**그래프의 표현 방법**

* 그래프의 용어
* 정점: 노드라고도 하고, 데이터가 저장되는 기본 원소이다.
* 간선: 링크라고도 하고, 정점 간의 관계를 나타낸다.
* 인접 정점: 하나의 정점에서 간선에 의해 연결되어 있는 정점을 뜻한다.
* 차수: 정점에 연결된 간선의 수를 말한다.
* 경로: 간선을 따라 갈 수 있는 길을 말하고, 정점을 나열하여 표시한다.
* 경로의 길이: 경로를 구성하는 데에 사용된 간선의 수를 말한다.
* 단순 경로: 경로 중에서 반복되는 간선이 없는 경로이다.
* 사이클: 시작 정점과 종료 정점이 같은 단순 경로를 뜻한다.

그래프는 무방향, 방향, 가중치, 완전 그래프가 있다.

그중 방향 그래프는 두 정점을 연결하는 간선에 방향이 존재하고, 간선의 방향으로만 이동할 수있는 그래프이다.

방향 그래프의 코드는 다음과 같다.

#include <iostream>

#include <vector>

using namespace std;

// 그래프의 엣지를 저장할 데이터 구조

struct Edge {

    int src, dest;

};

// 그래프 객체를 나타내는 클래스

class Graph

{

public:

    // 인접 목록을 나타내는 Vector의 Vector

    vector<vector<int>> adjList;

    // 그래프 생성자

    Graph(vector<Edge> const &edges, int n)

    {

        // 'vector<int>' 유형의 'n' 요소를 포함하도록 Vector 크기를 조정한다.

        adjList.resize(n);

        // 방향 그래프에 간선 추가

        for (auto &edge: edges)

        {

            // 끝에 삽입

            adjList[edge.src].push\_back(edge.dest);

            // 무방향 그래프에 대해 다음 코드의 주석 처리를 제거한다.

            // adjList[edge.dest].push\_back(edge.src);

        }

    }

};

// 그래프의 인접 리스트 표현을 출력하는 함수

void printGraph(Graph const &graph, int n)

{

    for (int i = 0; i < n; i++)

    {

        // 현재 정점 번호를 출력

        cout << i << " ——> ";

        // 정점 `i`의 모든 인접 정점을 인쇄한다.

        for (int v: graph.adjList[i]) {

            cout << v << " ";

        }

        cout << endl;

    }

}

// STL을 사용한 그래프 구현

int main()

{

    // 위의 다이어그램에 따라 그래프 가장자리의 Vector이다.

    vector<Edge> edges =

    {

        {0, 1}, {1, 2}, {2, 0}, {2, 1}, {3, 2}, {4, 5}, {5, 4}

    };

    // 그래프의 총 노드 수(0에서 5까지 레이블 지정)

    int n = 6;

    // 그래프 생성

    Graph graph(edges, n);

    // 그래프의 인접 목록 표현을 인쇄한다.

    printGraph(graph, n);

    return 0;

}

출처: <https://www.techiedelight.com/ko/graph-implementation-using-stl/>

가중 방향 그래프는 방향을 가리키는 간선이 가중치도 가지고 있는 그래프이다.

가중 방향 그래프의 코드는 다음과 같다.

