

Data Structure Lab. Project #3

2023 년 11 월 17 일

Due date: 2023 년 12 월 14 일 목요일 23:59:59 까지

본 프로젝트에서는 그래프를 이용해 그래프 연산 프로그램을 구현한다. 이 프로그램은 그래프 정보가 저장된 텍스트 파일을 통해 그래프를 구현하고, 그래프의 특성에 따라 BFS, DFS, Kruskal, Dijkstra, Bellman-Ford, FLOYD 그리고 KwangWoon 총 7 가지 알고리즘을 통해 그래프 연산을 수행한다. 주어지는 그래프 데이터는 기본적으로 방향성(Direction)과 가중치(Weight)를 모두 가지고 있으며, 데이터 형태에 따라 List 그래프와 Matrix 그래프로 저장한다. 하지만 그래프 탐색 알고리즘마다 방향성을 고려해야 하기도 하고 고려하지 않아도 된다는 조건이 존재한다. 다음은 7 가지 연산에 대한 설명이다. 그래프 탐색 방법들에 대한 자세한 사항은 “Program Implementation 2. 그래프 연산”에서 설명한다.

-BFS & DFS-

(방향, 무방향 가능)

BFS 와 DFS 는 그래프의 방향성과 가중치를 고려하고, 그래프 순회 또는 탐색 방법을 수행한다.

-Kruskal-

(무방향만 가능)

Kruskal 알고리즘은 최소 비용 신장 트리(MST)를 만드는 방법으로 방향성이 없고, 가중치가 있는 그래프 환경에서 수행한다.

-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 수행한다. 만약 weight 가 음수일 경우 Dijkstra 는 에러를 출력하며, Bellman-Ford 에서는 음수 사이클이 발생한 경우 에러, 음수 사이클이 발생하지 않았을 경우 최단 경로와 거리를 구한다.

-FLOYD-

(방향, 무방향 가능)

FLOYD 에서는 음수 사이클이 발생한 경우 에러, 음수 사이클이 발생하지 않았을 경우 최단 경로 행렬을 구한다.

-KwangWoon-

(무방향만 가능, List graph 만 이용해 구현함)

KwangWoon 이라는 그래프 탐색법은 다음과 같은 조건을 만족하며 그래프를 탐색하게 된다. 현재 정점에서 방문할 수 있는 정점(방문한적이 없는 정점들)들이 홀수개면 탐색할 수 있는 정점 번호들의 가장 큰 정점 번호로 방문을 시작하고, 짝수개면 가장 작은 정점 번호로 방문을 시작한다.

각 그래프 연산들은 'GraphMethod' 헤더 파일에 일반 함수로 존재하며, **그래프 형식(List, Matrix)에 상관없이 그래프 데이터를 입력 받으면 동일한 동작을 수행하도록 일반화하여 구현한다.** 또한 충분히 큰 그래프에서도 모든 연산을 정상적으로 수행할 수 있도록 구현한다. 프로그램의 동작은 명령어 파일에서 요청하는 명령에 따라 각각의 기능을 수행하고, 그 결과를 출력 파일(log.txt)에 저장한다.

□ Program implementation

1. 그래프 데이터

프로그램은 그래프 정보가 저장되어 있는 텍스트 파일(graph_M.txt 또는 graph_L.txt)을 읽어, 해당 정보를 Adjacency List 또는 Adjacency Matrix에 저장한다. 텍스트 파일의 첫 번째 줄에는 그래프의 형식 정보(L: List, M: Matrix)가 저장되어 있고 두 번째 줄에는 그래프 크기가 저장되어 있다. 이후 데이터는 그래프 형식에 따라 구분된다. **그래프의 모든 vertex의 값은 1 이상의 정수이며, 1부터 시작하여 1, 2, 3, 4... 순으로 정해진다고 가정한다(적어도 정점이 10 개 이상은 있습니다).**

List 그래프의 경우 표 1-1 과 같이 edge의 출발 vertex를 의미하는 데이터(형식-1), edge의 도착 vertex와 weight가 쌍으로 이루어진 데이터(형식-2)로 구성된다. 만약 edge가 없는 vertex의 경우(형식-1)만 존재할 수 있다.

