데이터 구조

3차 프로젝트

학　　과: 컴퓨터정보공학부

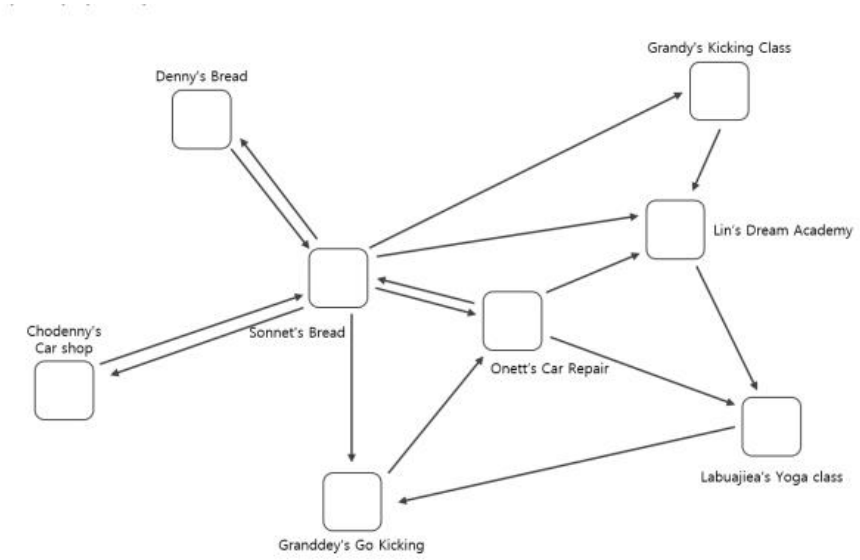
담당교수: 이기훈교수님

학　　번: 2017202087

성　　명: 홍 세 정

1. Introduction

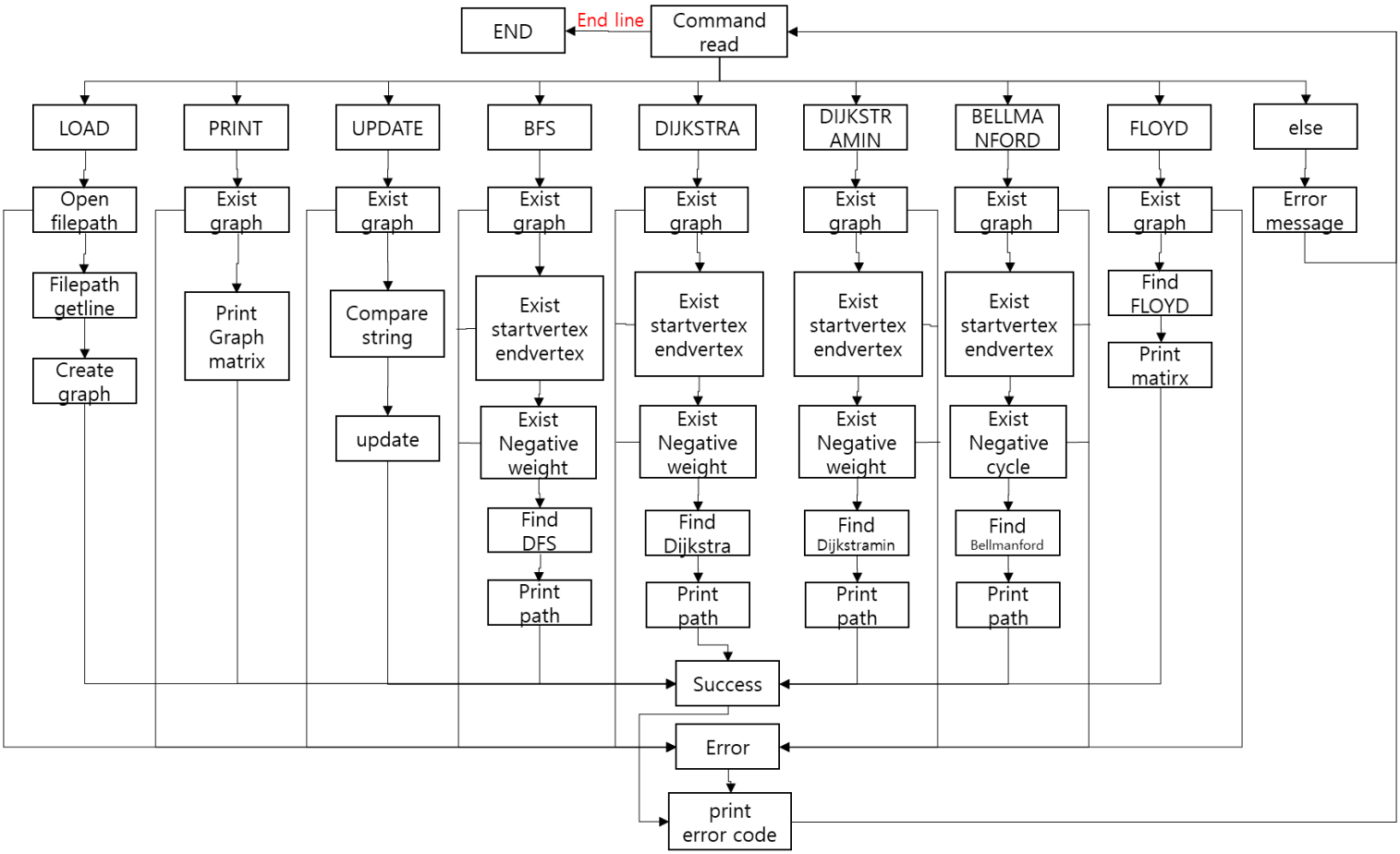
이번 3차 프로젝트에서는 도로 네트워크를 구현하고 DFS, Dijkstra, BellmanFord, Floyd 알고리즘을 이용하여 최단 경로를 찾는 알고리즘을 구현한다. 관광 프로그램을 제안하기 위한 최적의 코스를 제안한다. 또한 '라빈카프 공국 공식 압축 법칙'에 따라서 글자를 압축한다.



