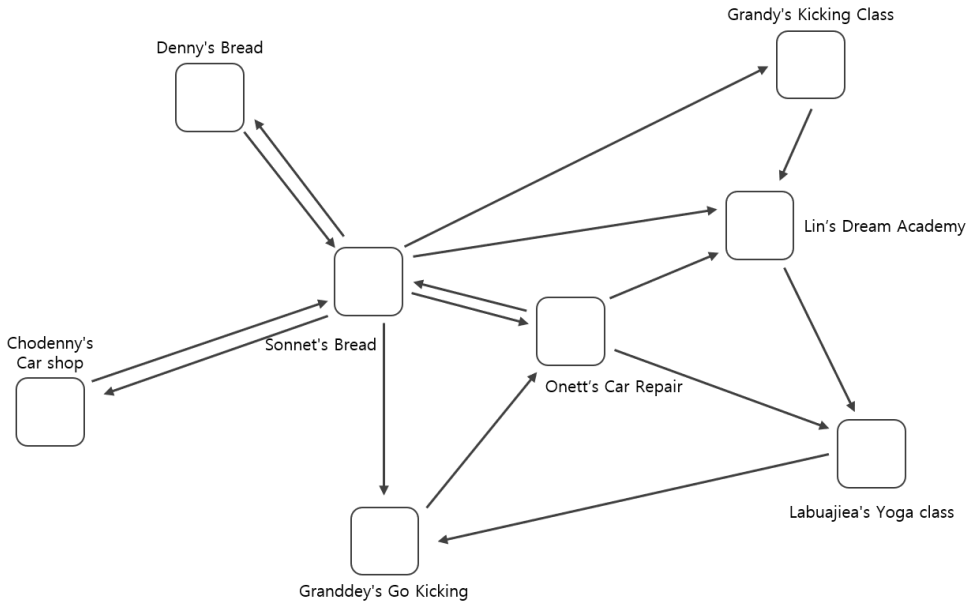
Data Structure Lab. Project #3

당신은 라빈카프 공국에 온지 얼마되지 않은 신입 공무원이다. 당신의 업무는 라빈카프 공국 관광프로그램을 제안하기 위한 최적의 코스를 제안하는 것이다. 당신은 라빈카프 공국의 가게 지도를 바탕으로 가장 합당한 관광코스를 제안해야 한다. 이를 위해서 그래프 최단 경로 탐색 알고리즘을 작성하여 최단 경로를 제안한다. 제안된 여행경로를 본 당신의 상관은 너무나도 감명받은 나머지 여행 경로를 광장에 새기고자 한다. 하지만 광장에 글자 새기는 비용이 만만치 않았기 때문에 상관은 당신에게 '라빈카프 공국 공식 압축 법칙'에 따라서 글자를 압축해 오기를 바라게 되는데...

본 프로젝트에서는 그래프를 통한 최단 경로 탐색 프로그램을 구현한다. 이프로그램은 각 노드(상점)간의 경로를 찾게 된다. 이 프로그램은 도시에 대한 정보를 가지고 있는 로그 파일을 바탕으로 그래프를 생성하게 되며 구현된 그래프를 바탕으로 경로 탐색을 시행하게 된다. 상점의 거리 그래프 정보는 방향성과 가중치를 모두 가지고 있으며 Matrix 형태로 저장되어있다. 그래프를 생성한 이후 문자 비교를 통해 일부 그래프 가중치의 값을 변경하고 최소 비용 경로를 다익스트라,벨만포드,플로이드 알고리즘을 통해 탐색한다. 만약 Weight가 음수일 경우 다익스트라는 에러를 출력하며, 벨만-포드에서는 음수 사이클이 발생할 경우 에러 음수 사이클이 발생하지 않았을 경우 최단 경로와 거리를 구한다. 플로이드 에서는 음수 사이클이 발생한 경우 에러, 음수 사이클이 발생하지 않았을 경우 최단 경로와 거리를 구한다. 이후 다시 라빈 카프 알고리즘을 통해 문자열을 수정하게 된다.



