|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Report**  제목: Shortest Path   |  |  |  | | --- | --- | --- | | **<과제물 제출 전 체크리스트>** | | | | 1 | 이 과제물은 내가/우리가 직접 연구하고 작성한 것이다. | O | | 2 | 인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다. | O | | 3 | 인용한 자료의 표현이나 내용을 왜곡하지 않았다. | O | | 4 | 정확한 출처 제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다. | O | | 5 | 정확한 출처 제시 없이 여러 사람의 글이나 아이디어를 짜깁기하지 않았다. | O | | 6 | 과제물 작성 중 도표나 데이터를 조작(위조 또는 변조)하지 않았다. | O | | 7 | 과제물은 다른 사람으로부터 받거나 구매하여 제출하지 않았다. | O | | 8 | 이 과제물에 실질적으로 참여하지 않은 사람을 공동 제출자로 명기하지 않았다. | O | | 9 | 이 과제물과 동일한 내용을 다른 교과목의 과제물로 제출한 적이 없다. | O |  |  |  | | --- | --- | | 과목명 | IMEN281 | | 담당교수 | 김병인 | | 학 과 | 산업경영공학과 | | 학 번 | 20130220 | | 성 명 | 이다진 | | 제출일 | 2017년 11월 10일 | |

목차

**Ⅰ. 목적1**

**Ⅱ. 방법과 알고리즘1**

1. 전체 구조2

2. 알고리즘2

3. 클래스 구성4

**Ⅲ. 수행 결과에 대한 토의6**

1. 토론6

2. 결론6

**Ⅳ. 실행 화면 스크린샷7**

Ⅰ. 목적

본 프로그램은 C++로 텍스트 파일의 데이터를 읽고 알맞은 클래스에 저장한 후 Dijkstra 알고리즘과 A\* 알고리즘을 이용하여 “최단 경로”를 찾는다. 프로그램 구현을 통해 C++을 통한 파일 입출력, 클래스 구성, 데이터 구조를 활용법을 익히고 Dijkstra 알고리즘과 A\* 알고리즘을 이해한다. cmd에서 프로그램 실행을 통해 Text User Interface 환경에 익숙해진다. 이러한 과정을 통해 컴퓨터로 정보를 처리하고 문제의 최적해를 찾는 기술을 익힌다.

Ⅱ. 방법과 알고리즘

1. 전체 구조

(1) 노드 파일을 읽어 노드 객체를 컨테이너에 담는다.

(2) 엣지 파일을 읽어 엣지 객체와 노드 객체를 함께 컨테이너에 담아 그래프를 형성한다.

(3) 메인 함수를 실행한다.

(4) 1 입력: Dijkstra 알고리즘을 이용하여 최단 경로를 찾아 결과를 텍스트 파일로 출력한다.

(5) 2 입력: A\* 알고리즘을 이용하여 최단 경로를 찾아 결과를 텍스트 파일로 출력한다.

(6) 메인 프로그램을 종료한다.

2. 알고리즘

알고리즘은 전체 구조의 순서대로 작성되었다.

(1) 노드 파일을 읽어 노드 객체를 컨테이너에 담는다.

read\_nodefile {

① 노드 파일을 오픈하고 실패 시 예외처리 하고 종료한다.

② 첫번째 줄을 제외한 나머지 줄에 대하여 ‘\t’을 구분자로 하여 nodeID, x좌표, y좌표를 저장한다.

③ 저장한 값으로 노드 객체를 생성하고 컨테이너 nodes에 담는다.

④ 파일의 끝에 도달하면 종료한다.

}

(2) 엣지 파일을 읽어 엣지 객체와 노드 객체를 함께 컨테이너에 담아 그래프를 형성한다.

read\_edgefile {

① 엣지 파일을 오픈하고 실패 시 예외처리 하고 종료한다.

② 첫번째 줄을 제외한 나머지 줄에 대하여 ‘\t’을 구분자로 하여 edgeID, start\_nodeID, end\_nodeID를 저장한다.

③ 엣지의 거리를 소수점 둘째짜리까지 계산하여 저장한다.

④ 엣지 객체를 생성하여 key가 start\_nodeID인 엣지 컨테이너의 value로 추가한다.

