Taiwan Online Programming Contest (TOPC) 2021

난이도 순서: A J B D E C G F H I

A. Olympic Ranking

단순 구현 문제입니다. 나라 이름에 공백이 들어갈 수 있음에 주의합시다.

```
#include <bits/stdc++.h>
using namespace std;
using T = tuple<int,int,int,string>;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    T res(-1,-1,-1,"");
    for(int i=0; i<N; i++){
        int a, b, c; string s;
        cin >> a >> b >> c; cin.get();
        getline(cin, s);
        res = max(res, T(a,b,c,s));
    }
    cout << get<3>(res);
}
```

B. Aliquot Sum

일반적으로 약수의 개수를 세는 것보다는 배수의 개수를 세는 것이 편합니다. 즉, i의 약수의 개수 A_i를 구하는 대신, n 이하의 자연수 i에 대해 $A_{2i}, A_{3i}, A_{4i}, \cdots$에 1씩 더하는 방식으로 계산하는 것이 더 효율적입니다. 시간 복잡도는 $\sum_{i=1}^{n} n/i = n\sum_{i=1}^{n} 1/i \approx n\int_1^n 1/x \, \mathrm{d}x = O(n\log n)$입니다.

```
#include <bits/stdc++.h>
using namespace std;

int A[1010101];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    for(int i=1; i<1010101; i++) for(int j=i+i; j<1010101; j+=i) A[j] += i;
    int T; cin >> T;
    while(T--){
        int n; cin >> n;
        if(A[n] > n) cout << "abundant\n";
        else if(A[n] < n) cout << "deficient\n";
        else cout << "perfect\n";
    }
}
```

C. A Sorting Problem

$|A_x - A_y| = 1$인 두 원소 A_x, A_y만 교환할 수 있다는 조건이 붙어있을 때, 1부터 n까지의 수로 구성된 순열을 정렬하는 문제입니다. 순열이므로 역배열($A[B[i]] = i$를 만족하는 B)를 정렬한다고 생각해도 되고, 이는 인접한 두 원소만 교환해서 B를 정렬하는 문제와 같습니다. 따라서 이 문제는 전형적인 Inversion Counting 문제가 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 18;

int N, A[SZ], T[SZ];
long long R;

void Add(int x, int v){ for(x+=3; x<SZ; x+=x&-x) T[x] += v; }
int Get(int x){ int r = 0; for(x+=3; x; x-=x&-x) r += T[x]; return r; }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,t; i<=N; i++) cin >> t, A[t] = i;
    for(int i=N; i>=1; i--) R += Get(A[i]), Add(A[i], 1);
    cout << R;
}
```

D. Drunk Passenger

술에 취한 승객이 앉는 자리에 따라 나눠서 생각해 봅시다. 술에 취한 승객이 바로 n번째 자리에 앉을 확률은 $1/(n-1)$입니다.

술에 취한 승객이 $1/(n-1)$의 확률로 $n-1$번째 자리에 앉는다면 $2, 3, \cdots, n-2$번째 승객은 올바른 위치에 앉을 것이고, $n-1$번째 승객은 $1/2$의 확률로 첫 번째 또는 n번째 자리에 앉게 될 것입니다. 따라서 이때 n번째 자리를 선점당할 확률은 $1/2(n-1)$입니다.

술에 취한 승객이 $1/(n-1)$의 확률로 $n-2$번째 자리에 앉는다면 $2, 3, \cdots, n-3$번째 승객은 올바른 위치에 앉을 것이고, $n-2$번째 사람은 $1, n-1, n$번째 자리 중 한 곳에 앉을 수 있습니다. $n-2$번째 사람이 n번째 자리에 앉을 확률은 $1/3$이며, $1/3$의 확률로 첫 번째 자리에 앉는다면 n번째 자리는 선점되지 않습니다. $n-2$번째 사람이 $1/3$의 확률로 $n-1$번째 자리에 앉는다면 $n-1$번째 사람은 $1/2$의 확률로 첫 번째 또는 n번째에 앉게 될 것입니다. 따라서 술에 취한 승객이 $n-2$번째 자리에 앉으면서 n번째 자리가 선점당할 확률은 $1/(n-1) \times \{1/3 + 1/3 \times 1/2\} = 1/2(n-1)$입니다.

비슷한 방법으로 계산하면, 술에 취한 승객이 $2, 3, \cdots, n-1$번째 자리에 앉았을 때의 확률이 매번 $1/2(n-1)$임을 알 수 있습니다. 따라서 정답은 $\frac{1}{n-1} + \frac{n-2}{2(n-1)} = \frac{n}{2(n-1)}$입니다.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n; cin >> n;
    cout << fixed << setprecision(10) << n / 2. / (n - 1);
}
```

