데이터구조설계

DS Project 3

학　　과　컴퓨터정보공학부
학　　번　2024402027
이　　름　김효정

제 출 일 자　2025-12-07

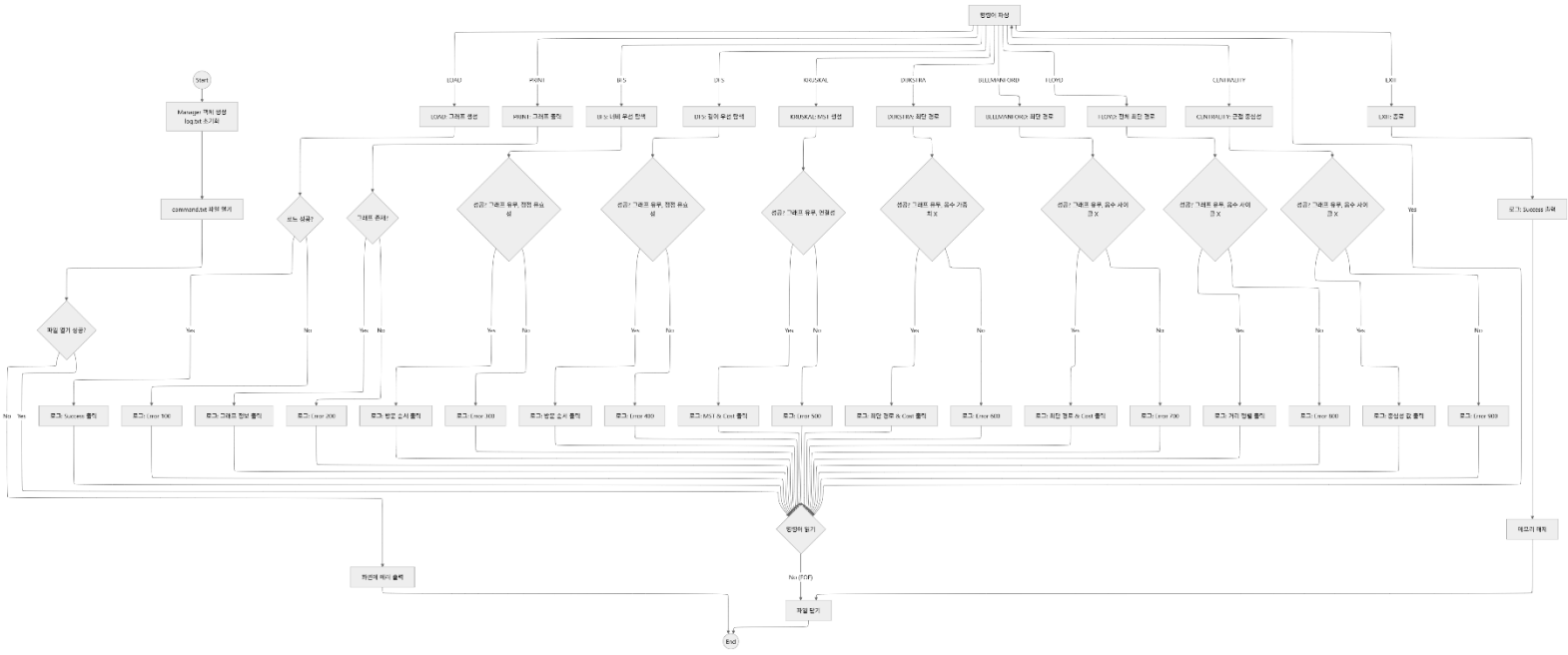
1. Introduction

본 프로젝트는 텍스트 파일 형태로 제공되는 그래프 데이터를 기반으로 다양한 그래프 알고리즘을 실행하는 프로그램을 구현하는 것을 목표로 한다. 프로그램은 입력 파일(graph_L.txt 또는 graph_M.txt)에 기록된 그래프 저장 방식(List 또는 Matrix), 정점 수, 간선 정보(도착 정점 및 가중치)를 읽어 내부 그래프 구조를 생성하며, 이후 command.txt에 포함된 명령들을 순차적으로 처리하여 결과를 log.txt 파일에 출력하도록 설계되었다.

프로그램이 수행하는 알고리즘은 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford, Floyd, Closeness Centrality로 구성되며, 각 알고리즘은 방향성 옵션(O/X), 가중치 처리 방식, 음수 가중치 및 음수 사이클 검출 여부 등 명세서에서 요구하는 조건을 정확히 반영하였다. 전체 흐름은 Manager 클래스에서 제어하며, 그래프의 저장 방식에 따라 ListGraph와 MatrixGraph가 독립적으로 동작할 수 있도록 구조를 분리하였다.

프로그램 개발 과정에서 핵심적으로 고려한 부분은 명령어별 유효성 검사, 그래프 로딩과 출력의 일관성, 동적 메모리 해제 처리, 그리고 알고리즘 결과 출력 형식의 정확한 준수였다. 특히 log.txt 출력 규격은 채점 기준과 직결되기 때문에 성공·실패 여부와 에러 코드 출력 형식을 명세와 동일하게 유지하도록 구현하였다. 또한 실행 종료 시에는 모든 동적 메모리를 해제하여 메모리 누수가 발생하지 않도록 하였으며(Valgrind로 검증), 알고리즘 특성에 맞는 방문 순서와 경로 계산이 정확히 이루어지도록 세부 동작을 세심하게 구성하였다.

2. Flowchart



본 프로그램의 전체 동작 흐름은 Manager 객체 생성부터 시작되며, command.txt 파일에 기록된 명령어를 한 줄씩 읽어 처리하는 구조로 이루어진다. 플로우차트는 크게 초기 파일 처리 단계, 명령어 분기 처리 단계, 각 명령어별 예외 검사, 그리고 정상/오류 출력 및 반복 처리 단계로 구성된다.