⑤ two-way이므로 동일한 엣지 객체를 key가 end\_nodeID인 엣지 컨테이너의 value로 추가한다.

⑥ 파일의 끝에 도달하면 종료한다.

}

(3) 메인 함수를 실행한다.

try {

①-1. argv[5]이 1인 경우

파일을 읽고 Dijkstra 알고리즘으로 최단 경로를 구하는 메소드를 호출한다.

①-2. argv[5]이 2인 경우

파일을 읽고 A\* 알고리즘으로 최단 경로를 구하는 메소드를 호출한다.

② 최단 경로와 경과 시간을 출력하는 메소드를 호출한다.

}

catch {

③ throw 값에 해당하는 예외 처리를 실행한다.

}

(4) Dijkstra 알고리즘(A\*알고리즘)을 이용하여 최단 경로를 찾는다.

>> STEP 1

① start\_node를 permanentNodes 컨테이너에 담는다.

② start\_node와 인접한 노드 i에 대하여

s\_dis에 두 노드 사이의 거리를 저장하고 preNode로 start\_node를 저장한다.

③ i를 newReachableNodes 컨테이너에 담는다.

>> STEP 2

① newReachableNodes 컨테이너에서 s\_dis가 가장 작은 node를 삭제한다.

② 해당 노드를 permanentNodes 컨테이너에 담는다.

>> STEP 3

① STEP 2에서 저장된 permanent\_node가 end\_node와 같다면 메소드를 종료한다.

>> STEP 4

① STEP 2에서 저장된 permanent\_node와 인접하고 permanentNodes에 들어있지 않은 노드 j에 대하여 s\_dis[j] = min(s\_dis[j], s\_dis[i]+distance[i][j]) 로 갱신한다.

② 노드 j의 preNode로 permanent\_node를 저장한다.

③ 노드 j가 newReachableNodes 컨테이너에 들어있지 않다면 추가한다.

exception 1. nodes에 start\_nodeID가 없는 경우 예외 처리 후 메소드 종료

exception 2. path가 없는 경우 예외 처리 후 메소드 종료

A\*알고리즘은 아래의 부분을 제외하면 Dijkstra 알고리즘과 동일하다.

>> STEP 2-①

newReachableNodes 컨테이너에서 s\_dis + end\_node까지의 거리가 가장 작은 node를 삭제한다.

(5) 결과를 텍스트 파일로 출력한다.

① temp\_node가 end\_node일 때부터 temp\_node의 preNode가 없을 때까지 {}를 반복한다

{ temp\_node의 preNode를 path 컨테이너에 담는다.

temp\_node를 temp\_node의 preNode로 갱신한다 }

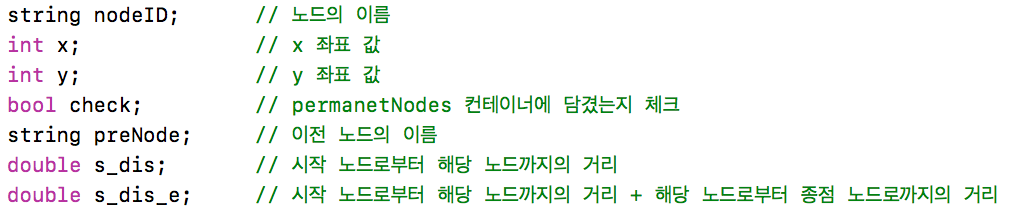
② path의 순서를 뒤집는다.

③ path의 노드의 x좌표, y좌표, start\_node로부터의 거리를 출력한다.