E. Eatcoin

$p \le q \times d^5$가 될 때까지만 버틸 수 있다면 그 이후로는 항상 10^{99}원을 만들 수 있습니다. 따라서 $p \le q \times d^5$를 만족하는 가장 작은 d를 d'이라고 하면, $x = p \times d' - q \times (1^5 + 2^5 + \cdots + (d'-1)^5)$입니다. x를 구했으면 y는 이분 탐색을 이용해 찾을 수 있습니다. Python이나 Java의 BigInteger를 사용하거나, C++에서 큰 수 연산을 직접 구현해야 문제를 해결할 수 있습니다.

```python
def s5(n):
    return n * n * (n + 1) * (n + 1) * (2 * n * n + 2 * n - 1) // 12

def calc(p, q, x, n):
    return x - n * p + q * s5(n)

p, q = map(int, input().split())

day, x = 1, -1
while True:
    if p <= q * day**5:
        x = p * day - q * s5(day-1)
        break
    day += 1

print(x)

l, r = 0, 10**99
while l < r:
    m = (l + r) // 2
    if calc(p, q, x, m) >= 10**99: r = m
    else: l = m + 1

print(r)
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using poly = vector<int>;

ostream& operator << (ostream &out, const poly &v){
    if(v.empty()) out << 0;
    else for(int i=(int)v.size()-1; i>=0; i--) out << v[i];
    return out;
}

void normalize(poly &v){
    v.push_back(0);
    for(int i=0; i+1<v.size(); i++){
        if(v[i] < 0){
            int b = (abs(v[i]) + 9) / 10;
            v[i+1] -= b; v[i] += b * 10;
        }
        else v[i+1] += v[i] / 10, v[i] %= 10;
    }
    while(v.size() > 1 && !v.back()) v.pop_back();
}

bool ge(const poly &a, const poly &b){
    if(a.size() != b.size()) return a.size() > b.size();
```

```
    for(int i=(int)a.size()-1; i>=0; i--) if(a[i] != b[i]) return a[i] > b[i];
    return true;
}

poly operator + (const poly &a, const poly &b){
    poly res(max(a.size(),b.size())+1);
    for(int i=0; i<a.size(); i++) res[i] += a[i];
    for(int i=0; i<b.size(); i++) res[i] += b[i];
    normalize(res); return res;
}

poly operator - (const poly &a, const poly &b){
    assert(ge(a, b)); // ensure a - b >= 0
    poly res(max(a.size(),b.size())+1);
    for(int i=0; i<a.size(); i++) res[i] += a[i];
    for(int i=0; i<b.size(); i++) res[i] -= b[i];
    normalize(res); return res;
}

poly operator * (const poly &a, const poly &b){
    poly res(a.size() + b.size());
    for(int i=0; i<a.size(); i++) for(int j=0; j<b.size(); j++) res[i+j] += a[i]
* b[j];
    normalize(res); return res;
}

poly operator / (poly a, const poly &b){
    if(!ge(a, b)) return {0};
    vector<int> res(a.size()-b.size()+1);
    for(int i=a.size()-b.size(); i>=0; i--){
        poly target(a.begin()+i, a.end());
        poly now = {0}; normalize(target);
        while(res[i] + 1 < 10){
            auto nxt = now + b;
            if(ge(target, nxt)) swap(nxt, now), res[i]++;
            else break;
        }
        target = target - now;
        for(int j=i; j<a.size(); j++) a[j] = j - i < target.size() ? target[j-i]
: 0;
    }
    normalize(res);
    return res;
}

const poly one{1}, two{2}, twelve{12};

poly long2poly(long long n){
    poly res;
    while(n) res.push_back(n % 10), n /= 10;
    if(res.empty()) res.push_back(0);
    return res;
}

poly s5(poly n){
    if(n == poly{0}) return {0};
    return n * n * (n + one) * (n + one) * (two * n * n + two * n - one) /
twelve;
```

```
}

poly calc(poly p, poly q, poly x, poly n){
    return x + q * s5(n) - n * p;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    long long _p, _q; cin >> _p >> _q;
    auto p = long2poly(_p), q = long2poly(_q);

    poly x;
    for(int i=1; ; i++){
        poly day = long2poly(i);
        if(ge(q * day * day * day * day * day, p)){
            x = p * day - q * s5(long2poly(i-1));
            break;
        }
    }

    poly l = {0}, r(99, 0); r.push_back(1);
    while(!ge(l, r)){
        poly m = (l + r) / two;
        if(calc(p, q, x, m).size() >= 100) r = m;
        else l = m + one;
    }

    cout << x << "\n" << r;
}
```

