

데이터 구조 설계
3차 프로젝트

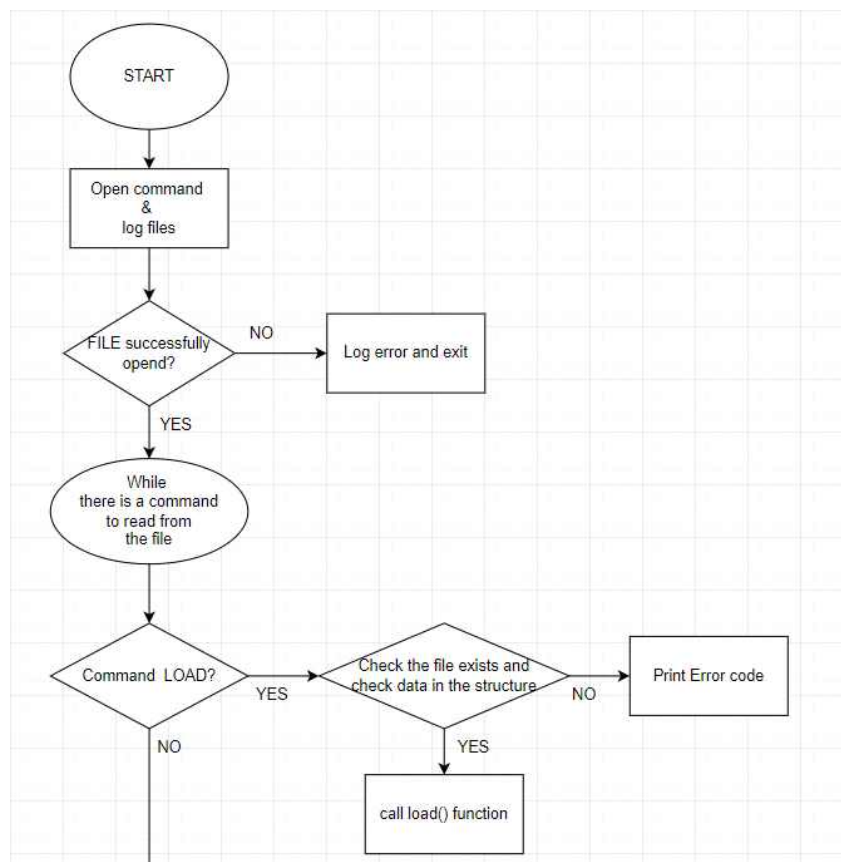
학번: 2018202084
이름: 김민재

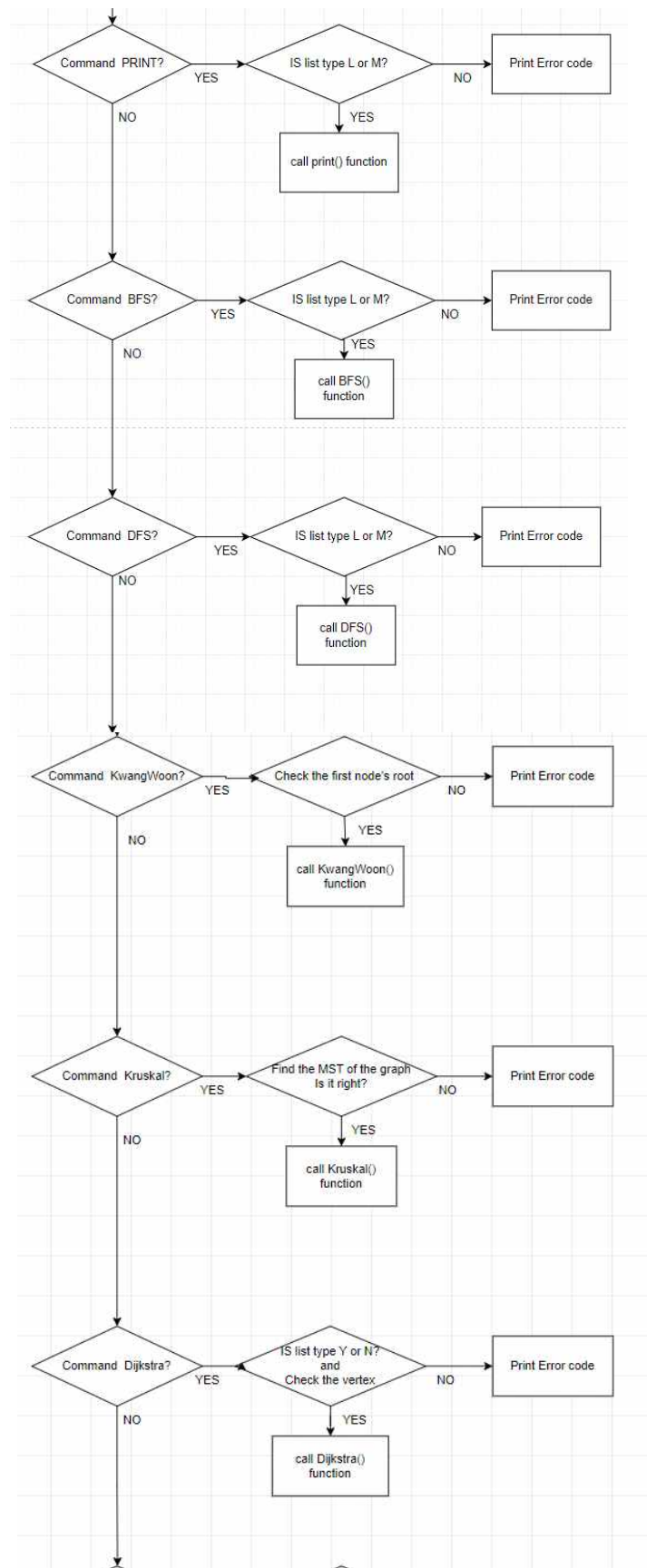
- Introduction

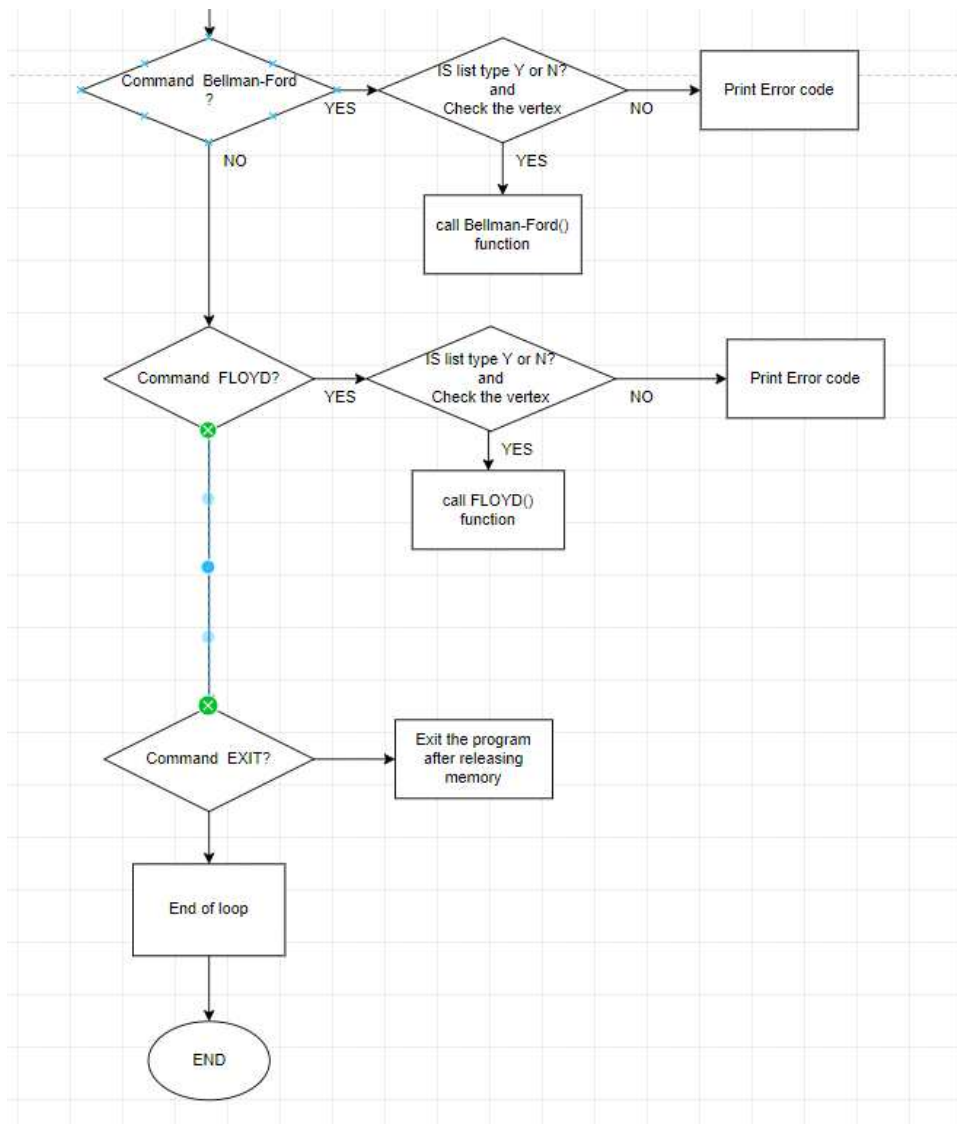
graph를 이용한 연산 프로그램을 구현한다. 저장되어 있는 txt 파일을 이용하여 graph를 구현하고, 7개의 그래프 특성에 따른 연산을 수행하도록 한다. 그리고 KwangWoon 알고리즘을 추가하여 연산하도록 한다. 주어진 data 형태에 따라서 List와 Matrix 형태로 수행하도록 한다. DFS와 BFS 알고리즘은 direction과 weight를 고려한다. 수행 방법으로는 순회 또는 탐색을 한다. 입력받은 vertex를 이용하여 명령어에 따른 알고리즘 활용으로 모든 vertex에 방문하도록 한 후에 방문 순서대로 명령어에 따른 결과를 출력하도록 한다. 값이 작은 vertex를 먼저 방문하는 것을 가정으로 생각하며, BFS는 Queue 형태를 가지며, DFS는 stack 구조를 가지게 된다. BFS 그래프는 입력에 따라서 방향성이 다르다. Y는 방향성이 있으며, N은 무방향을 나타내도록 한다. 방문 순서에 따른 vertex 순서로 결과를 출력한다. DFS 또한 BFS와 유사하며, 입력한 vertex를 기준으로 하여 DFS를 수행하도록 한다. 결과는 log.txt에 출력하도록 한다. Kruskal 알고리즘은 최소 신장 트리를 만들며 direction은 없지만 weight는 존재하는 환경에서 수행하도록 한다. 