[그림1] 프로젝트 최단경로 찾기 예시

□ **Program implementation**

1. 가게 정보 데이터

프로그램은 방향성과 가중치를 가지고 있는 그래프 정보를 형태로 섬의 거리 데이터를 저장한 Matrix텍스트 파일을 명령어를 통해 읽어 해당 정보를 클래스에 저장한다. 가게 데이터 텍스트 파일의 첫번째 줄에는 상점의 개수가 저장되어 있고 다음 줄 부터 섬의 거리 데이터가 저장되어 있다. 거리 데이터의 행과 열은 각각 Edge의 Start Vertex와 End Vertex의 Weight를 의미한다. 이중 첫번째 열은 '상점의 이름' 이다.

줄 수	내용
1	그래프 크기(n)
2...n+1	[가게이름]/[weight_1] [weight_2] ... [weight_n]

표 1. 도로 정보 데이터 형식

```
6
Denny's Bread / 0 6 13 0 0 1
Chodenny's Car shop / 0 0 5 6 0 5
Granddey's Go Kicking / 2 0 0 7 4 7
Labuajiea's Yoga class / 0 6 0 0 3 5
Grandy's Kicking Class / 0 0 5 2 0 9
Jeawon's Computer Academy / 5 6 8 4 5 0
```

그림 2. 상점 정보 데이터가 저장되어 있는 텍스트 파일의 예

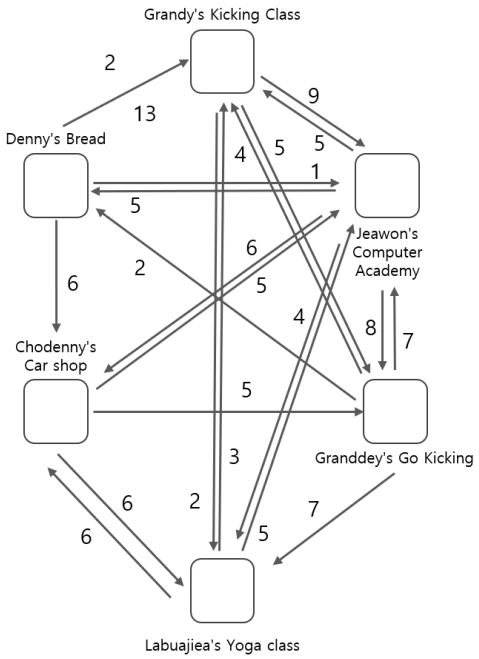


그림 3. 그림2의 맵 연결 예

Vertex와 Edge는 Linked-List를 이용하여 연결하며, 맵 데이터를 순차적으로 읽어 Linked-List 의 가장 끝에 연결한다(순차적으로 연결하여 반드시 Vertex 오름차순으로 정렬된 형태로 저장되어 있어야 한다). 만약 **Linked-List의 값이 변경되어야 한다면 노드의 값을 수정하는게 아닌 새로운 노드를 생성하고 교체할 기존 노드를 삭제 해야 한다. (오름차 순으로 정렬된 상태이여야 함에 주의한다.)**

