Data Structure Project

Project #3

담당교수 : 이기훈

제출일 : 2018. 12. 12.

학과 : 컴퓨터공학과

학번 : 2017202088

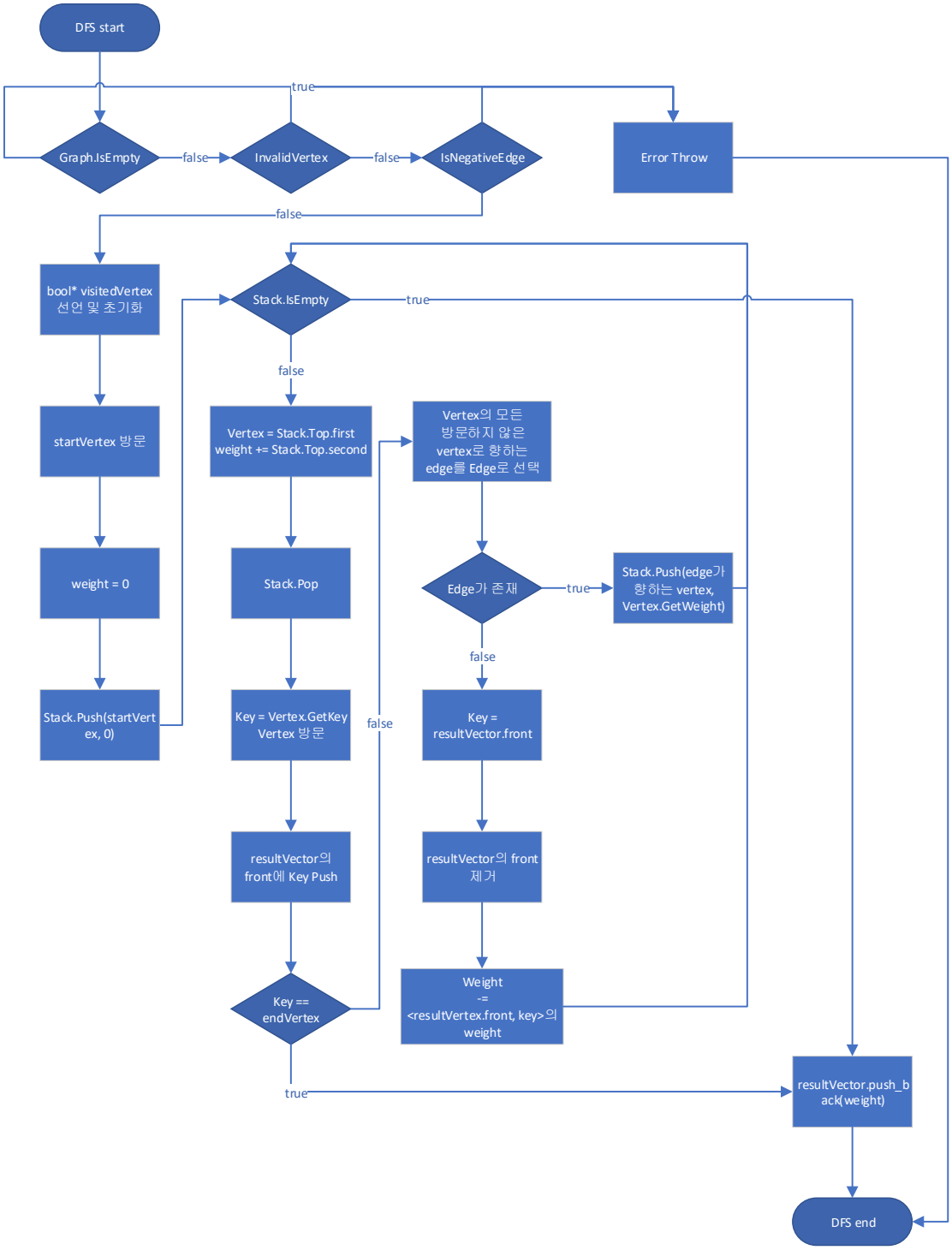
이름 : 신 해 담

1. Introduction

- 도로 그래프를 이용하여 최단경로를 찾는 프로그램을 구현한다. 도로 정보가 Matrix 형태로 저장되어 있는 텍스트 파일을 읽어 그래프를 구현한다. 그래프 정보는 방향성(Direction)과 가중치(Weight)를 가진다. 그리고 그래프와 DFS, Dijkstra, Bellman-Ford, FLOYD 알고리즘을 이용하여 도로의 경로와 거리를 구한다. 음수 Weight가 존재하면 DFS와 Dijkstra는 에러를 출력하며, Bellman-Ford와 FLOYD에서는 음수 사이클이 존재하면 에러를 출력한다. 프로그램의 동작은 명령어 파일에서 요청하는 기능을 수행하고, 결과를 로그(log.txt) 파일에 저장한다.