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;    // 그래프의 엣지를 저장할 데이터 구조  struct Edge {      int src, dest, weight;  };    typedef pair<int, int> Pair;    // 그래프 객체를 나타내는 클래스  class Graph  {  public:      // 인접 목록을 나타내는 쌍 Vector의 Vector      vector<vector<Pair>> adjList;        // 그래프 생성자      Graph(vector<Edge> const &edges, int n)      {          // vector<Edge> 유형의 'n' 요소를 포함하도록 Vector 크기를 조정한다.          adjList.resize(n);            // 방향 그래프에 간선 추가          for (auto &edge: edges)          {              int src = edge.src;              int dest = edge.dest;              int weight = edge.weight;                // 끝에 삽입              adjList[src].push\_back(make\_pair(dest, weight));                // 무방향 그래프에 대해 다음 코드의 주석 처리를 제거한다.              // adjList[dest].push\_back(make\_pair(src, weight));          }      }  };    // 그래프의 인접 리스트 표현을 출력하는 함수  void printGraph(Graph const &graph, int n)  {      for (int i = 0; i < n; i++)      {          // 주어진 정점의 모든 인접 정점을 출력하는 함수          for (Pair v: graph.adjList[i]) {              cout << "(" << i << ", " << v.first << ", " << v.second << ") ";          }          cout << endl;      }  }    // STL을 사용한 그래프 구현  int main()  {      // 위의 다이어그램에 따라 그래프 가장자리의 Vector이다.      vector<Edge> edges =      {          // (x, y, w) —> 가중치가 `w`인 `x`에서 `y`까지의 가장자리          {0, 1, 6}, {1, 2, 7}, {2, 0, 5}, {2, 1, 4}, {3, 2, 10}, {5, 4, 1}, {4, 5, 3}      };        // 그래프의 총 노드 수(0에서 5까지 레이블 지정)      int n = 6;        // 그래프 생성      Graph graph(edges, n);        // 그래프의 인접 목록 표현을 인쇄한다.      printGraph(graph, n);        return 0;  } |

출처: <https://www.techiedelight.com/ko/graph-implementation-using-stl/>

**다익스트라 알고리즘**

: 다익스트라 알고리즘은 음의 가중치가 없는 그래프의 한 정점에서 모든 정점까지의 최단 거리를 각각 구하는 알고리즘이다.

알고리즘 구현 과정

1. 출발점으로부터의 최단 거리를 저장할 배열 d[v]를 만들고, 출발 노드에는 0을, 출발점을 제외한 다른 노드들에는 매우 큰 값인 INF를 채워 넣는다.
2. 현재 노드를 나타내는 변수 A에 출발 노드의 번호를 저장한다.
3. A로부터 갈 수 있는 임의의 노드 B에 대해, d[A]+P[A][B]와 d[B]의 값을 비교한다.
4. 만약 d[A]+P[A][B]가 더 짧은 경로라면, d[B]의 값을 이 값으로 갱신한다.
5. A의 모든 이웃 노드 B에 대해 위의 작업을 수행한다.
6. A의 상태를 방문 완료 상태로 업데이트 한다.(더 이상 A 사용 x)
7. 아직 미방문 상태인 노드들 중, 출발점으로부터의 거리가 가장 짧은 노드 하나를 골라서 그 노드를 A에 저장한다.
8. 도착 노드가 방문 완료 상태가 되거나, 혹은 더 이상 미방문 상태의 노드를 선택할 수 없을 때까지 3~7번 과정을 반복한다.
9. 모든 작업을 마친 후 도착 노드에 저장된 값이 A로부터의 최단 거리이고, 이 값이 INF라면 도중에 길이 끊긴 것임을 의미한다.

다익스트라 알고리즘의 코드는 다음과 같다.

#include <vector>

#include <iostream>

#include <queue>

#define MAX 100        // 최대 정점의 개수

#define INF 99999999

using namespace std;

vector<int> dijkstra(int start, int V, vector<pair<int,int> > adj[]) {

    vector<int> dist(V, INF);    // 전부 INF로 초기화

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

    dist[start] = 0;

    pq.push(make\_pair(0, start));    // 시작 정점 방문

    while (!pq.empty()) {

        int cost = -pq.top().first;    // 방문한 정점의 dist 값

        int cur = pq.top().second;    // 현재 방문한 정점

        pq.pop();

        for (int i = 0; i < adj[cur].size(); i++) {    // 현재 방문한 정점의 주변 정점 모두 조사

            int next = adj[cur][i].first;    // 조사할 다음 정점

            int nCost = cost + adj[cur][i].second;    // 현재 방문한 정점을 거쳐서 다음 정점을 갈때의 비용

            if (nCost < dist[next] ) {     // 기존 비용보다 현재 방문한 정점을 거친 비용이 더 싸다면

                dist[next] = nCost;    // 갱신

                pq.push(make\_pair(-nCost, next));    // pq에 저장

            }

        }

    }

    return dist;

}

int main()