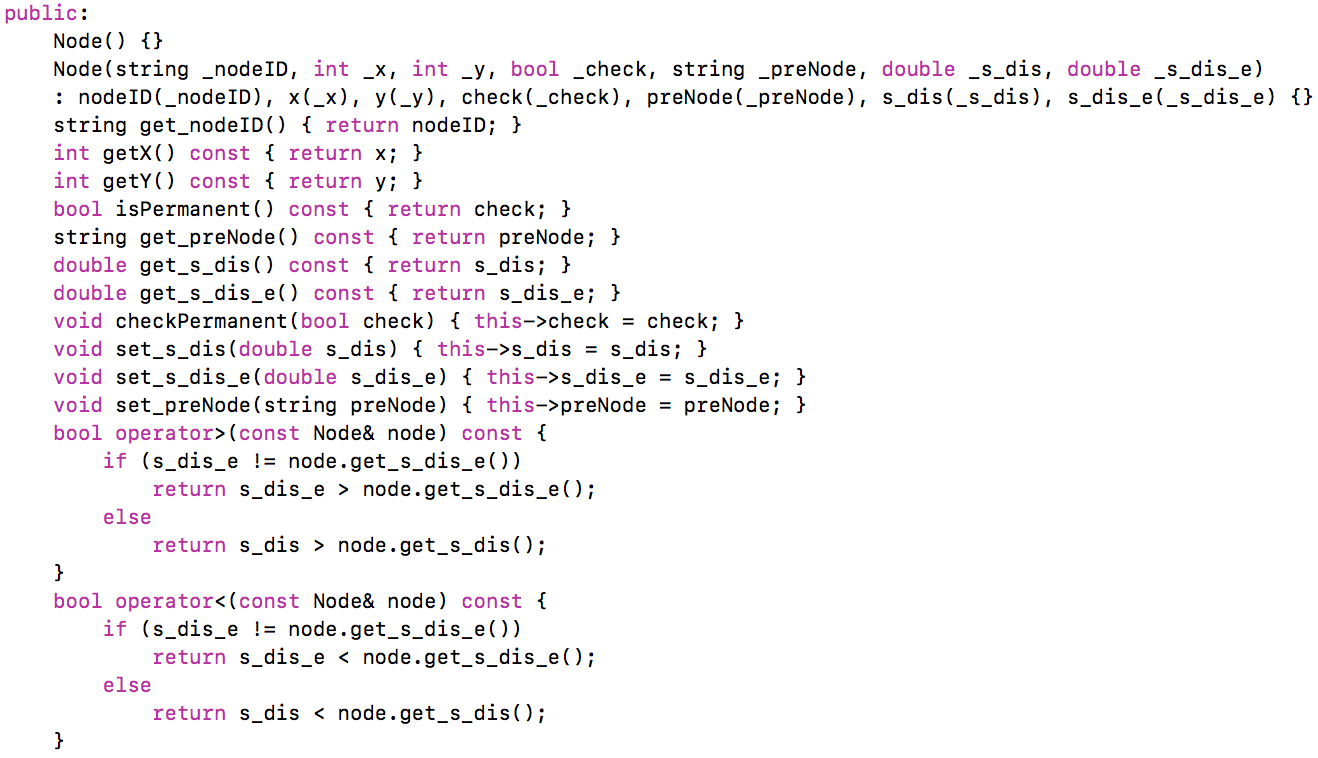
3. 클래스 구성

(1) class Node

- 멤버 변수

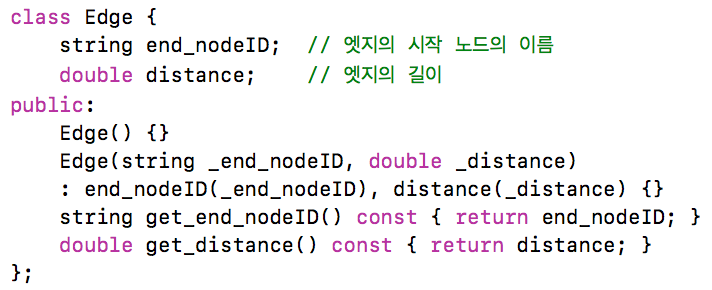
****

- 메소드



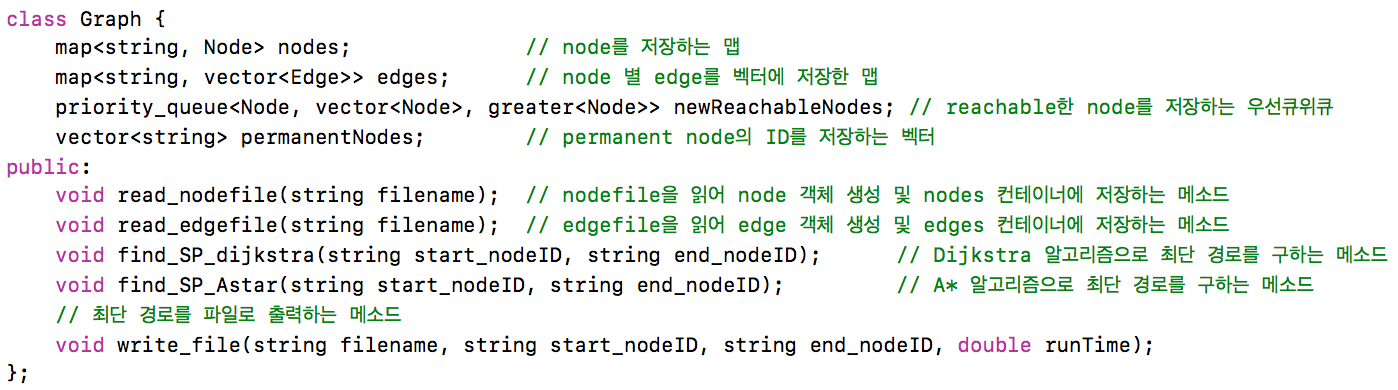
(설명) Node는 노드 하나에 대한 정보 및 메소드를 담고 있는 클래스이다. get~, set~ 메소드는 private한 멤버 변수에 접근 및 값 변경을 위한 메소드이다. 연산자 ‘>’, 연산자 ‘<’에 대한 오버로딩은 newReachableNodes 컨테이너가 우선순위큐이므로 remove 메소드를 사용하기 위해 사용하였다. Dijkstra 알고리즘에서는 시작 노드로부터 해당 노드까지의 거리(s\_dis)에 대하여, A\* 알고리즘에서는 시작 노드로부터 해당 노드까지의 거리 + 해당 노드로부터 종점 노드까지의 거리(s\_dis\_e)에 대하여 크기를 비교하였다.

(2) class Edge



(설명) Edge는 엣지 하나에 대한 정보 및 메소드를 담고 있는 클래스이다. get~ 메소드는 private한 멤버 변수에 접근하기 위한 메소드이다.

(3) class Graph



(설명) Graph는 노드 집합과 엣지 집합으로 이루어진 그래프이다. 메소드에 대한 자세한 설명은 위의 <Ⅱ.2.알고리즘>에 서술되어있다.

- 멤버 변수

