Data Structure Lab. Project #3

본 프로젝트에서는 그래프를 통한 최단 경로 탐색 프로그램과 라빈 카프 알고리즘을 이용한 문자열 비교 프로그램을 구현한다. 이 프로그램은 각 노드 간의 최단 경로를 찾게 된다. 이 프로그램은 도로에 대한 정보를 가지고 있는 텍스트 파일(Matrix)로부터 장소의 이름과 거리의 weight를 읽어와 Linked list (Adjacency List)로 저장한다. 그래프를 생성한 이후 최소 비용 경로를 BFS, 다익스트라, 벨만포드, 플로이드 알고리즘을 통해 탐색한다. 만약 Weight가 음수일 경우 다익스트라는 에러를 출력하며, 벨만포드에서는 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. 이 프로그램은 target company의 report 제목에 대한 정보를 가지고 있는 reportdata.txt 파일로부터 읽어서 report 제목들을 list로 저장한다. 이후 사용자가 입력한 문자열에 대해서 라빈카프 알고리즘을 통한 리스트의 Report 제목들과 문자열 비교를 하여 같은 문자열이 포함 되어있는지 여부와 포함 되어있다면, 포함된 제목을 출력하여 준다.

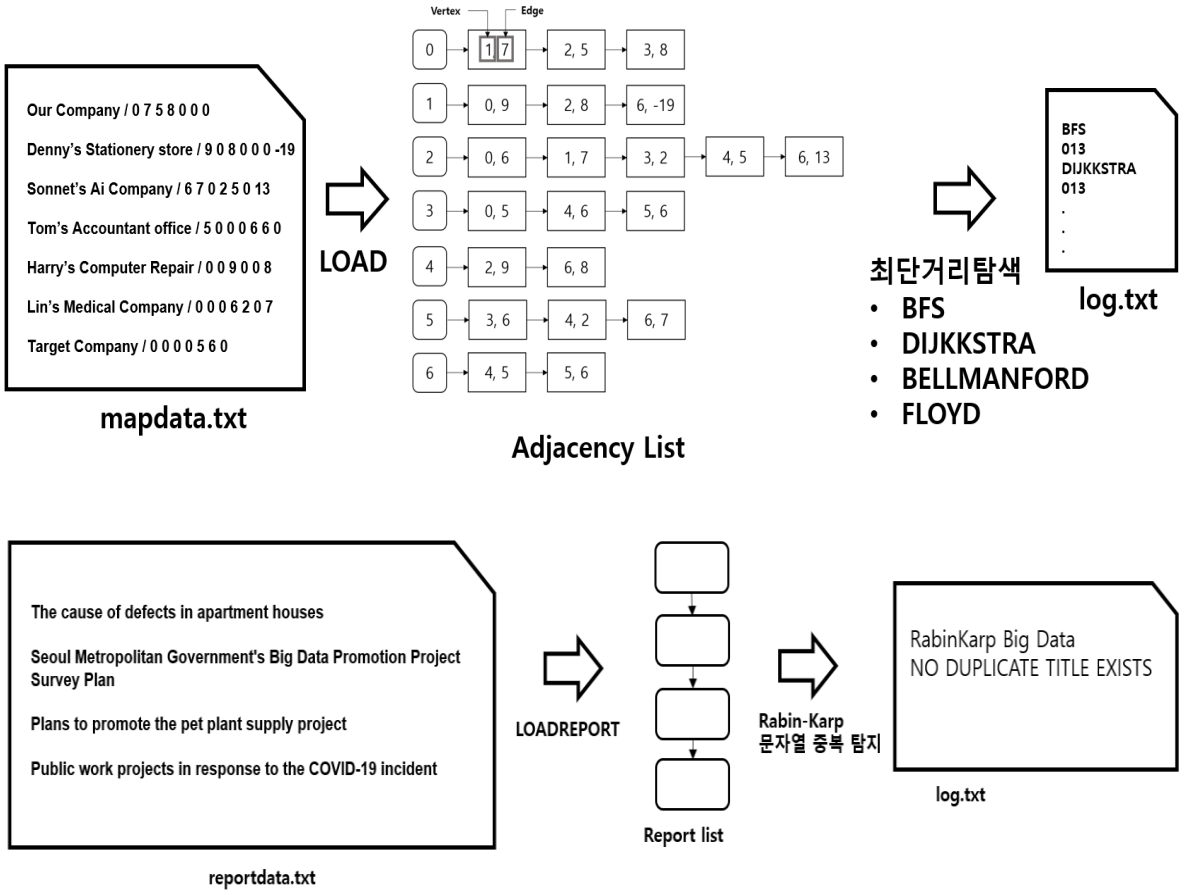


그림 1. 프로젝트 최단경로 찾기 프로그램 / 중복 보고서 탐지 프로그램

□ **Program implementation**

1. 거리 정보 데이터

