과제 10 - 최단 경로

자료구조 3분반
B411114 신재협

2017. 12. 21

<개요>

HW10은 최단 경로 알고리즘을 이용하여 주어진 지하철 노선도에서의 시간이 최소로 걸리는 경로를 찾아내는 과제입니다.

<요구사항>

입력으로 주어지는 출발역과 도착역 간의 경로 중 비용(시간)을 최소로 갖는 경로를 찾아내어 해당되는 경로의 출발역부터 도착역까지의 구성 노드(역)들을 출력합니다.
다른 헤더파일 없이 hw10.cpp 파일에 시간 비용에 대한 인접 행렬 정보와 그에 필요한 Node(역) 클래스, 배열들과 스택, 함수들이 작성되어 있습니다.

<hw10.cpp 전체 코드>

```
#include<iostream>
#include<stack>
using namespace std;

#define INF 10000


class Node
{
public:
Node(char l = '\0', int n = 0)
{
line = l;
num = n;
}
char line; //노선
int num;//역 번호
};

Node sts[43]; //역 들의 모임. 총 43개 역이 있다.
bool visit[43]; //방문했는지 확인을 위한 배열
double dist[43];//출발역으로부터 모든 역에 대한 최단 경로를 저장하기 위
한 배열
int route[43]; //경로 출력을 위해 이용하는 배열
stack<int> stk;
//각 역들간의 시간 비용 담기 위한 인접 행렬
double timecost[43][43] =
{
{ 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF,INF, INF, INF, 0.5, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF },
{ 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, 1.0, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF,
```

```
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, 0.5, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, 0.5, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0,
1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0,
0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
```

```
1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 0.5, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, 0.5, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, 1.0, 0.0, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, 0.0, 1.0, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, 0.5, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, 0.5, INF, INF, INF, INF, 1.0, 0.0, 1.0,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
```

```
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0,
1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF, 0.5,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, 1.0, 0.0, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF },
{ 0.5, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, 0.0, 1.0, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0, INF, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, INF,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0, 1.0,
INF, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
```

```
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, 0.5, INF, INF, INF, INF, INF, INF, INF, 1.0, 0.0,
1.0, INF, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, 0.5, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, 1.0,
0.0, 1.0, INF, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
1.0, 0.0, 1.0, INF, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, 1.0, 0.0, 1.0, INF, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, 0.5, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, 1.0, 0.0, 1.0, INF },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, 1.0, 0.0, 1.0 },
{ INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF,
INF, INF, INF, INF, 1.0, 0.0 }
};

void graphInit()
{
for (int i = 0; i < 20; i++) //a 노선에 대한 정보 입력
{
sts[i].line = 'a';
sts[i].num = i + 1;
}
for (int i = 20; i < 30; i++) //b 노선에 대한 정보 입력
{
sts[i].line = 'b';
sts[i].num = i - 19;
}
for (int i = 30; i < 43; i++) //c 노선에 대한 정보 입력
{
sts[i].line = 'c';
```

```
sts[i].num = i - 29;
}
}

int findSt(Node s)
{
for (int i = 0; i < 43; i++)
{
if (sts[i].line == s.line&&sts[i].num == s.num)
{
return i;
}
}
return -1;
}

void sPath(Node s, int n)
{
//탐색을 진행하기 위한 기본적인 세팅
for (int i = 0; i < n; i++)
{
visit[i] = false;
dist[i] = timecost[findSt(s)][i]; // 최초 최단거리는 s와의 cost
}

for (int i = 0; i < n-1; i++)
{
int min = INF + 1;
int idx = -1;
for (int j = 0; j < n; j++)
{
//기존 정점과 연결된 가장 작은 정점을 찾아서
if (visit[j] == false && min > dist[j])
{
min = dist[j];
idx = j;
}
}

visit[idx] = true;
for (int j = 0; j < n; j++)
{
if (dist[j] > dist[idx] + timecost[idx][j])
{
```

```
dist[j] = dist[idx] + timecost[idx][j];
route[j] = idx;
}
}
}
}

void printPath(Node s, Node e)
{
//입력 받은 역 정보 범위 확인
if (findSt(s) == -1 || findSt(e) == -1)
{
cout << "존재하지 않는 역을 입력하셨습니다." << endl;
cout << "프로그램 종료." << endl;
return;
}
sPath(s, 43);

//시작 노드는 스택을 이용하지 않고 따로 출력
cout << sts[findSt(s)].line << sts[findSt(s)].num;

//스택을 이용하여 도착역부터 역으로 추적
int idx = findSt(e);
while (idx != 0)
{
stk.push(idx);
idx = route[idx];
}
//시작역 다음역부터 도착역까지 출력
while (!stk.empty())
{
cout << " -> " << sts[stk.top()].line << sts[stk.top()].num;
stk.pop();
}cout << endl;

double tmp = 60 * dist[findSt(e)];
int min = (int)tmp / 60;
int sec = (int)tmp % 60;
cout << "소요시간 : " << min << "분 " << sec << "초 " << endl;
}

int main()
{
graphInit(); //지하철 노선도 초기화
```

```
char start[4], end[4];
cout << "출발역: ";
cin >> start;
cout << "도착역: ";
cin >> end;
cout << "==========================================================================
<< endl;

//출발 역에 대한 노드, 도착 역에 대한 노드class
Node s, e;

//입력 받은 출발역의 정보를 노드s에 입력
s.line = start[0];
s.num = start[1] - 48;
if (start[2] >= 48 && start[2] <= 57)
{
int tmp = s.num * 10;
s.num = start[2] - 48;
s.num += tmp;
}
//입력 받은 도착역의 정보를 노드e에 입력
e.line = end[0];
e.num = end[1] - 48;
if (end[2] >= 48 && end[2] <= 57)
{
int tmp = e.num * 10;
e.num = end[2] - 48;
e.num += tmp;
}

printPath(s, e);

return 0;
}
```

