데이터 구조실습 3차 보고서

제출일자: 2021년 12월 03일 (금)

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 목요일 3, 4

학 번: 2018202028

성 명: 유해찬

1. Introduction

이번 프로젝트는 최단 경로를 찾는 알고리즘과 라빈카프를 이용해서 중복된 문자열을 찾는 알고리즘을 수행하는 프로젝트입니다.

첫 번째로는 최단 경로 알고리즘부터 소개하겠습니다.

최단 경로 알고리즘으로는 BFS, Dijkstra, BellmanFord, Floyd 총 4가지의 알고리즘을 다루는 것을 목표로 합니다. 또한, 이를 이용할 때 사용하는 Graph는 텍스트 파일로 부터 읽어와 Adjacency List형식으로 Graph를 저장하게 됩니다. 이러한 것들을 LOAD라는 메서드를 이용해서 읽어옵니다.

이제 각각의 최단 경로를 찾는 알고리즘의 특징은 다음과 같습니다. 특징을 설명하기 전에 이들의 Graph는 항상 Target 회사와의 경로와 거리를 구하는 것을 목표로합니다.

(1) BFS

첫 번째로는 BFS입니다. 일단 BFS는 음수 사이클이 발생할 경우에는 Error를 발생합니다. 또한, 음수인 간선이 Graph에 존재할 수 있는데 이런 경우에는 음수인 간선을 제거하고 최단 경로를 구하게 됩니다.

(2) DIJKSTRA

다익스트라 알고리즘은 Set이라는 명령어를 사용하여 구현을 완료해야합니다. 이를 구현하기 위해서는 만약 음수사이클이 발생하는 Graph는 최단 경로를 구할 수 없기 때문에 에러를 출력합니다. 또한, 음수인 간선에 대해서는 제거하고 최단 경로를 구하게 됩니다.

(3) BELLMANFORD

벨만포드 알고리즘은 다익스트라와 비슷한 알고리즘입니다. 하지만 다른점이 있는데 이는 음수 간선의 경우에도 정상 작동하게 됩니다. 또한, 음수 Weight에 사이클이 발생할 경우에는 에러를 출력하게 됩니다.

(4) FLOYD

플로이드 알고리즘은 모든 쌍에 대해서 최단 경로 행렬이 결과로 나옵니다. 또한, Weight가 음수인 경우에도 작동을 합니다. 하지만 음수 Weight에 사이클이 발생한 경우에는 알맞은 에러를 출력하도록 합니다.

또한, 라빈카프 알고리즘도 이번 프로젝트에 존재합니다. 회사의 보고서 중 같은 주제의 보고서가 있는지 찾기 위해서 제목의 포함여부를 통해서 찾아 내는 알고리즘 입니다. 만약에 포함된 문자열이 있는 문장이 있다면 그 문장 전체를 답으로 불러오게 됩니다. 또한, 라빈카프 알고리즘은 대소문자는 구분하지 않고, 공백 또한 아스키 문자로 값이 나오기 때문에 공백도 포함하여 구할 수 있게 합니다. 이러한 비교문제는 최대 10글자이며 값을 있는지 없는지 계속해서 Hash값을 찾아서 비교를 합니다.

첫 번째로 라빈카프도 textfile을 읽어와야합니다. 이를 읽어오기 위해서는 LOADREPORT

라는 메서드를 통해서 값을 읽어올 수 있도록 합니다. 만약 읽을 수 없다면 에러를 출력하게 됩니다. 읽어왔다면 RABINKARP 알고리즘을 통해서 똑같은 주제가 있는지 찾을 수 있습니다. 계속해서 해쉬 값을 포함하여 가져올 수 있습니다.

마지막으로는 에러 처리입니다. 이번 프로젝트에는 다양한 에러들이 있습니다. 최단 경로 찾는 문제만 하더라도 음의 간선이 안되거나 음의 사이클이 안되는 알고리즘이 많습니다. 이러한 경우에 모두 에러를 처리해 주는 것을 목표로 합니다. 에러들은 다음과 같이 존재합니다.

(1) LoadFileNotExist

에러코드는 101로 LOAD명령어를 수행했을 경우 로드하는 텍스트 파일이 없을 경우 에러코드를 출력한다.

(2) FaildtoUpdatePath

에러코드는 005로 LOADREPORT를 사용해서 보고서를 읽을 때 그 텍스트 파일이 존재하지 않는 경우는 005를 출력한다.

(3) GraphNotExist

에러코드는 202이고 Graph를 사용하는 명령어인 PRINT, BFS, DIJKSTRA, BELLMANFORD, FLOYD 명령어를 사용했을 때 거래처의 거리 정보 데이터가 존재하지 않으면 이 오류 코드를 출력한다.

(4) InvalidVertexKey

명령어의 인자가 잘못 들어간 경우인데 경로 탐색을 하는 알고리즘은 각각 인자를 받지 않기 때문에 사용이 되는지는 잘 모르겠다.

(5) VertexKeyNotExist

에러코드는 201로 Graph에서 Target회사로 가려고 할 때 막혀 있어서 갈 수가 없을 때 이 오류를 출력하게 된다.

(6) InvalidAlgorithm

에러코드는 203으로 음수간선이 들어 있으면 안되는 BFS와 DIJSTRA에서 그래프에 음수간선이 있을 경우에 출력하게 된다.

(7) NegativeCycleDetected

에러코드는 204이고 음수인 사이클이 그래프에 존재할 경우 출력한다.

(8) Success

결과 출력의 에러코드가 0인 경우 출력한다. 이는 LOAD만 출력한다

(9) CommandFileNotExist

오류코드는 100이고 Command 파일이 존재하지 않을 경우는 SYSTEM을 사용하여 명령어를 출력한다.

(10) NonDefinedCommand

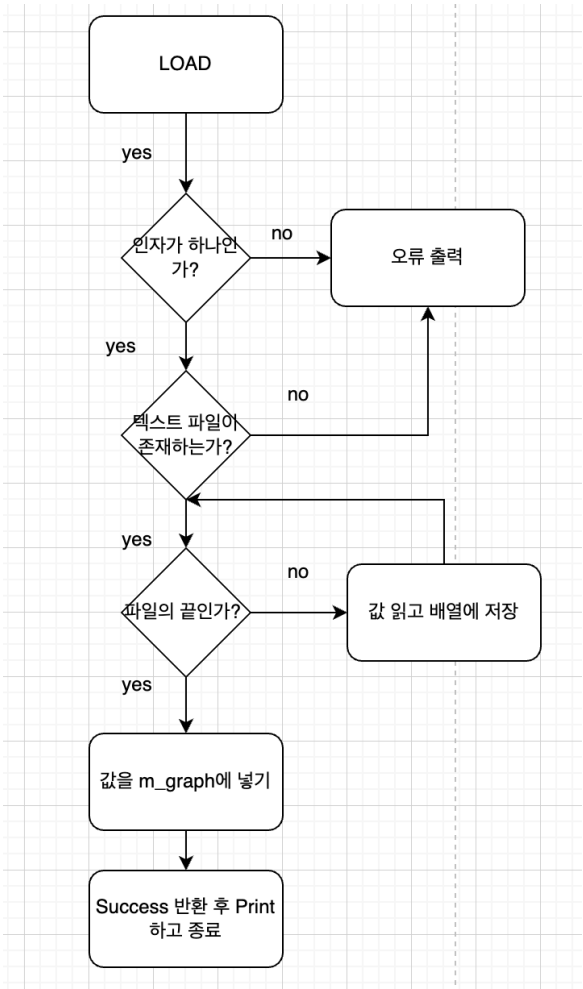
존재하지 않는 명령어인 경우에 출력하게 된다. 오류코드는 300이다.

(11) InvalidOptionNumber

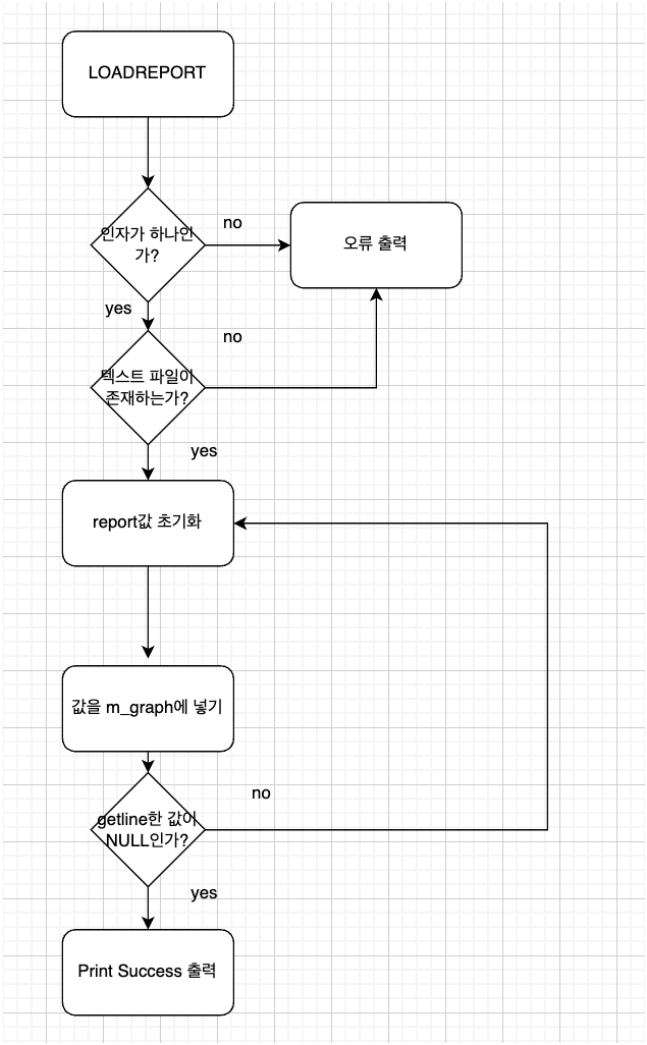
라빈카프 알고리즘에서 인자의 길이가 10이 넘어갈 경우에 출력을 하게 된다. 오류코드는 100이다.