2.1. 초기 실행 단계

프로그램이 시작되면 Manager 객체가 생성되고, 출력 로그를 기록하기 위한 log.txt가 초기화된다. 이후 command.txt 파일을 열고, 파일 열기에 실패할 경우 즉시 오류 메시지를 출력한 뒤 종료한다.

2.2. 명령어 읽기 및 분기

파일 열기에 성공하면, 프로그램은 command.txt의 내용을 한 줄씩 읽으며 다음 명령어가 EOF인지 검사한다. 읽은 명령어는 파싱 후 아래 명령어 중 하나로 분기된다.

LOAD

PRINT

BFS

DFS

KRUSKAL

DIJKSTRA

BELLMANFORD

FLOYD

CENTRALITY

EXIT

각 명령어는 별도의 조건 검사 단계를 갖고 있으며, 검사 결과에 따라 성공 또는 에러 로그를 출력하고 다음 명령어 처리를 이어간다.

2.3. 명령어별 처리 흐름

1) LOAD

그래프 파일을 읽어 그래프 구조를 생성한다.

성공 시: Success 로그 출력

실패 시: Error 100 출력

2) PRINT

현재 그래프 존재 여부를 확인한 뒤, 그래프가 존재하면 인접 리스트/행렬 정보를 출력, 없으면 Error 200

3) BFS / DFS

두 명령어 모두 "그래프 존재 여부"와 "시작 정점 유효성"을 검사한다.

통과 시: 탐색 순서 출력

실패 시: 각각 Error 300, Error 400

4) KRUSKAL

그래프가 존재하고 연결되어 있어야 MST 생성이 가능하다.

가능하면 MST와 비용 출력

아니면 Error 500

5) DIJKSTRA

음수 가중치가 포함된 그래프는 사용할 수 없으며, 시작 정점 유효성도 검사한다.

성공 시: 최단 경로 + 비용 출력

실패 시: Error 600

6) BELLMANFORD

음수 간선은 허용되지만 음수 사이클이 존재하면 실패한다.

성공 시: 최단 경로 출력

실패 시: Error 700

7) FLOYD

모든 쌍 최단 경로 알고리즘을 수행하며, 음수 사이클 존재 여부를 검사한다.

성공 시: 거리 행렬 출력

실패 시: Error 800

8) CENTRALITY

모든 정점 간 최단 거리 계산이 필요하므로, 음수 사이클이 없어야 한다.

성공 시: 중심성 결과 출력

실패 시: Error 900

9) EXIT

메모리를 해제하고 Success를 출력한 뒤 프로그램을 종료한다.

2.4. 반복 및 종료

각 명령어 처리 후 프로그램은 다시 명령어 읽기 단계로 돌아가며, EOF가 되었을 때 파일을 닫고 정상적으로 종료된다.

3. Algorithm

3.1 BFS (Breadth-First Search)

BFS는 시작 정점에서 가까운 정점부터 탐색을 확장해 나가는 방식으로, 큐(FIFO)를 사용해 방문 가능한 정점을 순서대로 처리한다. 수업자료("Graph Search Methods – Part II")에서 설명하는 기본 구조를 그대로 따르며, 프로그램에서는 option이 'O'이면 방향 그래프,

'X'이면 무방향처럼 간선 정보를 처리한다. 시작 정점을 방문 처리해 큐에 넣고, 큐에서 정점을 하나씩 꺼내 인접 정점을 확인하며 방문하지 않은 정점을 큐에 추가한다. 큐가 빌 때까지 반복하며, 결과적으로 방문 순서는 간선 수 기준으로 가장 가까운 정점이 먼저 나오게 된다.

3.2 DFS (Depth-First Search)

DFS는 한 방향으로 깊게 진행하다가 더 이상 갈 곳이 없을 때 되돌아오는 방식의 탐색이다. 수업자료("Graph Search Methods – Part II")에서 설명하는 DFS 구조와 동일하며, 본 프로젝트에서는 스택 기반 반복 구조로 구현하였다. option 값에 따라 방향성 여부를 반영해 인접 정점을 탐색한다. 시작 정점을 스택에 넣고 방문 처리한 뒤, 스택 top의 인접 정점 중 방문하지 않은 정점을 push하고, 더 이상 갈 곳이 없으면 pop하여 이전 단계로 돌아간다. 탐색이 끝날 때까지 이 과정을 반복하며, 생성되는 DFS 방문 흐름은 자료에서의 DFS spanning tree와 동일한 형태가 된다.

3.3 Kruskal Algorithm (최소 신장 트리)

Kruskal 알고리즘은 사이클을 만들지 않는 최소 비용 간선을 선택해 그래프의 MST를 구성하는 알고리즘이다. 수업자료 "Minimum Spanning Trees"에서 설명된 절차를 그대로 따르며, 모든 간선을 가중치 기준으로 정렬한 뒤 Union-Find를 사용해 사이클 여부를 판별한다. 프로젝트에서는 ListGraph 또는 MatrixGraph에서 모든 간선을 (가중치, (u, v)) 형태로 모아 정렬한 후, 정렬된 간선을 순서대로 확인해 두 정점이 같은 집합이면 제외하고 그렇지 않으면 Union 후 MST에 포함한다. 간선이 n-1개 선택되면 MST가 완성되며, 명세서의 "무방향 + 가중치 고려" 조건을 정확히 반영한 구현 방식이다.