<코드 설명>

과제의 전체 코드를 보고서에 담았습니다.
작성한 함수와 배열,스택 및 변수의 의미는 다음과 같습니다.

Node sts[43]
: 역을 의미하는 클래스 Node의 클래스 배열입니다.
인덱스는 0부터 42를 갖습니다. 인덱스 0~19는 a노선을, 20~29는 b노선을, 30~42는 c노선을 의미합니다.
이러한 인덱스 운용은 배열 visit,dist,route에도 동일하게 적용하였습니다.

bool visit[43]
: 인덱스에 해당되는 역이 방문되었는지 bool자료형으로 체크하는 배열입니다.

double dist[43]
: 출발역부터 노선에 존재하는 전체 역에 대한 비용(시간)을 저장하는 배열입니다.

int route[43]
: 과제에서 요구하는 최단 경로에 대한 방문 순서를 출력하기 위한 배열입니다.

stack<int> stk
: route 배열을 이용해서 경로 순서를 출력할 때에 원하는 순서인 출발역~도착역를 만들기 위해 이용되는 스택입니다. STL stack을 이용하였습니다.

double timecost[43][43]
: 노선에 존재하는 3개의 노선 42개 역들의 연결 정보를 담은 인접행렬입니다. 다익스트라 알고리즘을 적용하기 위해서 그래프상에서 직접적으로 연결되지 않았다면 매크로 선언된 INF(매우 큰 수, 구현할 땐 10000으로 설정)으로 입력했습니다.

void graphInit()
: 클래스 배열 sts의 각 역들의 노선과 역 번호를 초기화 시켜주는 함수입니다.

int findSt(Node s)
: 매개변수로 입력받은 역을 반복문을 통해 검색하는 함수입니다. 클래스 배열의 노선과 역 번호 값을 비교하여 검색하고자 하는 역이 존재한다면 그 역의 인덱스 값을 반환하고, 아니라면 -1를 반환합니다.

void sPath(Node s, int n)
: 다익스트라 알고리즘을 이용하여 출발역 s의 관점에서 그래프 상의 n개 역

에 대한 최단경로 비용을 구하는 함수입니다. 알려진 다익스트라 알고리즘과 같이 작동합니다. 다익스트라 알고리즘에 대해서는 <알고리즘의 선택 및 이유>에서 설명하도록 하겠습니다.

`void printPath(Node s, Node e)`
: sPath함수를 호출하여 구한 최소 비용을 이용하여 출발역 s부터 도착역 e까지의 최단 경로 순서와 비용을 출력하는 함수입니다.

<class설계 내용 및 이유>

```
class Node
{
public:
Node(char l = '\0', int n = 0)
{
line = l;
num = n;
}
char line; //노선
int num;//역 번호
};
```

역에 대한 노선과 번호를 더 쉽게 관리하기 위해서 Node라는 이름의 클래스를 구현하여 사용하였습니다.
생성자를 통해 노선 정보를 갖는 line과 역의 번호 정보를 갖는 num 변수를 초기화합니다. 매개변수가 전달되지 않으면 각각 '\0'과 0으로 초기화합니다.