2. Flowchart

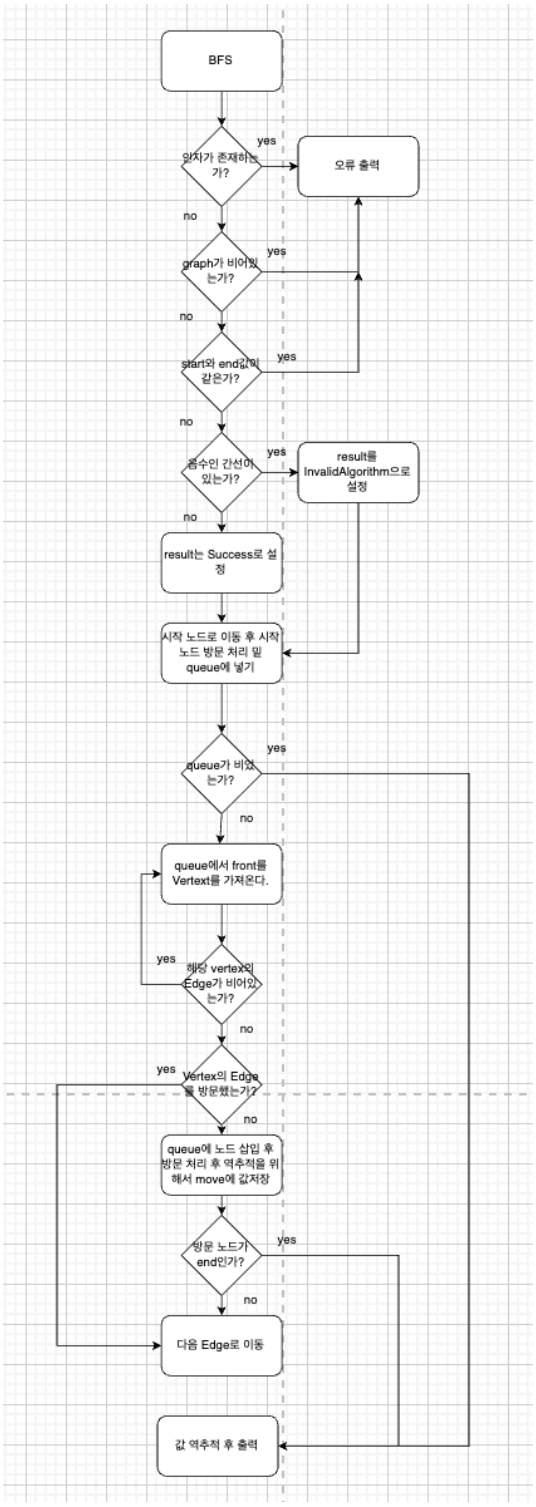
1. LOAD



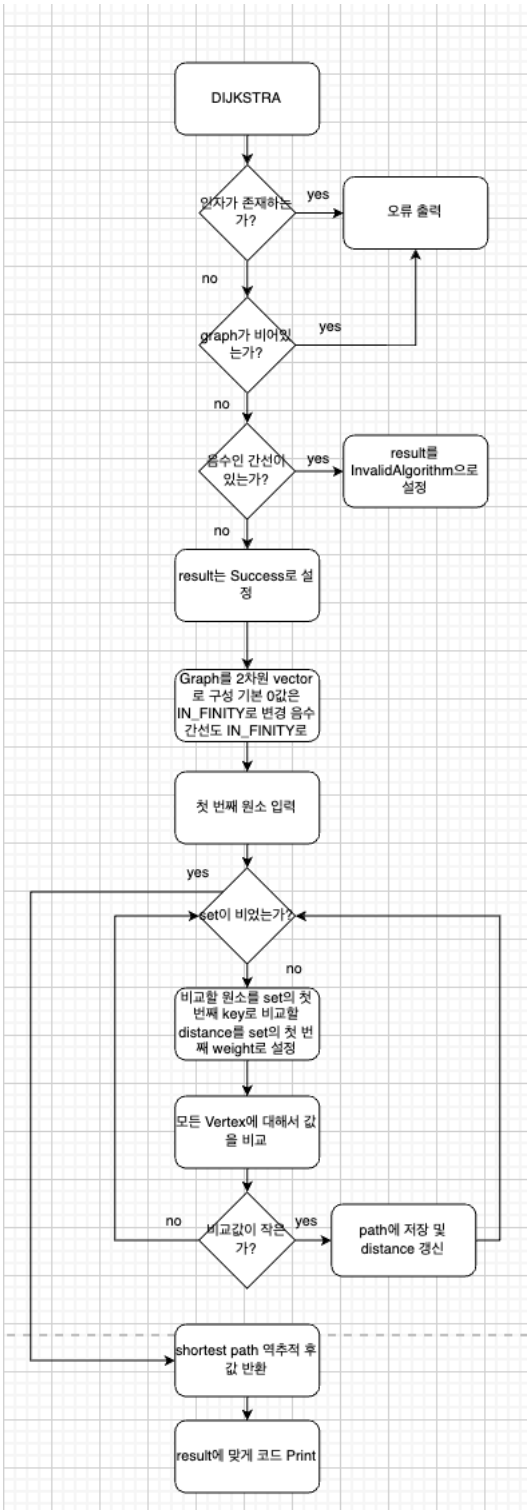
2. LOADREPORT


LOADREPORT
인자가 하나인가?
no
오류 출력
yes
no
텍스트 파일이 존재하는가?
yes
report값 초기화
값을 m_graph에 넣기
no
getline한 값이 NULL인가?
yes
Print Success 출력

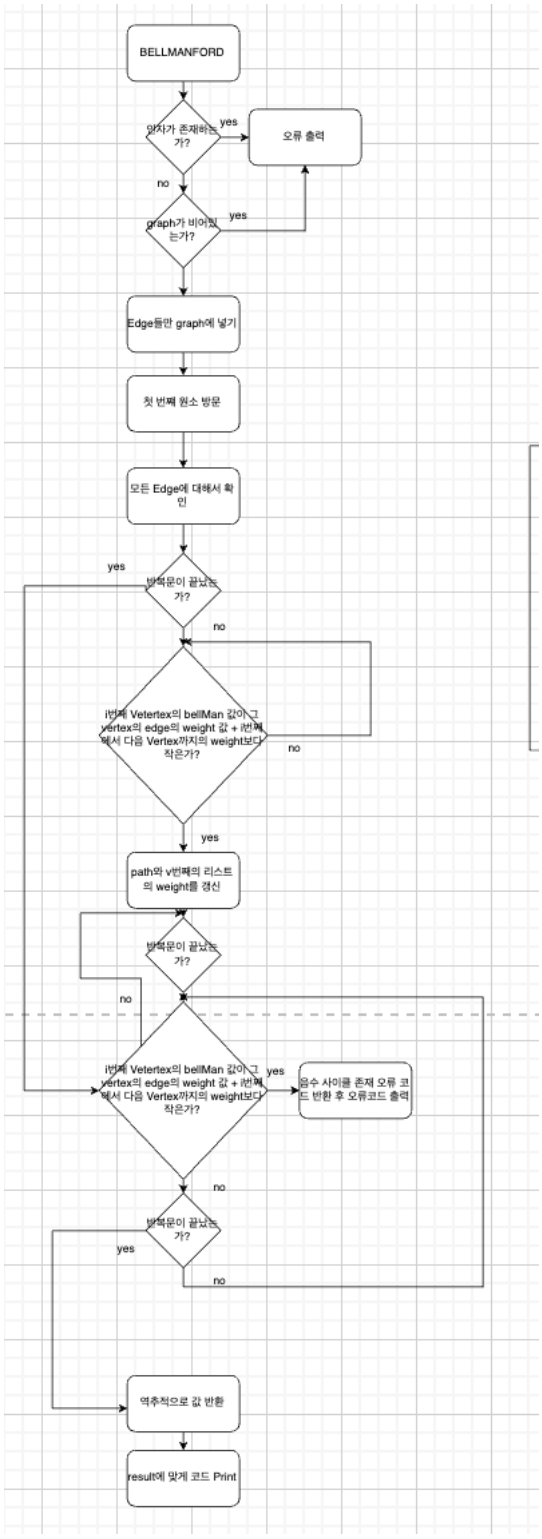
3. BFS


BFS
yes
인자가 존재하는가?
오류 출력
no
yes
graph가 비어있는가?
no
yes
start와 end값이 같은가?
no
yes
음수인 간선이 있는가?
result를 InvalidAlgorithm으로 설정
no
result는 Success로 설정
시작 노드로 이동 후 시작 노드 방문 처리 및 queue에 넣기
yes
queue가 비었는가?
no
queue에서 front를 Vertext를 가져온다.
yes
해당 vertex의 Edge가 비어있는가?
no
yes
Vertex의 Edge를 방문했는가?
no
queue에 노드 삽입 후 방문 처리 후 역추적을 위해서 move에 값저장
yes
방문 노드가 end인가?
no
다음 Edge로 이동
값 역추적 후 출력