3.4 Dijkstra Algorithm

Dijkstra 알고리즘은 음수 가중치가 없는 그래프에서 단일 시작 정점 기준 최단 경로를 구하는 알고리즘이다. 수업자료 "Shortest Paths"의 흐름을 따르며 dist 배열과 방문 배열을 이용해 구현하였다. dist 배열을 시작 정점은 0, 나머지는 INF로 초기화한 뒤, 방문하지 않은 정점 중 dist가 가장 작은 정점을 선택하고, 해당 정점을 기준으로 인접 정점의 거리를 dist[w] = min(dist[w], dist[u] + weight(u, w)) 형태로 갱신한다. 모든 정점을 처리할 때까지 반복하면 최단 경로가 확정되며, 음수 가중치가 포함되어 있으면 명세서에 따라 에러 코드(600)를 출력한다.

3.5 Bellman-Ford Algorithm

Bellman-Ford는 음수 가중치가 있어도 최단 경로 계산이 가능한 알고리즘이며, 수업자료 "Shortest Paths Part II/III"에서 소개된 구조와 동일하다. dist 배열을 초기화한 뒤, 모든 간선(u, v)을 이용해 dist[v] = min(dist[v], dist[u] + weight) 연산을 (정점 수 - 1)번 반복한다. 이후 한 번 더 모든 간선을 확인했을 때 값이 갱신되면 음수 사이클이 존재하는 것으로 판단한다. 프로젝트에서도 동일하게 음수 사이클이 감지되면 즉시 에러 코드(700)를 출력하도록 구현되어 있다.

3.6 Floyd Warshall Algorithm (모든 쌍 최단 경로)

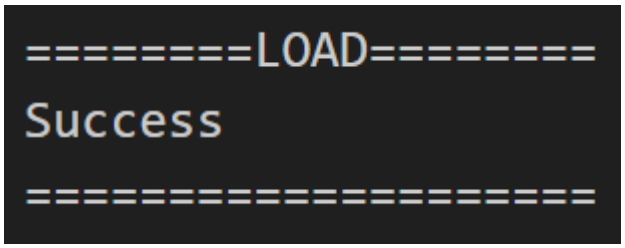
Floyd 알고리즘은 모든 정점 쌍의 최단 경로를 계산하는 DP 기반 알고리즘으로, 수업자료 "Shortest Paths Part III"에서 제시된 점화식 A[i][j] = min(A[i][j], A[i][k] + A[k][j])을 그대로 사용한다. 초기 행렬은 직접 연결된 간선 가중치 또는 INF로 설정하고, k를 중간 정점으로 두고 i → k → j 경로가 더 짧으면 값을 갱신하는 방식이다. A[i][i]가 음수가 되는 경우 음수 사이클로 판단하며, 프로젝트에서도 MatrixGraph 기반 행렬을 이용해 동일한 방식으로 수행한다.

3.7 Closeness Centrality

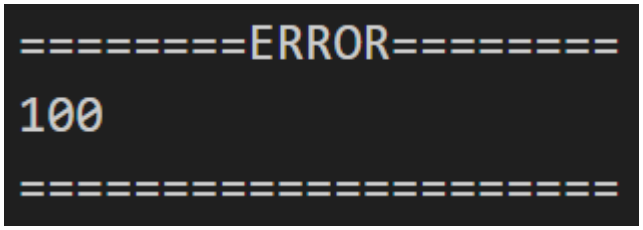
Closeness Centrality는 특정 정점이 다른 모든 정점과 얼마나 가까운지를 평가하는 지표이며, 명세서 기준으로 방향성은 고려하지 않고 가중치만 반영해 계산한다. 각 정점 v에 대해 v에서 모든 정점까지의 최단 경로 비용을 계산하고, (정점 수 - 1)을 비용 합으로 나눈 값을 중심성 값으로 사용한다. 비용 합이 가장 작은 정점이 가장 높은 중심성을 갖게 된다. 도달할 수 없는 정점이 있는 경우에는 중심성 값 대신 'x'를 출력하며, 전체 계산은 프로젝트에서 구현한 Floyd 결과를 기반으로 수행한다.

4. Result Screen

1) LOAD 명령어

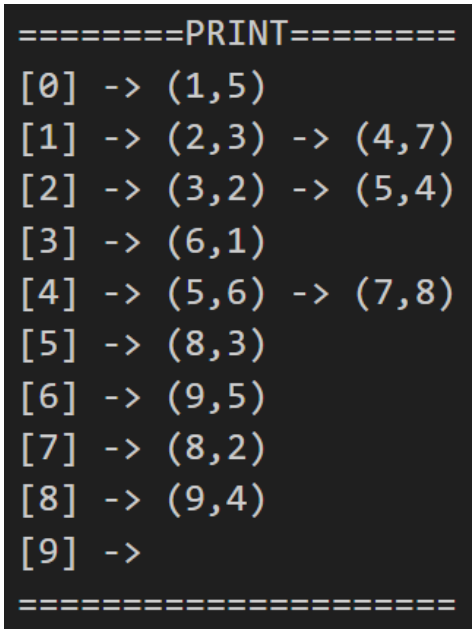


이 로그는 지정한 그래프 파일이 정상적으로 열리고 모든 정점·간선 정보가 요구된 형태로 파싱되었음을 의미한다. 즉, 그래프가 프로그램 내부 구조(ListGraph 또는 MatrixGraph)로 성공적으로 초기화되었다는 뜻이다.

