**REPORT**

**| 알고리즘 설계과제**

**| 내비게이션 프로그램 설계**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

**| kin16742@naver.com**

# **개요**

**설계의 목적**

2019 - 1학기 알고리즘 과목 설계 과제는 내비게이션 프로그램 설계로 이 프로그램은 출발지에서 목적지로 가는 경로 중 침수된 지역을 피하여 안전하고 가장 빠른 경로를 안내하는 프로그램입니다. 이 프로그램을 설계하는 목적은 인접 리스트 기반 그래프 자료구조를 생성하고 관리하는 것과 Dijkstra 알고리즘을 이용하는 것입니다.

**요구사항**

다음의 세 가지 기능을 수행해야 합니다.

1. **데이터 입력**

지역 정보, 지역 간 도로 정보를 입력 받아 인접리스트 기반 무향 그래프로 저장

1. **최단거리 찾기**

출발, 도착 지역을 입력 받아 최단거리를 찾는다

1. **최단경로 찾기**

출발, 도착 지역을 입력 받아 최단경로를 찾는다.

**개발환경**

운영체제 – Windows 10, x64

개발 언어 – C++

사용 프로그램 – Visual Studio 2017

# **필요한 자료구조 및 기능**

**필요한 자료구조**

1. **지역 정보**

* **지역 정보 구조체**

struct loc {

int locNum; 지역번호 : 100,000 ~ 999,999 사이의 정수 (기준 키, 유일함)

char locName[20]; 지역이름 : 공백 없는 20Byte 이내의 문자열

bool flood; 침수여부 : 1 또는 0. “1” (침수된 경우), “0” (침수되지 않은 경우)

bool operator < (loc l) { 지역 번호 순으로 정렬하기 위한 연산자 오버로딩

return this->locNum < l.locNum;

}

};

* **지역 정보를 저장할 배열**

loc location[100005]; 지역 정보가 저장되는 배열

int locIdx[1000005]; 지역 정보의 지역 번호를 지역 정보 배열 내의 인덱스와 매핑해 저장

* location[1]의 지역 번호가 123456이라면 locIdx[123456] = 1

1. **그래프**

* **인접 리스트 기반 무향 그래프**

vector<vector<pair<int, int>>> navi;

2차원 벡터로 인접 리스트 기반 무향 그래프 구현

* navi[i].push\_back({ j, w }); : i번 정점에 j번 정점으로 향하는 가중치가 w인 간선을 그래프에 추가

1. **Dijkstra**

* **각 set에 들어갈 edge 정보 구조체**

struct edge {

int to; 도착점의 정점 인덱스

int from; 출발점의 정점 인덱스

int weight; 간선의 가중치

};

* **Tree set**

vector<edge> tree;

* **Fringe set**

priority\_queue<pair<int, pair<int, int>>> fringe;

* Fringe set을 priority queue로 구현, top은 가중치가 가장 적은, 가중치가 같다면 도착점의 인덱스가 가장 낮은 간선
* fringe.push({ w, { d, s } }); : 도착점이 d, 출발점이 s, 가중치가 w인 edge를 fringe set에 추가
* **그 외**

bool inTree[100005]; “1” ( Tree set 내에 있음 ) “0” ( Tree set 내에 없음 )

int dist[100005]; edge가 Tree set에 추가될 때 해당 edge의 도착점으로의 최단 경로가 저장되는 배열

**기능 상세**

1. **데이터 입력**
2. **지역 정보 입력**

n개의 지역 정보를 입력 받는다.

1. **지역 번호 순 정렬**

지역 정보가 저장된 배열을 지역 번호 순으로 정렬한다.

1. **지역 번호와 인덱스 매핑**

각 지역 번호와 지역 정보가 저장된 배열의 인덱스를 매핑한다.

1. **도로 정보 입력 ( 그래프 )**

m개의 도로 정보를 입력 받아 그래프에 추가한다.

( 무향 그래프이므로 양쪽 정점에 모두 추가 )

1. **최단거리 찾기**
2. **Dijkstra**

출발 지역에서 도착 지역까지의 경로를 계산한다.