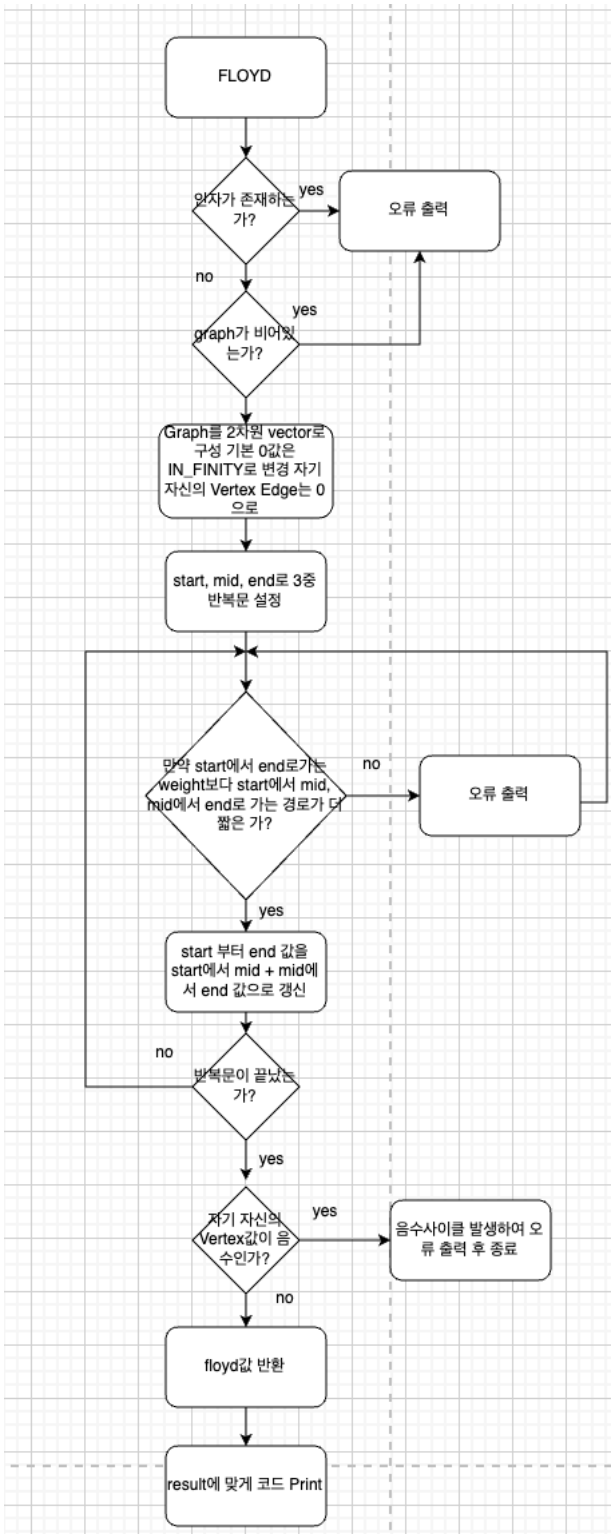
4. DIJKSTRA



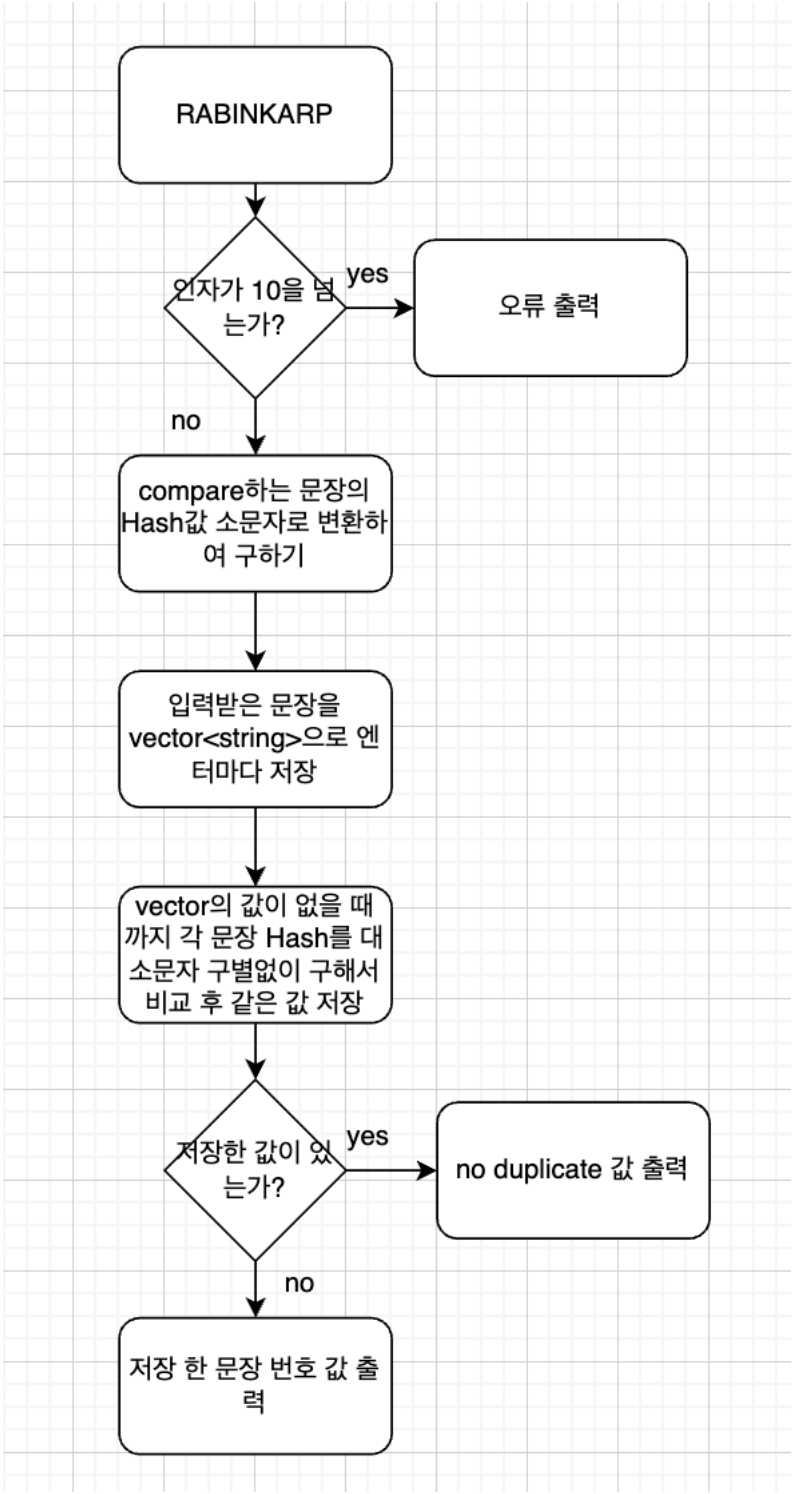
5. BELLMANFORD


BELLMANFORD
음자가 존재하는가?
yes
오류 출력
no
graph가 비어있는가?
yes
Edge들만 graph에 넣기
첫 번째 원소 방문
모든 Edge에 대해서 확인
yes
반복문이 끝났는가?
no
i번째 Vetertex의 bellMan 값이 그 vertex의 edge의 weight 값 + i번째 에서 다음 Vertex까지의 weight보다 작은가?
no
yes
path와 v번째의 리스트의 weight를 갱신
반복문이 끝났는가?
no
i번째 Vetertex의 bellMan 값이 그 vertex의 edge의 weight 값 + i번째 에서 다음 Vertex까지의 weight보다 작은가?
yes
음수 사이클 존재 오류 코드 반환 후 오류코드 출력
no
반복문이 끝났는가?
yes
no
역추적으로 값 반환
result에 맞게 코드 Print

6. FLOYD


FLOYD
인자가 존재하는가?
yes
오류 출력
no
graph가 비어있는가?
yes
Graph를 2차원 vector로 구성 기본 0값은 IN_FINITY로 변경 자기 자신의 Vertex Edge는 0으로
start, mid, end로 3중 반복문 설정
만약 start에서 end로가는 weight보다 start에서 mid, mid에서 end로 가는 경로가 더 짧은 가?
no
오류 출력
yes
start 부터 end 값을 start에서 mid + mid에서 end 값으로 갱신
no
반복문이 끝났는가?
yes
자기 자신의 Vertex값이 음수인가?
yes
음수사이클 발생하여 오류 출력 후 종료
no
floyd값 반환
result에 맞게 코드 Print

7. RABINKARP



3. Algorithm

- BFS

  우선 방문을 확인할 수 있는 Vertex의 개수의 크기의 배열을 생성한다.