Mapdata.txt에 그래프에 대한 정보가 들어있다. 상점의 size와 이름, 상점간의 거리의 정보가 들어있다. 이 txt를 보고 그래프를 구현할 수 있다.

그래프를 생성하고 문자 비교를 통해 일부 그래프 가중치의 값을 변경하고 최소 비용 경로를 DFS, Dijkstra, BellmanFord, Floyd 알고리즘을 통해 탐색한다. Weight가 음수일 경우를 판단하여 다익스트라에서는 에러를 출력하고 음수 사이클이 발생할 경우 벨만포드에서는 에러를 출력한다.

2. Flowchart

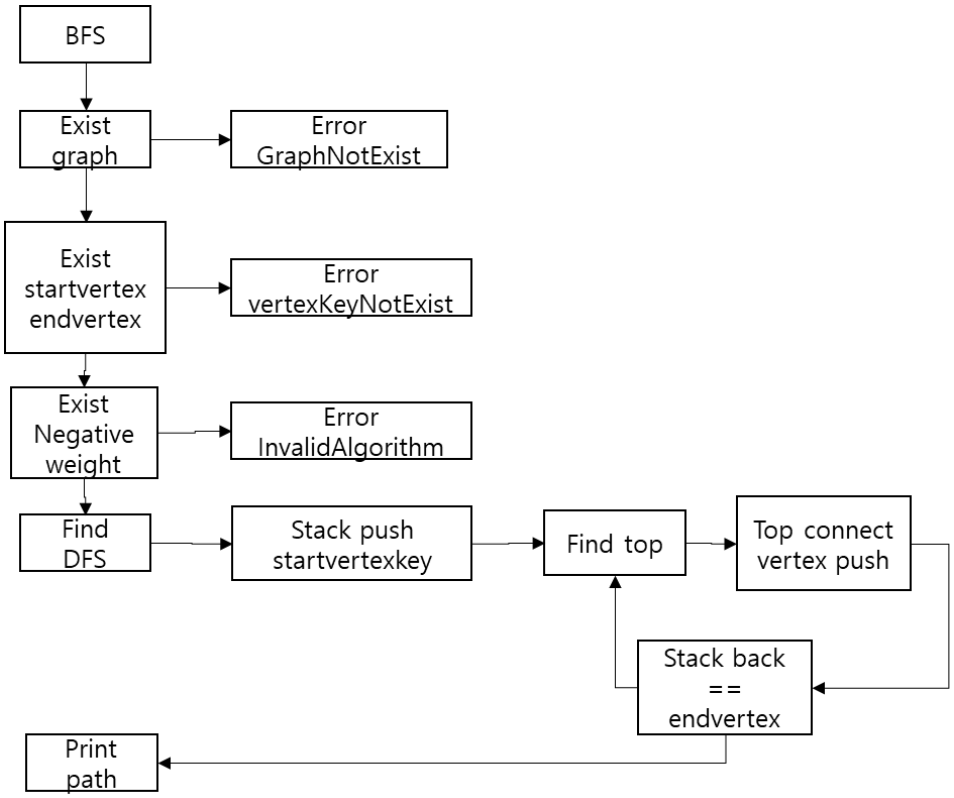


Flowchart를 다음과 같이 볼 수 있다.

main함수에서 manager 클래스를 불러와 명령어에 따라 결과를 출력할 수 있다.

명령어는 LOAD, PRINT, UPDATE, BFS, DIJKSTRA, DIJKSTRA, BELLMANFORD, FLOYD가 있다.