Matrix 그래프의 경우 표 1-2 의 (형식-3)과 같이 그래프의 정보가 저장되어 있다. 행렬의 행과 열은 각각 to vertex와 from vertex를 의미하고, 행렬의 값은 from vertex와 to vertex 사이를 잇는 edge의 weight를 의미한다.

텍스트 파일에 포함되어 있는 그래프의 형식 및 그래프 크기 정보는 'Graph' 클래스에 저장하며, 그래프 연결 정보는 그래프 형식(List or Matrix)에 따라 알맞은 '그래프 클래스'에 저장한다. Adjacency List 정보는 'ListGraph' 클래스에 저장하고 Adjacency Matrix 정보는 'MatrixGraph' 클래스에 저장한다. 그래프 클래스들은 'Graph' 클래스를 상속받는다. 그래프 정보가 포함되어 있는 텍스트 파일의 형식은 그림 1-1 과 그림 1-2 와 같다. 각 클래스의 멤버 변수에 대한 자세한 내용은 '구현 시 반드시 정의해야 하는 Class 및 멤버 변수'에 작성되어 있다.

형식	내용
1	시작 vertex
2	[도착 vertex] [weight]

표 1-1. List 그래프 데이터 형식

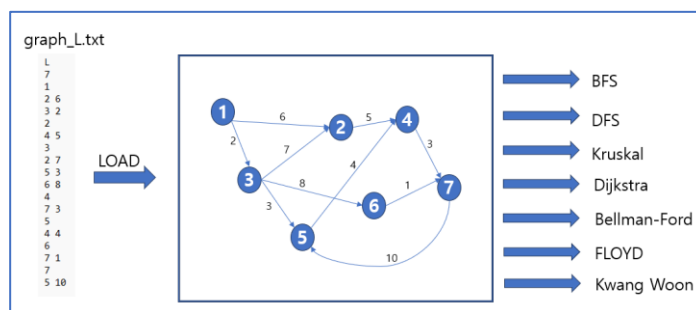
형식	내용
3	[weight_1] [weight_2] [weight_3] ... [weight_n]

표 1-2. Matrix 그래프 데이터 형식

```

L
7
1
2 6
3 2
2
4 5
3
2 7
5 3
6 8
4
7 3
5
4 4
6
7 1
7
5 10

```

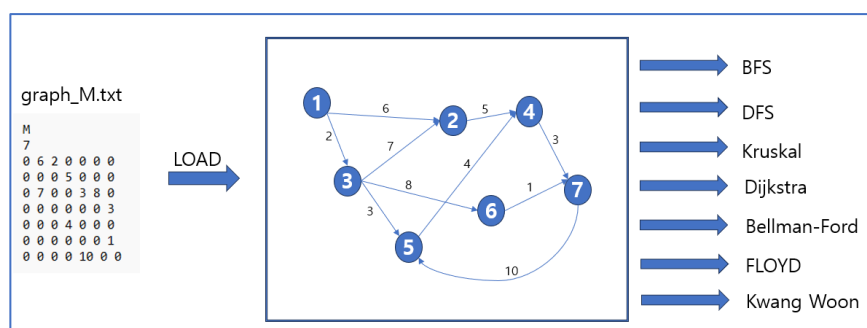


<그림 1-1. graph_L.txt 파일 예시>

```

M
7
0 6 2 0 0 0 0
0 0 0 5 0 0 0
0 7 0 0 3 8 0
0 0 0 0 0 0 3
0 0 0 4 0 0 0
0 0 0 0 0 0 1
0 0 0 0 10 0 0

```



<그림 1-2. graph_M.txt 파일 예시>

2. 그래프 연산

프로그램은 텍스트 파일로부터 그래프 정보를 읽은 뒤, 명령어에 따라 그래프 특성에 맞는 그래프 연산을 수행할 수 있다.