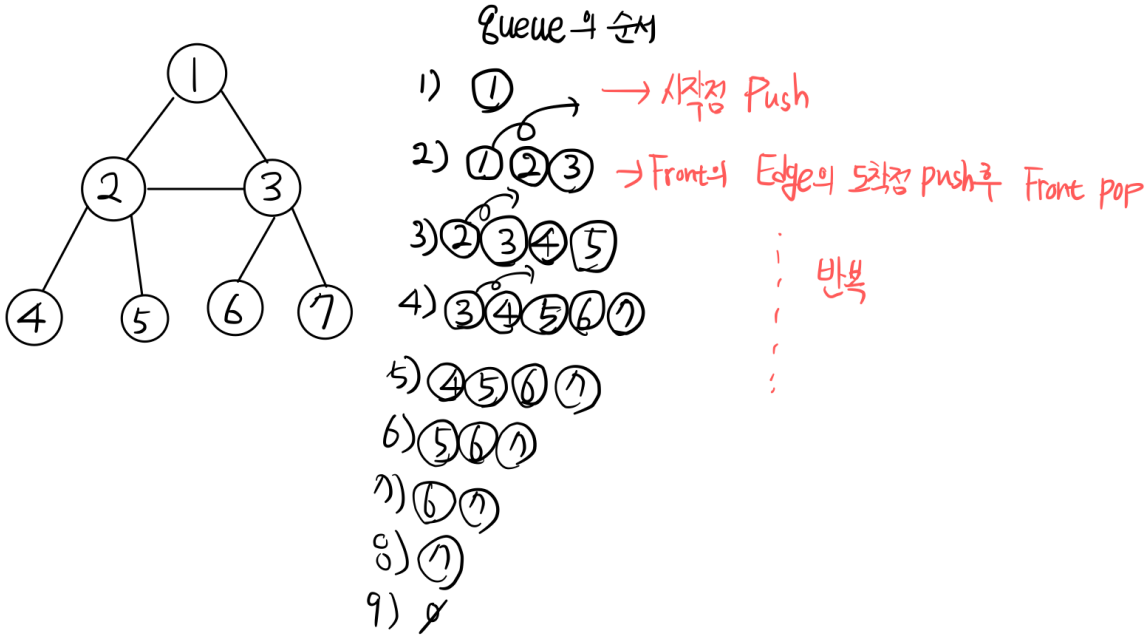
  만약, 생성했으면 start 원소가 가리키는 Vertex로 이동을 하고 queue에 값을 넣은 뒤 방문 했다고 하고 이제 반복문에 들어간다.

  반복문은 queue에 데이터가 없을 때 까지 반복을 한다. 또한, end value에 도착을 했

다면 그 시점으로 끝나게 된다. 이에 대한 경로를 알기 위해서 move라는 stack에 값을 저장한다. 반복문에서는 Front에서 갈 수 있는 Graph들을 모두 가져와서 queue에 넣는다. 그 뒤에 queue의 front가 방문 처리를 하고 삭제가 된다. 그리고 move 컨테이너에 부모의 key와 현재 key 값을 저장하고 넣는다. 마지막으로 end와 비교하는 값이 같다면 종료를 한다.

종료를 완료하고 move를 확인한다. 만약 move 마지막 value의 second가 end와 같지 않다면 이는 못 찾은 것 이므로 아무것도 존재하지 않은 result 벡터를 반환한다.

그렇지 않다면 제대로 나온 것 이므로 계속해서 반복문을 통해서 경로를 역추적하여 result라는 백터에 값을 집어넣게 된다. 이 때 값은 거꾸로 들어가기 때문에 나중에 출력을 할 때 주의를 해야한다. BFS알고리즘의 예시는 다음과 같다.



- DIJKSTRA

다익스트라 알고리즘은 얼핏보면 다이나믹프로그래밍 문제이다. 이때 구현을 하기위한 조건이 있는데 set을 사용을 해야한다는 것이다. 이 대한 알고리즘은 다음과 같다.

8. 우선 다익스트라를 사용하기 위해서 Graph에 대해서 IN_FINITY로 초기화 되고 Vertex의 개수만큼의 원소를 가지고 있는 vector를 하나 생성하고 graph를 인접 행렬처럼 가져오게 된다.
9. Graph를 m_graph를 이용하여 초기화 해준다. 이 때 Edge끼리 연결이 되어있지 않는 값들은 모두 IN_FINITY로 초기화를 해준다.
10. Graph를 초기화 하였다면, 우선 set에 출발하는 첫 번째 값을 넣어준다.
11. Set의 값이 empty가 될 때 까지 반복문을 계속한다. 반복문은 다음과 같다.
    i. Set의 begin값을 가져와서 current는 key distance는 value의 값으로 가져온

뒤 이러한 set의 begin값을 삭제한다.

ii. 만약에 Dijkstra(INF로 초기화가 된 vector 배열)의 current위치에 들어가있는 배열보다 distance보다 작다면 continue를 한다. -> 이미 작으면 이미 방문했다는 것으로 생각하면 되기 때문이다.

iii. 이 후 current Vertex가 가리키는 graph의 모든 값을 확인하여 거쳐간다면 얼마 만에 도착을 하는지 확인을 한다.

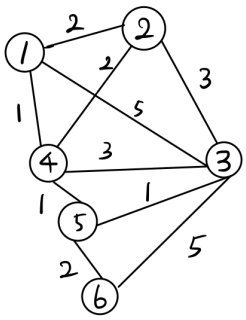
iv. 확인을 하는데 만약 기존의 distance보다 도착하는 값이 작다면 Dijkstra 배열의 비교하는 값을 갱신해준다.

v. 갱신한 배열을 set에 넣어준다. 이는 자동적으로 다시 정렬이된다.

vi. 나중에 역추적하기 위해서 path라는 컨테이너에 어떤 Vertex를 부모로 이동을 했는지 저장을 해준다.

vii. 이를 모든 Vertex를 돌때까지 반복한다.

이러한 반복문이 끝났다면 path에 저장된 값을 통해서 end value로부터 역추적을 하여 result 백터에 값을 넣는다. 이도 거꾸로 들어가기 때문에 나중에 출력할 때 거꾸로 출력해야한다. 예시는 다음과 같다.



DIJKSTRA 배열

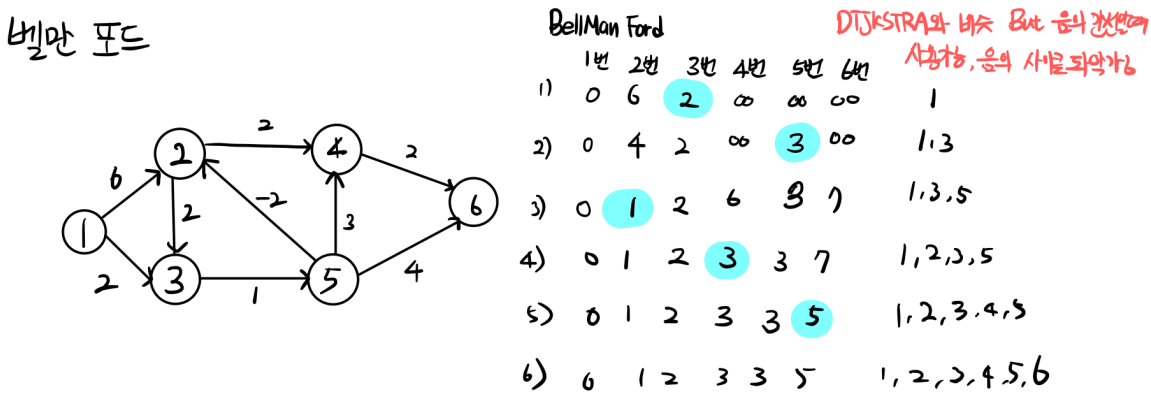
제일 작은 값 선택.

	1번	2번	3번	4번	5번	6번	
1번꼬1)	0	2	5	1	∞	∞	→ 시작점에서 갈수있는곳
4번꼬2)	0	2	4	1	2	∞	⇒ 3번꼬는 1→3 보다 1→4→3이 빠르게 갱신!

반복하면 값을 얻을 수 있다!

- BELLMANFORD

벨만포드를 구현하기 위해서는 우선 graph에 대해서 Edge가 있는 값들만 저장한다. 이러한 Edge가 저장된 graph를 모든 경우의 수를 돌도록 확인한다. 비교하는 edge의 시작점과 wieght를 더한 값이 비교하는 도착점에 저장된 weight값보다 작을경우 바꾸어 준다. 또한, 이러한 weight를 변경하는 것이 끝나게 된다면 한 번더 반복을 하게 되어 값이 변하는지를 확인한다. 정상적인 graph라면 변할 수가 없는데 음수사이클이 있다면 그 사이클을 반복하면 무한정히 weight를 줄일 수 있기 때문에 값이 갱신될 것이다. 그렇기 때문에 갱신이 된다면 음수 사이클이 존재한다고 볼 수 있다.



