**Computer Algorithm**

**Chapter 09**

**Path**

**[실습] Shortest Path 구현**

**실습 목표**

* 그래프에서 최단경로알고리즘을 이해한다.
* Graph를 인접행렬로 구현하고 주어진 해당 그래프의 노드간의 최단 경로를 구하는 Dijkstra알고리즘을 구현한다.

**요구사항**

* **실습과제) (100점)**

그래프를 인접행렬로 변환하여 정점 간 최단경로 알고리즘(Dijkstra알고리즘)를 구현한다.

1. 출발 정점은 1로 한다.
2. 그래프와 알고리즘은 하단의 “배경지식”을 참고한다.
3. 프로그램의 출력은 아래의 실행예를 참고한다.

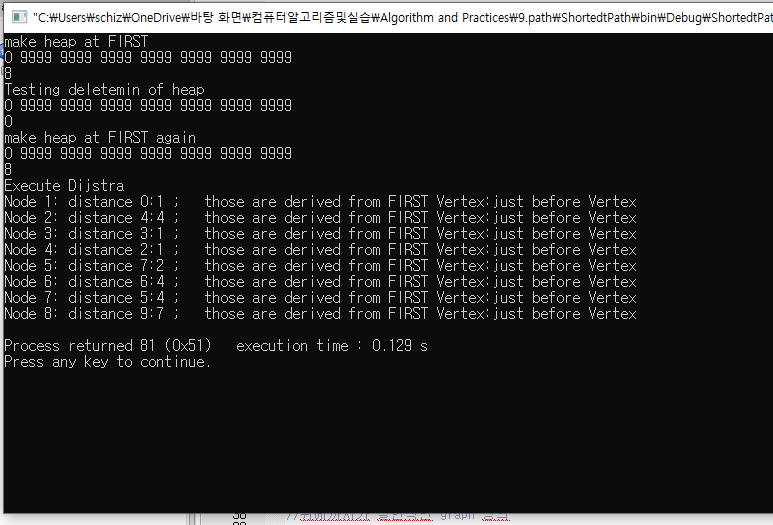
(실행 예)

Node 1: distance 0: 0 🡨 노드번호, 거리, 앞노드번호를 차례로 출력한다.

Node 2: distance 4: 4

Node 3: distance 1: 3

Node 4: distance 1: 2

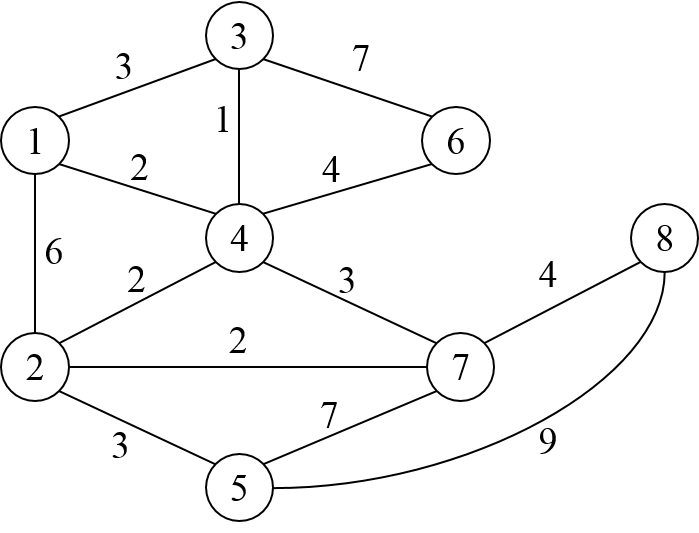


**배경지식**

Shortest Path문제는 두 정점 u와 v를 연결하는 경로 중 간선들의 가중치 합이 최소가 되는 경로를 찾는 문제이다. 간선들의 가중치는 경우이 따라 비용, 거리, 시간 등으로 해석될 수 있으며 이 문제를 해결하는 알고리즘 중에 Dykstra 알고리즘이 가장 널리 알려져 있다. Dykstra 알고리즘은 Cycle이 존재하지 않는 그래프에서 가중치가 모두 0이상의 값을 가지는 경우에 Shortest Path문제를 해결하는 데 있어 가장 좋은 효율을 보인다고 알려져 있다.

본 실습에서는 Dykstra알고리즘을 사용하여 Shortest Path문제를 해결하는 법을 구현하도록 한다.

그래프의 각 edge에 가중치(weight)가 부여되어 있는 그래프를 가중치 그래프 (weighted graph) 또는 네트워크라고 한다. 다음 그림에서 edge의 길이를 두 인접한 vertex 사이의 거리라고 한다면 a 🡪 h의 최단거리는 a 🡪 f 🡪 h 로 12이다. 이러한 그래프는 인접행렬으로 나타낼 수 있으며 연결되지 않은 vertex 사이의 edge의 가중치는 가능한 큰 값 (무한대라고 가정할 만한 값) 을 넣는다.