<결과 분석>

프로그램에 입력으로 출발역과 도착역이 주어지면, 출발역부터 도착역까지의 최단 경로에 포함되는 역들의 정보가 방문하는 순서대로 출력하고 마지막으로는 최소 비용(시간)을 분과 초 단위를 이용하여 출력합니다.

최단 경로의 비용과 최단 경로대로 방문하는 순서를 출력하기 위해서 정수형 배열 route와 스택 stk를 이용하였습니다. 이 두 가지 결과 값을 구하는 원리는 다음과 같습니다.

1. 최단 경로와 비용을 찾는 함수 sPath(Node s, int n)에서 아래의 코드를 보시면 다음과 같습니다.

```
if (dist[j] > dist[idx] + timecost[idx][j])
{
dist[j] = dist[idx] + timecost[idx][j];
route[j] = idx;
}
```

: \출발역부터 인덱스 j 역까지의 비용"에 해당하는 기존에 저장해두었던 dist[j]보다 \출발역부터 인덱스 idx 역까지의 비용" + \인덱스idx부터 인덱스 j 역까지의 비용"이 더 작다면 dist[j]를 더 낮은 비용으로 교체합니다. 이를 통해 출발역부터 인덱스 j 역까지의 최단거리 비용을 구할 수 있습니다.

2. 최단 경로에 대한 방문 순서를 출력하기 위해서는 printPath(Node s, Node e)의 다음 부분과 같이 작성하였습니다.

```
//스택을 이용하여 도착역부터 역으로 추적
int idx = findSt(e);
while (idx != 0)
{
stk.push(idx);
idx = route[idx];
}
//시작역 다음역부터 도착역까지 출력
while (!stk.empty())
{
cout << " -> " << sts[stk.top()].line << sts[stk.top()].num;
stk.pop();
}cout << endl;
```

: 앞서 설명드린 코드 route[j] = idx; 는 인덱스 j 역이 인덱스가 idx인 역으로부터 직접 연결되어있다는 것을 의미합니다. 이를 이용하여 도착역부터 출발역의 다음 역까지의 배열 인덱스 값을 알 수 있습니다. 하지만 과제에서 요구하는 것과 같이 경로를 출력하려면 도착역이 가장 마지막에 출력되어야 합

니다. 따라서 스택 stk를 이용해서 순서를 거꾸로 만들었고, 이를 통해 방문 순서를 출력하였습니다.

<그래프 구성 방식>

각 역 간의 시간 비용 정보를 인접 행렬 "double timecost[43][43]"를 이용하여 그래프를 구성하였습니다. 인접 리스트보다는 시간복잡도에서 불리하지만, 다익스트라 알고리즘을 적용하여 구현하기에는 행렬이 더 편리해 보였기 때문에 인접 행렬을 이용하였습니다. 다양한 함수에서 쉽게 접근하기 위해 전역으로 선언하였습니다.

<알고리즘의 선택 및 이유>

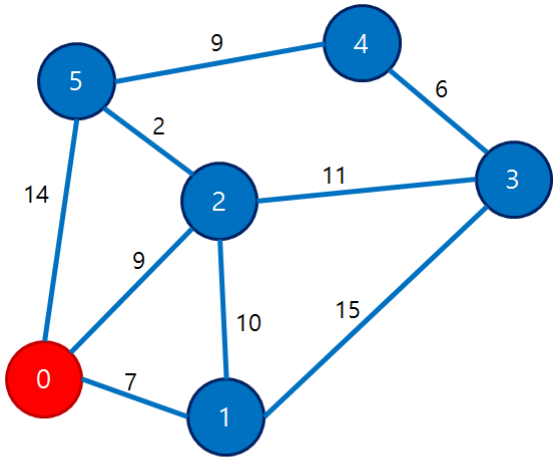
이번 최단경로 과제를 구현하기 위해서 다익스트라 알고리즘을 선택하였습니다. 처음에는 DFS를 응용하여 구현하려 했으나, 환승을 구현할 때의 어려움이 발생하였기에 대안으로 이와 같이 판단했습니다. 다익스트라 알고리즘은 하나의 노드에서부터의 최단경로를 구할 수 있고, 또한 주어진 과제에서는 가중치(weigh)가 음수인 경우가 없었기 때문에 문제 해결을 위해 적용하기 적절했다고 생각됩니다.

다익스트라 알고리즘은 다음과 같은 최단 경로 탐색 원리를 갖습니다.