F. Flip

흔히 금광 세그먼트 트리 등의 이름으로 불리는 유형입니다. 세그먼트 트리의 각 정점에서

1. 구간에 포함된 alternating subarray의 개수
2. 0101... 형태의 가장 긴 prefix의 길이
3. 1010... 형태의 가장 긴 prefix의 길이
4. ...1010 형태의 가장 긴 suffix의 길이
5. ...0101 형태의 가장 긴 suffix의 길이

를 관리하면 세그먼트 트리와 레이지 프로퍼게이션을 이용해 $O(N + Q \log N)$ 시간에 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int SZ = 1 << 18;

struct Node{
    ll len, sum, fr[2], bk[2];
    Node(){ len = sum = fr[0] = fr[1] = bk[0] = bk[1] = 0; }
    void set(int v){
        sum = len; fr[!v] = bk[!v] = 0;
        fr[v] = bk[v] = len != 0 ? 1 : 0;
    }
    void flip(){ swap(fr[0], fr[1]); swap(bk[0], bk[1]); }
    friend Node operator + (const Node &l, const Node &r){
```

```cpp
        Node res;
        res.len = l.len + r.len;
        res.sum = l.sum + r.sum + l.bk[0] * r.fr[1] + l.bk[1] * r.fr[0];
        res.fr[0] = l.fr[0]; if(l.fr[0] == l.len) res.fr[0] += r.fr[l.len%2];
        res.fr[1] = l.fr[1]; if(l.fr[1] == l.len) res.fr[1] += r.fr[1-l.len%2];
        res.bk[0] = r.bk[0]; if(r.bk[0] == r.len) res.bk[0] += l.bk[r.len%2];
        res.bk[1] = r.bk[1]; if(r.bk[1] == r.len) res.bk[1] += l.bk[1-r.len%2];
        return res;
    }
};

int N, Q, A[SZ];
Node T[SZ<<1]; int L[SZ<<1];

void Push(int node, int s, int e){
    if(!L[node]) return;
    T[node].flip();
    if(s != e) L[node<<1] ^= 1, L[node<<1|1] ^= 1;
    L[node] = 0;
}

void Init(int node=1, int s=1, int e=N){
    if(s == e){ T[node].len = 1; T[node].set(A[s]); return; }
    int m = (s + e) / 2;
    Init(node<<1, s, m);
    Init(node<<1|1, m+1, e);
    T[node] = T[node<<1] + T[node<<1|1];
}

void Update(int l, int r, int node=1, int s=1, int e=N){
    Push(node, s, e);
    if(r < s || e < l) return;
    if(l <= s && e <= r){ L[node] ^= 1; Push(node, s, e); return; }
    int m = (s + e) / 2;
    Update(l, r, node<<1, s, m);
    Update(l, r, node<<1|1, m+1, e);
    T[node] = T[node<<1] + T[node<<1|1];
}

Node Query(int l, int r, int node=1, int s=1, int e=N){
    Push(node, s, e);
    if(l <= s && e <= r) return T[node];
    int m = (s + e) / 2;
    if(r <= m) return Query(l, r, node<<1, s, m);
    if(m < l) return Query(l, r, node<<1|1, m+1, e);
    return Query(l, r, node<<1, s, m) + Query(l, r, node<<1|1, m+1, e);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    for(int i=1; i<=N; i++) cin >> A[i];
    Init();
    for(int q=1; q<=Q; q++){
        int op, l, r; cin >> op >> l >> r;
        if(op == 1) Update(l, r);
        else cout << Query(l, r).sum << "\n";
    }
```