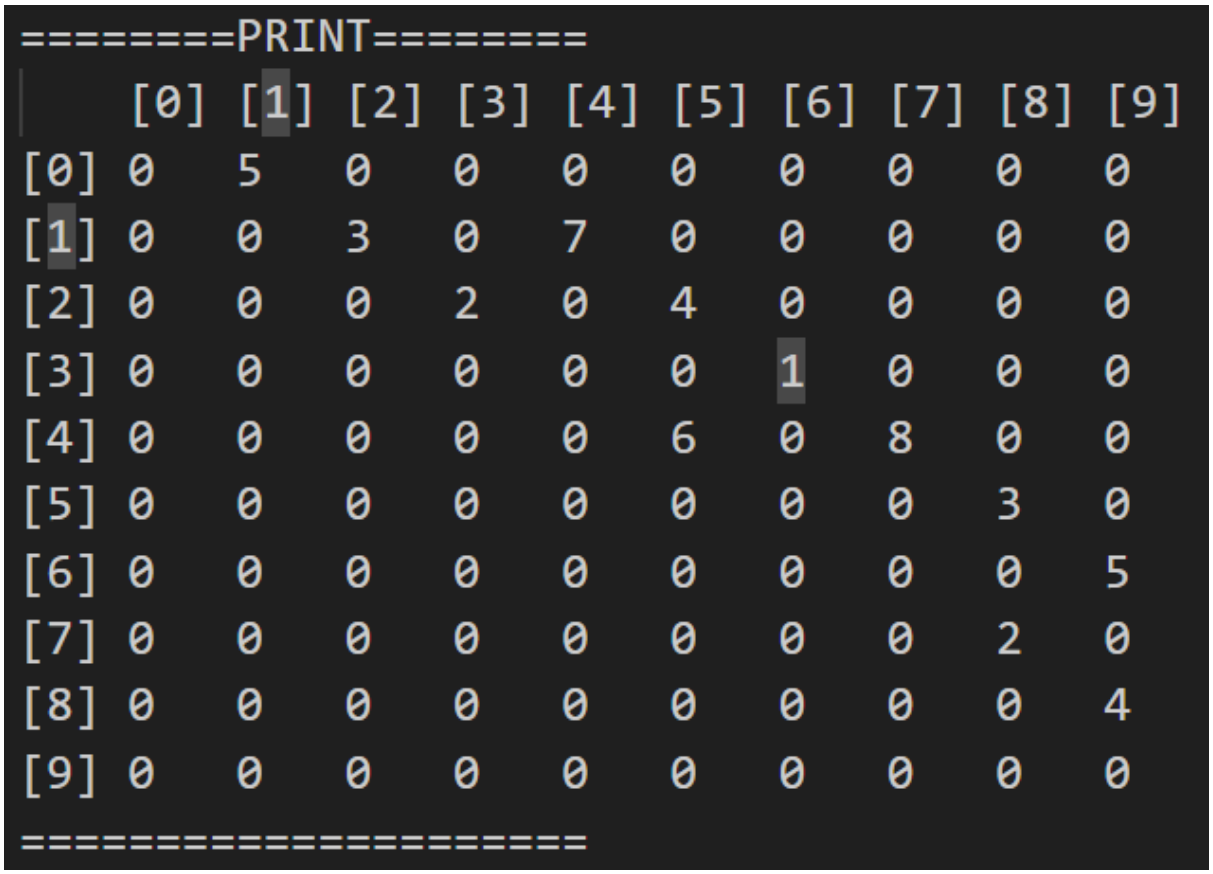


LOAD 실행 시 파일명이 존재하지 않거나(graph_L.txt, graph_M.txt 외), 파일 형식이 명세와 맞지 않아 로드에 실패한 경우 출력되는 결과이다. Error 100은 "그래프 로드 실패"를 의미하며 초기화가 이루어지지 않았음을 뜻한다.

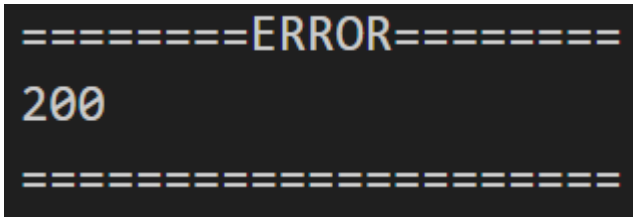
2) PRINT 명령어



ListGraph 출력은 각 정점의 인접 리스트를 (도착정점, 가중치) 형식으로 나열한 것이다. 이 로그는 LOAD한 그래프의 구조가 올바르게 저장되었고, PRINT가 명세대로 인접 리스트 정보를 읽어왔다고 확인해준다.

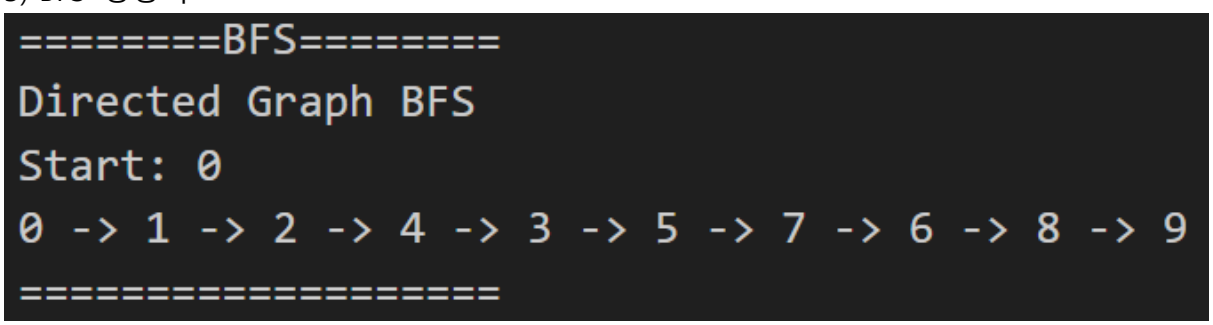


MatrixGraph 출력은 10×10 가중치 행렬로, 간선이 존재하지 않는 경우 0 또는 x로 표시된다. 이 출력은 LOAD graph_M.txt가 정상적으로 실행되었음을 보여준다.