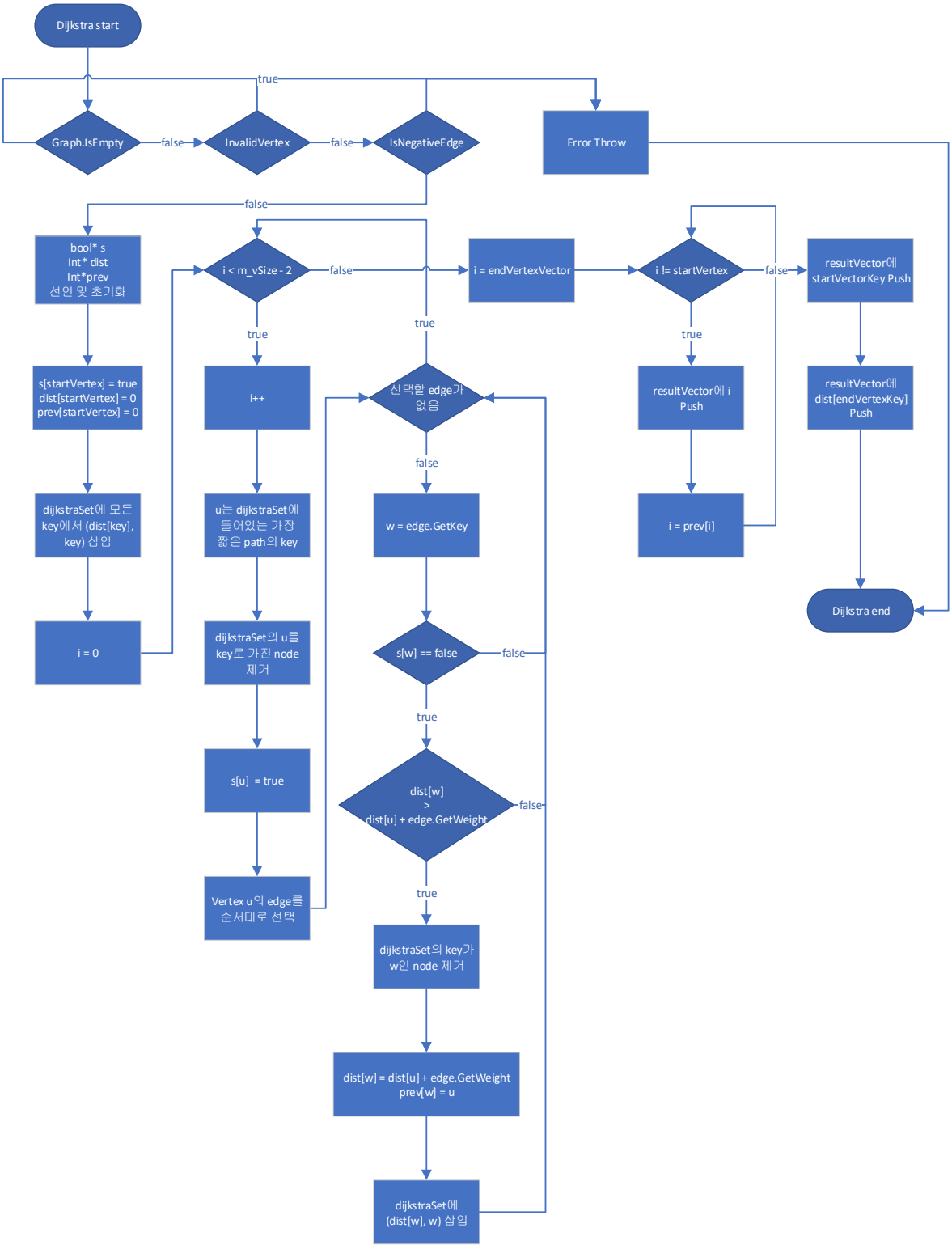
2. Flow Chart

- DFS의 Flow Chart