1. 아직 방문하지 않은 정점들 중 거리가 가장 짧은 정점을 하나 선택하여 방문합니다.
2. 해당 정점에서 인접하고 아직 방문하지는 않은 정점들의 거리를 갱신합니다.

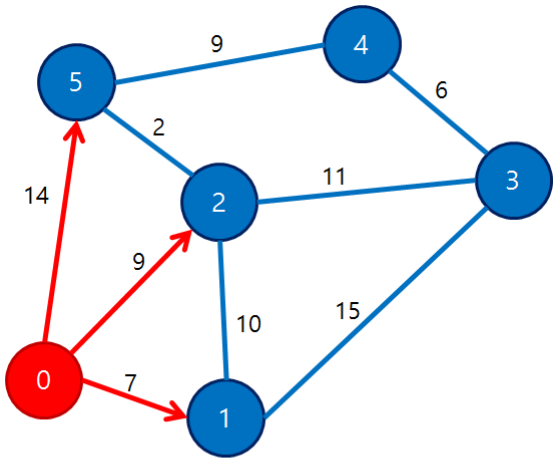
작동 원리대로, 그래프에서 한 정점에서 다른 정점으로 가는 최소 비용을 찾아내는 예시 그림은 다음 페이지에 배치하였습니다.
예시 그림의 아래 표는 0번 노드에서 다른 n(1~5)번 노드로 가는 비용을 나타냅니다.

1. 시작점은 0번 정점입니다. 0번과 다른 정점간의 최소비용을 그림 아래 배열(표)에 기록합니다.



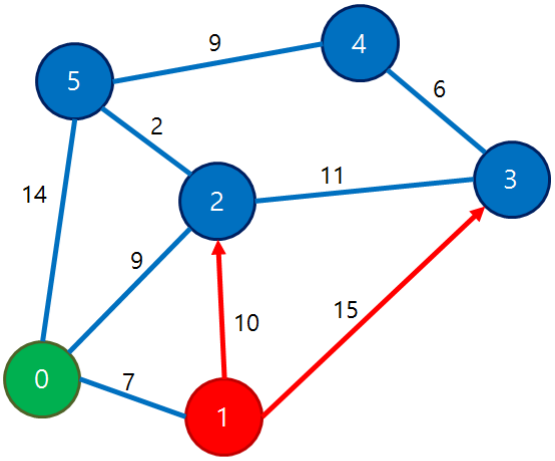
0	1	2	3	4	5
0	∞	∞	∞	∞	∞

2. 시작 정점부터 방문합니다. 0번 정점에 연결된 정점들 간의 비용(가중치)와 기존에 배열에 입력된 비용을 비교합니다.
기존의 비용보다 더 적은 비용이라면 갱신합니다.



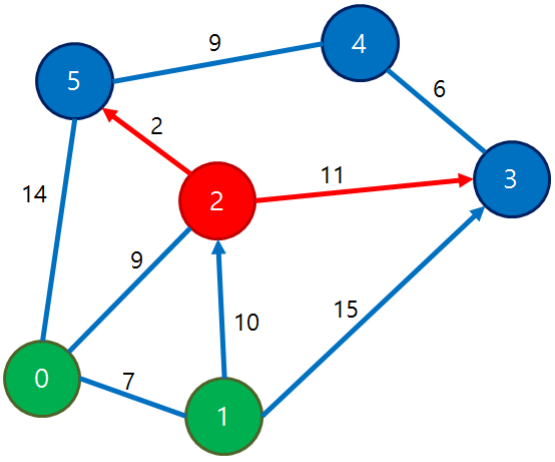
0	1	2	3	4	5
0	7	9	∞	∞	14

3. 0번 정점과 연결된 정점 중 1번 정점을 방문합니다. "0번 정점부터 1번 정점까지의 최소비용 + 1번 정점과 연결된 x번 정점간의 비용"을 기존 배열에 저장되어 있는 "0번 정점부터 x번 정점간의 비용"과 비교하여 최소 값을 배열에 갱신합니다.