프로그램은 방향성과 가중치를 가지고 있는 그래프 정보를 형태로 거래처의 거리 데이터를 저장한 Matrix 텍스트 파일(mapdata.txt)을 명령어를 통해 읽어 해당 정보를 클래스에 저장한다. 가게 데이터 텍스트 파일의 첫번째 줄에는 자신의 회사와 weight 정보가 저장되어 있고 다음 줄부터 거리 데이터가 저장되어 있다. 거리 데이터의 행과 열은 각각 Edge의 Start Vertex와 End Vertex의 Weight를 의미한다. 이중 첫번째 열은 '거리의 이름' 이다. 이후 텍스트 파일의 마지막 줄에는 Target Company, 즉 도착해야 하는 회사의 weight가 저장 되어있다.

줄 수	내용
1	Our Company / [weight 정보]
2...n-1	[거래처이름]/[weight_1] [weight_2] ... [weight_n]
n	Target Company / [weight 정보]

표 1. 도로 정보 데이터 형식

```
Our Company / 0 7 5 8 0 0 0
Denny's Stationery store / 9 0 8 0 0 0 -19
Sonnet's Ai Company / 6 7 0 2 5 0 13
Tom's Accountant office / 5 0 0 0 6 6 0
Harry's Computer Repair / 0 0 9 0 0 8
Lin's Medical Company / 0 0 0 6 2 0 7
Target Company / 0 0 0 0 5 6 0
```

그림 2. 거리 정보 데이터가 저장되어 있는 텍스트 파일의 예 (mapdata.txt)