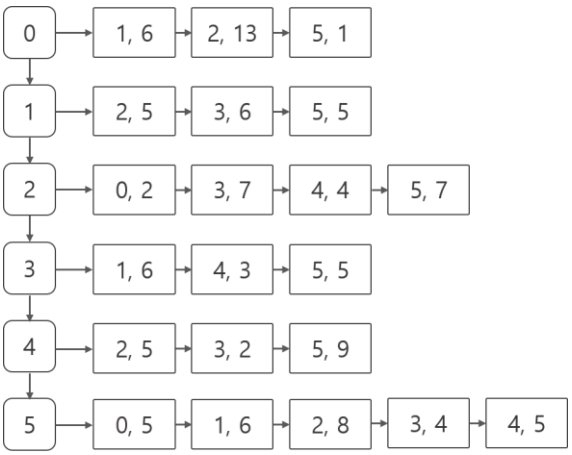


그림 4. Linked-List 연결 예시

그래프의 최단경로 탐색 알고리즘을 탐색하기 전에 그래프의 가중치 정보는 가게의 이름에 따른 규칙에 의해 변경되어야 한다. 규칙은 아래와 같다. (3번 항목을 제외하고 비교항목에 공백문자가 있을경우 포함해서 비교하도록 한다.)

라빈카프 공국 규칙 (대소문자는 무시한다)

1. 상점 가게 주인의 이름이 5글자 이상 동일한 가게(대소문자의 구분은 없다.) (예. Stubban's Shoes Store / Tubban's Care Rental)는 간선 비용이 10% 낮아진다. (소수점은 올림한다.)
2. 1의 알고리즘이 적용된 이후 가게 주인의 이름이 10글자 동일 했다면 1의 규칙이 적용된 이후 간선 비용을 10% 낮춘다. (소수점은 올림한다.)
3. 2의 규칙이 적용된 이후에 남은 간판명이 한 단어 이상 같은 가게는 간선비용이 20% 낮아진다. (예. Tor's toy car shop, Goy's car toy shop) 단어의 순서는 상관 없다. (소수점은 올림한다.)
4. 단어의 비교는 오직 라빈카프 알고리즘을 통해서만 이루어 져야 한다

~~이후 변경된 Graph의 정보를 입력 텍스트 파일과 같이 출력한다. (초기 1회~~) [1_2 업데이트]

2. 그래프 연산

프로그램은 텍스트 파일로부터 도로 정보 데이터를 읽은 뒤, 명령어에 따라 알맞은 그래프 연산을 수행할 수 있다.

- BFS

BFS 명령어는 Start Vertex와 End Vertex를 입력받아 경로와 거리를 구한다. 큐를 활용하여 구현한다. 이번과제에서 큐를 직접 구현하여 사용한다.

- DIJKSTRA

DIJKSTRA 명령어는 Start Vertex와 End Vertex를 입력받아 최단 경로와 거리를 구한다. Dijkstra 연산을 위해 STL set을 이용하여 최단 거리와 경로를 구하고, 결과로 찾은 최단 경로를 다시 순회하여 Start Vertex부터 End Vertex 순으로 경로를 구한다.

- DIJKSTRAMIN

DIJKSTRAMIN 명령어는 DIJKSTRA 명령어와 동작이 동일하며, Dijkstra 연산을 위해 STL set을 활용 할 수 있다. (직접 구현하여도 무방)

- BELLMANFORD

BELLMANFORD 명령어는 Start Vertex와 End Vertex를 입력받아 최단 경로와 거리를 구한다. Weight가 음수인 경우에도 정상 동작하며, 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

- FLOYD

FLOYD 명령어는 입력 인자 없이 모든 쌍에 대해서 최단 경로 행렬을 구한다. Weight가 음수 인 경우에도 정상 동작하며, 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

3. 정렬 연산

정렬 연산은 최단 경로로 방문한 노드를 정렬하여 출력한다 이번 프로젝트에서는 다양한 정렬 알고리즘의 사용을 통한 성능 비교를 진행한다.

성능의 측정은 Chrono 라이브러리를 통한 수행 시간을 통해 비교를 진행한다. (Chrono라이브러리의 사용이 힘들 경우 time으로 대체한다.)

```
std::chrono::duration<double> sec = std::chrono::system_clock::now() - start;
```

Chrono 라이브러리를 통한 동작 시간 구하기 예제

구현하여 성능 비교를 진행해야 하는 정렬 알고리즘은 아래와 같다.