- BFS, DFS

각 명령어는 방향성과 가중치가 있는 그래프와 없는 그래프에서 모두 수행가능 하므로 방향성을 고려하여 그래프 탐색을 수행한다. 프로그램은 각 명령어(ex. BFS Y 1, DFS N 2)와 함께 입력 받은 vertex 를 시작으로 각 명령어에 맞는 알고리즘을 이용하여 모든 vertex 들을 방문하고, vertex 의 방문 순서를 명령어의 결과로 출력한다. 이때, 각 명령어는 vertex 의 값이 작은 vertex 를 먼저 방문한다고 가정한다. BFS 명령어는 Queue 를, DFS 명령어는 Stack 을 이용하여 동작한다.

- KRUSKAL

Kruskal 명령어는 방향성이 없고 가중치가 있는 그래프에서 수행 가능하다. 프로그램은 Kruskal 명령어 수행 시, 저장되어 있는 그래프 연결 정보를 이용하여 해당 그래프에 대한 MST 를 구하는 동작을 수행한다. 이때, MST 를 구하는 과정에 있어서 sub-tree 가 cycle 을 이루는지에 대한 검사를 수행하도록 한다.

- DIJKSTRA

Dijkstra 명령어는 방향성과 가중치가 있는 그래프와 방향성이 없는 그래프에서 모두 수행 가능하다. 프로그램은 명령어와 함께 입력 받은 vertex 를 기준으로 Dijkstra 알고리즘을 수행한 후, 모든 vertex 에 대해 shortest path 를 구하고, shortest path 에 대한 cost 값을 구한다.

- BELLMANFORD

Bellmanford 명령어는 방향성, Start Vertex 와 End Vertex 를 입력으로 받아 최단 경로와 거리를 구한다. Weight 가 음수인 경우에도 정상 동작하며, 음수 Weight 에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

- Floyd

Floyd 명령어는 입력 인자로 방향성만 있고 모든 쌍에 대해서 최단 경로 행렬을 구한다. Weight 가 음수인 경우에도 정상 동작하며, 음수 Weight 에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

- Kwang Woon



KwangWoon 탐색 방법은 광운대의 마스코트 우니가 고안한 그래프 탐색 방법이다. 과거에 우니와 팡이는 선형 탐색을 이용하여 배열의 구간 합을 구하는 C++ 프로그래밍 과제를 하고 있었다. 선형 탐색을 이용하여 배열의 구간 합을 구한다는 말은 배열[1,2,3,4,5]가 있을 때 특정 index 범위 내의 합을 구하기 위해, 해당 범위에 있는 원소를 하나씩 접근하여 구간 합을 구하는 것이다. 예를 들어 위의 배열에서 3 번부터 5 번의 구간 합을 구하려면 배열[3],[4],[5] 하나 하나를 모두 더해주면 구할 수 있다.