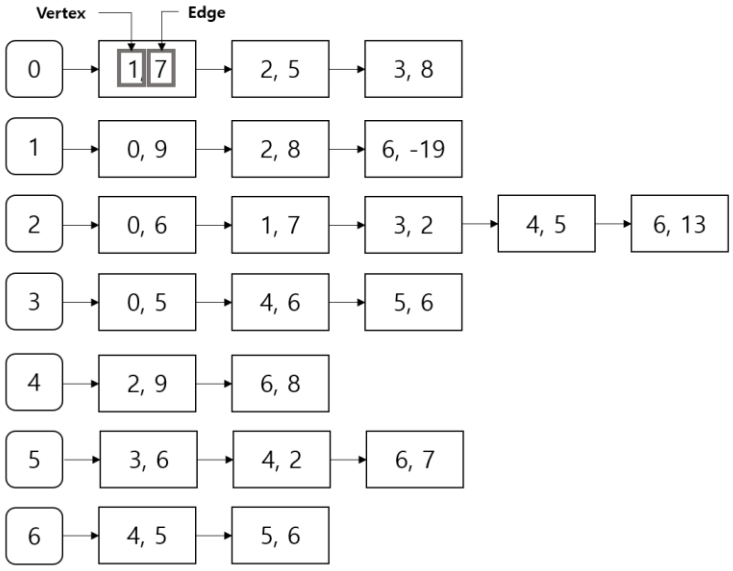
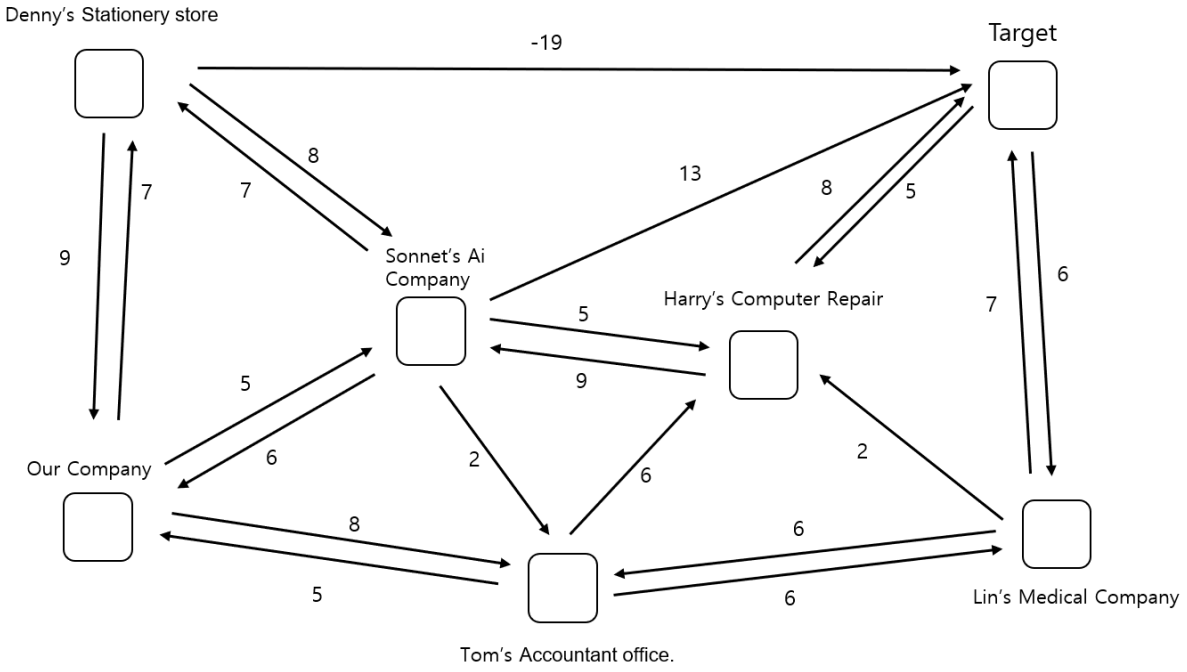


그림 3. Linked-List 연결 예시

Vertex는 정점이고 이 정점들을 잇는 선은 edge라고 부른다. Vertex와 Vertex 간의 Edge를 하나의 노드로 만들어 Linked-List를 이용하여 연결하며, text 파일을 순차적으로 읽어 Linked-List를 만든다. (순차적으로 연결하여 반드시 Vertex 오름차순으로 정렬된 형태로 저장되어 있어야 한다).

2. 그래프 연산

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



- BFS

BFS 명령어는 Target 회사와의 경로와 거리를 구한다. 큐를 활용하여 구현한다. 큐에 넣는 순서는 회사들의 Index 순서대로 들어가야한다. 음수 사이클이 발생하는 경우, 에러를 출력하고, Weight가 음수인 간선은 제거하고 최단경로를 구한다.

- DIJKSTRA

DIJKSTRA 명령어는 Target 회사와의 최단 경로와 거리를 구한다. Dijkstra 연산을 위해 STL 활용(vector, list 등)하여 최단 거리와 경로를 구하고, 결과로 찾은 최단 경로를 다시 순회하여 자신의 회사부터 Target 회사 순으로 경로를 구한다. 음수사이클이 발생하는 경우, 에러를 출력하고, Weight가 음수인 간선은 제거하고 최단경로를 구한다.

- BELLMANFORD

BELLMANFORD 명령어는 Target 회사와의 최단 경로와 거리를 구한 다. 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 구하고, 거기서 자신의회사와 Target 회사와의 최단거리를 출력해준다. Weight가 음수 인 경우에도 정상 동작하며, 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력한다.