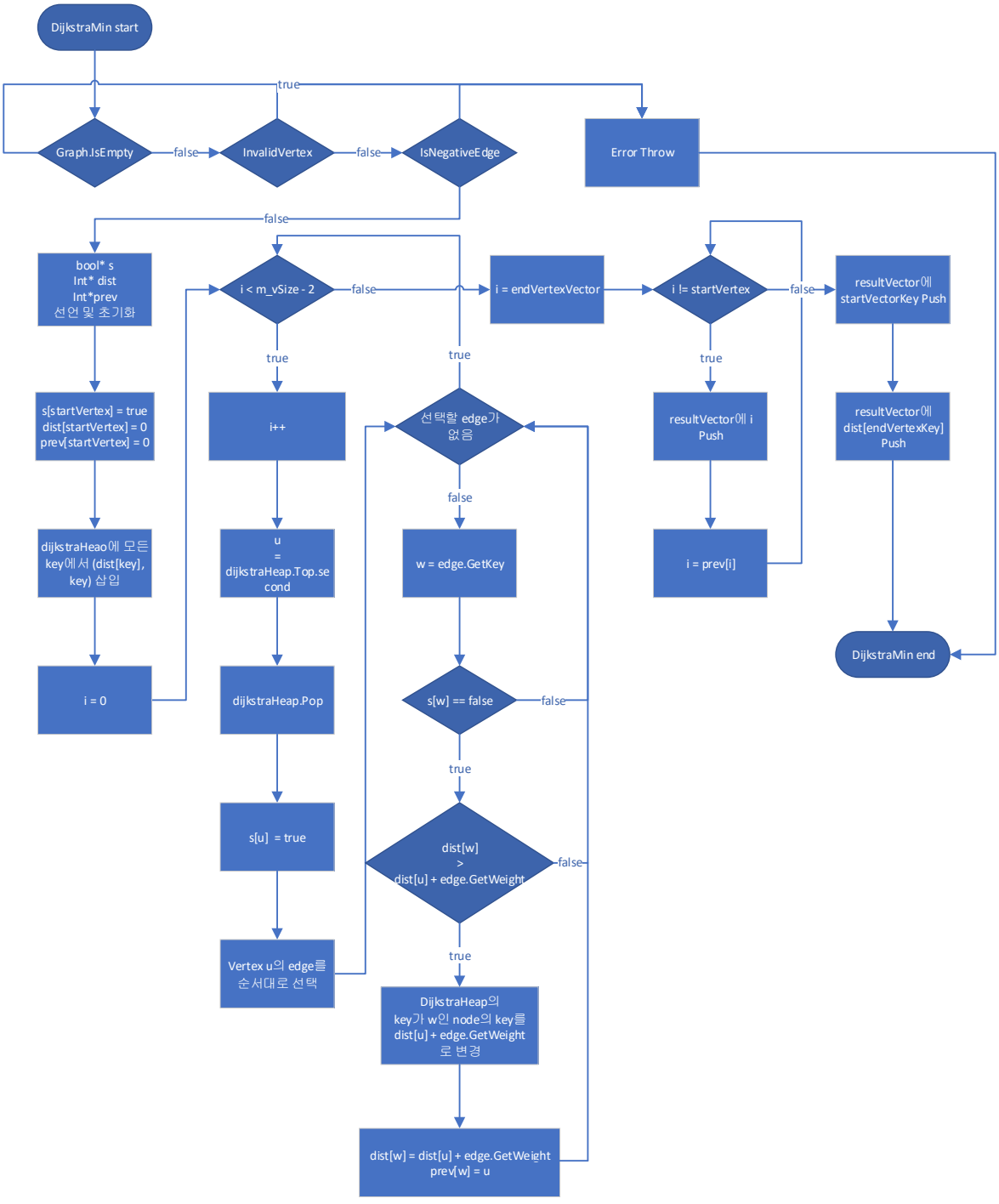
스택이 비었는지 확인하며 반복을 수행하여 DFS를 수행한다.