그래프가 로드되지 않은 상태에서 PRINT를 호출했을 때 출력되는 오류로, 출력할 그래프가 없음을 의미한다.

3) BFS 명령어



방향 그래프 조건에서 시작 정점 0부터 너비 우선 탐색을 수행한 결과이다. 인접 리스트의 정렬 순서를 기준으로 큐에서 방문 순서가 정확하게 반영되었다.

```
========BFS========
Undirected Graph BFS
Start: 0
0 -> 1 -> 2 -> 4 -> 3 -> 5 -> 7 -> 6 -> 8 -> 9
====================
```

무방향 옵션(X)을 선택하면 역방향 간선도 탐색에 포함되지만, 이 그래프에서는 결과가 동일하게 나타났다. 이는 그래프 구조 특성상 방문 순서에 영향을 주지 않았음을 의미한다.

```
========ERROR========
300
======================
```

시작 정점이 유효 범위를 벗어나거나(BFS O 100), 그래프 미로드, 인자 부족 등 BFS 수행 조건을 충족하지 못할 때 발생한다.

4) DFS 명령어

```
========DFS========
Directed Graph DFS
Start: 0
0 -> 1 -> 2 -> 3 -> 6 -> 9 -> 5 -> 8 -> 4 -> 7
====================
```

스택 기반 깊이 우선 탐색으로, 가능한 한 깊게 들어가며 좌측(작은 번호)부터 탐색되는 정상적인 DFS 결과이다.

```
========DFS========
Undirected Graph DFS
Start: 0
0 -> 1 -> 2 -> 3 -> 6 -> 9 -> 8 -> 5 -> 4 -> 7
====================
```

무방향 그래프에서는 역방향 간선도 포함되므로 방문 순서가 달라지는 것을 확인할 수 있다.

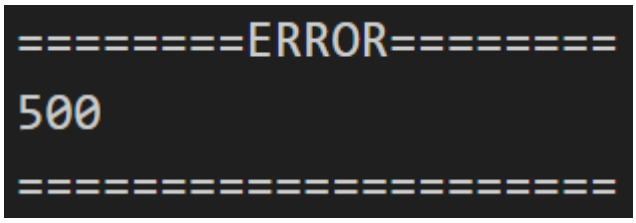
```
========ERROR========
400
======================
```

정점 번호 오류, 그래프 미로드, 옵션 오류 등의 이유로 DFS를 수행할 수 없을 때 Error 400이 출력된다.

5) KRUSKAL 명령어

```
========KRUSKAL========
[0] 1(5)
[1] 0(5) 2(3)
[2] 1(3) 3(2) 5(4)
[3] 2(2) 6(1)
[4] 5(6)
[5] 2(4) 4(6) 8(3)
[6] 3(1)
[7] 8(2)
[8] 5(3) 7(2) 9(4)
[9] 8(4)
Cost: 30
======================
```

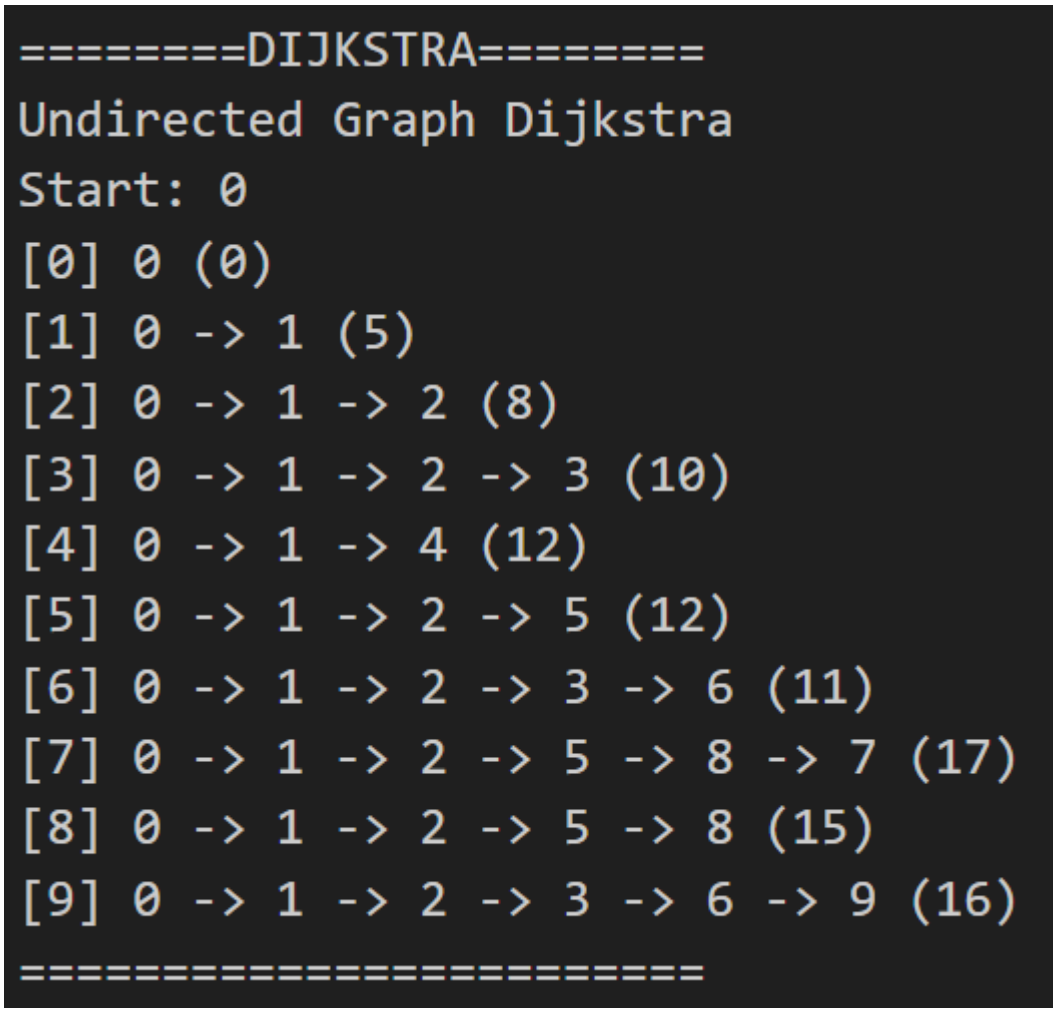
가중치 오름차순 정렬 → 사이클 검사 → MST 구성 절차가 정상 수행된 결과이다. Cost 30은 선택된 9개의 MST 간선 가중치 합이 정확하게 계산되었음을 의미한다.