저장된 연결 정보를 이용하여 MST를 구한다. MST를 구한 후 구성하는 vertex 값을 오름 차순으로 출력하도록 한다. Dijkstra 알고리즘은 하나의 정점을 두어 이와 다른 모든 정점을 최단 경로의 알고리즘을 고려하여 direction과 weight가 모두 있는 환경에서 연산을 진행하도록 한다. Dijkstra는 방향성을 고려하여 최단 경로와 cost 값을 log.txt에 출력하도록 한다. vertex, 최단 경로, cost 순서로 나타내며 최단 경로는 vertex에서 기준 vertex로의 역순으로 출력하도록 한다. 입력된 vertex를 기준으로 하여 수행한 후에 최단 경로에 따른 cost 값을 구하도록 한다. weight가 음수이면 Dijkstra는 에러를 출력하고, Bellman Ford 알고리즘은 음수 cycle이 있으면 Dijkstra처럼 에러를 출력하고, 아닌 경우는 최단 경로를 구하도록 한다. 시작 vertex 기준으로 끝의 vertex 까지의 최단 경로와 거리를 구한다. FLOYD는 방향, 무방향 알고리즘이 모두 가능하며 음수 cycle인 경우는 에러를 출력하고, 음수 cycle이 아닌 경우는 최단 행렬을 구하도록 한다. KwangWoon 알고리즘은 List graph를 이용하여 구하도록 하며, 위치한 정점에서 방문 가능한 vertex 값이 홀수 인 경우는 그 중에 제일 큰 값의 vertex 번호로 시작하고, 짝수 개인 경우는 가장 작은 vertex 번호를 이용하여 방문을 하도록 한다. 정렬 연산은 효율성을 위하여 세그먼트의 크기에 따라서 알고리즘을 구현하도록 한다. quick 정렬을 진행하며, 수행할 때, 재귀적으로 분할하는 경우에 6개의 크기 이하인 경우는 삽입 정렬, 7이상인 경우는 분할하도록 한다. LOAD 명령어는 graph 정보를 불러오며, txt 파일을