- BFS

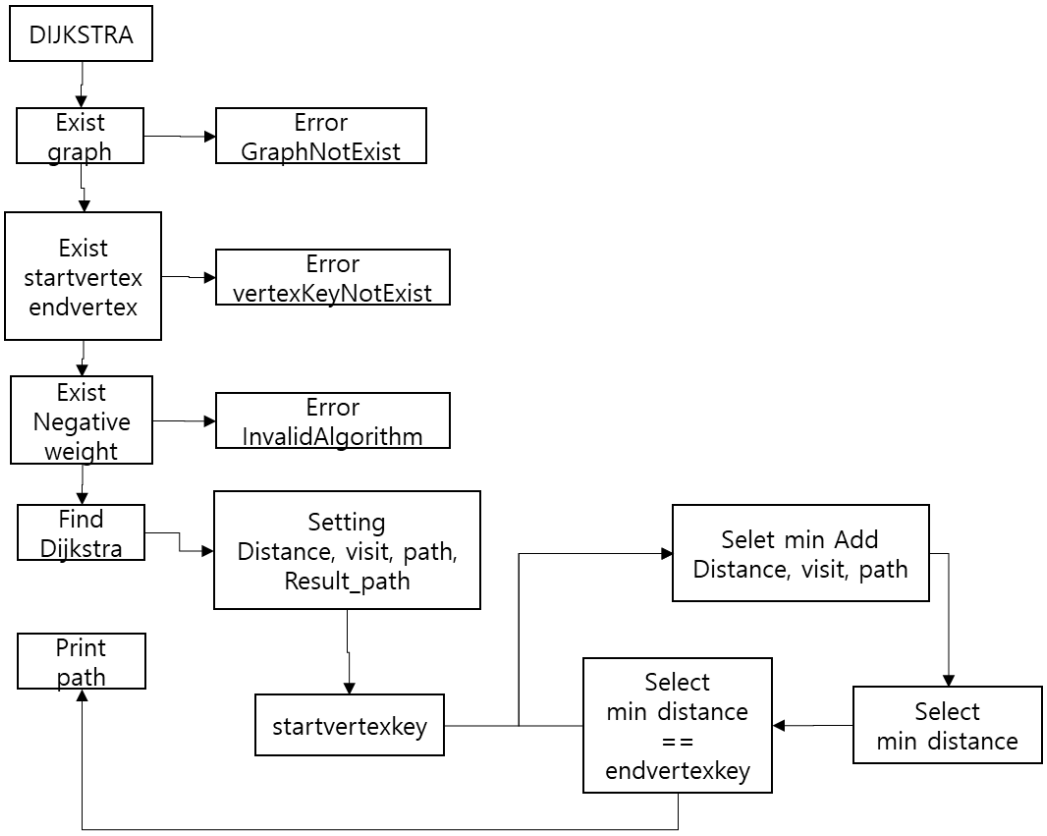


DFS를 알고리즘을 이용하여 최단 경로를 출력할 수 있다. BFS 명령어는 startvertexkey와 endvertexkey가 필수적으로 입력되어야 하고, vertexkey가 모두 vertex size안에 있어야한다. 그렇지 않으면 다음과 같은 error를 반환하게 된다.

Graph, startvertexkey, endvertexkey가 존재하고 negative weight가 발생하지 않는다면 DFS 알고리즘으로 최단경로를 출력할 수 있다.

DFS는 stack으로 구현할 수 있다. 첫번째 방문하는 vertex를 stack에 제일 먼저 저장한다. 그 후 제일 위에 있는 vertex, top을 선택하여 해당하는 vertex가 연결하는 edge를 모두 stack에 저장한다. Endvertex를 방문하였다면 반복문을 빠져나와 path를 출력할 수 있다. 반복문을 이용하여 top에서 이동할 수 있는 vertex를 stack에 저장하고 모두 저장하였다면 top을 다시 선택하여 stack에 저장하고, 이동할 경로가 없다면 다음 top으로 넘어가는 방식으로 DFS를 구현할 수 있다. Count가 0이 된다면 탐색 중 길이 막혔을 경우이다. 이미 지나갔다는 것을 표현하여 반복문을 종료하여 path를 출력할 수 있다.

- DIJKSTRA

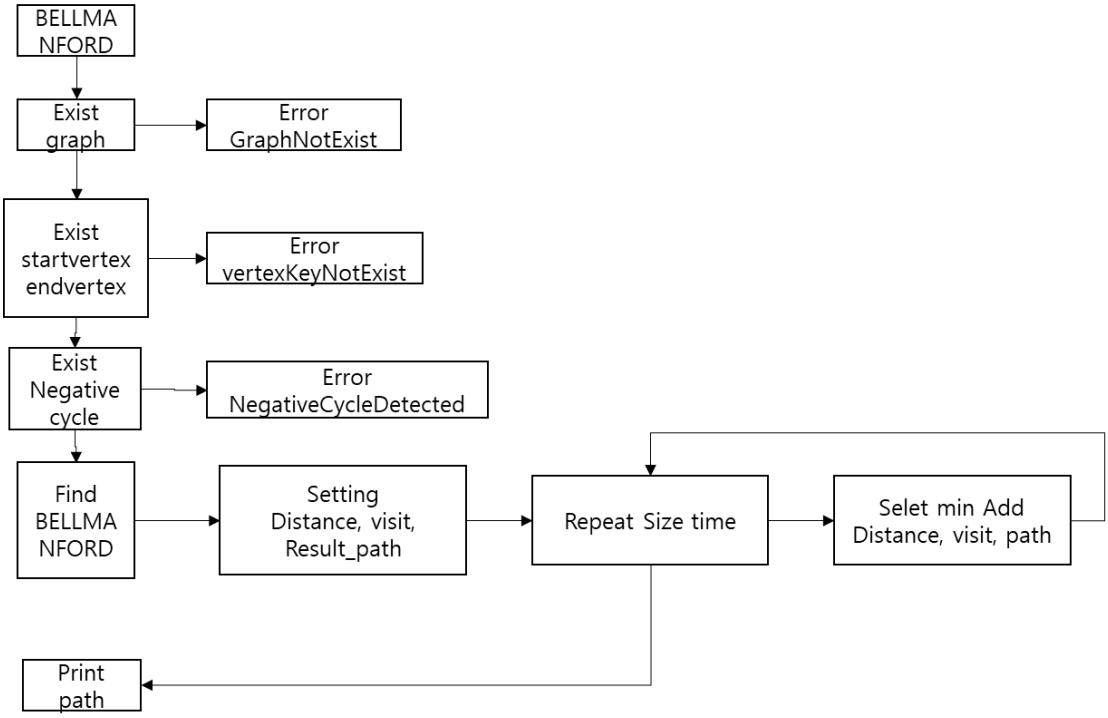


DIKSTRA는 다음과 같은 flowchart를 가지고 있다.

