**Programming Report #1**

**Shortest Path(최단경로)**

과 목 : 컴퓨터 알고리즘

담당교수 : 이상호

학 과 : 컴퓨터공학과

학 번 : 1605020

이 름 : 박소현

제출일자 : 2019년 4월 11일

**1. 문제 설명**

최단 경로 문제는 가장 짧은 경로에서 두 정점을 찾는 문제로, 가중치(weight) 그래프를 이용하여 경로의 가중치 합이 최소가 되도록 하는 경로를 찾는 문제이다. 예를 들면, 어느 도시에서 다른 도시로 이동할 때 가장 적은 시간이 걸리는 길을 찾는 것과 비슷한 문제이다. 이 때, 각 도로 구간에서 걸리는 시간을 가중치라 할 수 있다. 최단 경로 문제는 크게 네 가지로 분류할 수 있다.

1. 하나의 정점에서 다른 하나의 정점까지의 최단 경로를 구하는 문제

2. 하나의 정점에서 다른 모든 정점까지의 최단 경로를 구하는 문제

3. 하나의 목적지로 가는 모든 최단 경로를 구하는 문제

4. 모든 최단 경로를 구하는 문제

위의 네 가지 경우 모두 음의 가중치는 고려하지 않는다고 가정했을 때, Floyd-Warshall 알고리즘을 이용하여 해결하는 4번을 제외한 1번부터 3번까지는 모두 Dijkstra 알고리즘을 이용하여 해결했을 때 optimal solution이 제공된다. 이번 프로그래밍 문제도 2번에 해당하는 문제이므로 Dijkstra 알고리즘을 이용하여 해결하도록 하겠다.

**2. 입출력 예시**

(1) 입력 자료 형식의 예

|  |
| --- |
| 5 1 //첫 번째 줄 node = 5(정점의 개수), start = 1(시작 정점)  0 4 2147483647 25 12 // V1 정점의 가중치  4 0 32 11 1 // V2 정점의 가중치  2147483647 32 0 100 7 // V3 정점의 가중치  25 11 100 0 60 // V4 정점의 가중치  12 1 7 60 0 // V5 정점의 가중치 |

(2) 출력 형식의 예

|  |
| --- |
| 입력파일 : graph1.txt //입력된 파일의 이름  정점 1(으)로부터 각 정점까지의 최단경로 = (0, 4, 12, 15, 5) |

**3. 문제풀이 방법(알고리즘)**

먼저 입력 파일의 첫 번째 줄을 입력 받아온다. 첫 번째 값은 정점의 개수, 그리고 두 번째 값은 시작 정점을 나타내므로 각각의 값을 node 그리고 start 변수에 저장시켜준다. 이 두 변수는 향후 필요한 변수들의 값을 미리 초기화 해주는 함수인 initialize 함수, 그리고 Dijkstra 알고리즘을 실제로 구현한 dijkstra 함수를 호출할 때 사용된다. 이 때, start 변수는 -1을 해서 배열의 index 값으로 사용될 수 있도록 해준다.

Initialize 함수를 호출해서 graph 배열, vertex 배열, 그리고 length 배열을 초기화한다. Graph 배열은 for문을 두 번 반복하며 배열의 원소 한 개씩 입력 파일로부터 읽어온다. Vertex 배열은 정점이 선택 되었는지에 대해 판별하기 위해 필요한 변수로, 먼저 정점의 개수만큼 모두 0으로 초기화를 해준다. 즉, vertex 배열에서 특정 정점의 값이 0일 경우 아직 선택되지 않았음을 의미한다. 마지막으로, length 배열에는 graph 배열에서 시작 정점의 행을 복사해서 넣어준다. Length 배열은 나중에 Dijkstra 함수에서 어떻게 이용되는지 자세하게 설명하도록 하겠다.