- Dijkstra의 Flow Chart



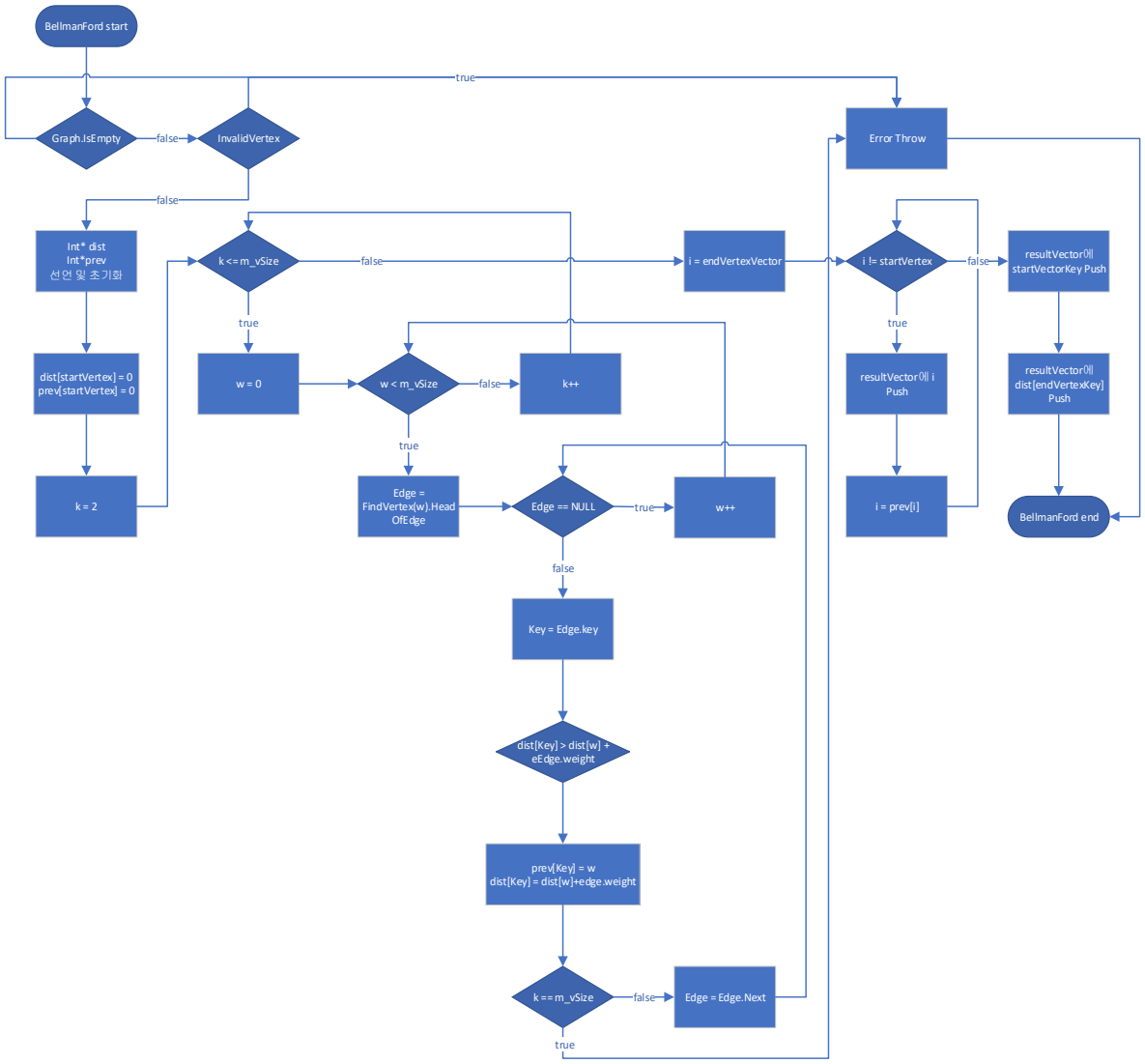
각종 선언과 초기화 후 i에 대하여 for loop를 돌며 dijkstra를 수행한다. 그리고 resultVector를 생성하여 반환하고 dijkstra를 종료한다.