DIKSTRA도 마찬가지로 startvertexkey와 endvertexkey를 가지고 최단경로를 찾아주는 알고리즘이다. DIKSTRA를 구현하는 방법은 이번 프로젝트는 두가지로 나누었다. Set으로 구현가능하고 minheap으로 구현가능하다. 가장 작고 방문하지 않은 vertex를 다음에 방문할 vertex로 지정하는 작업만 다르다. Min heap도 구현하였다.

Graph가 존재하지 않을 때, startvertexkey와 endvertexkey가 존재하지 않을 때 negative weight가 존재할 때는 error가 발생한다. Error가 발생하지 않을 때 최단 경로를 구할 수 있다. DIJKSTRA는 distance, visit, path, result_path를 vector로 선언해줄 수 있었다. Distance는 startvertex부터 index까지의 weight를 visit는 방문을 하였는지 여부를 path는 index까지의 최단 경로에서 바로 전 vertex를 나타낸다. 다음과 같이 setting을 먼저한다음, 이전 distance보다 바뀌는 distance가 작다면 distance를 변경해준다. 모두 변경해주었다면, distance에서 방문되지 않았던 vertex중 가장 작은 값을 선택하여 다음 path를 결정할 수 있다. 이때 선택된 vertex가 endvertex 이면 반복문을 종료하고 path를 반환하여 path를 출력할 수 있다.

- BELLMANFORD

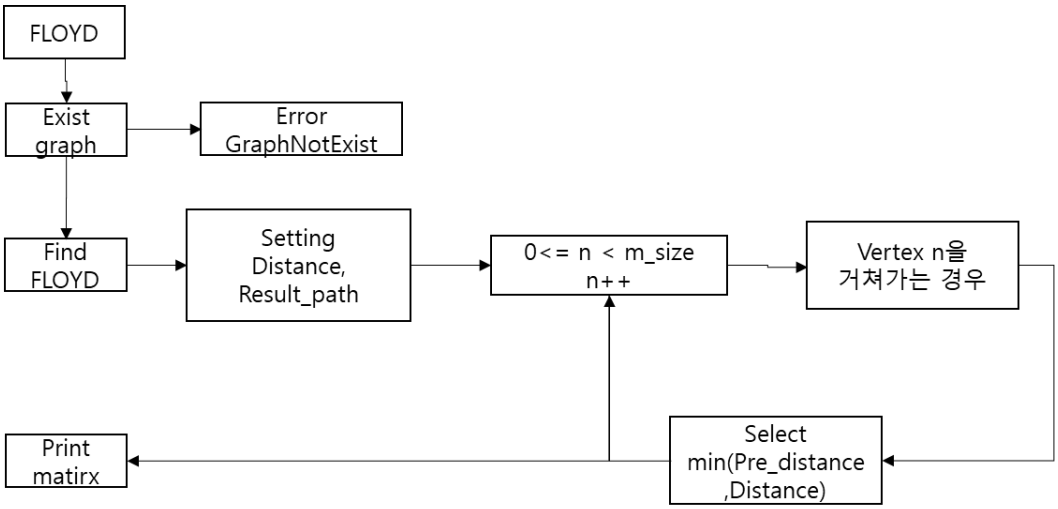


BELLMANFORD는 다음과 같은 flowchart를 가지고 있다.

Startvertexkey와 endvertexkey를 입력하여 최단경로를 찾는 알고리즘이다. Vertex를 몇 번 거쳐갔는지를 확인하여 최단 경로를 찾을 수 있다.

Graph가 존재하지 않을 때, Startvertexkey와 endvertexkey가 존재하지 않을 때 negative cycle이 존재할 때는 다음과 같은 error code가 수행된다. 최단 경로를 찾기 위한 조건이 모두 만족되었을 때 BELLMANFORD 알고리즘을 수행할 수 있다. 최단경로를 저장하고 있는 distance, 이전 방문한 vertex를 저장하고 있는 visit, startvertexkey에서 endvertexkey까지의 최단 경로를 저장하고 있는 result path를 선언할 수 있다. 다음 알고리즘을 구현하기 위해서 vertex를 0번 거쳐가는 경우부터 m_vSize번 거쳐가는 경우까지 총 m_vSize 반복하게 된다. N번 거쳐서 vertex까지 가는 경우가 N-1 거쳐서 가는 경우보다 작다면 distace와 visit의 값을 변경한다. 반복문을 마치면 최단 경로를 구할 수 있다. Visit를 이용하여 왔던 경로를 돌아가면서 result_path를 구할 수 있다.

- FLOYD



FLOYD 알고리즘의 flow chart이다.