- Quick Sort
- Insert Sort
- Merge Sort
- Heap Sort
- Bubble Sort

각각의 정렬 알고리즘을 구현하여 보고 정렬 알고리즘을 바꾸었을 때 각각의 명령어 수행에 몇 초 간의 수행시간이 걸렸는지 도표로 정리하여 보고서에 추가한다. (정렬 알고리즘을 사용하게 되는 명령어에 한하여) (추가적으로 구현하고 싶은 정렬 알고리즘이 있다면 자유롭게 작성 가능하다, 추가점수 有)

명령어 : DIJKSTRA 0 3	
Sort	Time
Quick Sort	2270100 nano second
Insert Sort	…
Merge Sort	…
Bogo Sort	…
Radix Sort	…

작동 결과 정리 예시

4. 문자열 압축하기

최단 경로를 구하는 모든 알고리즘에 대하여 최단 경로를 찾고 난 후의 최종 출력물에 대하여 광장에 글자 새기기를 위한 다음과 같은 추가 작업을 진행한다.

만약 3에서의 출력이 아래와 같다면

```
===== DIJKSTRA =====

order: 0 1 3

segment size: 3

sorted order: 0 1 3

total cost: 12

=================
```

이는 0 1 3 가게 이름의 조합 즉

Denny's Bread Chodenny's Car shop Labuajiea's Yoga class으로 나타낼 수 있다. 이러한 문자열은 아래와 같은 법칙으로 압축될 수 있다.

라빈카프 공국 공식 압축 법칙 (대소문자는 무시한다)

1. 단어의 비교는 오직 라빈카프 알고리즘을 통해서만 이루어 져야 한다
2. 주어진 문자열에 대하여, 두 번 이상 등장한 부분 문자열 중 가장 길이가 긴 문자열은 해당 단어 뒤에 '*'를 붙임으로서 압축 할 수 있다. (길이가 동일 할 경우 모든 단어에 대해서 압축 할 수 있다, 공백 또한 함께 압축한다.)

   예) Benney's Carshop Benney's Oakshop -> Benny's *Carshop Oakshop
3. 2번 규칙의 처리이후 Shop, Class, Academy 단어들은 최종적으로 문자열에서 한번 만 나타나게 한다.

   예) Benny's* Carshop Oakshop ->Benny's* Car Oakshop

따라서 최종 출력은 아래와 같게 된다.

```
===== DIJKSTRA =====
order: 0 1 3
segment size: 3
sorted order: 0 1 3
total cost: 12
Course : Bread Chodenny's* Car shop Labuajiea's Yoga class
================
```

☐ **Program implementation**

프로그램은 명령어를 통해 동작하며, 실행할 명령어가 작성되어있는 커맨드 파일(command.txt)을 읽고 명령에 따라 순차적으로 동작한다. 추가로 명령어를 주석 처리 할 수 있는 기능 또한 구현 한다. 각 명령어 앞에 '//'기호가 있다면 해당 명령은 무시되고 실행된다. 각 명령어의 사용법과 기능은 다음과 같다.