- FLOYD

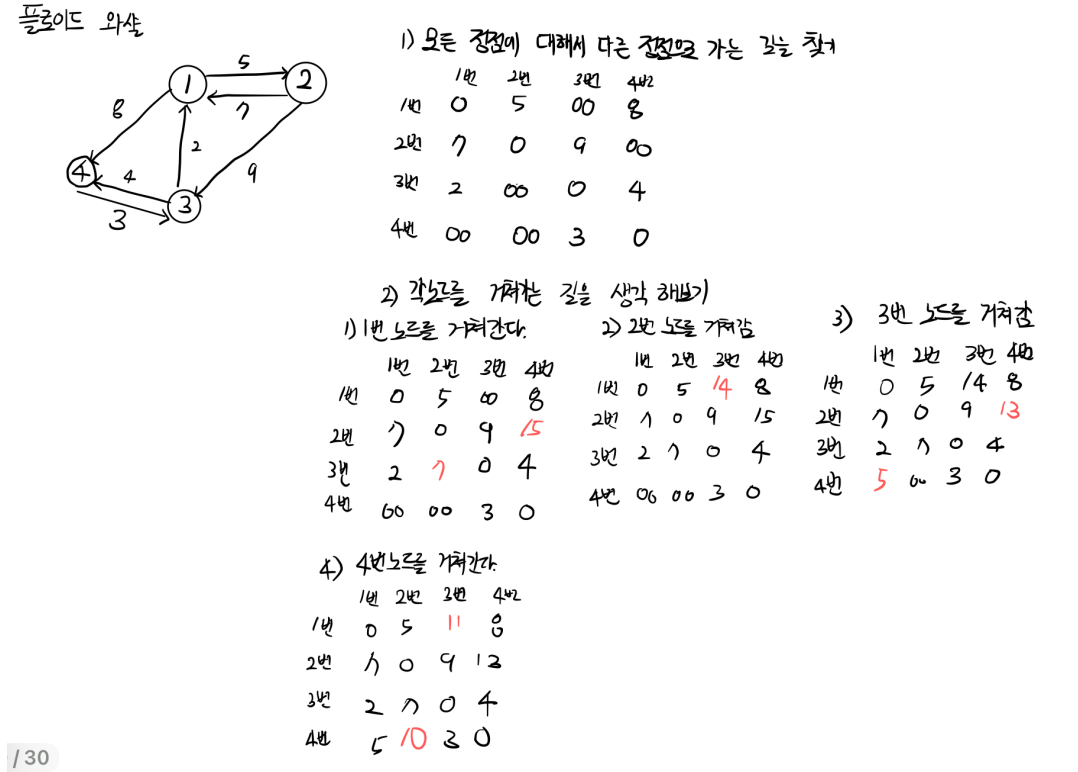
  플로이드 알고리즘은 모든 graph에 대해서 값이 나오게 된다.

  첫 번째로는 우선 Graph를 2차원 배열에 넣는다. 이 때 다익스트라와 벨만포드와는 달리 각 Vertex의 자기 자신은 IN_FINITY가 아닌 0으로 초기화 하게 된다. 그리고 이제 3중포문을 통해서 플로이드 알고리즘을 계산하기 시작한다.

  각각의 Vertex를 경유하는 경우를 생각하면서 반복문을 검사한다. 이를 각각 start, end, min로 나타내서 반복문을 결정하였다.

  1. Start부터 end까지의 거리가 start에서 mid까지의 거리와 mid에서 end까지의 거리보다 큰지 검사를 한다. 만약 그렇다면 해당 합계가 start에서 end까지의 거리보다 최적의 경로이기 때문에 테이블에 값을 저장하게 된다.
  2. 음수사이클이 발생할 경우를 생각하면 Vertex에서 Vertex까지의 값 즉 자기자신을 가리키는 값이 0으로 초기화 되어있는데 이 값들이 음수가 된다면 음수 사이클이 있다고 판별할 수 있기 때문에 이를 활용해서 음수사이클을 판별하게 된다.

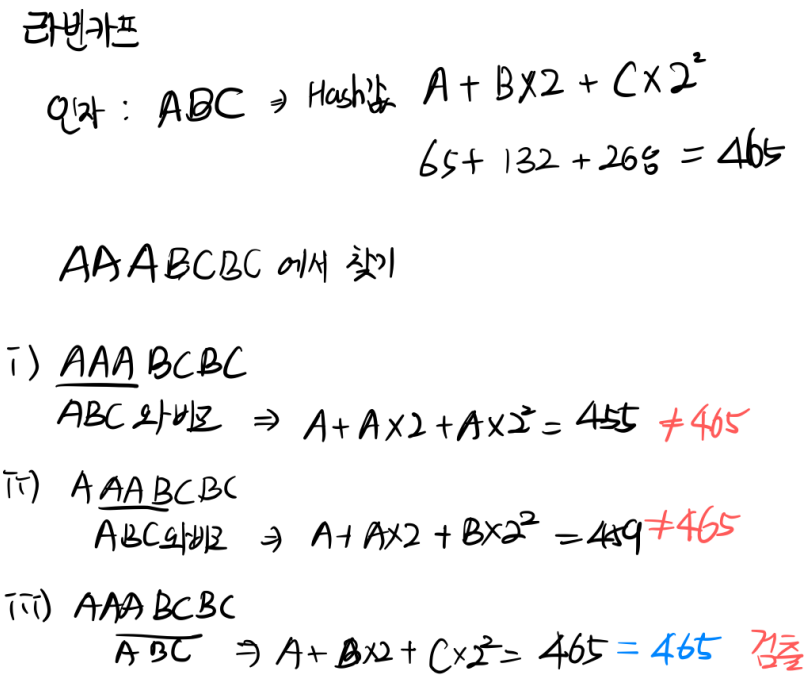
  이에대한 예시는 다음과 같다.



/ 30

- RABINKARP

라빈카프는 구현에 대해서는 쉬운 알고리즘이다. 우선 각각의 문자에는 아스키코드 값이 있다. 라빈카프 메서드를 실행하기 위해서 받은 인자값에 대해서 첫 번째 인자부터 n 번째 인자까지를 해당 아스키 코드 값 * 2^번째로 를 계산하여 Hash값을 저장해준다. 그 뒤에 모든 문장을 돌면서 이 값들을 확인해 주는 절차를 받게 된다. Hash값이 같다면 count라는 벡터에 값을 저장하고 마지막으로 count라는 벡터에서 해당 벡터의 문장 값을 출력하고 이를 종료한다. 만약 일치하는 단어가 없다면 에러 메시지를 출력 후 종료하게 된다.



- 값 비교

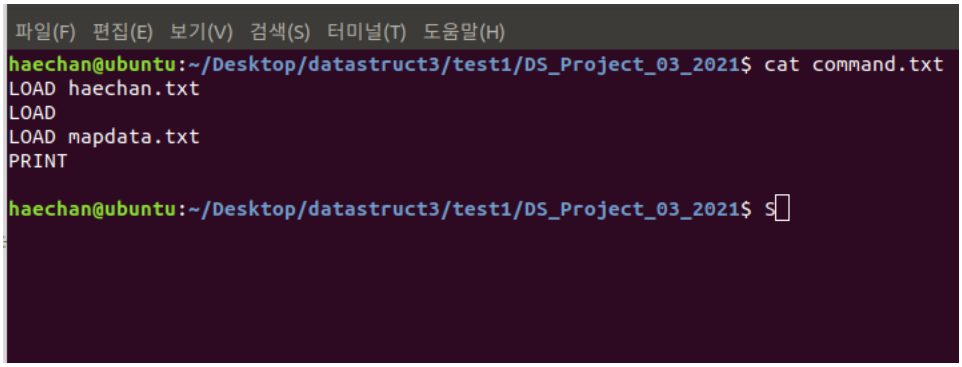
  그래서 이들의 알고리즘을 생각해보면 실제적인 알고리즘과 다를 수도 있다. 왜냐하면 어떤 것을 그냥 Graph그대로 값을 사용한 것들도 있고 아니면 Graph클래스를 사용하는 것이 아닌 Graph클래스에서 값을 가져와서 인접 리스트로 변환하여 알고리즘을 진행한 것도 있다. 그래도 비교를 해보자면 다음과 같이 값이 나오게 되었다.

```
BFS는 4e-06ms 입니다
DIJKSTRA 는 1.1e-05ms 입니다
BELLMANFORD는 2e-06ms 입니다
FLOYD는 3e-06ms 입니다
```

  일단 BELLMANFORD는 음수사이클에 대한 검사를 하기 때문에 분명히 DIJKSTRA보다 그냥 있어도 느릴 수 밖에 없다. 다익스트라는 O(E log V) (V는 Vertext의 개수이다) 이고 벨만포드는 또한 O(V E)이다. V는 정점의 개수 E는 간선의 개수이다. BFS는 O(E)이므로 남들보다 빠르다. 마지막으로 플로이드는 모든 정점에 대해서 돌아다녀야하므로 최악의 경우를 따지면 O(V^3) 이므로 이론적으로는 제일 느릴 수 밖에 없다. 하지만 결과값이 다른것을 보면 알고리즘에 대해서 좀더 복잡하게 하는 이유가 있던지 혹은 역추적할 때 시간이 더걸리던지 이러한 경우의 수가 충분히 있을 수 있다고 생각한다.

4. Result

1. LOAD



다음같이 LOAD 명령어에 텍스트가 없는 경우 인자가 없는 경우 인자가 제대로 들어간 경우를 확인하기 위해서 3가지를 넣고 확인해 본다.