- DijkstraMin의 Flow Chart



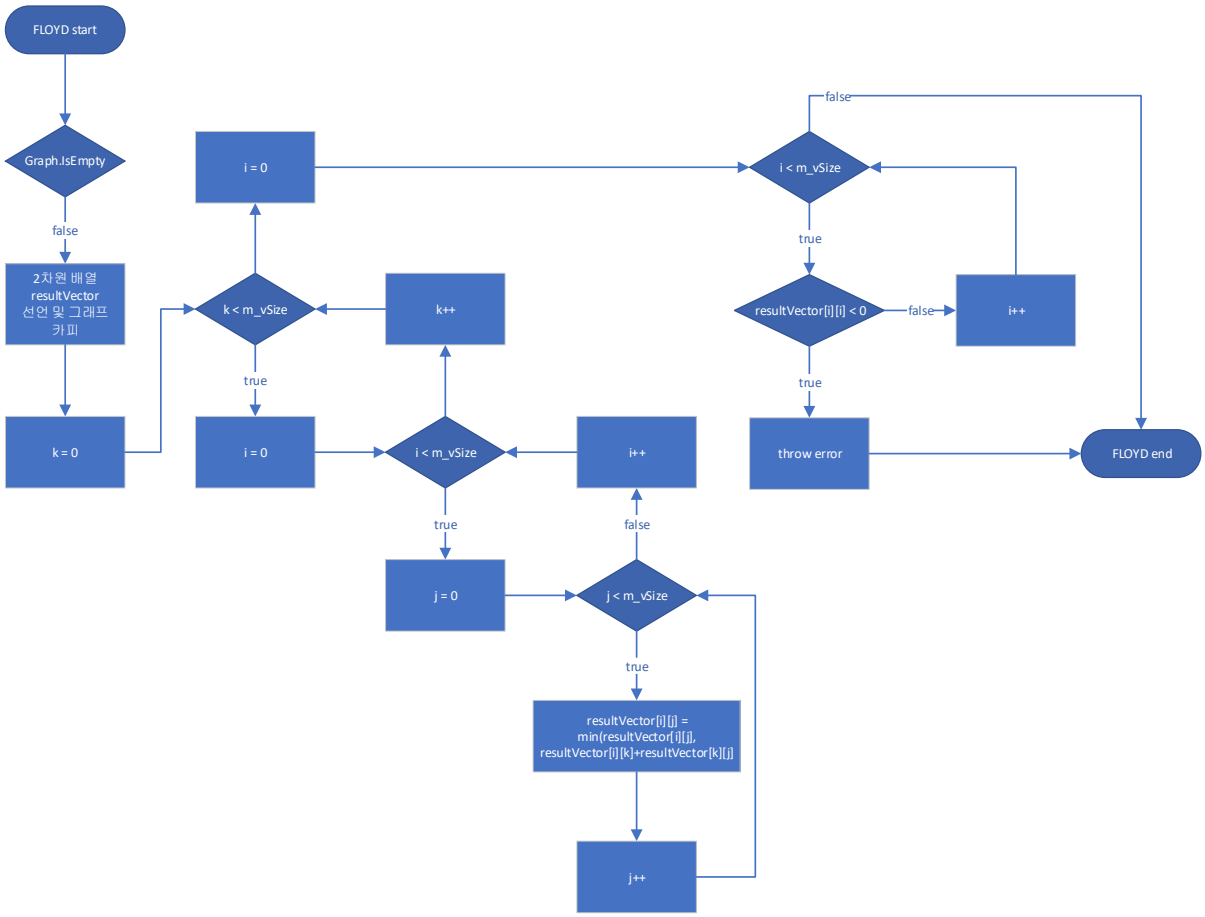
Dijkstra 알고리즘을 min heap을 사용하여 구현한 flow chart이다.

- Bellman-Ford의 Flow Chart



각종 선언과 초기화 후 k, w, edge에 대하여 루프를 돌며 Bellman-Ford를 수행한다. 그리고 resultVector를 생성하여 반환하고 종료한다.