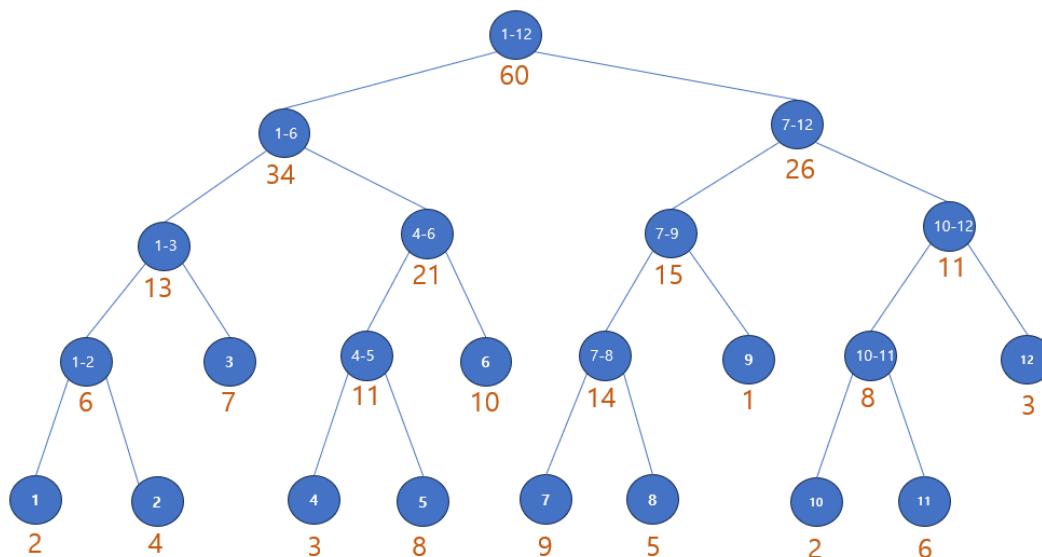
과제가 너무 쉽다며 광이에게 자신은 과제를 10 분만에 다 끝냈다고 자랑하고 있었다. 자랑이 다 끝난 후 우니는 광이에게 뒷통수를 한 방 맞았다. 그 이유는 과제의 제한사항 중 하나인 배열의 크기를 고려하지 않고 프로그래밍했기 때문이다. 우니가 구현한 방식으로 구간 합을 구하면 시간 복잡도는 $O(N)$ 으로 간단하지만 시간이 오래 걸린다는 단점이 있다. 또한 배열의 값을 바꾸고 다시 구간 합을 구하려면 수를 바꾸는데 $O(1)$, 수를 다시 더하는데 $O(N)$ 으로 M 번 수행한다면 최종적으로 $O(MN)$ 의 시간이 걸리기 때문이다. 광이는 우니에게 세그먼트 트리를 이용하여 구간 합을 구하면 된다는 것을 알려주고 조용히 떠났다. 광이의 도움으로 우니는 과제를 해결하였다고 한다.

이후에 우니는 그래프 탐색 방법을 배우고 KwangWoon 이라는 세그먼트 트리를 이용하여 자신만의 새로운 그래프 탐색 방법을 만들었다. KwangWoon 탐색 방법은 아래와 같다.

항상 정점 “1”부터 탐색을 시작한다. 따라서 정점 1 부터 탐색을 시작하여 탐색의 순서를 출력하면 된다. 현재 연결되어 있는 간선의 수가 짝수라면 가장 작은 정점 번호로 방문을 시작하고, 홀수라면 가장 큰 정점의 번호로 방문하도록 한다. **세그먼트 트리는 현재 정점에서 다른 정점으로 이동할 수 있는 이동 여부를 위해 사용된다.** 따라서 리프 노드는 다른 정점으로의 이동 여부를 담고 있다. 만약 그래프에 “1”번 정점이 없다면 에러를 출력한다.

<구간 합 세그먼트 트리 예시>

arr[12] = {2, 4, 7, 3, 8, 10, 9, 5, 1, 2, 6, 3}



노드 안에 적혀 있는 숫자는 배열의 index 를 나타낸다.

3. 정렬 연산

정렬 연산은 Kruskal 알고리즘의 edge 를 정렬하는 데 사용한다. 연산의 효율을 향상하기 위해 segment size 에 따라 정렬 알고리즘을 구현한다. 해당 프로젝트에서는 퀵 정렬을 수행하며 정렬 수행 시 segment size 를 재귀적으로 분할할 때, segment size 가 6 이하일 경우 삽입 정렬을 수행하며, segment size 가 7 이상인 경우엔 분할한다. 정렬은 제공된 의사코드에 기반하여 작성한다.

STL sort Algorithm

```
1 void quicksort(arr, low, high)
2   if(low < high)
3     if(high-low+1 ≤ segment_size)
4       insertionSort(arr, low, high)
5     else
6       pivot = partition(a, low, high)
7       quickSort(a, low, pivot-1)
8       quickSort(a, pivot+1, high)
```

그림 2. STL sort 의사코드

□ Functional Requirements

프로그램은 명령어를 통해 동작하며, 실행할 명령어가 작성되어 있는 텍스트 파일(command.txt)를 읽고 명령어에 따라 순차적으로 동작한다. 모든 명령어는 반드시 대문자로 입력하며, 각 명령어의 사용법과 기능은 다음과 같다. 명령어의 인자는 모두 공백(" ")으로 구분한다.