읽어서 graph를 구현하도록 한다. PRINT 명령어는 graph 상태를 출력하며, LIST 형은 adjacency list를 출력하도록 하고, Matrix 형은 adjacent matrix를 출력하도록 한다.

- Flowchart







- Algorithm

함수별 흐름도의 구성

Manager::LOAD

```
|
|— 파일 열기
|   |— 파일 열기 실패 시: 오류 처리 및 함수 종료 (false 반환)
|   └─ 파일이 비어 있는지 확인
|       └─ 파일이 비어 있으면: 오류 처리 및 함수 종료 (false 반환)
|
|— 기존 그래프 삭제 (if loaded)
|
|— 그래프 타입 읽기 ('L' 또는 'M')
|   |— 'L': ListGraph 객체 생성
|   |   └─ 파일에서 간선 데이터 읽기 및 간선 추가
|   └─ 'M': MatrixGraph 객체 생성
|       └─ 파일에서 간선 데이터 읽기 및 간선 추가
|
└─ 성공 메시지 출력 및 함수 종료 (true 반환)
```

Manager::PRINT

```
|
|— 그래프 존재 여부 확인
|   └─ 그래프가 없으면: 오류 처리 및 함수 종료 (false 반환)
|
└─ 그래프 타입에 따라 인쇄 메소드 호출 및 함수 종료 (true 반환)
```

