ID : team074

PW : mw8fvten

O(log n) 약수 찾기, 소수 판별

```
어떤 정수 n, a, b에 대해서
n = a*b, a <= b 이면 1 <= a <= logn 이 성립
```

에라토스테네스의 체 - 빠른 소인수분해

```
#include <iostream>
using namespace std;
int sieve[5000001] = { 0, };
int main() {
    for (int i = 2; i * i < 5000001; i++)
        if (sieve[i] == 0)
            for (int j = i * i; j < 5000001; j += i)
                if(sieve[j] == 0)
                    sieve[j] = i;

    int T; cin >> T;
    while (T--) {
        int k; cin >> k;
        while (sieve[k] != 0) {
            cout << sieve[k] << " ";
            k /= sieve[k];
        }
        if (k > 1) cout << k;
        cout << '\n';
    }
}
```

최대공약수(gcd) - 유클리드 알고리즘

```
ll gcd(ll a, ll b) {
    if (b == 0) return a;
    return gcd(b, a % b); // 만약 b가 a보다 크다면 이 과정에서 swap이 되므로 따로 예외처리를 해줄 필요가 없다.
}
```

모듈러 - 곱셈의 역원

```
// x^y
ll modpow(ll x, ll y) {
    if(y==0) return 1;
    ll ret = modpow(x, y/2);
    ret = modmul(ret, ret);
    if(y%2==1) ret = modmul(ret, x);
    return ret;
}

// 1/x
ll modinv(ll x) {
    return modpow(x, MOD - 2);
}
```

다익스트라

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;
using pii = pair<int, int>;

const int MAX = 123456789;
vector<int> path(800, MAX);
vector<vector<pii>> graph(800);

void daijkstra(int start) {
    path[start] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> Q;
    Q.push({0, start});
    while (!Q.empty()) {
        int cur_w = Q.top().first;
        int cur_v = Q.top().second;
        Q.pop();
        if(cur_w > path[cur_v]) continue;
        vector<pii>& adj = graph[cur_v];
        for (auto& a : adj) {
            if (path[a.first] > cur_w + a.second) {
                path[a.first] = cur_w + a.second;
                Q.push({path[a.first], a.first});
            }
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 0; i < m; i++) {
        int v1, v2, w;
        cin >> v1 >> v2 >> w;
        v1--;
```

```
        v2--;
        graph[v1].push_back({v2, w});
        graph[v2].push_back({v1, w});
    }
}
```

간선의 가중치가 없는 최단경로 - BFS

벨만-포드

음수 가중치 최단경로

```
#include <iostream>
#include <vector>

using namespace std;
using ll = long long;
using pii = pair<ll, ll>;

const ll INF = 123456789;

ll n, m;
vector<vector<pii>> graph(500);
ll path[500];

void path_init() {
    for (ll i = 0; i < n; i++) path[i] = INF;
}

bool bellman_ford(ll start) {
    path[start] = 0;
    // 정점 개수만큼 반복
    for (ll i = 0; i < n - 1; i++) {
        // edge relaxation
        for (ll c1 = 0; c1 < n; c1++) {
            if (path[c1] != INF) {
                vector<pii>& adj = graph[c1];
                for (auto& a : adj) {
                    ll c2 = a.first;
                    ll w = a.second;
                    if (path[c2] > path[c1] + w) {
                        path[c2] = path[c1] + w;
                    }
                }
            }
        }
    }
    // 음수 사이클 존재 여부 확인
    for (ll c1 = 0; c1 < n; c1++) {
        if (path[c1] != INF) {
            vector<pii>& adj = graph[c1];
            for (auto& a : adj) {
                ll c2 = a.first;
                ll w = a.second;
                if (path[c2] > path[c1] + w) {
                    return false;
                }
```

```
            }
        }
    }
    return true;
}

int main() {
    cin >> n >> m;
    for (ll i = 0; i < m; i++) {
        ll c1, c2, time;
        cin >> c1 >> c2 >> time;
        c1--;
        c2--;
        graph[c1].push_back({c2, time});
    }
    path_init();
    bool check = bellman_ford(0);
    if (check) {
        for (ll i = 1; i < n; i++) {
            if (path[i] != INF)
                cout << path[i] << '\n';
            else
                cout << "-1\n";
        }
    } else {
        cout << "-1\n";
    }
}
```

매개변수탐색

최소값 구하는 경우