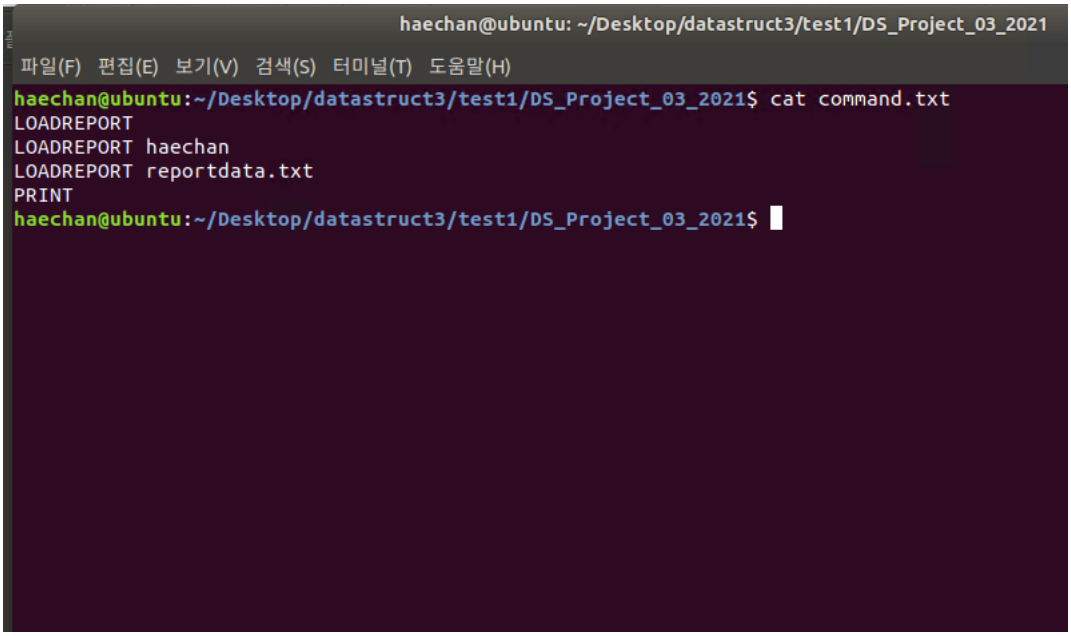
```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
======LoadFileNotExist=======
Error code: 101
==========================

======LoadFileNotExist=======
Error code: 101
==========================

==========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 0 6 2 0 7
0 0 0 0 5 6 0
===================================

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

NonDefinedCommand

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

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

NonDefinedCommand

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

다음은 공백이 두개가 들어가 있어서 NonDefinedCommand 오류가 두번 나온 것을 확인 할 수 있다.

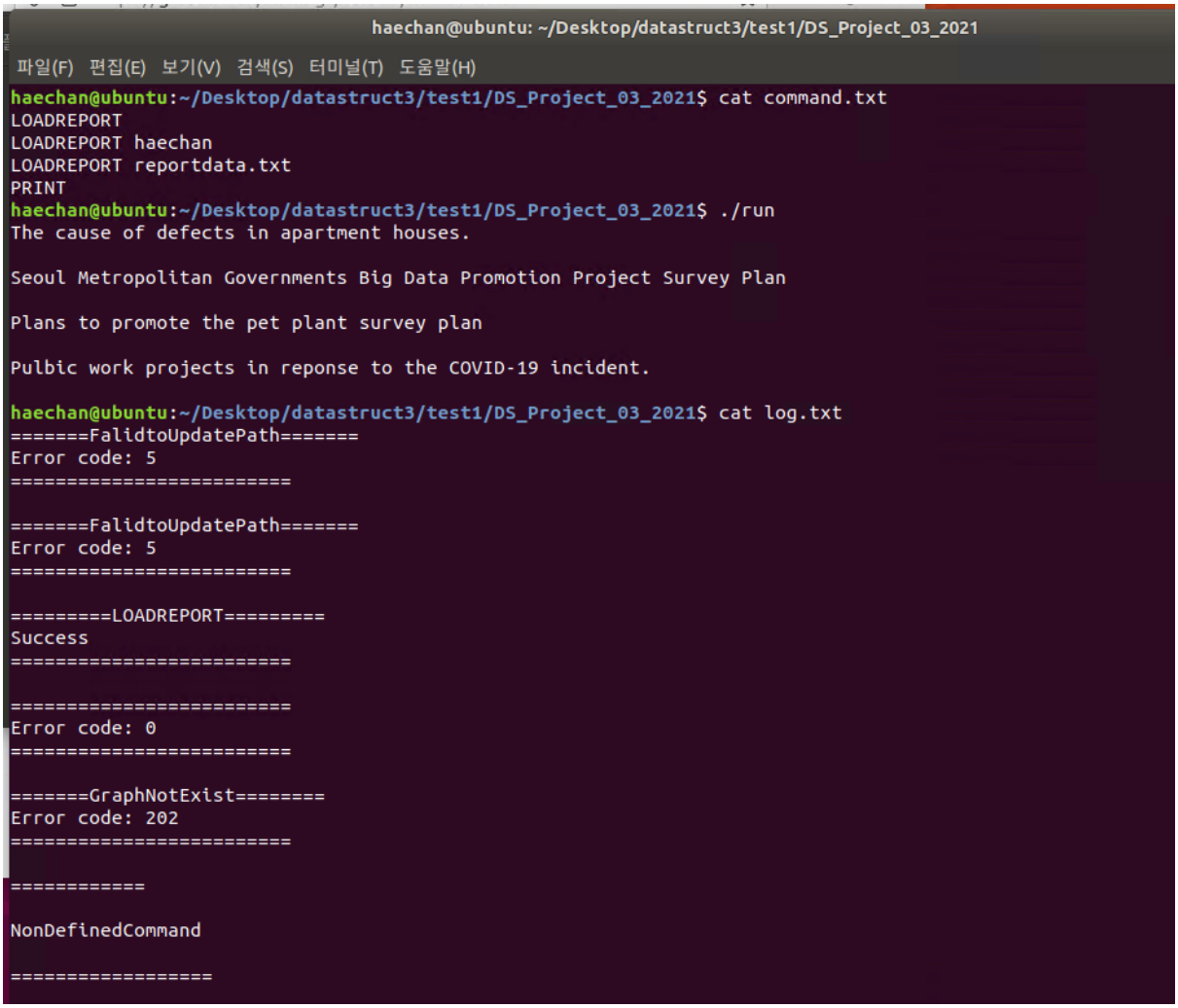
또한, 나머지 경우에 대해서는 제대로 LoadFileNotExist에러가 나게 되고 값은 제대로 PRINT로 확인해보니 들어간 것으로 볼 수 있다.

2. LOADREPORT

이것도 LOAD와 같은 방식으로 검증을 하였다.



다음과 같이 확인을 하고 결과값을 살펴보면 다음과 같다.



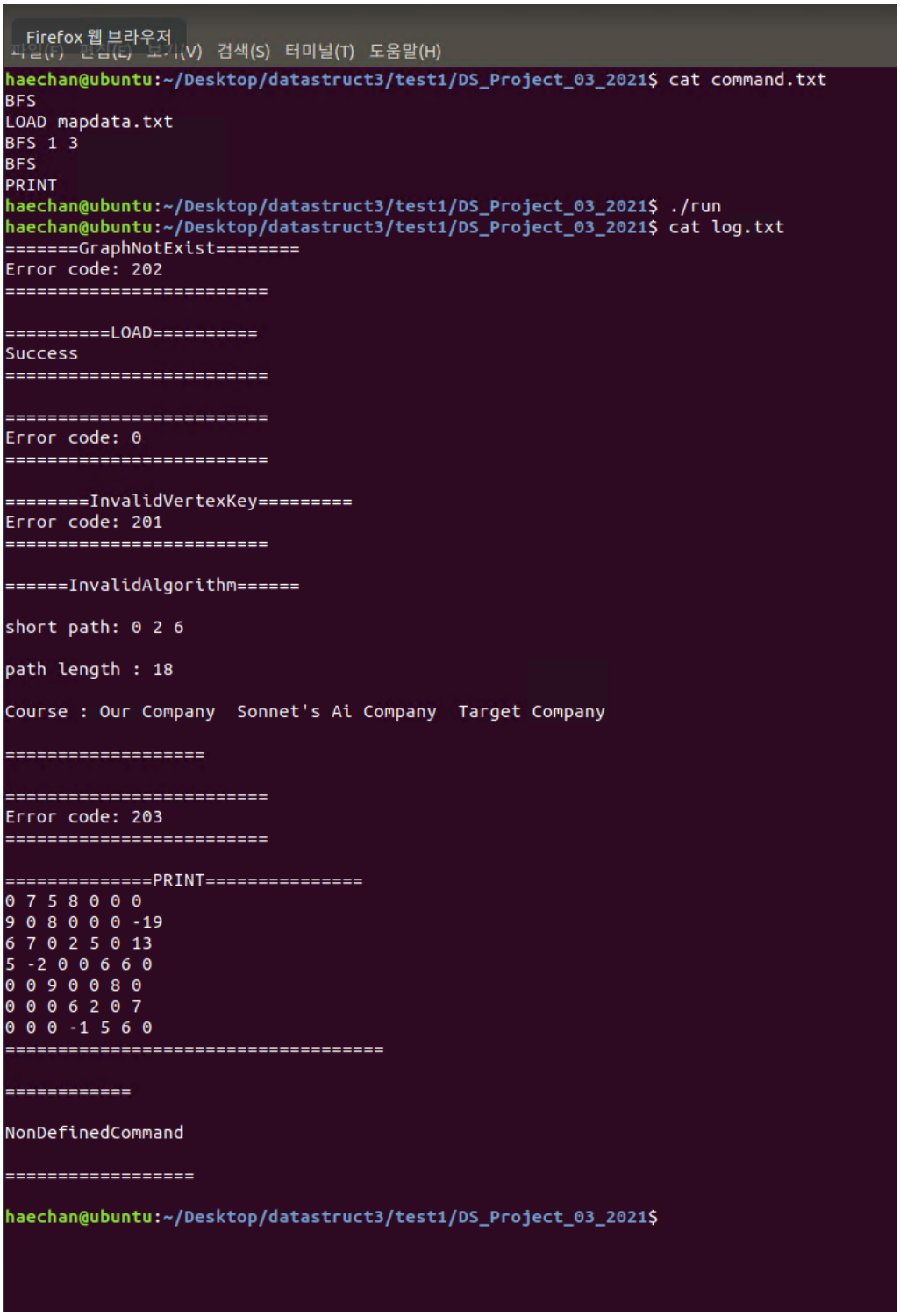
값이 제대로 출력되는지를 확인하기 위해서 일부러 LOADREPORT에 값을 출력하도록 만들었다. 제대로 값이 들어간 것을 볼 수 있다. 또한, 파일이 존재하지 않는 경우 이

기 때문에 제대로 FalidtoUpdatePath가 나오는 것을 볼 수 있다. 그 뒤에 제대로 입력한 인자에 대해서는 Success가 제대로 출력이 된 것을 볼 수 있다. PRINT는 잘못 입력하였는데 이것도 Graph가 없어서 제대로 출력된 것을 볼 수 있다. 마지막은 개행문자 인데 제대로 NonDfinedCommand가 나오는 것을 볼 수 있다.

3. BFS

   BFS는 두 번 확인 할 예정입니다. 첫 번째는 음수 간선이 있는지? 두 번째는 음수 간선이 없을 때

   첫 번째는 다음과 같이 확인 하였습니다.



Mapdata에 대해서 값이 잘 나온 것을 볼 수 있다. 또한, 음수인 간선이 존재해서 출력할 때 InvalidAlogrithm으로 출력이 된다.

LOAD를 하지 않아서 Graph가 없는 경우는 다음과 같이 202번 에러코드가 나오고 만약 인자가 없어야하는데 인자가 들어간 경우는 InvalidVertexKey가 나오게 된다.

만약, Graph가 음수 간선이 존재하지 않으면 다음과 같은 결과를 가진다.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
BFS
LOAD mapdata.txt
BFS 1 3
BFS
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
========================

==========LOAD==========
Success
========================

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

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

======InvalidAlgorithm======

short path: 0 2 6

path length : 18

Course : Our Company  Sonnet's Ai Company  Target Company

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

========================
Error code: 203
========================

==============PRINT===============
0 7 5 8 0 0 0
9 0 8 0 0 0 0
6 7 0 2 5 0 13
5 -2 0 0 6 6 0
0 0 9 0 0 8 0
0 0 0 6 2 0 7
0 0 0 -1 5 6 0
==================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

InvalidAlogrithm이 아닌 그냥 BFS로 나오게 된다.

4. DIJKSTRA

   다익스트라도 BFS와 비슷하게 검증하면 다음과 같다.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
DIJKSTRA
LOAD mapdata.txt
DIJKSTRA 1 3
DIJKSTRA
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
========================

==========LOAD==========
Success
========================

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

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

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

shortest path : 0 2 6

path length : 18

Course : Our Company  Sonnet's Ai Company  Target Company

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

==============PRINT===============
0 7 5 8 0 0 0
9 0 8 0 0 0 0
6 7 0 2 5 0 13
5 0 0 0 6 6 0
0 0 9 0 0 8 0
0 0 0 6 2 0 7
0 0 0 0 5 6 0
====================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

파일

다음과 같이 Graph를 로드하지 않았을 경우는 GraphNotExist를 한다.

그리고 로드를 한 뒤에 인자가 있는 경우는 없기 때문에 인자가 있을 경우는 InvalidVertexKey가 있다. 그리고 다익스트라를 보면 shortest path는 0 2 6이 확실하게 맞고 이에대한 length도 정확한 것을 볼 수 있다. 또한, Course또한 맞다.

마지막으로는 Weight가 음수인 경우를 확인해보자.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
DIJKSTRA
LOAD mapdata.txt
DIJKSTRA 1 3
DIJKSTRA
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
========================

==========LOAD==========