Manager::PRINT 함수의 흐름도

단계	작업 내용
1	graph 존재 여부 확인, 없으면 오류 메시지 출력 및 종료
2	graph->printGraph(&fout) 호출
3	함수 종료

mBFS

- |
- |— 그래프가 로드되었는지 확인
 - | |
 - | |— 로드되지 않았다면: 함수 종료 (false 반환)
- |
- |— 초기화
 - | |— 방문한 정점 추적을 위한 map (visited)
 - | |— BFS용 queue (q)
 - | |— BFS 결과 저장을 위한 vector (bfsResult)
- |
- |— 시작 정점 처리
 - | |— 시작 정점을 방문 처리
 - | |— 시작 정점을 큐에 추가
- |
- |— BFS 루프
 - | |
 - | |— 큐가 비어 있지 않은 동안
 - | | |— 현재 정점 = 큐의 맨 앞 정점
 - | | |— 현재 정점에서 방문 가능한 정점들 가져오기
 - | | |— 인접 정점을 정점 번호 순으로 정렬
 - | | |— 정렬된 인접 정점들을 큐에 추가
 - | |
 - | |— BFS 결과 벡터에 현재 정점 추가
- |
- |— 결과 출력
 - | |— "BFS 결과" 메시지 출력
 - | |— 각 정점과 경로 출력
 - | |— 함수 종료 (true 반환)

mDFS

- |
- |— 그래프 로드 확인
 - | — 그래프가 로드되지 않았다면: 함수 종료 (false 반환)
- |
- |— 초기 설정
 - | — 방문한 정점 추적을 위한 map
 - | — DFS용 스택
 - | — DFS 결과를 저장할 벡터
- |
- |— DFS 알고리즘 실행
 - | — 스택이 비어 있지 않는 동안:
 - | | — 현재 정점 추출
 - | | — 이미 방문한 정점이면 건너뛰
 - | | — 방문 처리 및 결과 벡터에 추가
 - | | — 인접 정점을 스택에 추가 (정렬된 순서로)
- |
- |— 결과 출력 및 함수 종료 (true 반환)

mDIJKSTRA

- |
- |— 그래프 로드 및 사이즈 확인
- |
- |— 초기 설정
 - | — 인접 리스트 가져오기
 - | — 최단 거리 벡터 (INF로 초기화)
 - | — 이전 정점 추적 벡터
 - | — 우선순위 큐
- |
- |— Dijkstra 알고리즘 실행
 - | — 우선순위 큐가 비어 있지 않는 동안:
 - | | — 현재 정점과 거리 추출
 - | | — 현재 정점을 기준으로 인접 정점 탐색

- | | └─ 최단 거리 업데이트 및 우선순위 큐에 추가
- |
- └─ 결과 출력 및 함수 종료 (true 반환)

mKRUSKAL

- |
- └─ 그래프 로드 확인
- | └─ 그래프가 로드되지 않았다면: 함수 종료 (false 반환)
- |
- └─ 초기 설정
- | └─ 간선 리스트 생성
- | └─ 유니온-파인드 배열 초기화
- | └─ 총 비용 초기화
- |
- └─ Kruskal 알고리즘 실행
- | └─ 모든 간선에 대해:
- | | └─ 사이클이 아닌 경우: 간선 선택 및 유니온 연산 수행
- | | └─ 선택된 간선을 저장
- |
- └─ 선택된 간선들을 정점별로 그룹화 및 정렬
- |
- └─ 결과 출력 및 함수 종료 (true 반환)

mBELLMANFORD

- |
- └─ 그래프 로드 및 사이즈 확인
- |
- └─ 초기 설정
- | └─ 인접 리스트 생성
- | └─ 최단 거리 및 이전 정점 배열 초기화
- |
- └─ Bellman-Ford 알고리즘 실행
- | └─ 간선 완화 (relaxation)

- | └─ 음수 사이클 감지
- |
- | └─ 결과 출력 및 함수 종료 (true 반환)

mFLOYD

- |
- | └─ 그래프 로드 및 사이즈 확인
- |
- | └─ 초기 설정
- | | └─ 거리 행렬 초기화
- |
- | └─ Floyd-Warshall 알고리즘 실행
- | | └─ 모든 쌍에 대한 최단 거리 계산
- | | └─ 음수 사이클 감지
- |
- | └─ 결과 출력 및 함수 종료 (true 반환)

mKwoonWoon

- |
- | └─ 그래프 타입 및 정점 검증
- |
- | └─ 초기 설정
- | | └─ 방문 배열 초기화
- | | └─ 시작 정점 설정 및 방문 처리
- |
- | └─ KwangWoon 알고리즘 실행
- | | └─ 현재 정점에서 이동 가능한 정점 찾기
- | | └─ 후보 정점 선정 (홀수/짝수 기준)
- | | └─ 선택된 정점 방문 및 이동
- |
- | └─ 결과 출력 및 함수 종료 (true 반환)