FLOYD 알고리즘은 startvertexkey와 endvertexkey가 존재하지 않고 모든 vertex의 최단 경로를 출력한다. 그래서 distance는 2차원 vector로 선언되어 있다. FLOYD 알고리즘은 0~vertex_size만큼 반복하여 0을 거쳐 갔을때와 이전 distance와 비교하여 weight가 더 작은 것을 선택한다.

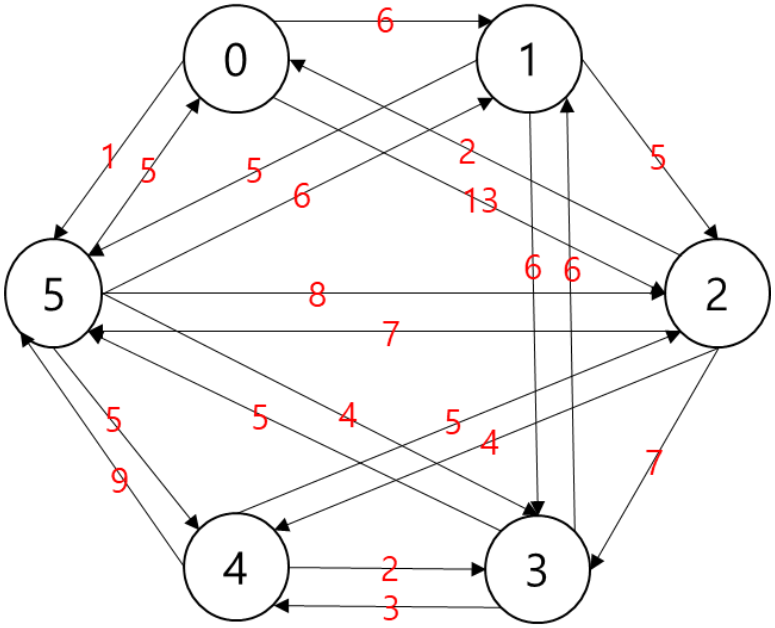
3. Algorithm

	0	1	2	3	4	5
0	0	6	13	0	0	1
1	0	0	5	6	0	5
2	2	0	0	7	4	7
3	0	6	0	0	3	5
4	0	0	5	2	0	9
5	5	6	8	4	5	0

다음 데이터를 가지는 graph를 예로 다음 알고리즘들을 설명한다.

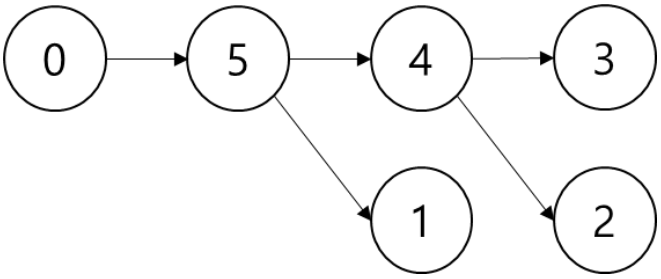
Vertex는 총 6개를 가지고 행과 열은 각각 edge의 stsart vertex와 end vertex를 의미한다.

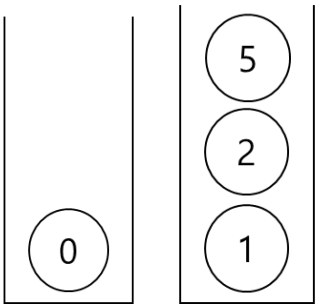
Start vertex와 end vertex가 같으면 0으로 가진다.



다음과 같은 graph를 가지게 된다.

- DFS 0 3





다음과 같이 stack에 startvertex를 넣은 후 top을 꺼내 top과 인접한 vertex를 stack에 넣는다. 이것을 모든 vertex를 방문할 때까지 반복한다.

Startvertex가 0이기 때문에 0부터 넣었다. 0을 꺼내어 0이 인접한 vertex를 vertexkey가 낮은 순으로 넣었다. 그 후 5를 pop하여 5와 인접한 index를 순서대로 push할 수 있다.

- DIJKSTRA

vertex	0	1	2	3	4	5
distance	0	IN_FINTY	IN_FINTY	IN_FINTY	IN_FINTY	IN_FINTY
visit	0	0	0	0	0	0
path	-1	-1	-1	-1	-1	-1

다음과 같이 초기화 할 수 있다. Startvertexkey가 0일 때이다.

vertex	0	1	2	3	4	5
distance	0	6	13	IN_FINTY	IN_FINTY	1
visit	0	0	0	0	0	0
path	-1	0	0	-1	-1	0

distance중 visit가 0이면서 가장 작은 수를 선택하고 visit를 1로 변경해준다.

vertex	0	1	2	3	4	5
distance	0	6	9	5	6	1
visit	1	0	0	0	0	1
path	-1	0	5	5	5	0

선택한 vertex를 거쳐가는 것 중 distace가 더 작으면 distance를 값을 변경한다.

vertex	0	1	2	3	4	5
distance	0	6	9	5	6	1
visit	1	0	0	1	0	1
path	-1	0	5	5	5	0

마찬가지로 반복한다.

vertex	0	1	2	3	4	5
distance	0	6	9	5	6	1
visit	1	1	0	1	0	1
path	-1	0	5	5	5	0

vertex	0	1	2	3	4	5
distance	0	6	9	5	6	1
visit	1	1	0	1	1	1
path	-1	0	5	5	5	0

vertex	0	1	2	3	4	5
distance	0	6	9	5	6	1
visit	1	1	1	1	1	1
path	-1	0	5	5	5	0

따라서 최종 path는 다음과 같다.