**nodes와 edges** : 모든 타입의 nodeID, edgeID를 저장할 수 있도록 string 형을 key로 하고, 해당 객체를 value로 갖는 map이다.

**newReachableNodes** : s\_dis(또는 s\_dis\_e)가 가장 작은 원소를 간단히 제거할 수 있는 우선순위큐이다.

**permanentNodes** : 단순히 추가되는 원소를 담는 벡터이다.

Ⅲ. 수행 결과에 대한 토의

1. 토론

프로그램 개발 과정 중 발생한 문제와 해결 방법에 대해 서술한다.

1) 노드와 엣지의 컨테이너 선택

노드와 엣지의 이름을 index화하여 map이 아닌 vector에 담고자 했다. 하지만 직관적이지않고 이름에 규칙성이 없는 경우 로직 에러가 발생할 수 있다고 판단하여 map를 선택했다. map은 key를 기준으로 자동 정렬해주기 때문에 텍스트 파일을 분석하여 map 정렬 기준(오름차순, 내림차순)을 선택하면 효율적일 것이다. 본 과제에서는 노드와 엣지의 수가 작아서 정렬 기준을 고려하지 않았다.

2) newReachableNodes 컨테이너 선택

Heap으로 구현한 우선순위큐를 사용하고자 했다. 하지만 Heap, Complete Binary Tree를 능숙히 다루지 못하여 vector로 구현한 기본 우선순위큐를 사용했다.

2. 결론

본 프로그램 작성을 통해 C++의 문법을 익히는 것뿐만 아니라 알고리즘을 이해하고 코드화하는 연습을 할 수 있었다. 특히 같은 알고리즘이라도 간결하고 효율적으로 코드를 작성해야 한다는 것을 알게 되었다. 또한 문제를 분석하여 가장 적절한 STL을 사용하기 위해 STL을 제대로 이해해야 한다는 것을 알게 되었다. 앞으로도 연습을 통해 좋은 알고리즘을 효율적으로 구현하는 데에 능숙해지길 기대한다.