명령어	명령어 사용 예 및 기능
LOAD	사용법) LOAD [textfile] 예시) LOAD graph_L.txt ✓ 프로그램에 그래프 정보를 불러오는 명령어로, 그래프 정보가 저장되어 있는 텍스트 파일(graph_L.txt 또는 graph_M.txt)을 읽어 그래프를 구성한다. ✓ 텍스트 파일이 존재하지 않거나, 비어 있는 경우, 출력 파일(log.txt)에 오류 코드(100)를 출력한다. ✓ 기존에 그래프 정보가 존재할 때 LOAD 가 입력되면 기존 그래프 정보는 삭제하고 새로 그래프를 생성한다.

PRINT	<p>사용법) PRINT</p> <ul style="list-style-type: none"> ✓ 그래프의 상태를 출력하는 명령어로, List 형 그래프일 경우 adjacency list 를, Matrix 형 그래프일 경우 adjacency matrix 를 출력한다. ✓ vertex 는 오름차순으로 출력한다. edge 는 vertex 를 기준으로 오름차순으로 출력한다. ✓ 만약 그래프 정보가 존재하지 않을 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
BFS	<p>사용법) BFS [방향성] [vertex] 예시) BFS Y 0 or BFS N 0</p> <ul style="list-style-type: none"> ✓ Y 는 방향성이 있는 그래프를 나타내고 N 은 방향성이 없는 무방향 그래프임을 나타낸다. ✓ 현재 그래프에서 입력한 vertex 를 기준으로 BFS 를 수행하는 명령어로, BFS 결과를 출력 파일(log.txt)에 출력한다. <p>BFS 명령어를 수행하면서 방문하는 vertex 의 순서로 결과를 출력한다.</p> <ul style="list-style-type: none"> ✓ 입력한 vertex 가 그래프에 존재하지 않거나, vertex 를 입력하지 않은 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
DFS	<p>사용법) DFS [방향성] [vertex] 예시) DFS Y 0 or DFS N 0</p> <ul style="list-style-type: none"> ✓ Y 는 방향성이 있는 그래프를 나타내고 N 은 방향성이 없는 무방향 그래프임을 나타낸다. ✓ 현재 그래프에서 입력한 vertex 를 기준으로 DFS 를 수행하는 명령어로, DFS 결과를 출력 파일(log.txt)에 출력한다. <p>DFS 명령어를 수행하면서 방문하는 vertex 의 순서로 결과를 출력한다.</p> <ul style="list-style-type: none"> ✓ 입력한 vertex 가 그래프에 존재하지 않거나, vertex 를 입력하지 않은 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.
KRUSKAL	<p>사용법) KRUSKAL</p> <ul style="list-style-type: none"> ✓ 현재 그래프의 MST 를 구하고, MST 를 구성하는 edge 들의 weight 값을 오름차순으로 출력하고 weight 의 총합을 출력 파일(log.txt)에 출력한다. ✓ 입력한 그래프가 MST 를 구할 수 없는 경우, 명령어를 수행할 수 없는 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.