필요한 변수들의 값이 초기화가 되었으므로 Dijkstra 알고리즘을 구현해놓은 dijkstra 함수를 호출해준다. 정점의 개수(node)와, 새로 선택되는 정점(start), 그리고 이전에 선택되었었던 정점(old)을 호출 인자로 받아오게 되는데, main 함수에서 처음 호출할 때는 이전에 선택되었던 정점이 없으므로 이전에 선택되었었던 정점을 받는 자리에 NULL을 지정해준다. 선택된 정점의 vertex 값이 1일 경우에는 고려하지 않으므로 if문을 이용하여 vertex의 값이 0이어야 함을 조건으로 둔다. 해당 정점의 vertex일 경우 if문이 실행되고 vertex의 값은 1로 바꿔준다. 정점의 개수만큼 for문을 반복시키며 이전에 선택된 가중치(sum)와 새로 지정된 정점에서의 가중치(graph[start][i])의 합이 이전에 선택되었던 정점에서의 가중치(length[i])보다 작을 경우 length[i]에 더 작은 값으로 업데이트해준다. 이 length 배열은 이처럼 선택되는 정점들을 돌며 최소 가중치들을 저장하게 되므로, 해당 배열이 결국 최종 답이 되게 된다. 다음, for문을 돌며 length 배열에서 제일 작은 값을 찾아준 뒤 min 변수에 저장해주고, 이 값으로 sum도 새로 업데이트 해준다. 최소값이 선택되었을 때의 i 값을 newnode 변수에 넣어주면, 이 newnode가 새로 선택될 정점이 된다. Dijkstra 함수가 호출될 때마다 index(처음에는 0으로 초기화) 값을 1씩 증가시켜주고 index 값이 정점의 개수-1 이 될 때까지 dijkstra 함수를 호출하여 모든 정점이 한 번씩 선택될 수 있도록 한다.

Dijkstra의 재귀 호출이 완료되면 다시 main 함수로 돌아오게 된다. Length 안에 저장되어 있는 값들을 출력해주면 Dijkstra 알고리즘으로 최단 경로 구하기가 완료된다.

**4. 소스 코드**

//1605020 박소현

//Dijkstra algorithm 구현

#include <stdio.h>

#include <string.h>

#define FILE\_NAME "graph1.txt" //불러올 파일의 이름을 이곳에 작성해주세요!

#define MAX 1000

#define SIZE 100

#define MAX\_INT 2147483647

int graph[SIZE][SIZE]; //입력 파일에서 받아오는 값을 저장

int vertex[SIZE]; //선택된 노드와 선택되지 않은 노드에 대한 구분을 해주는 변수

int length[SIZE]; //정점에서 다른 정점까지의 최단 경로를 저장

int index = 0; //dijkstra 함수 호출 횟수를 조절

int sum = 0;

FILE \*fp;

//필요한 변수들의 값을 초기화해주는 함수

void initialize(int node, int start) {

//입력 파일에서 둘째 줄부터 끝 줄까지 읽어서 graph배열에 값 저장

for (int i = 0; i < node; i++)

for (int j = 0; j < node; j++)

fscanf(fp, "%d", &graph[i][j]);

//노드의 개수만큼 반복하며 미리 0으로 초기화(0 : 아직 선택되지 않은 상태)

for (int i = 0; i < node; i++)

vertex[i] = 0;

//시작 정점의 행을 따로 다른 변수에 저장

for (int i = 0; i < node; i++)

length[i] = graph[start][i];

}

//실제 다익스트라 알고리즘의 구현

void dijkstra(int node, int start, int old) {

int min = MAX\_INT;

int newnode; //선택되는 노드가 저장되는 변수

//해당 노드가 0일 때

if (vertex[start] == 0) {

vertex[start] = 1; //해당 노트가 선택되었으므로 1로 바꿔준다.

for (int i = 0; i < node; i++) {

//이전에 선택된 가중치와 새로 지정된 노드에서 특정 경로의 합이 이전 노드에서의 경로보다 작을 경우

if ((sum + graph[start][i] < length[i]) && vertex[i] == 0 && graph[start][i] != MAX\_INT)

length[i] = graph[start][i] + sum;

else if (graph[start][i] == MAX\_INT)

continue;

}

for (int i = 0; i < node; i++) {

if( min > length[i]) {

if (vertex[i] == 1) //선택된 노드는 건너뛴다.

continue;

else {

min = length[i]; //최소 경로를 min에 저장

sum = min;

newnode = i; //선택된 노드의 값을 newnode에 저장

}

}

}

//index의 값이 노드의 개수-1 보다 크면 더 이상 재귀 호출을 하지 않음.

if(++index < node - 1)

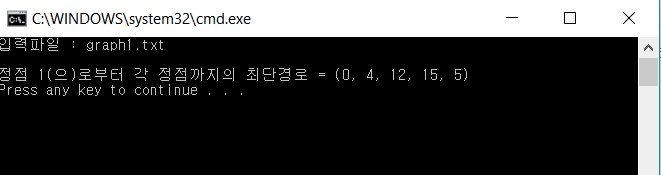
dijkstra(node, newnode, start); //선택된 노드에 대해서 함수 재귀 호출