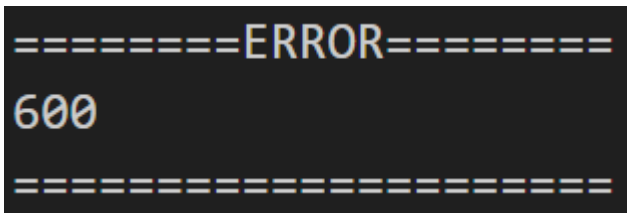
그래프가 비연결이거나 로드되지 않은 경우 출력된다. Error 500은 "MST 생성 불가"를 의미한다.

6) DIJKSTRA 명령어

도달 불가능한 정점은 x로 표기되며, 나머지는 최단 경로를 경로 형태로 출력한다.

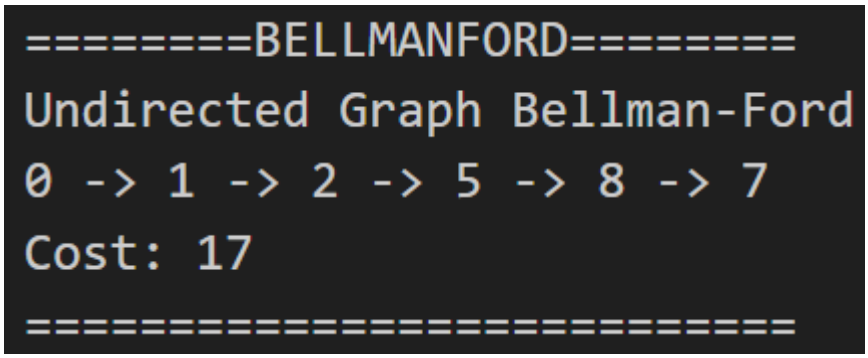


무방향 탐색에서는 역방향 이동이 허용되므로, Directed에서 x였던 정점도 경로가 생성될 수 있다.

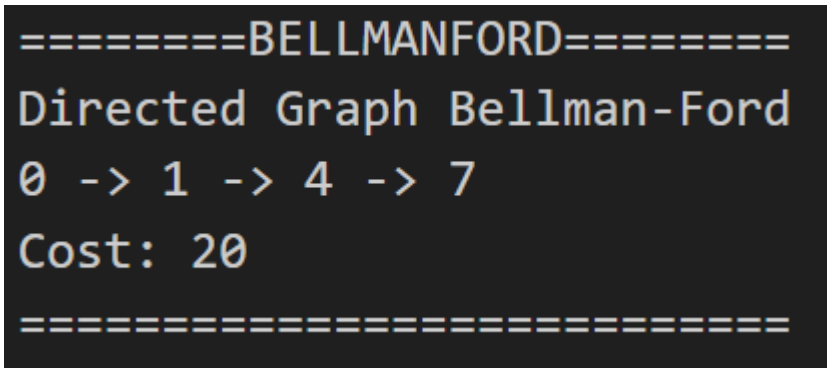


음수 가중치가 있거나, 정점 오류, 인자 부족 등이 있을 때 출력된다.

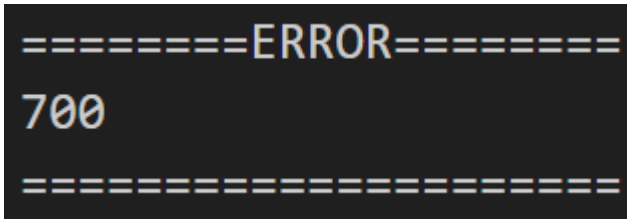
7) BELLMANFORD 명령어



무방향 그래프에서는 가능해진 경로이며, Bellman-Ford 알고리즘이 정상적으로 최단 비용을 계산했음을 보여준다.

