
| | |
|---|---|
| 과목명 | 자료구조 |
| 담당교수 | 우진운 교수님 |
| 학과 | 소프트웨어학과 |
| 학번 | 32153180 |
| 이름 | 이상민 |
| 제출일자 | 2018.12.3 |

소스 코드

```
#include <iostream>
using namespace std;
#define MAX 99999                                   // 충분히 큰 값 MAX 정의

class Graph
{
private:
    int **length;                                   // 이차원 배열
    int *dist;                                      // 일차원 배열
    int *path;                                      // 일차원 배열
    int *stack;                                     // 일차원 배열
    bool *s;                                        // 일차원 배열
    int n;                                          // 정점 수

public:
    Graph(const int vertices = 0) : n(vertices)
    {
        length = new int *[n];                      // 이차원 배열의 동적 생성
        for (int i = 0; i < n; i++)
            length[i] = new int[n];

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (i == j)                         // 자기 간선일 경우
                    length[i][j] = 0;               // 인접행렬 대각선 = 0
                else                                // 간선이 없을 경우
                    length[i][j] = MAX;             // MAX(충분히 큰 값)
            }
        }

        dist = new int[n];                          // 각 배열 동적 생성
        path = new int[n];
        stack = new int[n];
        s = new bool[n];
    }
    void ShortestPath(const int t);                 // 최단경로 함수
    void InsertEdge(int u, int v, int w);           // 간선삽입 함수
    int choose(const int n);                        // 최소 가중치 구하는 함수
    void MatrixPrint(int t);                        // 가중치를 갖는 인접행렬 출력
    void DistPrint(int t);                          // 최종 dist 출력
    void PathPrint(int t);                          // stack 이용해서 최단거리 출력
};
```

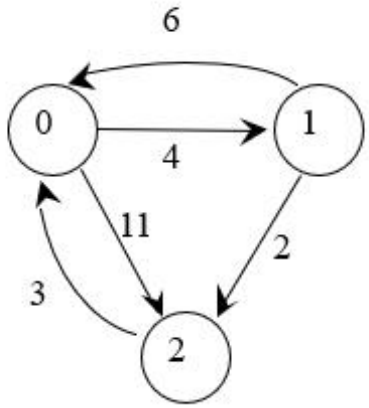
```
void Graph::ShortestPath(const int t)
{
    for (int i = 0; i < n; i++)
    {
        s[i] = false;                                   // 배열 s를 false로 초기화
        dist[i] = length[t][i];
        path[i] = t;
    }

    s[t] = true;                                        // 시작 정점 true
    dist[t] = 0;
    for (int i = 0; i < n - 2; i++)
    {
        int u = choose(n);                              // 최소 가중치 갖는 정점을
        s[u] = true;                                    // true로
        for (int w = 0; w < n; w++)
        {
            if (s[w] == false)                          // false인 곳만 비교
            {
                if (dist[u] + length[u][w] < dist[w])
                {                                       // 새로운 최단 경로
                    dist[w] = dist[u] + length[u][w];
                    path[w] = u;
                }
            }
        }
    }
}
void Graph::InsertEdge(int u, int v, int w)
{
    length[u][v] = w;                                   // u:행, v:열, w:가중치
}
int Graph::choose(const int n)
{
    int min = MAX;                                      // 충분히 큰 값 MAX
    int u = 0;

    for (int i = 0; i < n; i++)
    {
        if (s[i] == false)                              // false인 곳만 비교
        {
            if (dist[i] < min)
            {
                min = dist[i];
                u = i;
            }
        }
    }
    return u;                                           // 최솟값의 index 반환
}
```

```
void Graph::MatrixPrint(int t)
{
    for (int i = 0; i < n; i++)
    {
        cout << ' ';
        for (int j = 0; j < n; j++)
        {
            if (length[i][j] == MAX)                // 간선이 없는 경우
            {
                cout << "∞" << '₩t';                // ∞ 출력
            }
            else                                    // 간선이 있는 경우
            {
                cout << length[i][j] << '₩t';       // 가중치 출력
            }
        }
        cout << endl;
    }
    cout << endl;
}
void Graph::DistPrint(int t)
{
    for (int i = 0; i < n; i++)
    {
        cout << ' ';
        if (dist[i] == MAX)                         // 최단경로가 없는 경우
        {
            cout << "∞" << '₩t';                    // ∞ 출력
        }
        else                                        // 최단경로가 있는 경우
        {
            cout << dist[i] << '₩t';                // 최종 dist 출력
        }
    }
    cout << "₩n₩n";
}

void Graph::PathPrint(int t)
{
    if (dist[t] == MAX)                             // 경로가 없는 경우
        cout << "경로가 없습니다" << endl;

    int top = -1;
    do {
        stack[++top] = t;                           // stack에 삽입
        t = path[t];
    } while (t != 0);
    stack[++top] = t;                               // t=0까지 삽입

    while (top != 0)
    {
        int num = stack[top--];                         // stack에서 삭제
        cout << num << "->";                        // 삭제한 값 출력
    }
    cout << stack[top--] << endl;                   // 마지막으로 삭제한 값까지 출력
}
```

```
int main()
{
    int n, e;                                       // n:정점 수, e:간선 수
    int num, u, v, w;                               // u, v:정점, w:가중치
    int start;                                      // start:시작정점

    cout << "정점 수와 간선 수 입력 > ";
    cin >> n >> e;
    Graph g(n);

    for (int i = 0; i < e; i++)
    {
        num = i + 1;
        cout << num << "번째 간선(u,v)과 가중치 입력 > ";
        cin >> u >> v >> w;
        g.InsertEdge(u, v, w);
    }

    cout << "시작 정점 입력 > ";
    cin >> start;
    cout << endl;
    g.ShortestPath(start);

    cout << "<<<<<<<< 가중치를 갖는 인접행렬 >>>>>>>>" << endl;
    g.MatrixPrint(start);
    cout << "<<<<<<< 최종 결과인 배열 dist 값 >>>>>>>" << endl;
    g.DistPrint(start);
    cout << "<<<<<<< 각 정점까지의 최단 경로 >>>>>>>>" << endl;
    for (int i = 0; i < n; i++)
    {
        if (i != start)
        {
            cout << "정점 " << start << "부터 정점 " << i << "까지 최단 경로 : ";
            g.PathPrint(i);
        }
    }

    return 0;
}
```