- Result Screen

LOAD

The screenshot shows a terminal window and two gedit windows. The terminal window displays the following commands and output:

```
os2018202084@ubuntu: ~/Downloads/src
os2018202084@ubuntu:~/Downloads/src$ ./run
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
===== LOAD =====
Success
===== LOAD =====
Success
=====
os2018202084@ubuntu:~/Downloads/src$
```

The gedit window titled "graph_M.txt" contains the following data:

```
M
10
0 3 7 0 0 0 0 0 0 0
0 0 6 0 0 0 0 0 0 0
0 0 0 7 0 0 0 0 0 0
0 0 0 0 4 0 0 0 0 0
0 5 0 0 0 0 0 0 0 0
0 0 11 0 6 0 0 0 0 0
0 0 0 0 8 2 0 0 0 0
0 0 0 0 9 0 12 0 0 0
0 0 0 0 0 0 10 9 0 0
0 0 0 0 0 0 0 0 1 0
```

The gedit window titled "graph_L.txt" contains the following data:

```
L
10
1
2 3
3 7
2
3 6
3
4 7
```

The gedit window titled "command.txt" contains the following commands:

```
LOAD graph_L.txt
LOAD graph_M.txt
```

command.txt로부터 LOAD를 수행하여 성공 메시지를 출력하도록 한다.

The screenshot shows a terminal window and two gedit windows. The terminal window displays the following commands and output:

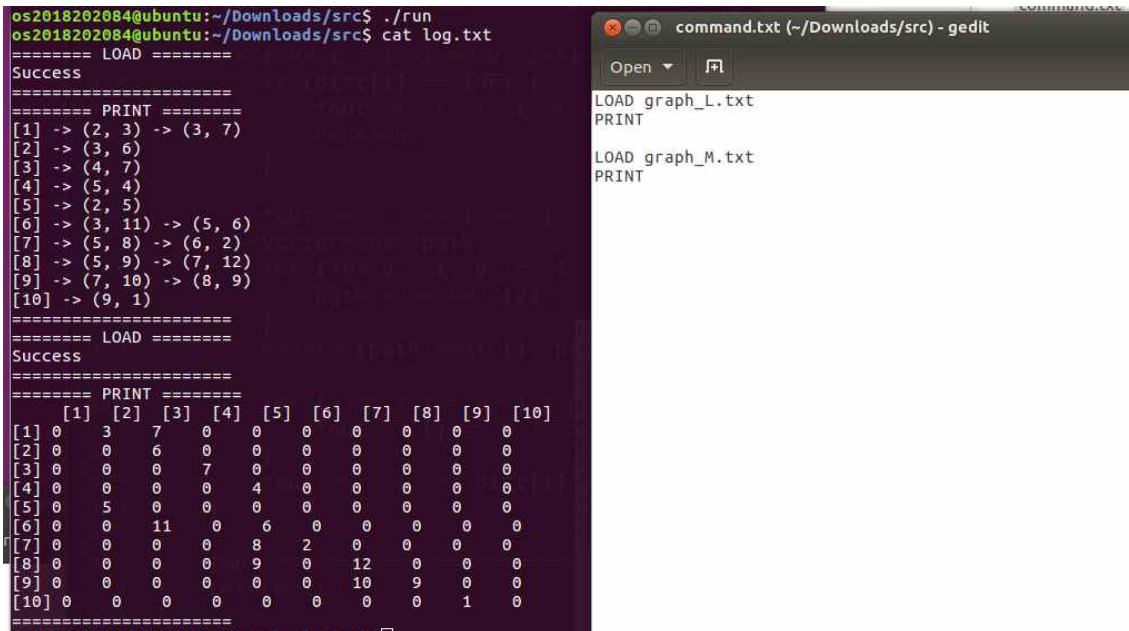
```
Success
=====
os2018202084@ubuntu:~/Downloads/src$ ./run
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
=====ERROR=====
100
=====
=====ERROR=====
100
=====
os2018202084@ubuntu:~/Downloads/src$
```

The gedit window titled "graph_M.txt" is empty.

The gedit window titled "graph_L.txt" is empty.

각 파일에 데이터가 없거나, 파일 이름이 잘못된 경우 에러를 출력한다.