- FLOYD의 Flow Chart

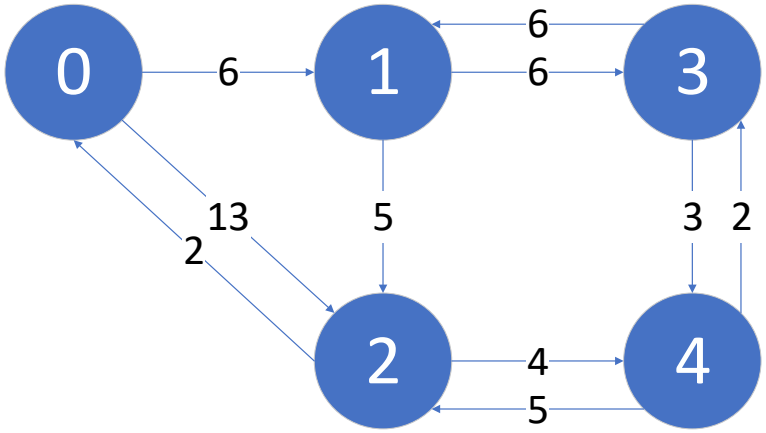


k, i, j에 대하여 중첩 루프를 돌며 FLOYD Matrix를 구한다. 그리고 음수 사이클 발생 여부를 체크하고 Matrix를 반환한다.

3. Algorithm

- DFS

  DFS(Depth-First Search)는 Stack에 시작 vertex를 넣고 edge 하나로 연결된 모든 방문하지 않은 vertex를 스택에 넣는다. 그리고 스택의 Top의 vertex를 방문하고 Pop한다. 그리고 그 vertex와 연결된 방문하지 않은 vertex를 스택에 넣는다. 이 과정을 반복하여 원하는 vertex를 찾는다.



  다음과 같은 그래프에서 0에서 3까지의 경로를 DFS로 찾는다고 가정한다. 스택에는 시작 vertex를 넣고 시작한다.

  Stack : 0 Path :

  그리고 0을 방문하고, 0과 연결된 vertex 1, 2를 스택에 넣는다.

  Stack : 1 2 Path : 0

  그 다음에 2를 방문하고, 2와 연결된 vertex 3, 4를 스택에 넣는다.

  Stack : 1 3 4 Path : 0 2

  그 다음 4를 방문하고, 4와 연결된 vertex 3을 스택에 넣는다.

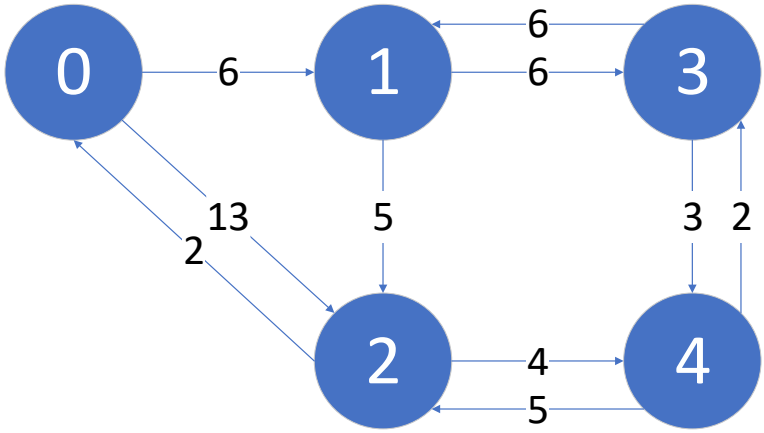
  Stack : 1 3 3 Path : 0 2 4

  그 다음 3을 방문하고, 목적 vertex에 도착했으므로 DFS를 종료한다.

  Stack : 1 3 Path : 0 2 4 3

- Dijkstra

  Dijkstra는 Source vertex를 기준으로 edge를 하나 사용하여 도달할 수 있는 vertex의 거리로 dist를 업데이트한다. 그리고 거리가 가장 짧은 vertex를 방문한다. 그 vertex를 기준으로 edge를 하나 사용하여 도달할 수 있는 vertex의 거리가 Source vertex에서의 거리보다 가까우면 업데이트한다. 다음으로 Source vertex에서 가장 가까운 방문하지 않은 vertex를 방문한다. 이 과정을 n–2번(초기화 포함) 반복하여 shortest path를 구한다.