{

    int V,E;

    vector<pair<int, int> > adj[MAX];

    cout << "정점의 개수 입력 : ";

    cin >> V;

    cout << "간선의 개수 입력 : ";

    cin >> E;

    for (int i = 0; i < E; i++) {

        int from, to, cost;

        cout << "그래프 입력 [정점 정점 가중치]: ";

        cin >> from >> to >> cost;

        adj[from].push\_back(make\_pair(to, cost));    // 양방향 그래프

        adj[to].push\_back(make\_pair(from, cost));

    }

    printf("\n===다익스트라 결과===\n");

    vector<int> dist = dijkstra(0, V, adj);

    for (int i = 0; i < V; i++) {

        printf("0번 정점에서 %d번 정점까지 최단거리 : %d\n", i, dist[i]);

    }

    return 0;

}

출처: <https://code-lab1.tistory.com/29>

**백준 1753**

#include <bits/stdc++.h>

#define endl '\n'

#define INF 1e9

using namespace std;

using piiz= pair<int, int>;

int v, e, start\_node;

vector<vector<pii>> graph;

void input(){

    int st, ed, len;

    cin >> v >> e >> start\_node;

    graph.resize(v + 1);

    for(int i = 0; i < e; i++) {

        cin >> st >> ed >> len;

        graph[st].push\_back({len, ed});

    }

}

void sol() {

    priority\_queue<pii, vector<pii>, greater<pii>> pq;

    bool visit[v + 1] = {false, };

    int dist\_arr[v + 1];

    fill(dist\_arr, dist\_arr + v + 1, INF);

    dist\_arr[start\_node] = 0;

    pq.push({0, start\_node});

    while(!pq.empty()) {

        int prev = pq.top().second;

        pq.pop();

        if (visit[prev]) continue;

        visit[prev] = true;

        for(const auto& elem : graph[prev]) {

            int next = elem.second, dist = elem.first;

            if (dist\_arr[next] > dist\_arr[prev] + dist) {

                dist\_arr[next] = dist\_arr[prev] + dist;

                pq.push({dist\_arr[next], next});

            }

        }

    }

    for(int i = 1; i <= v; i++) {

        if (dist\_arr[i] != INF) cout << dist\_arr[i];

        else cout << "INF";

        cout << endl;

    }

}

int main () {

    ios\_base::sync\_with\_stdio(false);

    cin.tie(nullptr);

    input();

    sol();

}

출처: https://luv-n-interest.tistory.com/1212

**백준 1238**

#include <iostream>

#include <vector>

#include <queue>

#define pii pair<int, int>

using namespace std;

int N, M, X;

const int INF = 1e9+7;

vector<pii > graph[1001];

vector<int> dist;

int resdist[1001];

void input(){

    int u, v, t;

    cin >> N >> M >> X;

    for(int i = 0; i < M; i++){

        cin >> u >> v >> t;

        graph[u].push\_back(make\_pair(t, v));

    }

}

void Dijstra(int S){

    dist.clear();

    dist.resize(N+1, INF);

    dist[S] = 0;

    priority\_queue<pii, vector<pii >, greater<pii > > que;

    que.push({0, S});

    while(!que.empty()){

        int min\_cost = que.top().first;

        int now = que.top().second;

        que.pop();

        if(min\_cost > dist[now]) continue;

        for(int i = 0; i < graph[now].size(); i++){

            int next = graph[now][i].second;

            int next\_cost = min\_cost + graph[now][i].first;

            if(next\_cost < dist[next]){

                dist[next] = next\_cost;

                que.push({next\_cost, next});

            }

        }

    }

}

int main(){

    ios\_base::sync\_with\_stdio(false); cin.tie(NULL); cout.tie(NULL);

    input();

    for(int i = 1; i <= N; i++){

        Dijstra(i);

        // i가 X로 가는 최단거리 half

        resdist[i] = dist[X];

    }

    Dijstra(X);

    int res = 0;

    for(int i = 1; i <= N; i++){

        resdist[i] += dist[i];

        res = max(res, resdist[i]);

    }

    cout << res;

    return 0;

}

출처: https://hyeo-noo.tistory.com/138