PRINT



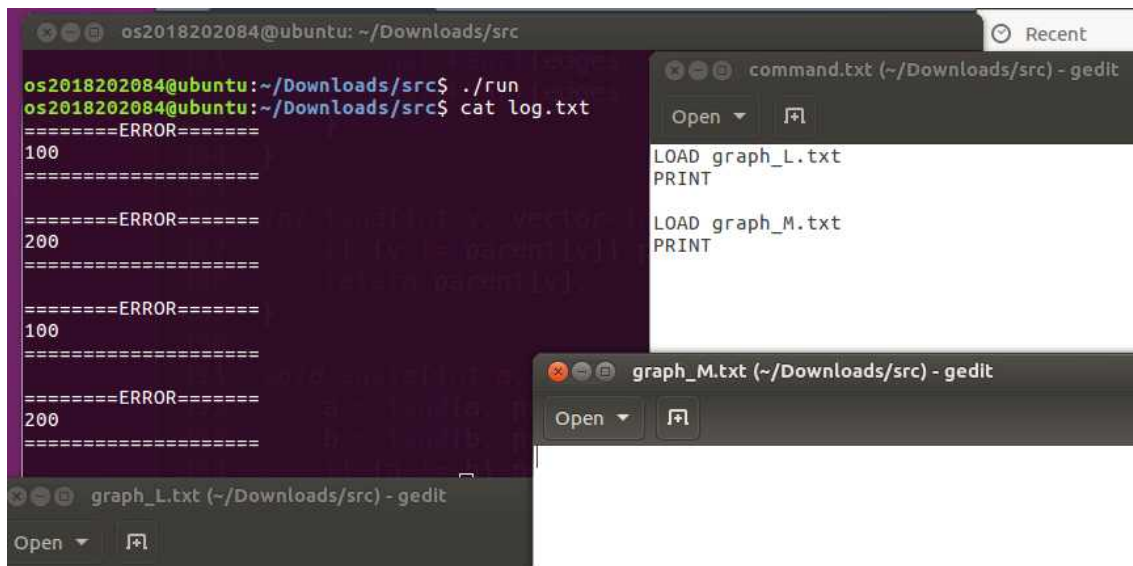
The terminal window shows the execution of a program that successfully loads and prints two graphs. The first graph is a list of edges, and the second is a matrix representation of a graph.

```
os2018202084@ubuntu:~/Downloads/src$ ./run
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
===== LOAD =====
Success
===== PRINT =====
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
[1] 0 3 7 0 0 0 0 0 0 0
[2] 0 0 6 0 0 0 0 0 0 0
[3] 0 0 0 7 0 0 0 0 0 0
[4] 0 0 0 0 4 0 0 0 0 0
[5] 0 5 0 0 0 0 0 0 0 0
[6] 0 0 11 0 6 0 0 0 0 0
[7] 0 0 0 0 8 2 0 0 0 0
[8] 0 0 0 0 9 0 12 0 0 0
[9] 0 0 0 0 0 0 10 9 0 0
[10] 0 0 0 0 0 0 0 0 1 0
```

The gedit window shows the contents of command.txt:

```
LOAD graph_L.txt
PRINT
LOAD graph_M.txt
PRINT
```

그래프의 상태를 출력한다. List 형 그래프일 경우 list를 Matrix 형 그래프일 경우 matrix를 출력한다. edge는 vertex를 기준으로 오름차순으로 출력하도록 한다.



The terminal window shows the execution of a program that encounters errors when loading and printing graphs. The errors are related to missing files or incorrect file names.

```
os2018202084@ubuntu:~/Downloads/src$ ./run
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
=====ERROR=====
100
=====ERROR=====
200
=====ERROR=====
100
=====ERROR=====
200
```

The gedit window shows the contents of command.txt:

```
LOAD graph_L.txt
PRINT
LOAD graph_M.txt
PRINT
```


The gedit window shows the contents of graph_M.txt:

```
graph_M.txt (~/Downloads/src) - gedit
Open
```

각 파일에 데이터가 없거나, 파일 이름이 잘못된 경우 에러를 출력한다.


BFS

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== BFS =====
Directed Graph BFS result
1 -> 2 -> 3 -> 4 -> 5
=====
===== BFS =====
Undirected Graph BFS result
2 -> 3 -> 4 -> 5
=====
```



Y 는 방향성이 있는 그래프를 나타내고 N 은 방향성이 없는 무방향 그래프임을 나타내어 입력한 vertex 를 기준으로 BFS 를 수행한다. BFS 수행 시 올바른 값을 출력하는 것을 확인할 수 있다.

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
=====ERROR=====
300
=====
=====ERROR=====
300
=====
os2018202084@ubuntu:~/Downloads/src$
```



위와 같이 99의 시작 vertex가 없거나 vertex를 못받으면 에러코드를 출력한다.