명령어	명령어 사용 예 및 기능
LOAD	사용법) LOAD textfile 프로그램에 도로 정보 데이터를 불러오는 명령어로, 도로 정보 데이

	터가 저장되어있는 파일을 읽어 그래프를 구성한다. 텍스트 파일이 존재하지 않을 경우, 로그 파일(log.txt)에 오류 코드(101)를 출력한다. 하나의 커맨드 파일에서 LOAD가 2번 이상 성공하는 경우는 고려하지 않는다.
UPDATE	사용법) UPDATE LOAD 명령어로 로드된 데이터에 대하여 라빈카프 공국 규칙에 따른값 업데이트를 진행한다. 업데이트에 문제가 있을 경우 로그 파일(log.txt)에 오류코드(005)를 출력한다.
PRINT	사용법) PRINT 도로 정보 데이터를 읽어 도로 정보를 출력하는 명령어로, Matrix 형태로 도로 정보를 출력한다. 도로 정보 데이터가 존재하지 않을 경우, 로그 파일(log.txt)에 오류 코드(202)를 출력한다.
BFS	사용법) BSF StartVertex EndVertex 예시) BSF 0 3 StartVertex를 기준으로 BFS를 수행하여 EndVertex까지의 경로와 거리를 구하는 명령어로, BFS결과를 로그 파일(log.txt)에 저장한다. BSF를 수행하고 경로를 StartVertex부터 EndVertex까지 순서대로 거리와 함께 저장한다. 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
DIJKKSTRA	사용법) DIJKSTRA StartVertex EndVertex 예시) DIJKSTRA 0 3 StartVertex를 기준으로 Dijkstra를 수행하여 EndVertex까지의 최단 경로와 거리를 구하는 명령어로 STL set을 이용하여 구현하며, Dijkstra 결과를 로그 파일(log.txt)에 저장한다. Dijkstra를 수행하고 최단 경로를 StartVertex부터 EndVertex까지 순서대로 최단 거리와 함께 저장한다. 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
DIJKSTRAMIN	사용법) DIJKSTRAMIN StartVertex EndVertex 예시) DIJKSTRAMIN 0 3 StartVertex를 기준으로 Dijkstra를 수행하여 EndVertex까지의 최단 경로와 거리를 구하는 명령어로 Min-Heap을 직접 구현하며, Dijkstra 결과를 로그 파일(log.txt)에 저장한다. Dijkstra를 수행하고 최단 경로를 StartVertex부터 EndVertex까지 순서대로 최단 거리와 함께 저장

	한다. 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
BELLMANFORD	사용법) BELLMANFORD StartVertex EndVertex 예시) BELLMANFORD 0 3 StartVertex를 기준으로 Bellman-Ford를 수행하여 EndVertex까지의 - 5 - 최단 경로와 거리를 구하는 명령어로, Bellman-Ford 결과를 로그 파일(log.txt)에 저장한다. 음수인 Weight가 있는 경우에도 동작해야 한다. 입력한 Vertex가 그래프에 존재하지 않거나, 입력한 Vertex가 부족한 경우, 또는 음수인 Weight를 가지는 사이클이 발생한 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
FLOYD	사용법) FLOYD 모든 도시의 쌍 (StartVertex, EndVertex)에 대해서 도시 StartVertex에서 EndVertex로 가는데 필요한 비용의 최솟값을 행렬 형태로 저장한다. FLOYD의 결과를 로그 파일(log.txt)에 저장한다.
CONFIG	사용법) CONFIG –-sort quick 사용할 정렬 알고리즘을 교체할 수 있는 명령이다. ‘—'를 통해 옵션을 변경 할 수 있으며 –sort의 경우 sort 알고리즘을 변경하는 옵션이다. 교체 할 수 있는 옵션은 quick, insert, merge, heap, bubble 이다. (자유롭게 추가 가능)

☐ **Requirements in implementation**

- 명령어와 함께 입력되는 Vertex는 숫자로 입력한다.
  - 명령어에 인자(Parameter)가 부족한 경우 오류 코드를 출력한다.
  - 예외처리에 대해 반드시 오류 코드를 출력한다.
- 출력은 ‘출력 포맷’을 반드시 따른다
- 모든 명령어는 커맨드 파일에 저장하여 순차적으로 읽고 처리한다.
  - 커맨드 파일 이름을 “command.txt”로 절대 고정하지 않는다.(반드시 Run 함수의 인자인 filepath를 사용)
- 로그 파일(log.txt)에 출력 결과를 반드시 저장한다.
  - 에러 발생시 에러 코드 및 에러 명 출력 필수.
  - 프로그램 실행 시 로그 파일이 이미 존재할 경우 이전 내용을 모드 지우고 새로

작성

- 로그 파일에 에러 결과를 반드시 저장한다.
  - 에러가 없는 경우 0(Success), 있는 경우 그에 맞는 에러 코드를 출력한다. (모든 명령어에 에러 코드가 출력되어야 한다.)
  - 프로그램 실행 시 로그 파일(log.txt)이 이미 존재할 경우 이전 내용을 모두 지우고 시작한다.
  -
- 프로그램이 종료 될 때 메모리 누수가 발생하지 않도록 한다.