1. **출력**

“N W SN DN” or “None”

N : 출발지부터 목적지까지 Tree vertex들의 set에 추가된 지역들의 수

W : 출발지부터 목적지까지의 최단거리

SN : 출발 지역이름

DN : 목적 지역이름

목적지가 침수되어 있거나 최단거리가 10^6을 초과하는 경우, “None”을 출력한다.

1. **최단경로 찾기**
2. **Dijkstra**

출발 지역에서 도착 지역까지의 경로를 계산한다.

1. **최단 경로 탐색**

Tree set의 간선 정보를 통해 최단 경로를 찾아 나간다.

1. **출력**

“N V0 V1 …. Vk” or “None”

N : 출발지부터 목적지까지 Tree vertex들의 set에 추가된 지역들의 수

V0 V1 …. Vk : 출발지부터 목적지까지의 최단경로 ( V0 : 출발지, Vk : 목적지 )

목적지가 침수되어 있거나 최단거리가 10^6을 초과하는 경우, “None”을 출력한다.

1. **제한사항**
2. 프로그램의 한 입력에 대해 정점 정보는 최대 10^5개, 간선 정보는 최대 2×10^5개, 질의는 최대 50개가 입력된다
3. 채점서버의 입력들에 대해 총 2초의 제한시간 이내에 수행되어야 한다.
4. 제시한 입출력 형식대로 표준 입출력을 사용하여 처리한다.
5. 문제에서 설명되지 않은 예외처리를 해야 할 질의는 입력되지 않는다**.**

# **기능별 알고리즘 명세**

**알고리즘 설명**

**Dijkstra**

Dijkstra 알고리즘은 가중치 그래프에서의 최단 경로와 거리를 계산하는 알고리즘입니다. 각 간선은 다음 3개의 속성 중 하나를 가집니다.

* + Tree : 최단 경로인 간선
  + Fringe : 최단 경로 후보 간선
  + Unseen : 아직 확인하지 않은 간선

알고리즘을 수행하기 위해서는 출발 정점을 설정해 주어야 합니다. 우선 출발 정점이 Tree에 추가되고, 출발 정점의 degree edge가 Fringe에 추가됩니다. 그 다음부터는 Fringe 상태인 간선 중 최소 가중치를 갖는 간선을 Tree에 추가하고, 추가된 간선의 도착점까지의 최단 거리를 갱신한 뒤 도착점의 degree edge를 Fringe에 추가하는 것을 반복합니다.

Dijkstra 알고리즘은 Fringe 상태인 최소 가중치 간선을 사용한다는 점과 갱신된 최단 거리로 그 다음에 추가되는 정점의 최단 거리를 구한다는 점에서 볼 때 그리디 테크닉이면 다이나믹 프로그래밍이라고 볼 수 있습니다.

최악의 경우, A에서 B까지의 최단 경로를 구하기 위해 그래프 내의 모든 정점을 확인해야만 하는 상황이 발생할 수 있습니다.

**기능별 시간 복잡도**

1. **데이터 입력**
2. **지역 정보 입력**

지역 정보 입력 : O(1)

n번 입력하므로 **O(n)**

1. **지역 번호 순 정렬**

STL 함수를 이용해 지역 번호를 기준으로 지역 정보 정렬 **O(nlogn)**

1. **지역 번호와 인덱스 매핑**

지역 정보 배열 내 지역 번호와 그 인덱스를 매핑

n개의 원소를 확인해 매핑하므로 **O(n)**

1. **도로 정보 입력 ( 그래프 )**

도로 정보 입력 : O(1)

그래프의 양쪽 정점에 간선 추가 : O(1)

m번 입력하므로 **O(m)**

1. **최단거리 찾기**
2. **Dijkstra**

**사용할 변수 초기화**

* + - Tree set 초기화 **O(n)**
    - Fringe set 초기화 **O(m)**
    - inTree 배열 초기화 **O(n)**
    - dist 배열 초기화 **O(n)**

**Dijkstra 알고리즘 실행**

1. 출발 정점 S 추가 O(1)
2. S 정점의 degree edge들 Fringe set에 추가 O(deg(S))
3. Fringe set에서 min값 추출 O(logm)