3. 문자열 비교하기

Target 회사에서 요구한 주제를 우리 회사 보고서 중 같은 주제의 보고서가 있는지 찾기 위해 제목의 포함여부를 통해 찾아야한다. 그러기 위해서 우리 회사 보고서 text 파일을 입력 받고 Target 회사의 요구한 주제를 매개변수로 넣어 제목에 포함된 보고서가 있는지 확인한다. 포함여부를 출력하고, 포함된 문자열이 있다면 해당 문자열을 모두 출력하여 준다. (매개변수는 공백을 포함한 10글자이내로 제한한다. 라빈카프 알고리즘을 사용하지 않고 구현할 시, 60% 감점)

The cause of defects in apartment houses.

Seoul Metropolitan Government's Big Data Promotion Project Survey Plan

Plans to promote the pet plant survey plan

Public work projects in response to the COVID-19 incident.

그림 4. 보고서 제목 데이터가 저장되어 있는 텍스트 파일의 예 (reportdata.txt)

사용자에게서 단어를 입력 받아 라빈카프 알고리즘을 이용하여 기존의 보고서 제목 데이터에서 해당 단어가 포함된 제목이 있는지 확인한다. 대소문자는 구분하지 않고, 공백은 포함한다.

라빈카프 알고리즘은 해싱(Hashing)을 사용하여 문자열에서 특정 패턴과 일치하는지 찾아주는 알고리즘이다. 기본적으로 비교할 문자열과 패턴을 Hash function을 통해 해시값으로 변환하고, 해시값의 비교를 통해 문자열에 패턴이 포함 되어있는지 확인한다. 해시값을 만들려면, 각 문자(ASCII 값)에 특정 수의 제곱 수를 차례대로 곱하여 모두 더하면 된다.

즉, 해시값이 일치하면 문자열이 같다라고 판단한다. 이는 해시 충돌이 없는 경우라고 가정한다.

Ex) 문자열 AABD에서 패턴 ABD이 있는지 라빈카프 알고리즘을 통해 탐색한다면

AAB의 해시값과 ABD의 해시값과 비교하여 일치여부를 판단한 뒤, 일치하지 않는 경우, 한 칸 씩 슬라이딩하여 ABD와 ABD의 해시값을 비교하게 되어 해당 패턴이 문자열에 있는지 파악할 수 있게 된다.

☐ Program implementation

프로그램은 명령어를 통해 동작하며, 실행할 명령어가 작성 되어있는 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 데이터가 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.
LOADREPORT	사용법) LOAD textfile 보고서 제목 데이터를 LOAD하는 명령어로 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) BFS 예시) BFS Our Company를 기준으로 BFS를 수행하여 target Company까지의 경로와 거리를 구하는 명령어로, BFS결과를 로그 파일(log.txt)에 저장한다. DFS를 수행하고 경로를 Our Company부터 target Company까지 순서대로 거리와 함께 저장한다. 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장하고, 음수사이클을 제외한 가장 빠른 경로를 저장한다.
DIJKKSTRA	사용법) DIJKSTRA 예시) DIJKSTRA

	Our Company를 기준으로 Dijkstra를 수행하여 target Company까지의 최단 경로와 거리를 구하는 명령어로 STL set을 이용하여 구현하며, Dijkstra 결과를 로그 파일(log.txt)에 저장한다. Dijkstra를 수행하고 최단 경로를 Our Company부터 target Company까지 순서대로 최단 거리와 함께 저장한다. 음수인 Weight가 있는 경우 로그 파일(log.txt)에 알맞은 오류 코드를 저장한다.
BELLMANFORD	사용법) BELLMANFORD 예시) BELLMANFORD Our Company 를 기준으로 Bellman-Ford를 수행하여 target Company 까지의 최단 경로와 거리를 구하는 명령어로, Bellman-Ford 결과를 로그 파일(log.txt)에 저장한다. 음수인 Weight가 있는 경우에도 동작해야 한다.
FLOYD	사용법) FLOYD 모든 도시의 쌍 (StartVertex, EndVertex)에 대해서 도시 StartVertex에서 EndVertex로 가는데 필요한 비용의 최솟값을 행렬 형태로 저장한다. FLOYD의 전체 matrix 결과를 로그 파일(log.txt)에 저장한다.
RABINKARP	사용법) RABINKARP text 인자로 받은 text를 보고서 제목 데이터의 제목들과 비교하여 같은 제목이 있는지 체크한다. 오류가 발생 시, log.txt에 오류 코드를 저장한다.

