKAL 명령어가 정상 동작하는가?	2
DIJKSTRA 명령어가 정상 동작하는가?	2
BELLMANFORD 명령어가 정상 동작하는가?	2
FLOYD 명령어가 정상 동작하는가?	2
CENTRALITY 명령어가 정상 동작하는가?	3
총합	**15**

– 채점 기준 이외에도 조건 미달 시 감점 (linux 컴파일 에러, 파일 입출력 X, 주석 미흡 등)
  - linux 컴파일 에러는 50% 감점, 나머지는 각각 10% 감점
– log.txt 파일에서는 동작하는 것처럼 보이지만, 실제로는 구현이 제대로 되지 않은 경우 감점
  - 제대로 구현되지 않은 기능마다 최종 점수에서 20점 감점

✓ 보고서

채점 기준	점수
Introduction을 잘 작성하였는가?	1
Flowchart를 잘 작성하였는가?	2.5
Algorithm을 잘 작성하였는가?	3
Result Screen을 잘 작성하였는가?	2.5
Consideration을 잘 작성하였는가?	1
총합	**10**

✓ Result Screen 작성 시 정점의 개수가 최소 10개인 그래프를 사용할 것
✓ 최종 점수는 (코드 점수 x 보고서 점수)로 계산됩니다.

□ 제출기한 및 제출방법

✓ 제출기한

- 2025년 12월 7일 일요일 23:59:59까지 클라스(KLAS)에 제출

✓ 제출방법

- 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 제출
  - 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음 (제출 시 감점 - 10%)
  - 보고서 파일 확장자가 pdf가 아닐 시 감점 - 10%

✓ 제출 형식

- 학번_DS_project3.tar.gz (ex. 2024123456_DS_project3.tar.gz)
  - 제출 형식을 지키지 않은 경우 감점 - 10%

✓ 보고서 작성 형식 및 제출 방법

- Introduction : 프로젝트 내용에 대한 설명
- Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명
- Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명
- Result Screen : 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명
- Consideration : 고찰 작성
  - 위의 각 항목을 모두 포함하여 작성
  - 보고서 내용은 한글로 작성
  - 보고서에는 소스코드를 포함하지 않음