DFS

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== DFS =====
Directed Graph DFS result
4 -> 5 -> 2 -> 3
=====
===== DFS =====
Undirected Graph DFS result
3 -> 4 -> 5 -> 2
=====
=====ERROR=====
400
=====
=====ERROR=====
400
=====
os2018202084@ubuntu:~/Downloads/src$
```

```
command.txt (~/.Downloads/src) - gedit
Open [v] [F]
LOAD graph_L.txt
PRINT
DFS Y 4
DFS N 3|

DFS Y
DFS N 22
```

DFS 또한 BFS와 유사하게 탐색을 수행하며, vertex 입력이 없으면 에러 값을 출력하고, 이외의 vertex여도 에러를 출력하도록 한다. DFS 수행 시 올바른 값을 출력하는 것을 확인할 수 있다.

KRUSKAL

```
os2018202084@ubuntu:~/Downloads/src$ ./run
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== Kruskal =====
[1] 2(3)
[2] 1(3) 3(6) 5(5)
[3] 2(6)
[4] 5(4)
[5] 2(5) 4(4) 6(6) 8(9)
[6] 5(6) 7(2)
[7] 6(2)
[8] 5(9) 9(9)
[9] 8(9) 10(1)
[10] 9(1)
cost: 45
=====
os2018202084@ubuntu:~/Downloads/src$
```

command.txt (~Downloads/src) - gedit

```
Open
LOAD graph_L.txt
PRINT
KRUSKAL
```

그래프의 MST 를 구하고, MST 를 구성하는 edge 들의 vertex 값을 오름차순으로 출력한다. weight 의 총합을 출력 파일에 보여준다.

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== Kruskal =====
[1] 2(3)
[2] 1(3) 3(6) 5(5)
[3] 2(6)
[4] 5(4)
[5] 2(5) 4(4) 6(6) 8(9)
[6] 5(6) 7(2)
[7] 6(2)
[8] 5(9) 9(9)
[9] 8(9) 10(1)
[10] 9(1)
cost: 45
=====
=====ERROR=====
100
=====
=====ERROR=====
200
=====
=====ERROR=====
600
=====
os2018202084@ubuntu:~/Downloads/src$
```

command.txt (~Downloads/src) - gedit

```
Open
LOAD graph_L.txt
PRINT
KRUSKAL
EXIT
```

graph_L.txt (~Downloads/src) - gedit

```
Open Save
```

데이터가 없는 경우에는 600 에러코드를 출력한다.

DIJKSTRA

```
os2018202084@ubuntu:~/Downloads/src$ cat log.txt
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== Dijkstra =====
Directed Graph Dijkstra result
startvertex: 1
[1] 1 (0)
[2] 1 -> 2 (3)
[3] 1 -> 3 (7)
[4] 1 -> 3 -> 4 (14)
[5] 1 -> 3 -> 4 -> 5 (18)
[6] x
[7] x
[8] x
[9] x
[10] x
=====
os2018202084@ubuntu:~/Downloads/src$
```

그래프의 방향성을 고려하여 명령어의 결과인 shortest path 와 cost를 출력한다. 기준 vertex 에서 도달할 수 없는 vertex 의 경우 'x'를 출력하도록 한다.

```
[8] x
[9] x
[10] x
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
=====ERROR=====
700
=====
os2018202084@ubuntu:~/Downloads/src$
```

vertex 값이 없으면 에러 값을 출력하도록 한다.

BELLMANFORD

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== Bellman-Ford =====
Undirected Graph Bellman-Ford result
2 -> 3 -> 4 -> 5
cost: 17
=====
===== Bellman-Ford =====
Undirected Graph Bellman-Ford result
1 -> 10 x
=====
os2018202084@ubuntu:~/Downloads/src$
```

```
command.txt (~/.Downloads/src) - gedit
Open
LOAD graph_L.txt
PRINT
BELLMANFORD N 2 5
BELLMANFORD N 1 10
```

방향성을 고려하여 StartVertex 를 기준으로 Bellman-Ford 를 수행하여 EndVertex 까지의 최 단 경로와 거리를 구하는 명령어로, Bellman-Ford 결과를 출력 파일 (log.txt)에 저장한다. StartVertex 에서 EndVertex 로 도달할 수 없는 경우 'x'를 출력한다.


```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
=====ERROR=====
800
=====
=====ERROR=====
800
=====
os2018202084@ubuntu:~/Downloads/src$
```

```
command.txt (~/.Downloads/src) - gedit
Open
LOAD graph_L.txt
PRINT
BELLMANFORD Y 1 33
BELLMANFORD Y 1
```

입력한 vertex 가 그래프에 존재하지 않거나, 입력한 vertex 가 부족한 경우, 명령어를 수행할 수 없는 경우, 음수 사이클이 발생한 경우 출력 파일에 오류 코드를 출력한다.

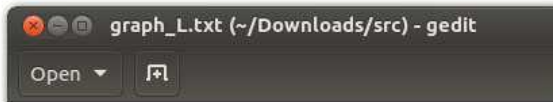
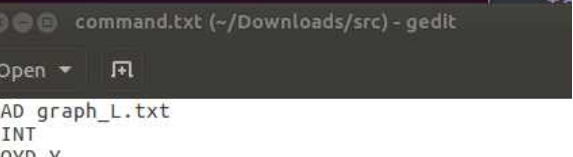
FLOYD

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== FLOYD =====
Directed Graph FLOYD result
[1] 0 3 7 14 18 x x x x x
[2] x 0 6 13 17 x x x x x
[3] x 16 0 7 11 x x x x x
[4] x 9 15 0 4 x x x x x
[5] x 5 11 18 0 x x x x x
[6] x 11 11 18 6 0 x x x x
[7] x 13 13 20 8 2 0 x x x
[8] x 14 20 27 9 14 12 0 x x
[9] x 23 23 30 18 12 10 9 0 x
[10] x 24 24 31 19 13 11 10 1 0
=====
os2018202084@ubuntu:~/Downloads/src$
```