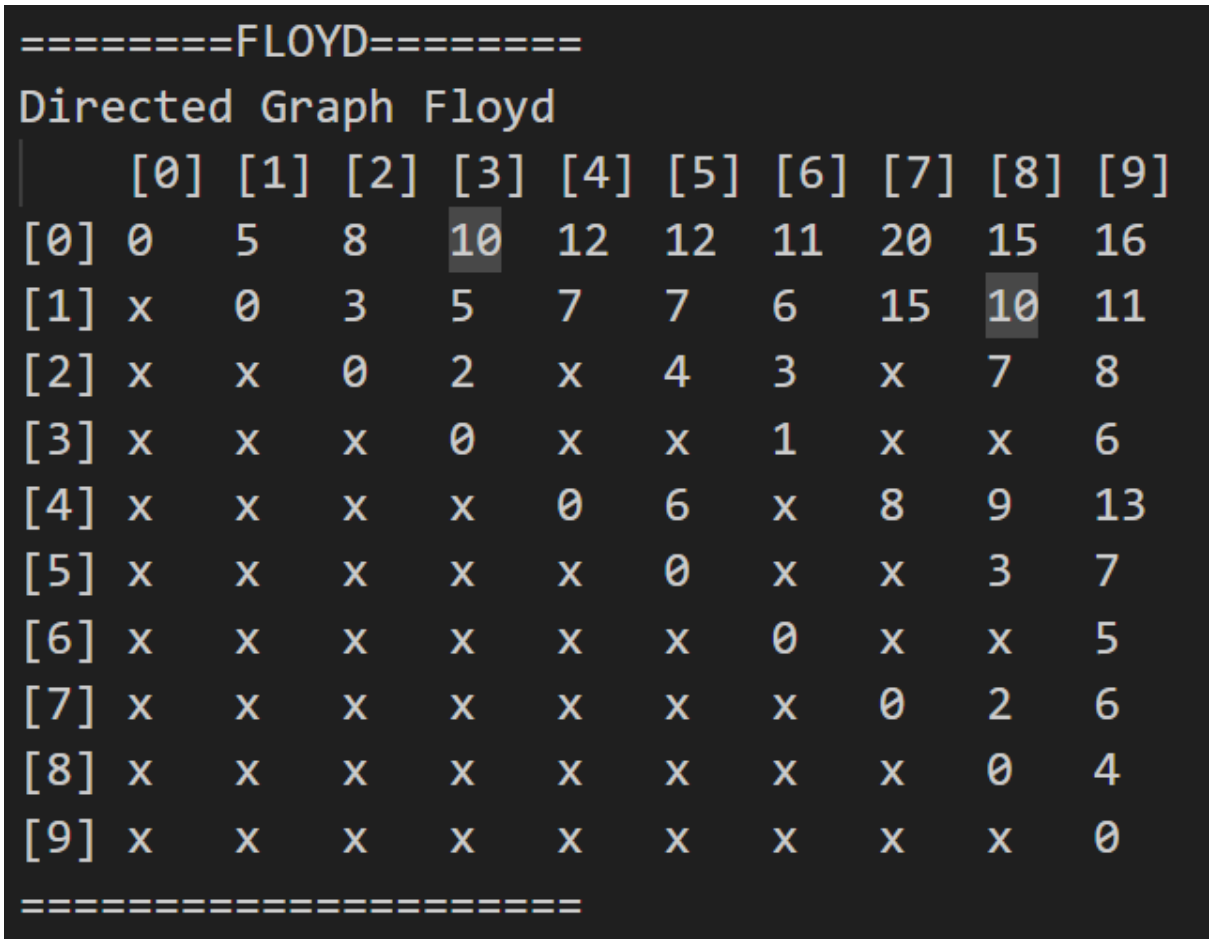


방향 그래프에서는 가능해진 경로이며, Bellman-Ford 알고리즘이 정상적으로 최단 비용을 계산했음을 보여준다.



음수 사이클 존재 또는 인자 오류일 때 출력된다.

8) FLOYD 명령어



10×10 행렬이 출력되며, x는 경로가 존재하지 않음을 의미한다. 중간 정점을 통한 업데이트가 정상적으로 수행되었음을 확인할 수 있다.

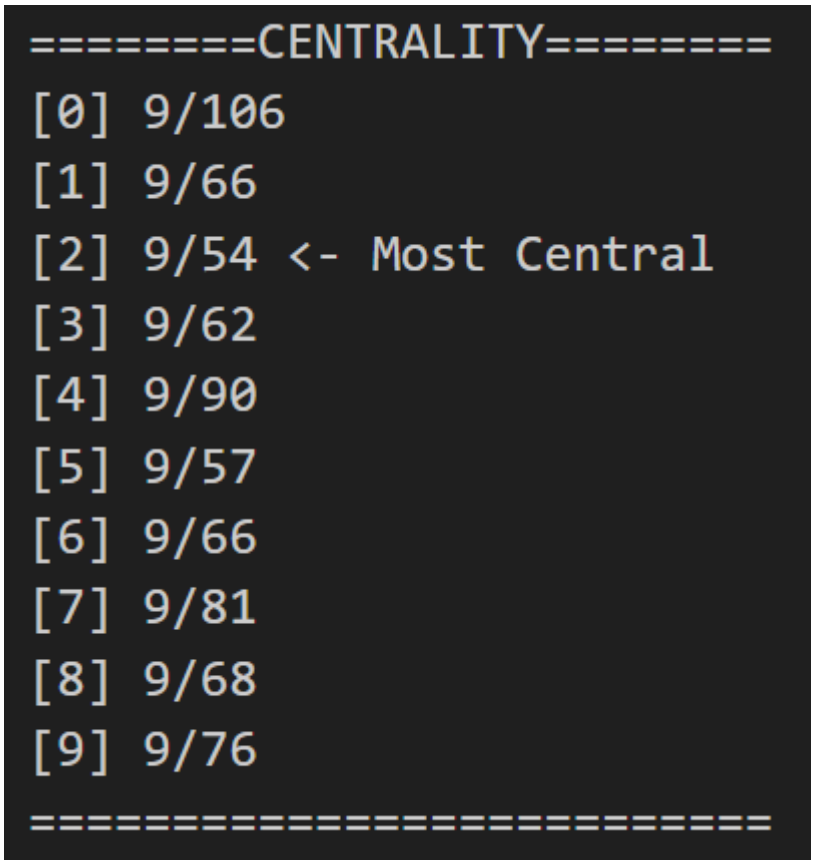
```
========FLOYD========
Undirected Graph Floyd
    [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
[0] 0   5   8   10  12  12  11  17  15  16
[1] 5   0   3   5   7   7   6   12  10  11
[2] 8   3   0   2   10  4   3   9   7   8
[3] 10  5   2   0   12  6   1   11  9   6
[4] 12  7   10  12  0   6   13  8   9   13
[5] 12  7   4   6   6   0   7   5   3   7
[6] 11  6   3   1   13  7   0   11  9   5
[7] 17  12  9   11  8   5   11  0   2   6
[8] 15  10  7   9   9   3   9   2   0   4
[9] 16  11  8   6   13  7   5   6   4   0
=====================
```