Success
========================

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

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

=========InvalidAlgorithm==========

shortest path : 0 2 6

path length : 18

Course : Our Company  Sonnet's Ai Company  Target Company

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

========================
Error code: 203
========================

==============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 0 6 2 0 7
0 0 0 0 5 6 0
====================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

음수인 경우는 다음과 같이 InvalidAlgorithm과 같이 나오게 된다.

5. BELLMANFORD

BELLMANFORD는 2가지로 검증을 한다. 검증 방법은 첫 번째는 음수 사이클이 없는 경우 두 번째는 음수사이클이 있는 경우이다.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
BELLMANFORD
LOAD mapdata.txt
BELLMANFORD 1 3
BELLMANFORD
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
========================

==========LOAD==========
Success
========================

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

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

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

shortest path : 0 1 6

path length : -12

Course : Our Company  Denny's Stationery store  Target Company

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

==============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 0 6 2 0 7
0 0 0 0 5 6 0
==================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

이것 또한 Graph가 없는 경우는 다음과 같이 GraphNotExist가 출력된다. 그리고 인자가 있는 경우는 InvalidVertexKey가 나오고 마지막으로는 BELLMANFORD는 제대로 값이 나오는 것을 알 수 있다. 0 1 6에 대해서 shoft path는 정답이고 length도 제대로 나온것이 맞다. Course는 다음과 같이 제대로 나오는 것을 볼 수 있다.

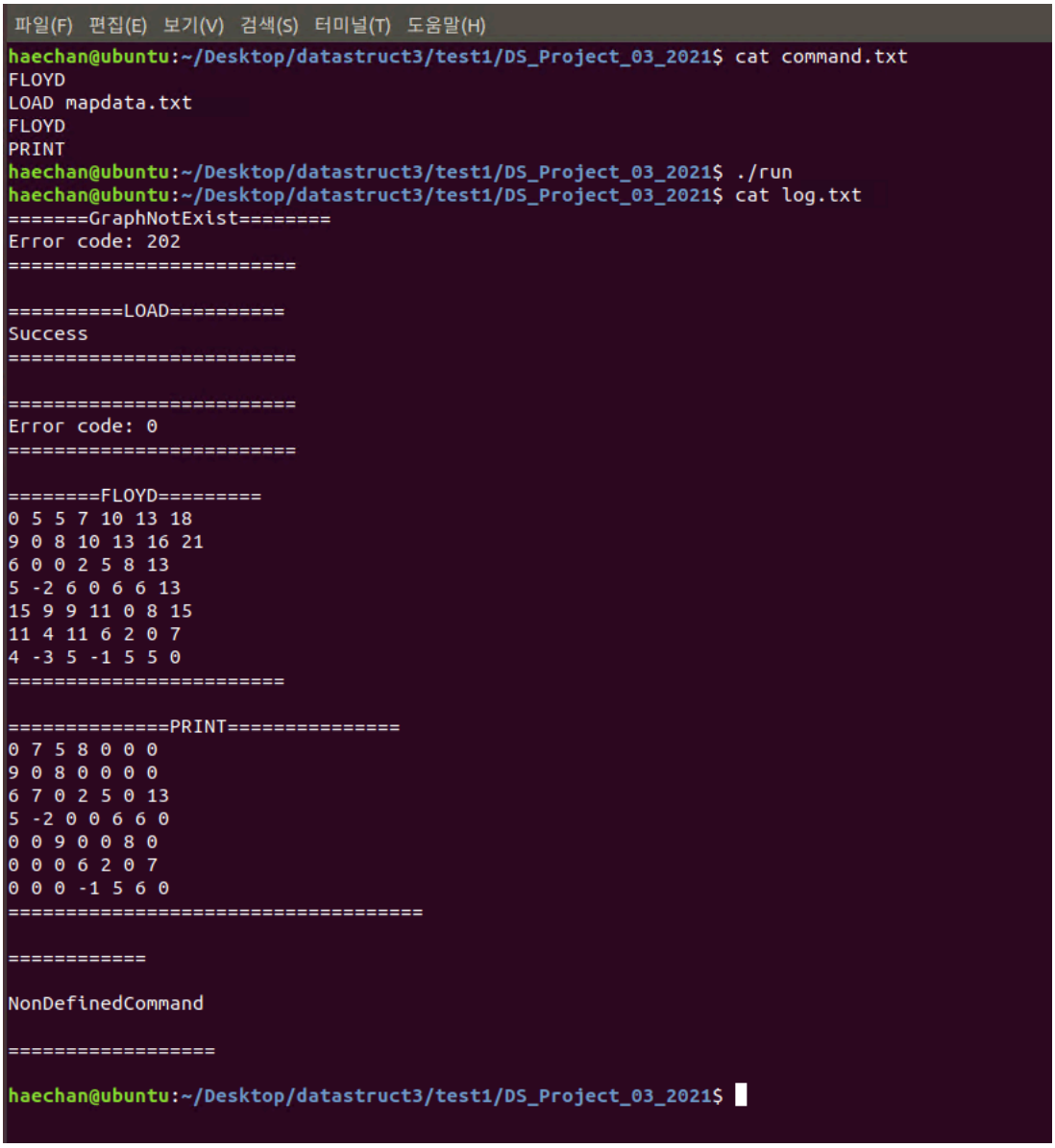
그리고 음수 사이클이 있는 경우는 다음과 같다.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
BELLMANFORD
LOAD mapdata.txt
BELLMANFORD 1 3
BELLMANFORD
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
========================

==========LOAD==========
Success
========================

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

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

=========NegativeCycleDetected========
Error code: 204
========================

==============PRINT===============
0 7 5 8 0 0 0
9 0 8 0 0 0 -19
6 7 0 2 5 0 13
5 -2 0 0 6 6 0
0 0 9 0 0 8 0
0 0 0 6 2 0 7
0 0 0 -1 5 6 0
==================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

다음과 같이 NegaticeCycleDetected가 나오게 된다.

6. FLOYD

FLOYD를 확인해보자 이것도 마찬가지로 하는데 인자가 입력되는 경우는 없기 때문에 그것은 빼고한다.



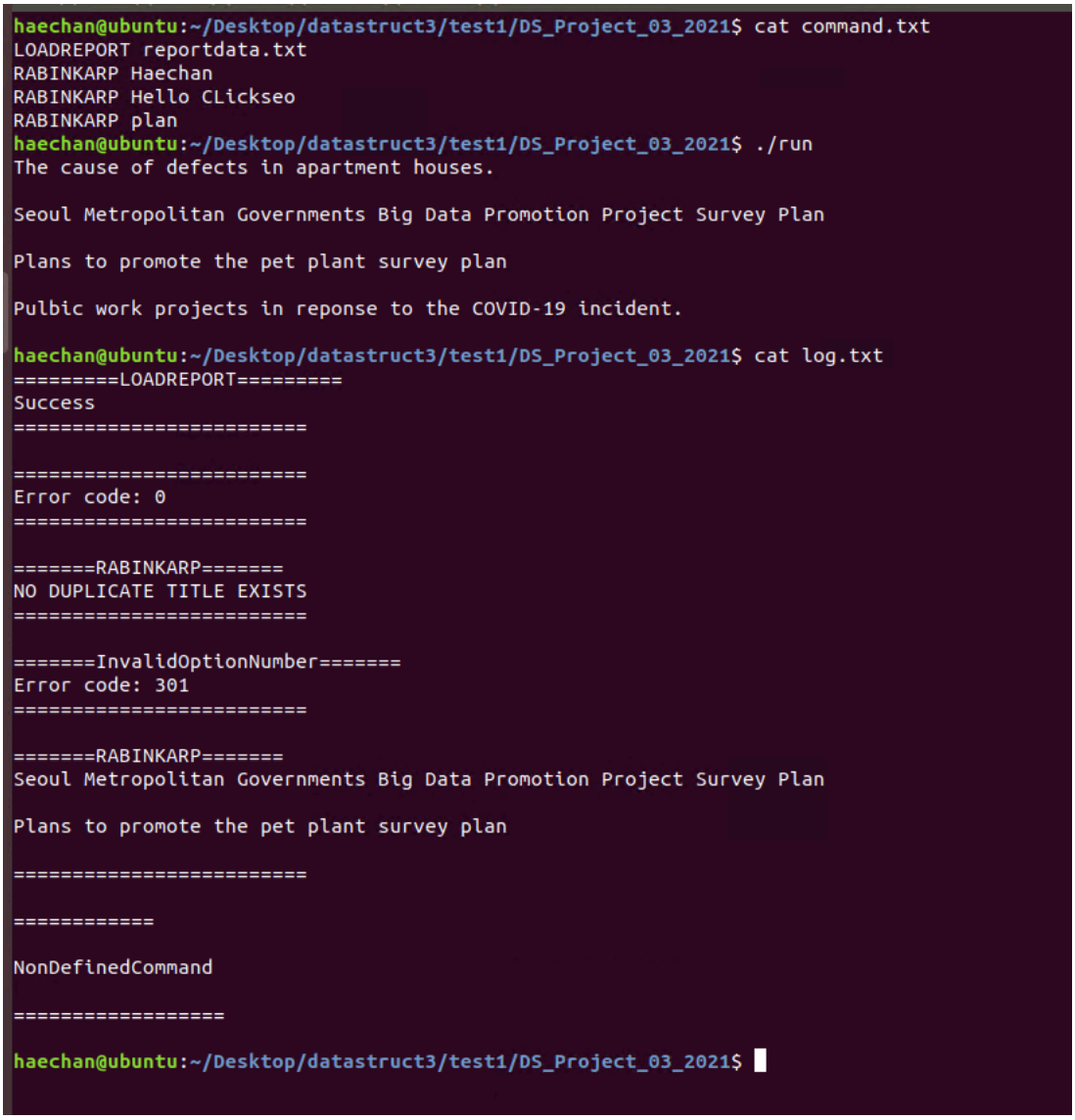
Floyd를 확인해보면 처음에는 GraphNotExist가 Load를 안했기 때문에 나타나고 마지막으로 음수사이클이 아닐 때 FLOYD는 정상값을 가지게 된다 만약 음수 싸이클일 겨우는 다음과 같다.

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
FLOYD
LOAD mapdata.txt
FLOYD
PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
=======GraphNotExist========
Error code: 202
=========================

==========LOAD==========
Success
=========================

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

=========NegativeCycleDetected========
Error code: 204
=========================

==============PRINT===============
0 7 5 8 0 0 0
9 0 8 0 0 0 -19
6 7 0 2 5 0 13
5 -2 0 0 6 6 0
0 0 9 0 0 8 0
0 0 0 6 2 0 7
0 0 0 -1 5 6 0
==================================

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

확실하게 음수 사이클이 판별되는 것을 볼 수 있다.

7. RABINKARP

라빈카프는 10글자 제한이 있다. 3가지를 나눠서 판단할 것이다. 우선 인자가 많은 경우와 인자가 글안에 없는 경우 인자가 글에 있는 경우를 확인할 것이다.



확인해보면 다음과 같다. 우선 Haechan이라고 검색을 하면 값이 없기 때문에 NO DUPLICATE TITLE EXISTS라고 오류가 나온다. 또한, 만약에 인자가 넘어간다면 InvalidOptionNumber이므로 값을 확인해보면 잘 나오는 것으로 볼 수 있다. 마지막으로는 plan을 찾아보면 대소문자 구별이 없기 때문에 확인해보면 Plan도 검색한 것을 볼 수 있다.

8. 명령어 잘못