```
mid = (left+right)/2

right = mid
// 조건을 만족하지 않는 경우
left = mid+1
```

최대값 구하는 경우

```
mid = (left+right+1)/2

left = mid
// 조건을 만족하지 않는 경우
right = mid-1
```

삼분탐색 (전봇대 문제)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstdlib>
using namespace std;
typedef long long ll;

ll getans(vector <ll> pole, ll a) {
   ll count = 0;
   for (int i = 1; i < pole.size(); i++) {
      count += abs(a * i - pole[i]);
   }
   return count;
}

int main() {
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   //이웃한 두 전봇대 사이 거리가 일정하도록 옮긴다.
   //이동해야하는 전봇대들의 거리의 합이 최소가 되야한다.
   //이웃한 두 전봇대 사이의 거리가 답일 때 전봇대들의 거리의 합이 최소가 된다.
   //이웃한 두 전봇대 사이의 거리가 답보다 크거나 작으면 전봇대들의 거리의 합이 최소보다 항상
크다.
   ll N;
   cin >> N;
   vector <ll> pole(N);
   for (auto& i : pole) cin >> i;
   sort(pole.begin(), pole.end());
   //이웃한 두 전봇대 사이의 거리를 범위로 정하자.
   //right의 범위를 더 좋은 방식으로 조절할 수 있을 것 같다.
   ll left = 0, right = pole.back()+1;
   ll a, b;
   while (left < right) {
      a = (left + left + right) / 3;
      b = (left + right + right) / 3;
      ll res_a = getans(pole, a);
      ll res_b = getans(pole, b);
      //res_a <= res_b인 경우 극소점은 b보다 왼쪽에 존재한다.
      if (res_a <=  res_b) right = b;
      //res_a > res_b인 경우 극소점은 a보다 오른쪽에 존재한다.
      else  left = a+1;
   }
   cout << getans(pole, left);
}
```

DP의 특성

중복되는 부분 문제(Overlapping Subproblems)

작은 부분 문제를 해결해 큰 문제를 해결하는 데에 사용합니다. 피보나치 함수를 구할 때에는 F(x)를 구할 때 보다 작은 문제인 F(x−1)과 F(x−2)를 사용합니다. 이들 중 F(x−1)은 이후에 F(x+1)을 구할 때에 다시 사용됩니다. 이렇게 부분 문제의 해법을 반복적으로 사용하는 구조에서 동적 계획법을 사용할 수 있습니다.

메모이제이션(Memoization)

메모이제이션은 이미 계산한 값을 저장해 나중에 다시 활용하는 기법입니다. 피보나치 함수를 구할 때 캐시에 계산한 값을 저장해 나중에 사용하는 부분이 메모이제이션입니다. 부분 문제의 수가 제한되어 있다면 메모이제이션을 통해 계산 횟수를 크게 줄일 수 있습니다.

최적 부분 구조(Optimal Substructure)

어떤 부분 해를 구하는 과정이 유일하고 다시 계산해도 변하지 않는다는 것이 보장되어야 합니다. 위의 피보나치 함수를 구하는 문제는 최적 부분 구조가 성립하는 것이 자명합니다. 답을 구하는 과정에서 분기가 발생할 수 있는 다른 문제들은 조금 더 복잡한 과정을 통해 이런 특성을 확보해야 할 때가 많이 있습니다.

점화식

문제의 해답을 어떤 점화식 형태로 표현할 수 있고, 그 점화식이 순환 참조하지 않는다면 DP를 고려해볼 수 있습니다.

DSU

```
#include <iostream>
#include <vector>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<long, long>;

class DSU {
    vector<int> parent;
    vector<int> count;

   public:
    DSU(int n) : parent(n), count(n, 0) {
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
        }
    }
    int find(int n) {
        if (parent[n] == n) return n;
        else return parent[n] = find(parent[n]);
    }
    void merge(int a, int b) {
        int u = find(a);
        int v = find(b);
        if(u == v) return;
        if (count[u] > count[v]) {
            parent[v] = u;
```

```
        } else {
            parent[u] = v;
            if(count[u] == count[v]) count[v]++;
        }
    }
};
```

세그먼트트리

```
#include<vector>
#include<iostream>

using namespace std;

struct seg {
    int n;
    vector<long long> S;
    long long e = 0;
    seg(const vector<long long>& A) {
        n = A.size();
        S.resize(4 * n);
        init(A, 0, n - 1, 1);
    }
    long long init(const vector<long long>& A, int nodeleft, int noderight, int
node) {
        if (nodeleft == noderight) {
            return S[node] = A[nodeleft];
        }
        int mid = (nodeleft + noderight) / 2;
        return S[node] = init(A, nodeleft, mid, node * 2)
            + init(A, mid + 1, noderight, node * 2 + 1);
    }

    long long query(int left, int right, int node, int nodeleft, int noderight)
{
        if (right < nodeleft || noderight < left) return e;
        if (left <= nodeleft && noderight <= right) return S[node];

        int mid = (nodeleft + noderight) / 2;
        return query(left, right, node * 2, nodeleft, mid)
            + query(left, right, node * 2 + 1, mid + 1, noderight);
    }
    long long query(int left, int right) {
        return query(left, right, 1, 0, n - 1);
    }
    void update(int index, long long newvalue, int node, int nodeleft, int
noderight) {
        if (index < nodeleft || noderight < index) return;
        if (nodeleft == noderight) {
            S[node] = newvalue;
            return;
        }
        int mid = (nodeleft + noderight) / 2;
        update(index, newvalue, node * 2, nodeleft, mid);
        update(index, newvalue, node * 2 + 1, mid + 1, noderight);
```

```
        S[node] = S[node * 2] + S[node * 2 + 1];
    }
    void update(int index, long long newvalue) {
        update(index, newvalue, 1, 0, n - 1);
    }
};
```