DIJKSTRA	<p>사용법) DIJKSTRA [방향성] [vertex] 예시) DIJKSTRA Y 1 or DIJKSTRA N 1</p> <ul style="list-style-type: none"> ✓ 그래프의 방향성을 고려하여 명령어의 결과인 shortest path 와 cost 를 출력 파일(log.txt)에 출력한다. 출력 시 vertex, shortest path, cost 순서로 출력하며, shortest path 는 해당 vertex 에서 기준 vertex 까지의 경로를 역순으로 출력한다. ✓ 기준 vertex 에서 도달할 수 없는 vertex 의 경우 'x'를 출력한다. <p>입력한 vertex 가 그래프에 존재하지 않거나, vertex 를 입력하지 않은 경우, 명령어를 수행할 수 없는 경우, 출력 파일(log.txt)에 알맞은 오류 코드를 출력한다.</p>
BELLMANFORD	<p>사용법) BELLMANFORD [방향성] [vertex(start)] [vertex(end)] 예시) BELLMANFORD Y 0 3</p> <ul style="list-style-type: none"> ✓ 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. ✓ 음수인 weight 가 있는 경우에도 동작해야한다. ✓ StartVertex 에서 EndVertex 로 도달할 수 없는 경우 'x'를 출력한다. <p>입력한 vertex 가 그래프에 존재하지 않거나, 입력한 vertex 가 부족한 경우, 명령어를 수행할 수 없는 경우, 음수 사이클이 발생한 경우 출력 파일 (log.txt)에 알맞은 오류 코드를 출력한다.</p>
FLOYD	<p>사용법) FLOYD [방향성] 예시) FLOYD N</p> <ul style="list-style-type: none"> ✓ 방향성을 고려하고 모든 vertex 의 쌍에 대해서 StartVertex 에서 EndVertex 로 가는데 필요 한 비용의 최솟값을 행렬 형태로 출력한다. ✓ 기준 vertex 에서 도달할 수 없는 vertex 의 경우 'x'를 출력한다. ✓ 명령어를 수행할 수 없는 경우, 음수 사이클이 발생한 경우 출력 파일 (log.txt)에 알맞은 오류 코드를 출력한다.
KwangWoon	<p>사용법) KWANGWOON</p> <ul style="list-style-type: none"> ✓ 현재 점에서 방문할 수 있는 정점(갈 수 있고 방문한적이 없는 정점)들이 홀수개면 그 정점 번호들의 가장 큰 정점 번호로 방문을 시작하고, 짝수개면 가장 작은 정점 번호로 방문을 시작하여 결과를 출력 파일(log.txt)에 저장한다. ✓ 해당 그래프 탐색 방법은 List Graph 에 저장된 데이터만 이용한다. 즉, graph_L.txt 데이터만 사용한다.

EXIT	<p>사용법) EXIT</p> <p>✓ 프로그램 상의 메모리를 해제하며, 프로그램을 종료한다.</p>
------	--

표 2. 명령어 기능 및 사용 예

□ Requirements in implementation

- ✓ 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
- ✓ 모든 명령어는 반드시 대문자로 입력하며, vertex는 숫자로 입력한다.
- ✓ 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력 받을 경우 오류 코드를 출력한다.
- ✓ 예외처리에 대해 반드시 오류 코드를 출력한다.
- ✓ 출력은 “출력 포맷”을 반드시 따른다.
- ✓ log.txt 파일에 출력 결과를 반드시 저장한다.
 - log.txt가 이미 존재할 경우 텍스트 파일 가장 뒤에 이어서 추가로 저장한다.

□ Error Code

명령어에 인자가 모자라거나 필요 이상으로 입력 받을 경우, 또는 각 명령어마다 오류 코드를 출력해야 하는 경우, 출력 파일(log.txt)에 명령어 별로 알맞은 오류 코드를 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 출력하여 저장한다. 각 명령어별 오류 코드는 다음과 같다.

동작	오류 코드
LOAD	100
PRINT	200
BFS	300
DFS	400
KWANGWOON	500
KRUSKAL	600
DIJKSTRA	700
BELLMANFORD	800

FLOYD	900
잘못된 명령어	1000

표 3. 에러 코드

□ Print Format

명령어 동작이 성공하였거나 실패한 경우, 출력 파일(log.txt)에 해당 내용을 지정된 형식에 맞추어 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 저장한다. 명령어별 출력 형식은 다음과 같다.