지금까지 구한 것은 startvertex가 0인 path였다. 나머지도 구해보면

Startvertex가 1일때

0	1	2	3	4	5
0	0	5	6	0	5
7	0		6	9	5
7	0		6	9	
7	0			9	
	0			9	

Startvertex가 2일때

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

Startvertex가 3일때

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

Startvertex가 4일때

0	1	2	3	4	5
0	0	5	2	0	9
0	8	5		0	7
7	8			0	7
	8			0	7
	8			0	

Startvertex가 5일때

0	1	2	3	4	5
5	6	8	4	5	0
5	6	8		5	0
	6	8		5	0
	6	8			0
		8			0

다음과 같이 DIJKSTRA로 최단 경로를 구할 수 있다.

- BELLMANFORD

	0	1	2	3	4	5
0	0	INFINTY	INFINTY	INFINTY	INFINTY	INFINTY
1	0	6	13	INFINTY	INFINTY	1
2	0	6	9	5	6	1
3	0	6	9	5	6	1
4	0	6	9	5	6	1
5	0	6	9	5	6	1

행과 열은 각각 vertex와 vertex 거쳐간 횟수를 나타낸다.

이 graph는 거쳐가 횟수가 2번째 될 때 모두 바뀌는 단순한 graph지만, weight가 음수가 존재하거나 더 복잡하다면 거쳐간 횟수가 m_vsize일 때 최종 path가 나올 수 있다.

	0	1	2	3	4	5
0	0	INFINTY	INFINTY	INFINTY	INFINTY	INFINTY
1	0	0	0	INFINTY	INFINTY	0
2	0	0	5	5	5	0
3	0	0	5	5	5	0
4	0	0	5	5	5	0
5	0	0	5	5	5	0

Visit를 나타낸 것이다. 이전 vertex를 나타냈다.

- FLOYD

	0	1	2	3	4	5
0	0	6	13	INFINTY	INFINTY	1
1	INFINTY	0	5	6	INFINTY	5
2	2	INFINTY	0	7	4	7
3	INFINTY	6	INFINTY	0	3	5
4	INFINTY	INFINTY	5	2	0	9
5	5	6	8	4	5	0

Floyd는 다음과 같이 초기 세팅을 해놓고 시작한다. Matrix를 다음과 같이 놓은 후 중간에 0 vertex를 지나가는 distance와 비교해서 더 작은 것을 선택하는 방식이다.

	0	1	2	3	4	5
0	0	6	13	INFINTY	INFINTY	1
1	INFINTY	0	5	6	INFINTY	5
2	2	8	0	7	4	3
3	INFINTY	6	INFINTY	0	3	5
4	INFINTY	INFINTY	5	2	0	9
5	5	6	8	4	5	0

0 vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	11	12	INFINTY	1
1	INFINTY	0	5	6	INFINTY	5
2	2	8	0	7	4	3
3	INFINTY	6	11	0	3	5
4	INFINTY	INFINTY	5	2	0	9
5	5	6	8	4	5	0

1 vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	11	12	15	1
1	7	0	5	6	9	5
2	2	8	0	7	4	3
3	13	6	11	0	3	5
4	7	13	5	2	0	8
5	5	6	8	4	5	0

2 vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	11	12	15	1
1	7	0	5	6	9	5
2	2	8	0	7	4	3
3	13	6	11	0	3	5
4	7	8	5	2	0	7
5	5	6	8	4	5	0

3vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	11	12	15	1
1	7	0	5	6	9	5
2	2	8	0	6	4	3
3	10	6	8	0	3	5
4	7	8	5	2	0	7
5	5	6	8	4	5	0

4vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	9	5	6	1
1	7	0	5	6	9	5
2	2	8	0	6	4	3
3	10	6	8	0	3	5
4	7	8	5	2	0	7
5	5	6	8	4	5	0

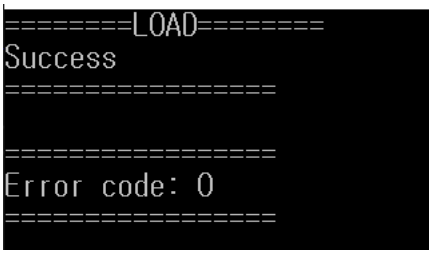
5vertex를 거쳐가는 path

	0	1	2	3	4	5
0	0	6	9	5	6	1
1	7	0	5	6	9	5
2	2	8	0	6	4	3
3	10	6	8	0	3	5
4	7	8	5	2	0	7
5	5	6	8	4	5	0

다음과 같은 과정으로 최종 path는 다음과 같다.

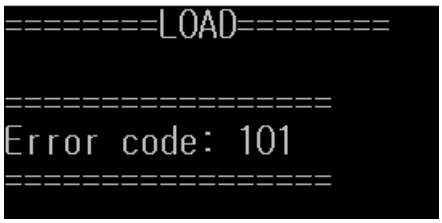
4. Result Screen

- LOAD



mapdata.txt를 읽어오는 것을 성공하고, graph를 구현하였다.