**과제에 소요된 시간**

알고리즘 및 STL 공부에 3시간

코드 작성 및 수정에 4시간

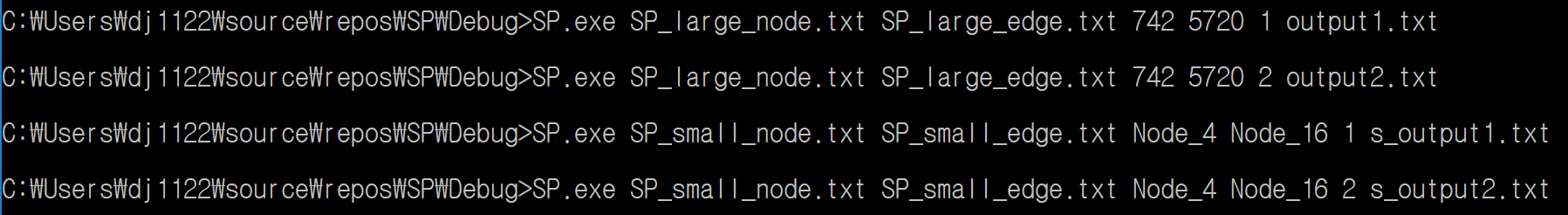
Mac에서 작성한 코드를 Window용으로 수정하고 확인에 30분

보고서 작성에 2시간

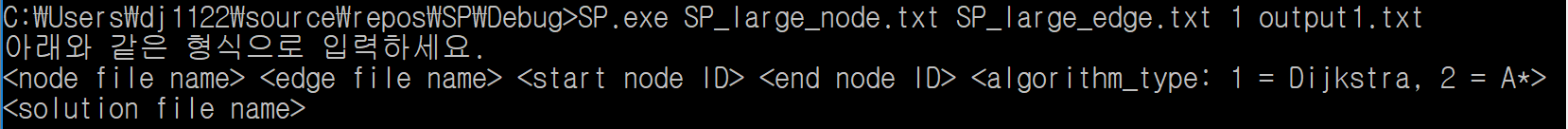
**총 9시간 30분 소요**

Ⅳ. 실행 화면 스크린샷

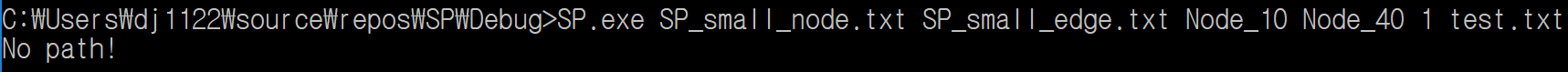
1. 최단 결로 찾기 성공



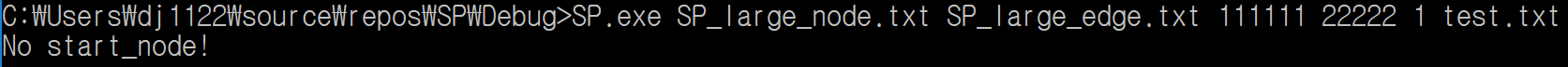
2. 잘못된 명령 인자 전달 경우



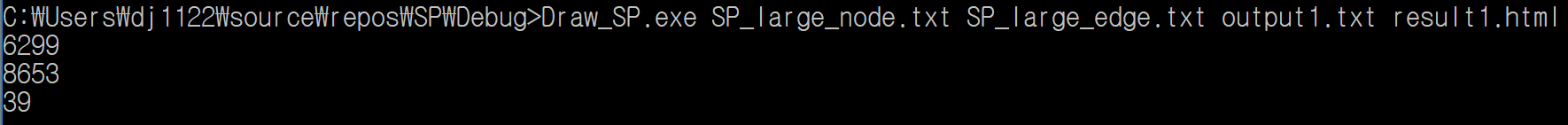
3. 경로가 없는 경우



4. 노드가 없는 경우



5. 찾은 최단 경로로 Draw.exe 실행



6. Draw.exe의 그림