☐ **Error Code**

명령어에 인자가 부족한 경우, 또는 각 상황마다 오류 코드를 출력해야하는 경우, 로그 파일(log.txt)에 알맞은 에러 코드 및 에러 명을 출력한다. 즉, 모든 명령어 동작에 에러 코드가 로그 파 일(log.txt)에 출력되어야 하며, 에러 명을 로그 파일(log.txt)에 출력해야 한다. 제공되는 Result.h를 반드시 사용하며, 표 3에 명시된 에러 코드 이외의 문제는 따로 고려하지 않는다. 이외의 내용은 Requirements in implementation을 참조한다. 각 명령어별 오류 코드는 다음과 같다.

명령어	에러 코드	내용
LOAD	101	에러 명: LoadFileNotExist - 섬의 거리 데이터 텍스트 파일이 존재하지 않을 경우
UPDATE	005	에러 명: FaildtoUpdatePath - 그래프의 업데이트에 실패 했을 때 출력되는 로그
PRINT BFS DIJKSTRA DIJKSTRAMIN BELLMANFORD	202	에러 명: GraphNotExist - 섬의 거리 정보 데이터가 존재하지 않을 경우

FLOYD		
BFS DIJKSTRA DIJKSTRAMIN BELLMANFORD	200	에러 명: VertexKeyNotExist - 명령어 인자(Vertex)가 부족한 경우
	201	에러 명: InvalidVertexKey - 인자로 입력한 Vertex가 섬의 거리 정보 데이터에 없는 경우
DIJKSTRA DIJKSTRAMIN	203	에러 명: InvalidAlgorithm - 섬의 거리 정보 데이터에 음수인 Weight가 존재할 경우
BELLMANFORD	204	에러 명: NegativeCycleDetected - 섬의 거리 정보 데이터에 Weight가 음수인 사이클이 존재할 경우
ETC	0	에러 명: Success - 표 2의 명령어 수행 시 에러가 발생하지 않은 경우 - 결과 출력이 없는 명령어일 경우, 위의 에러 명 출력(LOAD만 해당)

	100	에러 명: CommandFileNotExist - run 함수의 인자로 입력받은 커맨드 파일이 존재하지 않을 경우 - 결과 파일(result.log)에서 에러 명령어 자리에 "SYSTEM"을 사용 📁 커맨드 파일 이름을 "command.txt"로 절대 고정해서 사용하지 않음 - ex) ====== SYSTEM ====== CommandFileNotExist ======================
	300	에러 명: NonDefinedCommand - 존재하지 않는 명령어일 경우 📁 결과 파일(log.txt)에서 에러 명령어 자리에 오류가 발생한 명령어를 사용 📁 ex) ASTAR ====== ASTAR ====== NonDefinedCommand ====================
CONFIG	001	에러명 : InvalidAlgorithmName -존재하지 않는 Sorting 알고리즘 옵션을 주었을 경우

	003	에러명 : InvalidOptionName -존재하지 않는 옵션에 대한 설정을 하려고 했을 경우 예) CONFIG –-Repeat 10

□ Print Format

각 명령어 동작에 따라 로그 파일(log.txt)에 해당 내용을 지정된 형식에 맞추어 출력한다. 이외의 내용은 Requirements in implementation을 참조한다. 각 명령어별 출력 형식은 다음과 같다.