DFS와 BFS

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <map>
#include <stack>
#include <queue>
#include <deque>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<long, long>;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m, v;
    cin >> n >> m >> v;
    v--;

    vector<vector<int>> graph(n);
    while(m--) {
        int a, b;
        cin >> a >> b;
        a--; b--;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
    for(auto& g : graph) {
        sort(g.begin(), g.end());
    }

    // DFS
    vector<int> dfs_check(n, 0);
    stack<int> S;
    dfs_check[v] = 1;
    S.push(v);
    cout << v+1 << " ";
    while(!S.empty()) {
        int index = S.top();
        S.pop();
        vector<int>& adj = graph[index];
        for(auto& node : adj) {
            if(dfs_check[node] == 0) {
```

```
                dfs_check[node] = 1;
                S.push(index);
                S.push(node);
                cout << node+1 << " ";
                break;
            }
        }
    }
    cout <<'\n';

    // BFS
    vector<int> bfs_check(n, 0);
    queue<int> Q;
    bfs_check[v] = 1;
    Q.push(v);
    cout << v+1 << " ";
    while(!Q.empty()) {
        int index = Q.front();
        Q.pop();
        vector<int>& adj = graph[index];
        for(auto& node : adj) {
            if(bfs_check[node] == 0) {
                bfs_check[node] = 1;
                Q.push(node);
                cout << node+1 << " ";
            }
        }
    }
    cout <<'\n';
}
```

MST - 크루스칼

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <map>
#include <stack>
#include <queue>
#include <deque>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<long, long>;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin >> n >> m;
```

```
    vector<vector<pii>> graph(n);
    while(m--) {
        int a, b, c;
        cin >> a >> b >> c;
        a--;b--;
        graph[a].push_back({b, c});
        graph[b].push_back({a,c});
    }

    priority_queue<pii, vector<pii>, greater<pii>> Q;
    vector<int> visit(n, 0);
    Q.push({0, 0});
    int ret = 0;
    while(!Q.empty()) {
        int cur = Q.top().second;
        int w = Q.top().first;
        Q.pop();
        if(visit[cur] == 0) {
            visit[cur] = 1;
            ret += w;
        }
        else continue;

        for(auto& adj : graph[cur]) {
            if(visit[adj.first] == 0) {
                Q.push({adj.second, adj.first});
            }
        }
    }

    cout << ret << '\n';
}
```

SCC - 타잔 알고리즘

```
#include <algorithm>
#include <iostream>
#include <stack>
#include <vector>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<long, long>;

bool comp(vector<int>& a, vector<int>& b) {
    return a[0] < b[0];
}

vector<vector<int>> graph(10000);

vector<vector<int>> SCC;
```

```
stack<int> reals;
vector<int> num(10000, 10001);
vector<int> check(10000, 0);

int c = 1;

int tajan(int cur) {
    reals.push(cur);
    num[cur] = c++;  // 몇번째로 방문한 정점인지 체크

    int ret = num[cur];
    for (auto& g : graph[cur]) {
        if (num[g] == 10001) {
            // dfs 재귀함수로 구현.
            ret = min(ret, tajan(g));
        } else if (check[g] == 0) {
            ret = min(ret, num[g]);
        }
    }
    // SCC 최상단 정점인 경우
    if (ret == num[cur]) {
        vector<int> scc;
        while (!reals.empty() && reals.top() != cur) {
            scc.push_back(reals.top());
            check[reals.top()] = 1;
            reals.pop();
        }
        scc.push_back(reals.top());
        check[reals.top()] = 1;
        reals.pop();
        sort(scc.begin(), scc.end());
        SCC.push_back(scc);
    } else
        num[cur] = ret;

    return ret;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    // 입력
    int v, e;
    cin >> v >> e;
    while (e--) {
        int a, b;
        cin >> a >> b;
        a--;
        b--;
        graph[a].push_back(b);
    }

    // 타잔 알고리즘
    for (int start = 0; start < v; start++) {
        if (num[start] == 10001) {
            tajan(start);
        }
```

```
    }

    //출력
    sort(SCC.begin(), SCC.end(), comp);
    cout << SCC.size() << '\n';
    for (auto& scc : SCC) {
        for (auto& s : scc) {
            cout << s + 1 << " ";
        }
        cout << "-1\n";
    }
}
```