☐ **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.

☐ **Error Code**

명령어에 인자가 부족한 경우 등, 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 파일(log.txt)에 출력되어야 하며, 에러 명을 로그 파일(log.txt)에 출력해야 한다. 제공되는 Result를 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 - 거래처의 거리 데이터 텍스트 파일이 존재하지 않을 경우
LOADREPORT	005	에러 명: FaildtoUpdatePath - 보고서 제목 데이터 텍스트 파일이 존재하지 않는 경우
PRINT BFS DIJKSTRA BELLMANFORD FLOYD	202	에러 명: GraphNotExist - 거래처의 거리 정보 데이터가 존재하지 않을 경우

BFS DIJKSTRA BELLMANFORD	200	에러 명: VertexKeyNotExist - 명령어 인자(Vertex)가 잘못 들어간 경우
	201	에러 명: InvalidVertexKey - 인자로 입력한 Vertex가 거래처의 거리 정보 데이터에 없는 경우
BFS DIJKSTRA	203	에러 명: InvalidAlgorithm - 거래처의 거리 정보 데이터에 음수인 Weight가 존재할 경우
BELLMANFORD	204	에러 명: NegativeCycleDetected - 거래처의 거리 정보 데이터에 Weight가 음수인 사이클이 존재할 경우
ETC	0	에러 명: Success - 결과 출력이 없는 명령어의 경우, Success와 에러코드 0 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 ====================
RABINKARP	001	에러명 : InvalidOptionNumber - 너무 많은 인자가 들어온 경우 예) RABINKARP asian healthcare12345

□ 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 ==========
LOADREPORT	====== LOADREPORT ====== Success ========== ========== Error code: 0 ==========
PRINT	====== PRINT ====== 0 7 5 8 0 0 0 9 8 0 0 0 0 19 6 7 0 2 5 0 13 5 0 0 0 6 6 0 0 0 9 0 0 8 0 0 0 6 2 0 7 0 0 0 0 5 6 0 ==========

BFS	===== BFS ===== shortest path: 0 2 6 path length: 18 Course : OurCompany Sonnet'sAiCompany TargetCompany ==========
DIJKSTRA	===== DIJKSTRA ===== shortest path: 0 2 6 path length: 18 Course : OurCompany Sonnet'sAiCompany TargetCompany ==========
BELLMANFORD	===== BELLMANFORD ===== shortest path: 0 2 6 path length: 18 Course : OurCompany Sonnet'sAiCompany TargetCompany ==========

FLOYD	===== FLOYD ===== 0 7 5 8 10 14 18 9 0 8 10 13 16 21 6 7 0 2 5 8 13 5 12 10 0 6 6 13 15 16 9 11 0 14 8 11 18 11 6 2 0 7 20 21 14 16 5 6 0 ================
RABINKARP	----------존재하지 않을 경우---------- ======RABINKARP====== NO DUPLICATE TITLE EXISTS ================ ----------존재할 경우---------- ======RABINKARP====== DUPLICATE TITLE EXISTS Seoul Metropolitan Government's Big Data Promotion Project Survey Plan Plans to promote the pet plant survey plan ================

□ **동작 예시**

<table>
<tr><th>command.txt</th><th>mapdata.txt</th></tr>
<tr><td rowspan="2">LOAD mapdata.txt
PRINT
LOADREPORT reportdata.txt
BFS
BFS 1 3
DIJKSTRA
BELLMANFORD
FLOYD
RABINKARP surveyPlan
RABINKARP healthcare</td><td>Our Company / 0 7 5 8 0 0 0
Denny’s Stationery store / 9 0 8 0 0 0 -19
Sonnet’s Ai Company / 6 7 0 2 5 0 13
Tom’s Accountant office / 5 0 0 0 6 6 0
Harry’s Computer Repair / 0 0 9 0 0 8
Lin’s Medical Company / 0 0 0 6 2 0 7
Target Company / 0 0 0 0 5 6 0</td></tr>
<tr><td>reportdata.txt</td></tr>
<tr><td></td><td>The cause of defects in apartment houses
Seoul Metropolitan Government's Big Data Promotion Project Survey Plan
Plans to promote the pet plant survey plan
Public work projects in response to the COVID-19 incident</td></tr>
<tr><td colspan="2">실행 결과 화면</td></tr>
<tr><td colspan="2">log.txt</td></tr>
</table>