기능	출력 포맷
LOAD	====== LOAD ====== Success ========== ========== Error code: 0 ==========
UPDATE	====== UPDATE ====== Success ========== ========== Error code: 0

	==========
PRINT	====== PRINT ====== 0 6 13 0 0 0 0 5 6 0 2 0 0 7 4 0 6 0 0 3 0 0 5 2 0 ========== ========== Error code: 0 ==========
BFS	===== BFS ===== shortest path: 0 1 3 sorted nodes: 0 1 3 path length: 12 Course : Bread Chodenny's* Car shop Labuajiea's Yoga class ========== ========== Error code: 0

	=================
DIJKSTRA	===== DIJKSTRA ===== shortest path: 0 1 3 sorted nodes: 0 1 3 path length: 12 Course : Bread Chodenny's* Car shop Labuajiea's Yoga class ================ ================= Error code: 0 =================
DIJKSTRAMIN	====== DIJKSTRAMIN ====== shortest path: 0 1 3 sorted nodes: 0 1 3 path length: 12 Course : Bread Chodenny's* Car shop Labuajiea's Yoga class ================ =================

	Error code: 0 =================
BELLMANFORD	===== BELLMANFORD ===== shortest path: 1 2 4 sorted nodes: 1 2 4 path length: 9 Course : Bread Chodenny's* Car shop Labuajiea's Yoga class ================ ================= Error code: 0 =================
FLOYD	===== FLOYD ===== 0 6 11 12 15 7 0 5 6 9 2 8 0 6 4 10 6 8 0 3 7 8 5 2 0 ================ =================

	Error code: 0 ================
CONFIG	====== CONFIG LOG====== Sorted by : Merge Sorting ================ ================= Error code: 0 ================

□ 동작 예시

command.txt	mapdata.txt
LOAD mapdata.txt PRINT UPDATE PRINT DIJKSTRA 0 3 BFS 0 3 BELLMANFORD 1 4 DIJKSTRA -1 10	6 Denny's Bread / 0 6 13 0 0 1 Chodenny's Car shop / 0 0 5 6 0 5 Granddey's Go Kicking / 2 0 0 7 4 7 Labuajiea's Yoga class / 0 6 0 0 3 5 Grandy's Kicking Class / 0 0 5 2 0 9 Jeawon's Computer Academy / 5 6 8 4 5 0

BELLMANFORD
 	
ASTAR 1 4
 	
FLOYD
 	
//BFS
 	
//ASTAR 1 0	

실행 결과 화면	
log.txt	

```
===== LOAD =====

Success

=================


=================

Error code: 0

=================


===== PRINT =====

0 6 13 0 0

0 0 5 6 0

2 0 0 7 4

0 6 0 0 3

0 0 5 2 0

=================


=================

Error code: 0

=================

====== UPDATE ======

Success

====================


====================

Error code: 0
```

==========

===== BFS =====

shortest path: 0 1 3

sorted nodes: 0 1 3

path length: 12

Course : Bread Chodenny's* Car shop Labuajiea's Yoga class

==========

===== PRINT =====

0 3 5 0 0

0 0 3 3 0

1 0 0 3 2

0 3 0 0 2

0 0 5 1 0

==========

==========

Error code: 0

==========

==========

Error code: 0

==========

===== DIJKSTRA =====

shortest path: 0 1 3

sorted nodes: 0 1 3

path length: 12

Course : Bread Chodenny's* Car shop Labuajiea's Yoga class

==========

==========

Error code: 0

==========

===== BELLMANFORD =====

shortest path: 1 2 4

sorted nodes: 1 2 4

path length: 9

Course : Bread Chodenny's* Car shop Labuajiea's Yoga class

==========

==========

Error code: 0

==========

===== DIJKSTRA =====

InvalidVertexKey

==========

==========

Error code: 201

==========

```
===== BELLMANFORD =====
VertexKeyNotExist
=================


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


===== ASTAR =====
NonDefinedCommand
=================


=================
Error code: 300
=================


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


=================
```