}

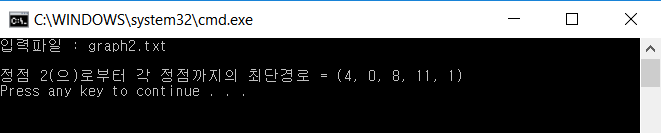
}

**5. 수행 결과**

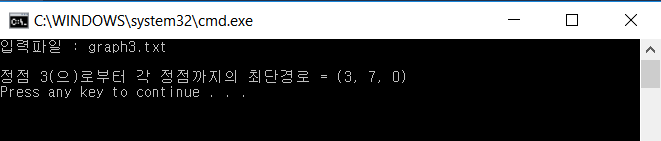
(1) graph1.txt



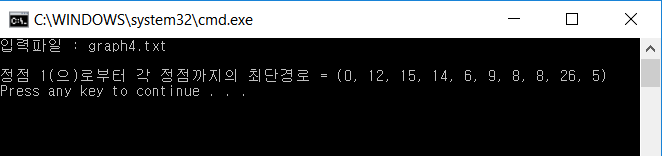
(2) graph2.txt



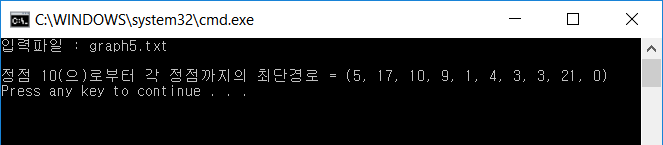
(3) graph3.txt



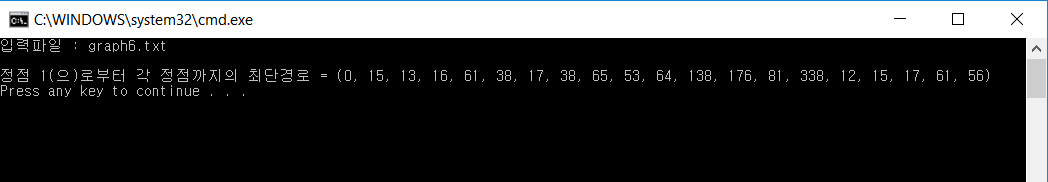
(4) graph4.txt



(5) graph5.txt

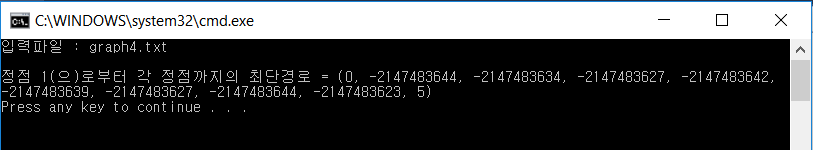


(6) graph6.txt

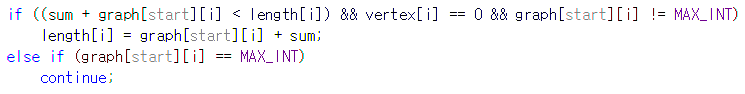


**6. 결과 분석 및 토의**

해당 문제에 대한 코드를 작성하면서 몇 개의 난관이 있었다. 처음에 실행했을 때에는 graph1.txt와 graph3.txt 입력 파일에 대해서는 결과가 잘 나왔지만, 나머지 입력 파일에 대해서는 런타임 에러가 발생했다. 이것저것 다시 수정해 보다가 다시 실행시켜 보았을 때는 이러한 결과가 나왔다.



결국, overflow 때문에 에러가 발생하고 있었다는 것을 알게 되었다. 가중치가 MAX\_INT일 때 해당 가중치에 sum의 값이 더해지면서 overflow가 발생하고 있었다. Overflow가 발생하게 되면 음수 가중치가 되는데 dijkstra 알고리즘에서는 음수 가중치일 경우를 처리해주는 코드가 없어서 에러가 났던 것이다. 그래서, 이 부분에 대해서 if문에 조건을 추가해 주었다.



가중치의 값이 MAX\_INT가 아닐 때에만 sum의 값을 더해주는 것으로 수정해주었다.

작성한 코드에 대한 시간 복잡도를 구해보겠다. 위의 함수에서 제일 시간이 오래 걸리는 연산은 dijkstra 함수의 첫 번째 for문 안에 있는 덧셈 연산이다. Dijkstra 함수가 한 번 실행될 때마다 덧셈 연산이 정점의 개수만큼 실행되고, Dijkstra 함수가 정점의 개수-1만큼 호출된다. 정점의 개수를 n이라고 두면, n(n2-1)이 되므로 결국 시간 복잡도는 O(n2)이 된다. 변수들을 생성할 때 최대 크기로 지정해서 사용하므로 공간의 효율성은 조금 떨어진다.