```
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
Haechan
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
======Haechan======

NonDefinedCommand

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

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

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

확인을 해보면 다음과 같다. Haechan이므로 이름이 오류가 난 것을 볼 수 있다. Haechan이라는 이름으로 공백은 아무것도 없기 때문에 값이 안 나온 것을 볼 수 있다.

9. 주석 기능

만약 주석을 이용하기 위해서 //을 입력하면 다음과 같다.

```
haechan@ubuntu: ~/Desktop/datastruct3/test1/DS_Project_03_2021
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
//LOAD
//LOAD
//LOAD mapdata.txt
//PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

다음과 같이 입력을 하고 run을 한다면 아무것도 시행이 안되는 것을 볼 수 있다. 하지만 개행문자에 대해서는 안되있기 때문에 오류코드가 발생하는 것을 볼 수 있다. 다음과 같은 결과를 낸다.

```
haechan@ubuntu: ~/Desktop/datastruct3/test1/DS_Project_03_2021
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat command.txt
//LOAD
//LOAD
//LOAD mapdata.txt
//PRINT
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ ./run
haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$ cat log.txt
============

NonDefinedCommand

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

haechan@ubuntu:~/Desktop/datastruct3/test1/DS_Project_03_2021$
```

다음과 같이 개행문자에만 오류가 생기는 것을 볼 수 있다. 개행문자는 아무것도 없기 때문에 명령어와 함께 개행문자가 나오지 않는다.

5. Consideration

이번 실험에 대해서는 Graph를 처음 학기중에서 처음 다루다 보니 많은 어려움을 겪었습니다. 처음에는 이러한 Graph클래스를 이용하여 Graph를 만드는 것 조차 어려움을 느꼈습니다. 그리고 최단 경로 알고리즘을 이용하면서 DP라는 알고리즘 기법에 대해서도 알게되었습니다. 다익스트라와 벨만포드를 하면서 Dynamic Programing이라고 해서 이게 뭐지? 라는 생각을 했는데 되게 간단하게 이전 계산 결과값을 저장하는 거라고 생각하면 편했습니다. 또한, 이번 프로젝트에서 Enum이라고 열거형을 처음 사용해보았는데 편하게 매크로 변수처럼 사용할 수 있다는 점이 좋았습니다. 이것도 잘 사용하면 활용방안이 많을 것이라고 생각하게 되었습니다. 또한, 이번 데이터구조 실습을 하면서 STL에 대해 많이 사용하게 되었는데 set이나 vector pair 같은 STL 컨테이너 들에 대해서 자주 사용하다 보니 활용력이 많이 올라가게 되었습니다. 또한, 벨만포드에 대해서는 분명히 다익스트라와 비슷한 알고리즘인데 왜 값이 다르게 나오지 라는 고민이 많아 어려웠습니다. 마지막으로는 BFS, 다익스트라, 벨만포드 다 구현은 하겠는데 Shortest path를 찾는 과정이 제일 어려웠던것 같습니다. 어떻게 하지? 라는 생각에 고민을 많이 했었는데 결론적으로는 값이 갱신될 때 마다 이를 path라는 변수를 따로 두어서 구한 뒤 역추적하는 방식을 택해서 shortest path를 구할 수 있게 되었습니다. 마지막으로는 라빈카프 알고리즘을 구할 때 이게 편하고 완벽하게 문자열을 찾을 수 있는 알고리즘이라 생각 했었는데 이게 중복되는 해시 값도 있구나 라는 것을 깨닫았다. Hi를 넣었는데 분명 문장에 Hi가 없는데 값이 나오는 것을 보고 디버깅을 해보았는데 해시 값이 같은 문자도 있다는 것을 깨닫았다.