기능	출력 포맷	
오류(ERROR)	<pre>===== ERROR ===== 100 =====</pre>	
LOAD	<pre>===== LOAD ===== Success =====</pre>	
PRINT	<pre>List 그래프의 경우) ===== PRINT===== [1]-> (2,6) -> (3,2) [2]-> (4,5) [3]-> (2,7) -> (5,3) -> (6,8) [4]-> (7,3) [5]-> (4,4) [6]-> (7,1) [7]-> (5,10) =====</pre>	<pre>Matrix 그래프의 경우) ===== PRINT===== [1] [2] [3] [4] [5] [6] [7] [1] 0 6 2 0 0 0 0 [2] 0 0 0 5 0 0 0 [3] 0 7 0 0 3 8 0 [4] 0 0 0 0 0 0 3 [5] 0 0 0 4 0 0 0 [6] 0 0 0 0 0 0 1 [7] 0 0 0 0 10 0 0 =====</pre>
BFS	<pre>===== BFS ===== (Undirected)Directed Graph BFS result <-방향성(Y, N)에 따라 선택함 startvertex: 1 1 -> 2 -> 3 -> 4 -> 5 -> 6 =====</pre>	
DFS	<pre>===== DFS ===== (Undirected)Directed Graph DFS result <-방향성(Y, N)에 따라 선택함 startvertex: 1 1 -> 2 -> 4 -> 3 -> 6 -> 5 =====</pre>	
KwangWoon	<pre>===== KWANGWOON===== startvertex: 1 1 -> 3 -> 4 -> 6 -> 5 =====</pre>	

Kruskal	<pre> ===== Kruskal ===== [1] 3(2) [2] 4(5) [3] 1(2) 5(3) [4] 2(5) 5(4) 7(3) [5] 3(3) 4(4) [6] 7(1) [7] 4(3) 6(1) cost: 16 ===== </pre>
Dijkstra	<pre> ===== Dijkstra ===== (Undirected)Directed Graph Dijkstra result <-방향성(Y, N)에 따라 선택함 startvertex: 5 [1]x [2]x [3]x [4]5 -> 6 -> 4 -> 3 (15) [5]5 -> 6 -> 4 (11) [6]5 -> 6 (1) ===== </pre>
Bellman-Ford	<pre> ===== Bellman-Ford ===== (Undirected)Directed Graph Bellman-Ford result <-방향성(Y, N)에 따라 선택함 0 -> 2 -> 5 -> 6 cost: 11 ===== </pre>
FLOYD	<pre> ===== FLOYD ===== (Undirected)Directed Graph FLOYD result <-방향성(Y, N)에 따라 선택함 [1] [2] [3] [4] [5] [6] [7] [1] 0 6 2 7 5 10 10 [2] 6 0 7 5 7 9 8 [3] 2 7 0 5 3 8 8 [4] 7 5 5 0 2 4 3 [5] 5 7 3 2 0 6 5 [6] 10 9 8 4 6 0 1 [7] 10 8 8 3 5 1 0 ===== </pre>

표 4. 출력 형식

동작 예시

command.txt
LOAD graph_M.txt PRINT BFS Y 3 DIJKSTRA N 5 EXIT
실행 결과 화면 및 log.txt
<pre> ===== LOAD ===== Success ===== ===== PRINT ===== [1] [2] [3] [4] [5] [6] [7] [1] 0 6 2 0 0 0 0 [2] 0 0 0 5 0 0 0 [3] 0 7 0 0 3 8 0 [4] 0 0 0 0 0 0 3 [5] 0 0 0 4 0 0 0 [6] 0 0 0 0 0 0 1 [7] 0 0 0 0 10 0 0 ===== ===== BFS ===== Directed Graph BFS result 3 -> 1 -> 4 -> 6 -> 0 -> 2 -> 5 ===== ===== Dijkstra ===== Undirected Graph Dijkstra result startvertex: 5 [1]x [2]x [3]x [4]5 -> 6 -> 4 -> 3 (15) [5]5 -> 6 -> 4 (11) [6]5 -> 6 (1) ===== </pre>

표 5. 동작 예시

구현 시 반드시 정의해야 하는 Class 와 헤더파일

1. Graph : Graph 클래스

항목	내용	비고
m_Type	그래프 형식	List 일 경우 0, Matrix 일 경우 1
m_Size	그래프 크기	

표 6-1. Graph 클래스 멤버 변수

2. ListGraph : ListGraph 클래스

항목	내용	비고
m_List	그래프 데이터	map < int, int > * (map[from vertex].insert([to vertex], [weight])) Vector<int> kw_graph

표 6-2. ListGraph 클래스 멤버 변수

3. MatrixGraph : MatrixGraph 클래스

항목	내용	비고
m_Mat	그래프 데이터	int** ([from vertex][to vertex] = weight)