다음과 같은 그래프에서 0에서 3까지의 shortest path를 찾는다고 한다. 우선 s와 dist, prev를 시작 vertex인 0을 기준으로 초기화한다.

s : o x x x x dist : 0 6 13 L L prev : - 0 0 - -

그리고 dist가 가장 작으면서 선택된 적 없는 vertex를 선택한다. 이번에 선택할 vertex는 dist가 6인 vertex 1이다. 그리고 그 vertex에서 연결된 vertex로 이동할 경우 경로가 짧아지는 path를 업데이트한다.

s : o o x x x dist : 0 6 11 12 L prev : - 0 1 1 -

다음으로 vertex 2를 선택하여 업데이트한다.

s : o o o x x dist : 0 6 11 12 15 prev : - 0 1 1 2

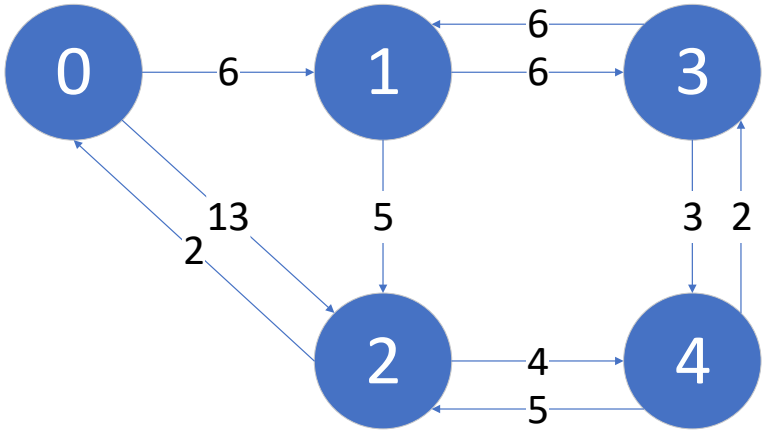
다음으로 vertex 3을 선택하여 업데이트한다.

s : o o o o x dist : 0 6 11 12 15 prev : - 0 1 1 2

3으로 가는 shortest path의 dist는 15이고, path는 prev[3]에서 역추적하여 얻으면 0 1 3 이다.

- Bellman-Ford

  Bellman-Ford는 Source vertex를 기준으로 edge를 하나 사용하여 도달할 수 있는 vertex의 거리로 dist를 업데이트한다. 그리고 각 vertex에서 기존에 구한 거리와, 어떤 vertex를 거치고 그 vertex에 연결된 vertex에 도달하는 거리를 비교하여 가까운 거리로 업데이트한다. 이를 n-2번(초기화 포함) 반복하여 shortest path를 구한다.



다음과 같은 그래프에서 1에서 4까지의 shortest path를 찾는다고 한다. 우선 dist와 prev를 초기화한다.

dist : L 0 5 6 L prev : - - 1 1 -

그리고 각 vertex를 거쳐 도달하는 vertex와 기존의 거리를 비교하여, 더 가까운 거리로 업데이트한다.

dist : 7 0 5 6 9 prev : 2 - 1 1 2

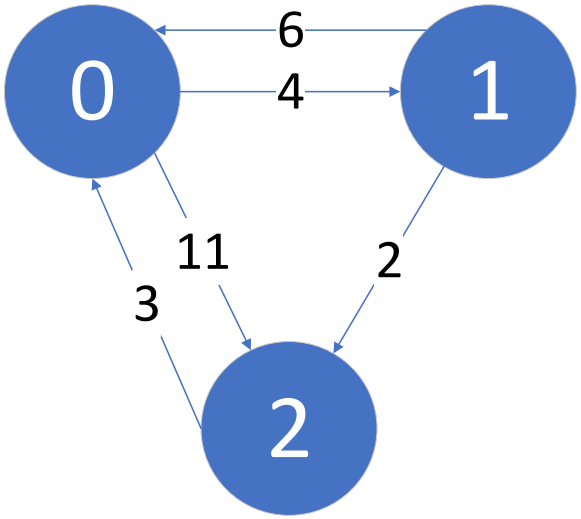
dist : 7 0 5 6 9 prev : 2 - 1 1 2

dist : 7 0 5 6 9 prev : 2 - 1 1 2

한 번만에 shortest path를 구하여 업데이트해도 거리가 변하지 않았다. 1에서 4까지의 dist는 9이며, path는 prev[4]에서 역추적하여 1 2 4 이다.

