1368 백준(오답노트)

물대기 성공다국어

한국어

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	5311	2368	1766	43.284%

문제

선주는 자신이 운영하는 N개의 논에 물을 대려고 한다. 물을 대는 방법은 두 가지가 있는데 하나는 직접 논에 우물을 파는 것이고 다른 하나는 이미 물을 대고 있는 다른 논으로부터 물을 끌어오는 법이다.

각각의 논에 대해 우물을 파는 비용과 논들 사이에 물을 끌어오는 비용들이 주어졌을 때 최소의 비용으로 모든 논에 물을 대는 것이 문제이다.

입력

첫 줄에는 논의 수 N(1 ≤ N ≤ 300)이 주어진다. 다음 N개의 줄에는 i번째 논에 우물을 팔 때 드는 비용 Wi(1 ≤ Wi ≤ 100,000)가 순서대로 들어온다. 다음 N개의 줄에 대해서는 각 줄에 N개의 수가 들어오는데 이는 i번째 논과 j번째 논을 연결하는데 드는 비용 Pi,j(1 ≤ Pi,j ≤ 100,000, Pi,j = Pj,i, Pi,i = 0)를 의미한다.

출력

첫 줄에 모든 논에 물을 대는데 필요한 최소비용을 출력한다.

예제 입력 1 복사

```
4
5
4
4
3
0 2 2 2
2 0 3 3
2 3 0 4
2 3 4 0
```

예제 출력 1 복사

```
9
```

Basic

- 우물(vertex)를 최소비용(minimal)로 연결하는 문제 -> MST
- MST를 구하는 알고리즘은 `Kruskal`과 `Prim`알고리즘이 있음.
- `Kruskal`은 각각의 트리들을 연결하는 방식임.
- `Prim`알고리즘은 하나의 트리를 기준으로 트리를 뻗어가는 방식임

내가 푼 방법 (Prim)

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#define MAX_VERT 301
#define endl '\n'
using namespace std;
class Edge
{
public:
    int a, b;
    long long w;

    Edge(const int& a_input, const int& b_input, const int& w_input) :
        a(a_input), b(b_input), w(w_input) {};
```

```
    bool operator < (const Edge& other) const
    {
        return this->w > other.w;
    }
    friend ostream& operator << (ostream& out, const Edge& E)
    {
        out << E.a << " " << E.b << " " << E.w << endl;
        return out;
    }
};

int V;
priority_queue<Edge, vector<Edge>> pq;
vector<int> cost(MAX_VERT, 0);
vector<bool> visited(MAX_VERT, false);
long long matrix[MAX_VERT][MAX_VERT];

void input()
{
    cin >> V;
    for (int i = 1; i <= V; ++i)
    {
        cin >> cost[i];
        pq.emplace(Edge{ i,i,cost[i] });
    }

    for (int i = 1; i <= V; i++)
    {
        for (int j = 1; j <= V; j++)
            cin >> matrix[i][j];
    }
}


void solve()
{
    int cnt = 0;
    long long res = 0;

    while (!pq.empty())
    {
        auto node = pq.top();
        pq.pop();

        if (node.a == node.b) // 땅을 파는 경우
        {
            if (visited[node.a]) continue; // 방문한 경우 지나간다.
            visited[node.a] = true; // 방문처리

            ++cnt;
            res += node.w;

            for (int i = 1; i <= V; i++)
            {
                if (i == node.a) continue;
                pq.push(Edge(node.a, i, matrix[node.a][i]));
            }
        }

        else
        {
            if (visited[node.b] == true) continue;
            visited[node.b] = true;

            ++cnt;
            res += node.w;

            for (int i = 1; i <= V; i++)
            {
                if (i == node.b) continue;
                pq.push(Edge(node.b, i, matrix[node.b][i]));
            }
        }

        if (cnt == V)
        {
            cout << res;
            return;
        }
```

```
    }


}

int main(void)
{
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    input();
    solve();
    return (0);
}
```

- 먼저 Prim알고리즘 방식으로 문제를 풀었음.
- 우선순위 큐를 상정한 다음, 정답 트리에 접한 Edge들을 포함하는 방법

다른 방법(Kruskal) ⭐

Prim방식이 약간 직관적이긴 하지만, 다른 방식, 즉 `Kruskal` 방식으로도 문제를 풀 수 있지않을까 궁금했음.

가상의 Vertex 0을 상정하고 Kruskal을 돌리는 방식을 생각함.
주의할 점은 Kruskal 알고리즘의 탈출 조건인 E = V-1이 아닌, E = V가 되어야함 (가상의 정점이 하나 더 추가되었으니)
또 우물을 파는데(연결x) 가장 적은 비용이 드는 정점은 "무조건" 포함되어야 함.

- 적어도 정점 하나는 무조건 파야 하니까.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int MAX_VERT = 301;

class Edge
{
public:
    int from, to, w;
    Edge(const int& f_input, const int& t_input, const int& w_input):
        from(f_input), to(t_input), w(w_input) {}
    constexpr bool operator > (const Edge& other)
    {
        return this-> w > other.w;
    }
    constexpr bool operator < (const Edge& other)
    {
        return this->w < other.w;
    }
    friend ostream& operator << (ostream& out, const Edge& E)
    {
        out << E.from << " " << E.to << " " << E.w;
        return out;
    }
};

int V, cnt, res;
int matrix[MAX_VERT][MAX_VERT];
vector<int> parents(MAX_VERT), weight(MAX_VERT,1), cost(MAX_VERT);
vector<Edge> edges;

template<typename T>
void showVector(const T& t)
{
    for(auto& ele: t)
        cout << ele << endl;
}

int Find(const int& a) noexcept
{
    if(parents[a] == a) return a;
    else
        return parents[a] = Find(parents[a]);
}

void Union(int a, int b) noexcept
{
    a = Find(a); b = Find(b);
```

```
    if(a==b) return;

    else if(weight[a] >= weight[b])
    {
        parents[b] = a;
        weight[a] += weight[b];
    }
    else
    {
        parents[a] = b;
        weight[b] += weight[a];
    }
}

void input() noexcept
{
    cin >> V;
    int input;
    for (int i = 0; i <= V; i++) parents[i] = i;

    for (int i = 1; i <= V; i++)
    {
        cin >> cost[i];
        edges.push_back(Edge{0,i,cost[i]});
    }
    sort(edges.begin(), edges.end());
    Union(edges[0].from, edges[0].to);
    res+= edges[0].w;
    ++cnt;

    for (int i = 1; i <= V; i++)
    {
        for (int j = 1; j <= V; j++)
        {
            cin >> matrix[i][j];
            if(i < j)
                edges.push_back(Edge{i,j,matrix[i][j]});
        }
    }
}

void kruskal() noexcept
{

    for(const auto& edge: edges)
    {
        auto [from, to, weight] = edge;

        if(Find(from)==Find(to)) continue; // 이미 포함된 경우
        // cout << "I am edge : " << edge << endl;
        Union(from, to);
        res += weight;
        ++cnt;
        if(cnt==V)
            break;
    }
    cout << res;
}


int main(void) noexcept
{
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    input();
    sort(edges.begin(),edges.end());
    //showVector(edges);
    kruskal();
    return (0);
}
```

⭐가상의 정점을 상정하고 MST를 구하는 테크닉을 알아 놓자!!