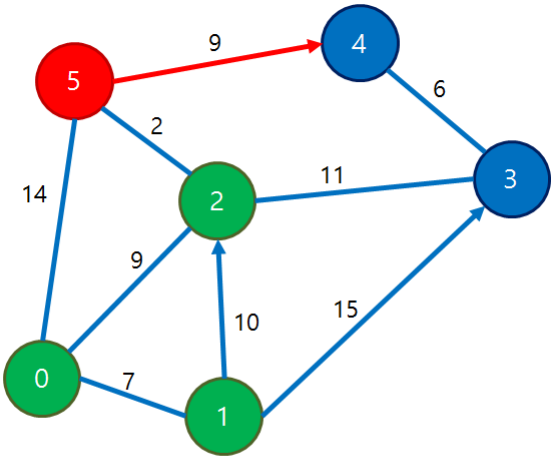
0	1	2	3	4	5
0	7	9	22	∞	14

4. 1번 정점과 연결되고 아직 방문하지 않은 정점 중 2번 정점을 방문합니다. 3.과 같은 일을 수행합니다.



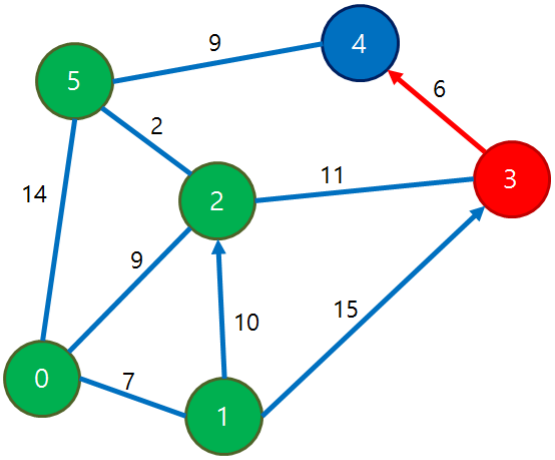
0	1	2	3	4	5
0	7	9	20	∞	11

5. 2번 정점과 연결되고 아직 방문하지 않은 정점 중 5번 정점을 방문합니다. 그리고 3.과 같은 일을 수행합니다.



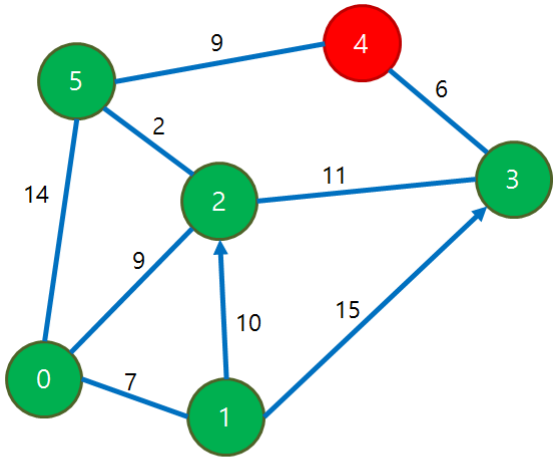
0	1	2	3	4	5
0	7	9	20	20	11

6. 2번 정점과 연결되고 아직 방문하지 않은 정점 중 3번 정점을 방문합니다. 그리고 3.과 같은 일을 수행합니다.



0	1	2	3	4	5
0	7	9	20	20	11

7. 마지막으로 방문하지 않은 하나의 정점, 즉 4번 정점을 방문합니다. 그리고 3.과 같은 일을 수행합니다.
결과적으로 시작점과 그 외 다른 정점들 간의 최단 거리 비용이 그림 아래 배열에 입력되었습니다.



0	1	2	3	4	5
0	7	9	20	20	11

* 다익스트라 알고리즘의 작동 흐름 (출처: https://goo.gl/eL6CTo)

<구현하면서 어려웠던 부분>

처음에는 DFS로 짤 수 있다고 판단하고 구현을 했었습니다만 환승의 경우에 대한 구현의 어려움을 느끼고 최단 경로 알고리즘 중 하나인 다익스트라 알고리즘으로 구현하기로 판단했습니다.

다익스트라 알고리즘을 이용하여 구현하던 중에, 역시 환승에 대한 구현을 어떻게 해야 할 지에 대한 어려움을 느꼈습니다. 이에 대한 해결책으로 기존의 생각했던 환승역들을 하나의 노드로 묶는 발상 대신 환승역끼리는 0.5의 가중치(시간 비용)를 갖는 관계로 구현하는 방법을 이용하였습니다.

또한, 최단 경로의 방문 순서를 출력하기 위해 고심했습니다. 처음에는 최단 경로를 찾는 함수 sPath에서 출발역->역에 대한 최단 거리가 구해질 때마다 출력하는 방식을 이용하려 했지만, 구현에 어려움을 느꼈고, <결과 분석>에서와 같이 도착역으로부터 해당 역의 이전 역을 추적하는 방법을 이용하여 해결할 수 있었습니다.