구현을 성공하였으므로 Success를 출력하고 error code를 0 출력한다. 0은 success를 의미한다.



알맞은 txt가 없을 때 에러 코드 101이 출력된다.

- PRINT

PRINT를 성공하였다. 다음과 같이 matrix가 출력되는 것을 확인할 수 있다.

각각 행과 열은 Edge의 Start Vertex와 End Vertex의 Weight를 의미한다.

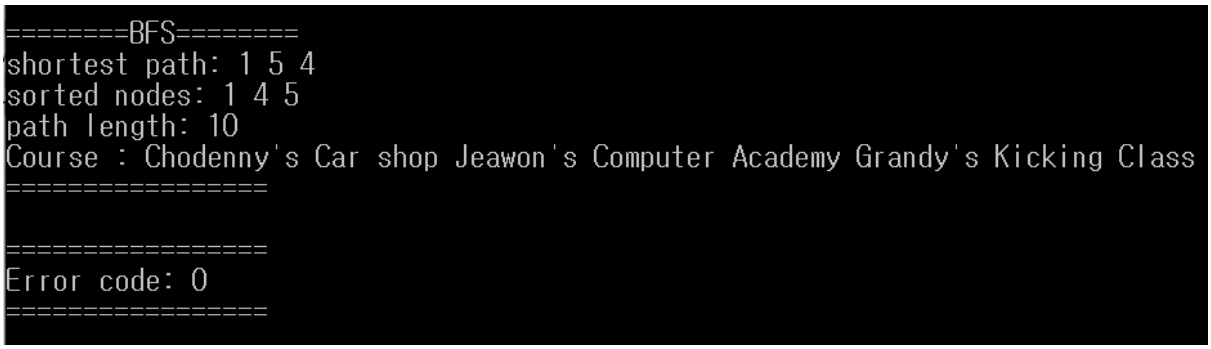
Print를 성공하였으므로 error code : 0 이다.

graph가 없을 때 다음과 같이 출력된다.

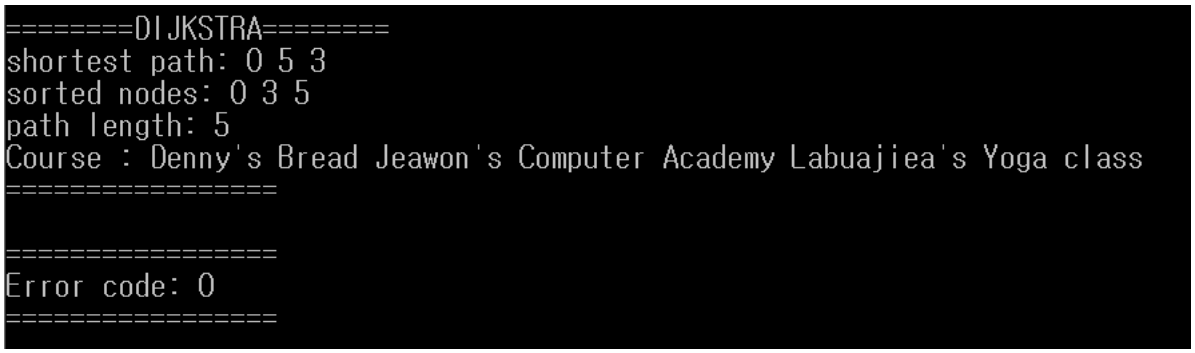
- BFS 0 3

BFS의 path와 sorted 된 노드와 course를 출력하는 것을 확인할 수 있었다.

알맞게 출력이 되었으므로 error code: 0으로 출력되었다.



- DIJKSTRA 0 3



shortest path와 sorted nodes, course가 출력된 것을 확인할 수 있다.

알맞게 출력이 되었으므로 error code: 0으로 출력되었다.

- DIJKSTRAMIN 0 3

```
========DIJKSTRAMIN========
shortest path: 0 5 3
sorted nodes: 0 3 5
path length: 5
Course : Denny's Bread Jeawon's Computer Academy Labuajiea's Yoga class
==================

==================
Error code: 0
==================
```

DIJKSTRAMIN도 DIJKSTRA와 같이 출력되었다.

- BELLMANFORD 0 3

```
========BELLMANFORD========
shortest path: 0 5 3
sorted nodes: 0 3 5
path length: 5
Course : Denny's Bread Jeawon's Computer Academy Labuajiea's Yoga class
==================

==================
Error code: 0
==================
```

- FLOYD

```
========FLOYD========
0 6 9 5 6 1
7 0 5 6 9 5
2 8 0 6 4 3
10 6 8 0 3 5
7 8 5 2 0 7
5 6 8 4 5 0
==================

==================
Error code: 0
==================
```