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

Success

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


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

Error code: 0

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


===== PRINT =====

0 7 5 8 0 0 0

9 0 8 0 0 0 -19

6 7 0 2 5 0 13

5 0 0 0 6 6 0

0 0 9 0 0 8

0 0 0 6 2 0 7

0 0 0 0 5 6 0

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


====== LOADREPORT ======

Success

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


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

Error code: 0

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

```
===== InvalidAlgorithm=====
shortest path: 0 2 6
sorted nodes: 0 2 6
path length: 18
Course : OurCompany Sonnet'sAiCompany TargetCompany
================
=================
Error code: 203
=================

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

===== InvalidAlgorithm=====
shortest path: 0 1 3
shortest path: 0 2 6
sorted nodes: 0 2 6
path length: 18
Course : OurCompany Sonnet'sAiCompany TargetCompany
================

===== BELLMANFORD =====
shortest path: 0 2 6
```

sorted nodes: 0 2 6

path length: 18

Course : OurCompany Sonnet'sAiCompany TargetCompany

==========

===== FLOYD =====

0 7 5 8 10 14 18

9 0 8 10 13 16 21

6 7 0 2 5 8 13

5 12 10 0 6 6 13

15 16 9 11 0 14 8

11 18 11 6 2 0 7

20 21 14 16 5 6 0

==========

======RABINKARP======

Seoul Metropolitan Government's Big Data Promotion Project Survey Plan

Plans to promote the pet plant survey plan.

==========

======RABINKARP======

NO DUPLICATE TITLE EXISTS

==========

☐ Files

✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일

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

✓ reportdata.txt : 보고서 제목 데이터를 저장하고 있는 파일

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

□ 채점 기준

채점 기준	인자	점수
LOAD, LOADREPORT	textfile	1
PRINT		1
BFS		2
DIJKKSTRA		3
BELLMANFORD		3
FLOYD		3
RABINKARP	text	2
총합		**15**

□ 제한사항 및 구현 시 유의사항

✓ 반드시 제공되는 코드(github 주소 참고)를 이용하여 구현하며, 작성된 소스 코드의 이름과 클래스, 함수 이름 및 형태를 임의로 변경하지 않는다.

✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.

✓ 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 리눅스(Ubuntu 18.04)에서 동작해야한다. (컴파일 에러 발생 시 감점)

- 제공되는 Makefile 을 사용하여 테스트하도록 한다.

□ 제출기한 및 제출방법

✓ 제출기한

- 2021년 12월 10일 23:59:58 까지 제출

✓ 제출 방법

- 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 제출
    - ■ 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음(.txt 파일 제출 X)
    - ■ 보고서 파일 확장자가 pdf 가 아닐 시 감점
- KLAS -> 과제 제출 -> tar.gz 로 과제 제출

✓ 제출 형식

- 학번_DS_project3.tar.gz (ex. 2020202001_DS_project3.tar.gz)

✓ 보고서 작성 형식

- ■ 보고서 내용은 한글로 작성
- ■ 보고서에는 소스코드를 포함하지 않음
- ■ 아래 각 항목을 모두 포함하여 작성

- Introduction : 프로젝트 내용에 대한 설명

- Flowchart : 설계한 프로젝트의 플로우 차트를 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 최단거리 탐지 알고리즘에 대한 성능 비교 및 성능 차이가 생긴 사유에 대한 description 진행,

- Result : 모든 명령어에 대해 결과 화면을 캡처하고 동작을 설명

- Consideration : 고찰 작성