표 6-3. MatrixGraph 클래스 멤버 변수

4. GraphMethod: GraphMethod 헤더 파일

- 그래프 연산을 수행하는 일반 함수들을 모아둔 헤더 파일

항목	내용	비고
BFS		BFS(Graph* graph, char option, int vertex)
DFS		DFS(Graph* graph, char option, int vertex)
KwangWoon		KWANGWOON(Graph* graph, int vertex)
Kruskal		Kruskal(Graph* graph)
Dijkstra		Dijkstra(Graph* graph, ,char option, int vertex)
Bellmanford		BellmanFord(Graph* graph, char option, int s_vertex, int e_vertex)
FLOYD		FLOYD(Graph* graph, char option)

표 6-4. 그래프 연산 일반 함수

5. Manager: Manager 클래스

- 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

항목	내용	비고
graph	그래프 클래스	

표 6-5. Manager 클래스 멤버 변수

Files

- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ graph_L.txt : graph 데이터를 adjacency list 형식으로 저장하고 있는 파일
- ✓ graph_M.txt : graph 데이터를 adjacency matrix 형식으로 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 구현한다.
- ✓ **클래스의 함수 및 변수는 자유롭게 추가 구현 및 변경이 가능하다.**
(Union, Find, Check, Sort, Compare 등의 함수 필요시 추가로 구현)
- ✓ 제시된 Class 를 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)
 - ✓ 프로그램은 반드시 리눅스(Ubuntu 18.04)에서 동작해야 한다. (컴파일 에러 발생 시 감점)
- 제공되는 Makefile 을 사용하여 테스트하도록 한다.
- ✓ 메모리 누수가 발생하지 않도록 프로그램을 작성한다. (메모리 누수 발생 시 감점)

□ 채점 기준

- ✓ 코드

채점 기준	점수
LOAD 명령어가 정상 동작하는가?	1
PRINT 명령어가 정상 동작 하는가?	1
BFS 명령어가 정상 동작 하는가?	1
DFS 명령어가 정상 동작 하는가?	1
KWANGWOON 명령어가 정상 동작 하는가?	3
KRUSKAL 명령어가 정상 동작 하는가?	2
DIJKSTRA 명령어가 정상 동작 하는가?	2
BELLMENFORD 명령어가 정상 동작 하는가?	2
FLOYD 명령어가 정상 동작하는가?	2
총합	15

✓ 보고서

채점 기준	점수
Introduction 을 잘 작성하였는가?	1
Flowchart 을 잘 작성하였는가?	2.5
Algorithm 을 잘 작성하였는가?	3
Result Screen 을 잘 작성하였는가?	2.5
Consideration 을 잘 작성하였는가?	1
총합	10

□ 제출기한 및 제출방법

✓ 제출기한

- 2023 년 12 월 14 일 목요일 23:59:59 까지 제출

- 추가 제출 기한: 2023 년 12 월 15 일 금요일 23:59:59 까지(10% 감점)

✓ 제출방법

- 소스코드(Makefile 과 텍스트 파일 제출 x)와 보고서 파일(pdf)을 함께 압축하여 제출
 - 확장자가 .cpp, .h, .pdf 가 아닌 파일은 제출하지 않음(.txt 파일 제출 X)
- KLAS -> 과제 제출 -> tar.gz 로 과제 제출

✓ 제출 형식

- 학번_DS_project3.tar.gz (ex. 2023202001_DS_project3.tar.gz)

✓ 보고서 작성 형식 및 제출 방법

- Introduction: 프로젝트 내용에 대한 설명
- Flowchart: 설계한 프로젝트의 플로우 차트를 그리고 설명
- Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명
- Result Screen: 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명, 그래프 방향성 및 가중치에 따라 탐색 결과가 어떻게 바뀌는지 작성(따라서 그래프 데이터 만드실 때, 적어도 10 개 이상의 노드로 예제 만들어 주셔야 합니다)
- Consideration: 고찰 작성
 - 위의 각 항목을 모두 포함하여 작성
 - 보고서 내용은 한글로 작성
 - 보고서에는 소스코드를 포함하지 않음