위 그래프에서 shortest path를 구하기 위해서는 위 그래프를 계산 가능한 자료구조로 표현해야 하는데 일반적으로 인접행렬(adjacency matrix)이 쓰인다.

밑의 표는 위 그래프를 인접행렬로 표현한 것이다.

1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 3 | 2 | MAX | MAX | MAX | MAX |
| 6 | 0 | MAX | 2 | 3 | MAX | 2 | MAX |
| 3 | MAX | 0 | 1 | MAX | 7 | MAX | MAX |
| 2 | 2 | 1 | 0 | MAX | 4 | 3 | MAX |
| MAX | 3 | MAX | MAX | 0 | MAX | 7 | 9 |
| MAX | MAX | 7 | 4 | MAX | 0 | MAX | MAX |
| MAX | 2 | MAX | 3 | 7 | MAX | 0 | 4 |
| MAX | MAX | MAX | MAX | 9 | MAX | 4 | 0 |

지난 주에 사용했었던 인접행렬(adjacency matrix)은 그래프상에 edge의 weight(가중치)가 없는 그래프를 표현한 것으로 두 node (vertex)간에 edge로 연결되어 있으며 1, 아니면 0으로 표현하였지만, 이번 실습에 쓰이는 그래프는 edge에 weight가 있으므로 두 node (verterx)가 edge로 연결되어 있으면 그 weight값을 행렬의 해당 원소의 값으로 하고 아니면 MAX 값으로 한다.

Dijkstra 알고리즘을 C언어 기반의 pseudo code로 작성하게 되면 다음과 같다.

Dikstra(G, first)

Input: 방향성을 가지는 Graph , 시작 노드 first

/\*

그래프의 노드 수 만큼의 크기를 가지는 두 배열 dist, prev를 각각 MAX와 null로 초기화한다. 여기서 null은 -1 값이나 또는 적절한 음수값을 사용할 수 있다.

for all vertices

dist[v] = MAX;

prev[v] = null;

dist[first] = 0; // 시작노드 first에서 first간의 거리는 0으로 한다.

while (heap is not empty)

extract min dist value from the heap

// update dist(u) and prev(u)

for all vertices u adjacent to v

if dist(u) > dist(v) + weight(edge(uv))

dist(u) = dist(v) + weight(edge(uv));

prev(u) = v;

위의 pseudocode를 C로 프로그램하기 위해서는 다음과 같은 자료구조가 필요하다.

#define VERTEX\_SIZE 8 // the number of vertices

#define MAX 9999 // MAX is 9999.

int dist[VERTEX\_SIZE]; // start vertex와 특정 정점과의 거리를 저장

// prev는 최단거리 경로를 기록하기 위한 배열로서 prev[v]는 v까지 이르는 최단거리 경로에서 바로 앞의 정점을 나타냄.

int prev[VERTEX\_SIZE];

int h[VERTEX\_SIZE]; priority queue를 위한 배열.

// 그래프의 내용을 인접행렬로 표현함. graph[u][v]의 값은 edge u-v의 weight값을 나타냄.

int graph[VERTEX\_SIZE][VERTEX\_SIZE] = {

{0, 6, 3, 2, MAX, MAX, MAX, MAX},

{6, 0, MAX, 2, ...},

... }

아래의 pseudocode를 참고하여 C로 프로그램할 수 있다.

void main ()

{

int FIRST; // 출발노드. 출발 노드는 미리 주어진다.

... // 필요한 변수들을 초기화한다.

for i = 0 to VERTEX\_SIZE -1

dist[i] = MAX;

dist[FIRST] = 0; // 출발 노드의 distance값을 0으로 함.

prev[FIRST] = FIRST; // 출발 노드의 prev를 FIRST로 함.

makeheap(h, dist); // dist 배열에서 heap을 만든다.

while ((v = deleteMin(h)) >= 0)

for all edges (u, v) in E // 인접행렬로 v와 인접한

// u를 찾는다.

// dist[u](u의 distance값)와 prev[u]를 업데이트함.

if (dist[u] > dist[v] + graph[v][u]) {

dist[u] = dist[v] + graph[v][u];

prev(u) = v;

}

Update된 dist 값에 따라 heap을 재구성한다.

Dist와 prev에 저장된 값에 따라 최단거리를 출력한다.

**제출방법**

* 보고서 작성방법: 실습문제 번호별로 결과가 나온 화면의 내용을

캡쳐하여 보고서에 붙여 놓는다.

* 소스코드의 파일이름에 연습문제 번호를 붙이는 것을 잊지 않는다. 예) ex-1.c, ex-2.c
* 결과 보고서에 이름과 작성 날짜를 기입하는 것을 잊지 않는다. 예) 김웅섭\_2020\_09\_01.doc
* 실행결과를 보고서에 작성하여 소스코드와 함께 제출한다.
* 제출 마감 : e-class 제출 마감시간까지