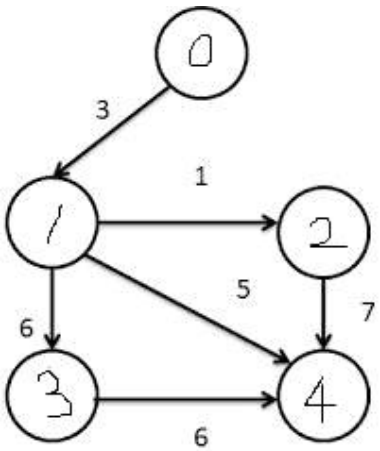
실행 파일

(1)



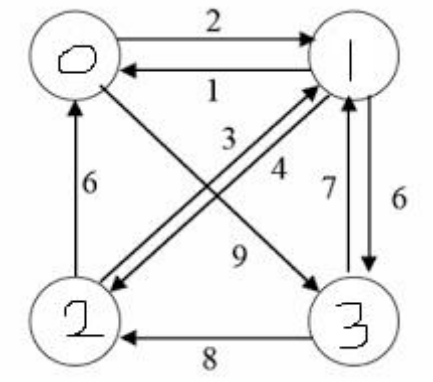

```
정점 수와 간선 수 입력 > 3 5
1번째 간선(u,v)과 가중치 입력 > 0 1 4
2번째 간선(u,v)과 가중치 입력 > 0 2 11
3번째 간선(u,v)과 가중치 입력 > 1 0 6
4번째 간선(u,v)과 가중치 입력 > 1 2 2
5번째 간선(u,v)과 가중치 입력 > 2 0 3
시작 정점 입력 > 0

<<<<<<<< 가중치를 갖는 인접행렬 >>>>>>>>
 0       4       11
 6       0       2
 3       ∞       0

<<<<<<< 최종 결과인 배열 dist 값 >>>>>>>
 0       4       6

<<<<<<< 각 정점까지의 최단 경로 >>>>>>>>
정점 0에서 1까지 최단 경로 : 0->1
정점 0에서 2까지 최단 경로 : 0->1->2
계속하려면 아무 키나 누르십시오 . . .
```

(2)




```
정점 수와 간선 수 입력 > 5 6
1번째 간선(u,v)과 가중치 입력 > 0 1 3
2번째 간선(u,v)과 가중치 입력 > 1 2 1
3번째 간선(u,v)과 가중치 입력 > 1 3 6
4번째 간선(u,v)과 가중치 입력 > 1 4 5
5번째 간선(u,v)과 가중치 입력 > 2 4 7
6번째 간선(u,v)과 가중치 입력 > 3 4 6
시작 정점 입력 > 0

<<<<<<<< 가중치를 갖는 인접행렬 >>>>>>>>
 0       3       ∞       ∞       ∞
 ∞       0       1       6       5
 ∞       ∞       0       ∞       7
 ∞       ∞       ∞       0       6
 ∞       ∞       ∞       ∞       0

<<<<<<< 최종 결과인 배열 dist 값 >>>>>>>
 0       3       4       9       8

<<<<<<< 각 정점까지의 최단 경로 >>>>>>>>
정점 0에서 1까지 최단 경로 : 0->1
정점 0에서 2까지 최단 경로 : 0->1->2
정점 0에서 3까지 최단 경로 : 0->1->3
정점 0에서 4까지 최단 경로 : 0->1->4
계속하려면 아무 키나 누르십시오 . . .
```

(3)




```
정점 수와 간선 수 입력 > 4 9
1번째 간선(u,v)과 가중치 입력 > 0 1 2
2번째 간선(u,v)과 가중치 입력 > 0 3 9
3번째 간선(u,v)과 가중치 입력 > 1 0 1
4번째 간선(u,v)과 가중치 입력 > 1 2 4
5번째 간선(u,v)과 가중치 입력 > 1 3 6
6번째 간선(u,v)과 가중치 입력 > 2 0 6
7번째 간선(u,v)과 가중치 입력 > 2 1 3
8번째 간선(u,v)과 가중치 입력 > 3 1 7
9번째 간선(u,v)과 가중치 입력 > 3 2 8
시작 정점 입력 > 0

<<<<<<<< 가중치를 갖는 인접행렬 >>>>>>>>
 0       2       ∞       9
 1       0       4       6
 6       3       0       ∞
 ∞       7       8       0

<<<<<<< 최종 결과인 배열 dist 값 >>>>>>>
 0       2       6       8

<<<<<<< 각 정점까지의 최단 경로 >>>>>>>>
정점 0에서 1까지 최단 경로 : 0->1
정점 0에서 2까지 최단 경로 : 0->1->2
정점 0에서 3까지 최단 경로 : 0->1->3
계속하려면 아무 키나 누르십시오 . . .
```