Fringe set에 들어갈 수 있는 원소 최대 개수는 간선의 개수와 같음

1. min값을 갖는 edge를 Tree에 추가 O(1)
2. 목적지 정점이 Tree에 추가되면 중지 O(1)
3. edge의 도착 정점 P의 degree edge들 Fringe set에 추가 O(deg(P))
4. 3 ~ 6을 모든 정점의 최단 경로를 구할 때 까지 반복 최대 m회

모든 정점의 최단 경로이므로 최대 반복 횟수는 간선의 개수와 같음

모든 정점의 degree edge를 Fringe set에 삽입하는 연산 수의 총합 = 간선의 개수

O(m)

그래프에서 간선의 개수 m은 정점 개수의 제곱 n^2보다 작다

m < n^2 => O(logm) = O(logn)

**∴ Dijkstra 알고리즘 시간 복잡도 = O(mlogm + m) = O(mlogm) =** **O(mlogn)**

1. **출력**

Tree set 크기, 최단거리, 지역 번호를 단순히 전역 변수를 참조해 출력 **O(1)**

1. **최단경로 찾기**
2. **Dijkstra**

2. - (1) 와 동일

1. **최단 경로 탐색**

목적지 정점을 Tree set에서 탐색 O(n)

Tree set 내의 정보를 Stack에 저장 후 이전 정점으로 이동 O(1)

최단 경로는 최악의 경우 모든 정점을 지나므로 n회 반복 -> **O(n^2)**

1. **출력**

Tree set 크기 출력 O(1)

Stack내 정보 출력 **O(n)**

# **평가 및 개선 방향**

**구현한 알고리즘의 장점**

1. 목적지 정점을 만났을 때 Dijkstra 알고리즘 진행을 멈추므로, 최악의 경우를 제외하고 빠른 수행 시간을 보여줄 것으로 예상됩니다.
2. 인접 리스트 기반 그래프로 구현하였으므로 인접 행렬 기반 그래프로 구현했을 때보다 메모리 사용량이 적고 실행 시간도 빠릅니다.
3. 100000 ~ 999999 사이의 유일한 지역 번호를 배열에서의 인덱스와 매핑해 따로 저장해 지역 번호로 지역 정보를 받아오는데 O(1) 시간이 걸립니다.
4. Fringe set에서 최소 가중치 간선을 찾아 사용하기 때문에 모든 정점으로의 최단 경로가 갱신되기까지 탐색할 범위가 적습니다.
5. Fringe set을 priority queue로 구현했기 때문에 탐색 시간이 빠릅니다.

**구현한 알고리즘의 단점**

1. 간선의 가중치가 음수일 때 사용할 수 없습니다.
2. 최악의 경우, 어떤 정점으로의 최단 경로를 탐색할 때 모든 정점에 대해 알고리즘을 수행해야 할 수 있습니다.
3. Dijkstra 알고리즘이 끝난 뒤, 최단 경로가 전체 정점을 포함하고 있을 때 출력하는 부분에서 시간이 오래 걸릴 수 있습니다.
4. 질의를 응답받음에 따라 최단 경로를 계산하므로 비효율적일 수 있습니다.

**향후 개선 방향**

위에 적어 둔 단점을 보완하는 방향으로 개선 방향을 생각해 보았습니다.

1. 간선의 가중치가 음수일 때, Floyd-warshall 알고리즘을 사용하면 되지만, 문제의 제한사항 때문에 이는 불가능할 것으로 보입니다.
2. 1과 동일합니다
3. 최단 경로를 갱신할 때, 최단 거리가 결정된 도착점의 이전 정점을 따로 계산하는 방법으로 개선될 수 있을 것으로 보입니다.
4. 각 질의를 받은 뒤 Single-source Shortest Path를 계산하는 것이 아닌 All-pair Shortest Path를 미리 계산해 두고 질의에 따른 답을 빠르게 출력하는 방법을 생각했으나, 이 또한 제한사항 때문에 불가능할 것으로 보입니다.

# **기타**

**결과**