- ASTAR 1 4

```
========ASTAR========
not command
==================
```

ASTAR 명령어는 구현되지 않았기 때문에 다음과 같이 not command로 출력된다.

```
========LOAD========
Success
==================

==================
Error code: 0
==================

========PRINT========
0 6 13 0 0 1
0 0 5 6 0 5
2 0 0 7 4 7
0 6 0 0 3 5
0 0 5 2 0 9
5 6 8 4 5 0
==================

==================
Error code: 0
==================

========BFS========
shortest path: 0 5 4 3
sorted nodes: 0 3 4 5
path length: 8
Course : Denny's Bread Jeawon's Computer Academy Grandy's Kicking Class Labuajiea's Yoga class
==================

==================
Error code: 0
==================

========FLOYD========
0 6 9 5 6 1
7 0 5 6 9 5
2 8 0 6 4 3
10 6 8 0 3 5
7 8 5 2 0 7
5 6 8 4 5 0
==================
```

========BFS========
shortest path: 0 5 4 3
sorted nodes: 0 3 4 5
path length: 8
Course : Denny's Bread Jeawon's Computer Academy Grandy's Kicking Class Labuajiea's Yoga class
==================

==================
Error code: 0
==================

========FLOYD========
0 6 9 5 6 1
7 0 5 6 9 5
2 8 0 6 4 3
10 6 8 0 3 5
7 8 5 2 0 7
5 6 8 4 5 0
==================

==================
Error code: 0
==================

========DIJKSTRA========
shortest path: 0 5 3
sorted nodes: 0 3 5
path length: 5
Course : Denny's Bread Jeawon's Computer Academy Labuajiea's Yoga class
==================

==================
Error code: 0
==================

========DIJKSTRAMIN========
shortest path: 0 5 3
sorted nodes: 0 3 5
path length: 5
Course : Denny's Bread Jeawon's Computer Academy Labuajiea's Yoga class
==================

```
========BFS========
shortest path: 1 5 4
sorted nodes: 1 4 5
path length: 10
Course : Chodenny's Car shop Jeawon's Computer Academy Grandy's Kicking Class
==================

==================
Error code: 0
==================

========BELLMANFORD========
shortest path: 0 5 3
sorted nodes: 0 3 5
path length: 5
Course : Denny's Bread Jeawon's Computer Academy Labuajiea's Yoga class
==================

==================
Error code: 0
==================

========DIJKSTRA========

==================
Error code: 201
==================

========BELLMANFORD========

==================
Error code: 200
==================

========ASTAR========
not command
==================
```

Log.txt도 다음과 같이 저장되어 있는 것을 확인할 수 있다.

5. Consideration

이번 프로젝트는 다양한 알고리즘으로 최단 경로를 구현하는 프로젝트였다.

프로젝트의 설명을 처음 읽었을 때는 1차,2차 프로젝트에 비해 어렵지 않을 것이라고 생각했지만 생각보다 구현할 양과 최단 경로 구하는 알고리즘의 이해가 쉽지 않았다. 설명을 들었을 때보다 구현할 때 생각보다 쉽게 구현되지 않았다.

PDF의 결과가 잘못되어 있어서 더 혼동이 왔던 것 같다. Print와 path가 잘못되어 있어서 올바른 결과가 나왔지만 잘못된 결과인 줄 알고 헤맸다. 확인해본 결과 알맞게 결과가 출력된 것 알 수 있었다. Print 명령어 수행할 때 vertex수에 맞도록 size x size로 출력되어야 한다. 이번 프로젝트 PDF에서 마지막 행과 열이 포함되어 있지 않아 나와 결과 값이 다르게 나왔다.

DIJKSTRA와 DIJKSTRAMIN의 차이가 무엇인지 이해할 수 없었다. DIJKSTRA는 강의를 이해하면서 차근차근 구현 가능하였다. DIJKSTRAMIN은 minheap을 이용하고 DIJKSTRA는 set을 이용하여 구현하는 것이 목표였다. 두개의 차이가 무엇인지 알 수는 없었지만, 내 생각은 최종 path를 set으로 구성하는가 minheap으로 구성하는가 라는 생각으로 구현할 수 있었다. Set은 STL로 쉽게 구현 가능하지만 minheap은 직접 minheap을 구현하여 DIJKSTRA를 구현해야 했다. Min heap을 구현하는 것도 생각보다 막막했다. 그래서 개념을 다시 잡고 어떤 역할을 해야하는지 파악하고 minheap을 구현할 수 있었다.

Negative cycle의 형태를 잘 이해를 하지 못했었다. 거리를 구하는데 거리라면 무조건 양수를 가져야할 것 같은데 weight가 음수로 나올 수 있다는 것도 이해하지 못하였다. 아직 이해는 하지 못하였지만 음수를 가질 수 있다고 가정하고 알고리즘을 구현할 수 있었다. 일단 음수 cycle을 가지고 있지 않다고 가정하고 알고리즘을 모두 구현한 후 음수 사이클을 생각하였다. 구글링하면서, 그래프를 직접 그려보면서 어떻게 해야 negative cycle을 갖는지 확인하고 구현할 수 있었다.

라빈카프 알고리즘에 대해 처음 들어보았다. 설명을 봐도 사실 무슨 말인지 잘 이해하지 못하였다. 그래서 라빈카프 알고리즘에 대해서 구글링과, 실습자료를 통해서 이해를 먼저 한 후 진행하였다. 처음은 알파벳 한글자라도 같으면 같다고 판단하는지에 대해서 의문을 가졌다. 하지만 글자 형태가 아닌 단어 형태로 비교하는 것이었다. 예시들과 여러 단어들로 조합해본 결과 알고리즘을 파악할 수 있었다.