Error code: 0

==========

□ **구현 시 반드시 정의해야하는 Class의 멤버 변수**

1. Manager: 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행.

항목	내용	비고
const char* RESULT_LOG_PATH	결과 파일 이름(log.txt)	값 수정 금지
std::ofstream fout	결과 파일의 FileStream	
Graph m_graph	그래프 클래스	

2. Graph

항목	내용	비고
Vertex* m_pVHead	Vertex Linked-List의 Head 포인터	
int m_vSize	Vertex의 수	

3. Edge

항목	내용	비고
int m_key	Edge(End Vertex)의 Key	
int m_weight	Edge의 Weight	
Edge* m_pNext	다음 Edge를 가리키는 포인터	

4. Vertex

항목	내용	비고
int m_key	Vertex(Start Vertex)의 Key	
int m_size	연결된 Edge의 수	
Edge* m_pEHead	Edge Linked-List의 Head 포인터	
Vertex* m_pNext	다음 Vertex를 가리키는 포인터	

☐ Files

✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일

✓ mapdata.txt: 도시 상점 정보 데이터를 저장하고 있는 파일

✓ log.txt: 모든 출력 결과를 저장하고 있는 파일 (파일 이름 변경 금지)

□ 제한사항 및 구현 시 유의사항

✓ 반드시 제공되는 압축파일(project3.tar.gz)을 이용하여 구현하며, 작성된 소스 코드의 이름과 클래스, 함수 이름 및 형태를 임의로 변경하지 않는다.

✓ 제공된 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.

- 제공된 코드의 변수 및 함수 프로토타입은 변경 불가능

    - 함수 및 변수 추가가 가능하되, 제공된 코드의 함수로 테스트하여 채점하기 때문에 제공된 코드의 함수 기능이 제대로 구현되어야 한다.

✓ 메인 함수(main.cpp)는 절대 변경하지 않는다.

✓ 제시된 클래스를 각 기능에 알맞게 모두 사용한다.

✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.

✓ 채점 시 코드를 수정해야하는 일이 없도록 한다.

✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점)

✓ 프로그램은 반드시 리눅스에서 동작해야한다. (컴파일 에러 발생 시 감점)

- 제공되는 Makefile을 사용하여 테스트하도록 한다.

- g++ 컴파일러의 버전은 4.8.4로 진행한다. (버전 확인 명령어: ~$ g++ --version)

□ 제출기한 및 제출방법

✓ 제출기한

-

제출방법

- 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 U-Campus에 제출

• 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음

• 보고서 파일 확장자가 pdf가 아닐 시 감점

• 제출형식

- 2000722000_DS_project3.tar.gz

• 압축방법

- 압축하기

```
ex) /home/user/pr1 에 있는 모든 파일을 2000722000_DS_project3.tar.gz로 압축
$ tar –zcvf 2000722000_DS_project3.tar.gz /home/user/pr1/*
```

- 압축풀기

```
ex) project3.tar.gz의 압축을 해제
$ tar –zxvf project3.tar.gz
```

• 보고서 작성 형식

- 하드카피는 제출하지 않음
- 보고서는 한글로 작성
- 보고서에는 소스코드를 포함하지 않음
- 반드시 포함해야하는 항목

★ Introduction : 프로젝트 내용에 대한 설명

★ Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명 (모든 명령어 및 정렬 알고리즘에 대하여 각각 그릴 것)

★ Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명 (BFS, Dijkstra, Bellman-Ford, FLOYD, 라빈-카프 알고리즘에 대하여 각각 예시를 들어 설명할 것) 이후 각 정렬 알고리즘에 대한 성능 비교 및 성능 차이가 생긴 사유에 대한 description 진행,

★ Result Screen : 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명(각 명령어 작동 화면에

대한 최소 10줄 이상의 description 필요)

★ Consideration : 고찰 작성 (사진 제외 본문 내용으로 1장 이상)