- FLOYD

  FLOYD는 모든 vertex에서 shortest path를 구하는 알고리즘이다. vertex에서 edge 하나로 도달할 수 있는 거리를 2D Matrix로 구현한다. 그리고 각 vertex가 vertex 0을 지날 수 있을 때, vertex 0, 1을 지날 수 있을 때, … , vertex n-1을 지날 수 있을 때로 점차 통과 가능한 vertex를 늘여가면서 Matrix를 업데이트한다.



다음과 같은 그래프를 FLOYD 알고리즘으로 shortest path를 찾는다. 우선 그래프를 나타내는 Matrix를 구현한다.

0 4 11

6 0 2

3 L 0

그리고 vertex 0을 거쳐서 거리가 짧아지는 vertex를 업데이트한다.

0 4 11

6 0 2

3 7 0

다음으로 vertex 1을 거쳐서 거리가 짧아지는 vertex를 업데이트한다.

0 4 6

6 0 2

3 7 0

다음으로 vertex 2를 거쳐서 거리가 짧아지는 vertex를 업데이트한다.

0 4 6

5 0 2

3 7 0

모든 vertex에서 수행하였으므로 FLOYD를 종료하고, 가장 마지막에 구한 Matrix가 결과 path이다.

4. Result Screen

- LOAD 파일명

  그래프 정보가 담긴 파일을 읽어 그래프를 구축하고 Success를 출력한다.

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

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

- PRINT

  그래프 정보를 출력한다.

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

- DFS startVertex endVertex

  startVertex에서 endVertex까지의 path를 DFS로 찾아 출력한다.

```
====== DFS ======
shortest path: 0 2 4 3
sorted nodes: 0 2 3 4
path lenth: 19
======================

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

- DIJKSTRA startVertex endVertex
  startVertex에서 endVertex까지의 path를 STL set을 사용한 DIJKSTRA로 찾아 출력한다.

```
====== DIJKSTRA ======
shortest path: 0 1 3
sorted nodes: 0 1 3
path lenth: 12
======================

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

- DIJKSTRAMIN startVertex endVertex
  startVertex에서 endVertex까지의 path를 min heap을 사용한 DIJKSTRA로 찾아 출력한다.

```
====== DIJKSTRAMIN ======
shortest path: 0 1 3
sorted nodes: 0 1 3
path lenth: 12
======================

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

- BELLMANFORD startVertex endVertex
  startVertex에서 endVertex까지의 path를 Bellman-Ford로 찾아 출력한다.

```
====== BELLMANFORD ======
shortest path: 1 2 4
sorted nodes: 1 2 4
path lenth: 9
======================

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

- FLOYD
  그래프에 FLOYD 알고리즘을 적용하여 얻은 Matrix를 출력한다.

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

5. Consideration

DFS를 구현할 때 알고리즘을 잘못 이해하여 인접한 vertex 중 가장 작은 vertex를 스택에 넣고 다시 그 vertex에서 인접한 가장 작은 vertex를 스택에 넣는 방식으로 잘못 인식하고 구현하였다. 스택에 모든 edge를 넣는 방식으로 다시 구현하여 바르게 구현하였다

Bellman-Ford 알고리즘을 list로 구현된 graph를 사용했음에도 $O(n^3)$이 나오도록 잘못 구현했다. Vertex와 edge loop의 순서를 바꿔도 같은 결과를 출력할 수 있음을 알고, 이를 반영하여 $O(n^2e)$가 되도록 수정했다.

ferr와 fout이 같은 파일을 열어 출력하면 두 출력값이 겹쳐서 출력되는 것을 발견했다. ferr을 fout과 별개의 파일로 열어야 올바른 출력이 되며, 이러한 경우에 대한 안내가 없어서 fout으로 출력하도록 통일했다.