방향성에 따른 vertex 의 쌍에 대해서 EndVertex 로 가는데 필요 한 비용의 최솟값을 행렬 형태로 보인다.

```
[8] x 14 20 27 9 14 12 0 x x
[9] x 23 23 30 18 12 10 9 0 x
[10] x 24 24 31 19 13 11 10 1 0
=====
=====ERROR=====
100
=====
=====ERROR=====
200
=====
=====ERROR=====
900
=====
os2018202084@ubuntu:~/Downloads/src$
```



데이터가 없을 때 에러 코드 900을 출력하도록 한다.

KwangWoon

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
===== KWANGWOON =====
startvertex: 1
1 -> 2 -> 3 -> 4 -> 5
=====
os2018202084@ubuntu:~/Downloads/src$
```

현재 점에서 방문할 수 있는 정점들이 홀수개면 그 정점 번호들의 가장 큰 정점 번호로 방문을 시작하고, 짝수개면 가장 작은 정점 번호로 방문을 시작하여 결과를 보인다.

EXIT

```
===== LOAD =====
Success
===== PRINT =====
[1] -> (2, 3) -> (3, 7)
[2] -> (3, 6)
[3] -> (4, 7)
[4] -> (5, 4)
[5] -> (2, 5)
[6] -> (3, 11) -> (5, 6)
[7] -> (5, 8) -> (6, 2)
[8] -> (5, 9) -> (7, 12)
[9] -> (7, 10) -> (8, 9)
[10] -> (9, 1)
=====
=====
os2018202084@ubuntu:~/Downloads/src$
```

프로그램 상의 메모리를 해제하며, 프로그램을 종료한다

- Consideration

그래프는 두 가지 형식, 즉 인접 리스트(ListGraph)와 인접 행렬(MatrixGraph)을 지원하는 것을 확인할 수 있었다. 본 프로젝트를 통해 복잡한 문제를 해결하는 데 필요한 능력과 접근 방식의 중요성을 깨달았다. 구현하고자 하는 명령어의 알고리즘의 선택과 구현, 데이터 구조의 이해가 효과적이었다고 생각한다. BFS와 DFS 같은 기본 알고리즘을 변형하여 다양한 시나리오에 적용할 수 있는 능력이 중요함을 깨달았다. 프로젝트를 진행하면서 코드의 효율성과 실행 시간을 최적화하는 것의 중요성을 알게 되었다. 불필요한 계산을 줄이고, 더 효율적인 데이터 구조를 사용함으로써 성능을 개선할 수 있었다. LOAD 함수는 주어진 파일로부터 그래프 데이터를 읽어오며, 파일 처리는 ifstream를 사용하여 수행된다. 파일의 존재 여부 및 비어 있는 파일인지 확인한다. 파일로부터 데이터를 읽어오는 부분에서 오류가 발생할 수 있으므로 이에 대한 오류 처리가 필요하다. 그래프 타입과 크기 정보를 기반으로 해당 그래프 객체를 생성한다. 코드에서 L 타입의 경우 ListGraph 객체를 생성하고, M 타입의 경우 MatrixGraph 객체를 생성한다. 이 부분에서 정상적인 객체 생성이 이루어지는지 확인해야 한다. 파일에서 읽어온 데이터를 기반으로 그래프의 간선 정보를 입력한다. 간선 정보는 insertEdge 함수를 통해 그래프에 추가된다. fout를 통해 디버그 정보를 출력하고 있으며, 각 간선이 추가되는지 확인한다. fout를 통해 디버그 정보를 출력하는 것은 디버깅에 유용하며, 각 단계에서 코드의 실행 여부를 확인할 수 있었다. 그래프의 출력 형식이 예상대로 되고 있는지 확인해야 한다. 이때, 인접 리스트(ListGraph)와 인접 행렬(MatrixGraph)에 따라 출력 형식이 다를 수 있으므로 다시 확인한다. 코드의 가독성을 높이고 성능을 개선하기 위한 최적화가 필요로 했다. 모든 코드를 작성하고 수정한 후에는 다양한 테스트 케이스를 활용하여 프로그램이 의도대로 작동하는지 확인해야 했다. 입력 데이터가 잘못된 형식이거나 예기치 않은 오류가 발생할 수 있는 상황에 대한 예외 처리도 고려하는 것에 시간이 오래 걸렸다.