무방향 처리로 인해 더 많은 경로가 유효해지므로 전체 행렬 값이 감소하거나 채워지는 부분이 증가한다.

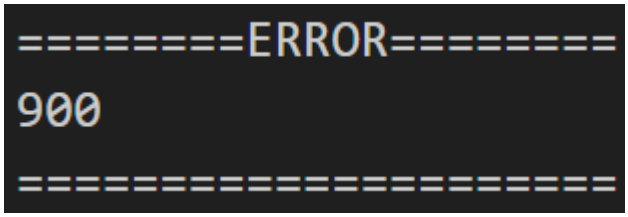
```
========ERROR========
800
=====================
```

그래프 미로드, 음수 사이클 등 Floyd 수행이 불가능할 때 발생.

9) CENTRALITY 명령어

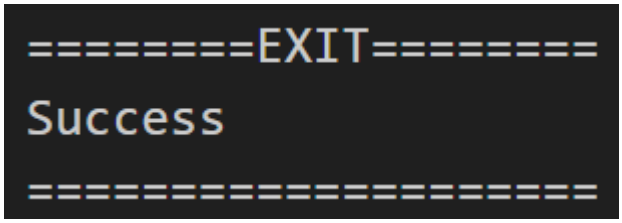


각 정점의 최단 거리 합을 계산한 값이며, 2번 정점이 가장 작은 값을 가져 중심 노드로 판정되었다.



음수 사이클 존재 또는 그래프 미로드 상태에서 CENTRALITY 수행 시 발생한다.

10) EXIT 명령어



모든 동적 메모리를 해제하고 프로그램이 정상적으로 종료되었음을 의미한다.

5. Consideration

프로젝트는 그래프 자료구조(List, Matrix)를 직접 구현하고, 이를 기반으로 다양한 탐색 및 최단 경로 알고리즘을 적용해보는 과정이었다. 단순히 알고리즘을 따라 적는 것이 아니라, 실제 코드 구조 안에서 요구사항을 만족하도록 조정하고 예외 상황을 처리하는 과정에서 배운 점이 많았다. 주요 시행착오는 다음과 같다.

첫째, DFS 방문 순서 문제다. 명세서에서 "방문 가능한 정점이 여러 개면 번호가 낮은 정점부터 방문"해야 하는데, 처음에는 BFS와 동일하게 인접 정점을 오름차순으로 정렬한 뒤 스택에 넣었다. 하지만 스택은 LIFO라 오름차순으로 넣으면 가장 큰 번호가 먼저 pop되어 잘못된 순서로 방문하게 됐다. 이를 해결하기 위해 GraphMethod.cpp의 DFS에서 인접 정점을 내림차순으로 정렬해 스택에 push하도록 수정했고, 그 결과 pop 시 작은 번호부터 방문하는 올바른 탐색 순서가 구현되었다.

둘째, 그래프 저장 방식(List vs Matrix)에 따른 처리 통합이다. 입력 파일에 따라 서로 다른 그래프 구조가 생성되는데, 알고리즘을 타입별로 따로 구현하면 중복 코드가 많아지고 유지보수도 어렵다. 이를 해결하려고 getAdjacencyList 헬퍼 함수를 만들어 내부 구조와 관계없이 공통된 vector<vector<pair<int,int>>> 형태로 인접 정보를 반환받도록 했다. 또한 Directed/Undirected 옵션에 따라 getAdjacentEdges와 getAdjacentEdgesDirect를 구분해 호출해, 그래프 방향성 요구사항도 정확히 반영했다.

셋째, LOAD 명령어 반복 호출 시의 메모리 관리 문제다. 기존 그래프를 해제하지 않고 새 그래프를 생성하면 메모리 누수가 발생할 수 있기 때문에, Manager의 LOAD 초입과 소멸자에서 delete graph를 명시적으로 수행하도록 했다. 이 과정에서 대규모 그래프를 반복적으로 로드하는 프로젝트 특성상 메모리 관리가 얼마나 중요한지 다시 확인할 수 있었다.

넷째, 음수 사이클 처리다. Bellman-Ford, Floyd, 그리고 Floyd 결과를 기반으로 하는 Centrality는 음수 사이클이 존재하면 정상적인 결과를 만들 수 없다. 특히 Centrality는 Floyd 결과에 의존하므로, Floyd 단계에서 음수 사이클을 감지하고 상위로 전달하는 방식이 필요했다. 이를 위해 Floyd 수행 후 d[i][i] 값이 음수인지 검사해 음수 사이클 여부를 판단하고, 문제가 있을 경우 false를 반환하도록 처리해 에러 코드(800, 900)가 정확히 출력되도록 했다.

이번 프로젝트를 통해 단순히 알고리즘을 구현하는 것을 넘어서, 자료구조 차이에 따른 처리 방식의 통합, 메모리 관리, 예외 상황 대응 등이 전체 프로그램 안정성에 얼마나 중요한 요소인지 직접 체감할 수 있었다.