```
}
```

G. Garden Park

간선을 가중치 오름차순으로 처리하면서, $D(v) := v$에서 끝나는 경로의 개수를 관리할 것입니다.

간선 $e = (u, v)$을 처리해야 하는 상황을 생각해 봅시다. e가 추가되기 전에는 u와 v가 서로 다른 컴포넌트에 속한 상태이며, e가 추가되면 u와 v가 연결됩니다. 따라서 e로 인해 추가되는 경로의 개수만 고려하면 되고, 지금까지 추가된 간선은 모두 e보다 가중치가 작기 때문에 $D(u)$와 $D(v)$만 신경써도 충분합니다. 이때 $D(u)$는 $D(v) + 1$ 만큼, $D(v)$는 $D(u) + 1$ 만큼 증가합니다.

가중치가 같은 간선이 여러 개 주어지는 경우에 주의해서 구현해야 합니다.

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, U[202020], V[202020], W[202020];
vector<int> C;
vector<pair<int,int>> E[202020];
ll D[202020], Delta[202020];

void Compress(vector<int> &v){
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++) cin >> U[i] >> V[i] >> W[i];
    Compress(C = vector<int>(W+1, W+N));
    for(int i=1; i<N; i++) W[i] = lower_bound(C.begin(), C.end(), W[i]) -
C.begin() + 1;
    for(int i=1; i<N; i++) E[W[i]].emplace_back(U[i], V[i]);

    for(int w=1; w<=C.size(); w++){
        vector<int> now;
        for(auto [u,v] : E[w]) now.push_back(u), now.push_back(v);
        Compress(now);
        for(auto [u,v] : E[w]) Delta[u] += D[v] + 1, Delta[v] += D[u] + 1;
        for(auto i : now) D[i] += Delta[i], Delta[i] = 0;
    }
    cout << accumulate(D+1, D+N+1, 0LL);
}
```

H. A Hard Problem

비트마다 독립적으로 생각해도 되므로 각 정점을 16개의 정점으로 분할($v \to (v, 0), (v, 1), \cdots, (v, 15)$)하고 문제를 해결해 봅시다.

어떤 정점의 값이 X_i로 주어지는 것은, 분할된 16개의 정점에 적힌 비트가 고정된다는 것을 의미합니다. 또한, 문제에서 주어진 그래프에서 어떤 두 정점 u, v가 연결되어 있다는 것은, (u, i)와 (v, i)의 값이 같을 때 비용이 1 만큼 증가한다는 것을 의미합니다. 이런 식으로 (1) A_i는 true가 되어야 한다, (2) A_i는 false가 되어야 한다, (3) A_i와 A_j가 다르면 비용이 발생한다는 조건에서 비용을 최소화하는 문제는 minimum cut을 이용해 해결할 수 있음이 잘 알려져 있습니다.

하지만 Q개의 조건 (t, u, i, v, j)가 더 주어진다면 어떨까요? $t = 0$인 정보, 즉 (u, i)와 (v, j)가 같아야 한다는 조건이 추가되더라도 여전히 minimum cut으로 해결할 수 있습니다. 그러나 $t = 1$인 정보, 즉 (u, i)와 (v, j)가 달라야 한다는 조건이 추가되면 MAX-2SAT 문제로 환원되기 때문에 더 이상 다항 시간에 해결하지 못합니다. $Q \le 8$이라는 조건을 잘 활용하면 문제를 풀 수 있다는 생각을 할 수 있습니다.

지금 문제가 되는 부분은 $(u, i) \ne (v, j)$ 조건입니다. 하지만 이런 형태의 조건은 최대 8개밖에 없으므로, (u, i)를 false로 고정하고 (v, j)를 true로 고정하는 경우와 (u, i)를 true로 고정하고 (v, j)를 false로 고정하는 경우를 모두 확인하더라도 시간 안에 문제를 해결할 수 있습니다.

전체 시간 복잡도는 $V = 16N + 2, E = 16N + 16M + 2Q$일 때 $O(2^Q N^2 M)$이지만, 최대 유량 문제가 늘 그렇듯 시간 제한 안에 잘 돌아갑니다.

```
#include <bits/stdc++.h>
using namespace std;

template<typename flow_t, flow_t MAX_U=(1<<30)>
struct Dinic{ // 0-based
    struct edge_t{ int v, r; flow_t c, f; };
    int n;
    vector<vector<edge_t>> g;
    vector<int> lv, idx;
    Dinic(int n) : n(n) { clear(); }
    void clear(){
        g = vector<vector<edge_t>>(n);
        lv = vector<int>(n, 0);
        idx = vector<int>(n, 0);
    }
    void add_edge(int s, int e, flow_t c1){
        g[s].push_back({e, (int)g[e].size(), c1, 0});
        g[e].push_back({s, (int)g[s].size()-1, c1, 0});
    }
    bool bfs(int s, int t, flow_t limit=1){
        fill(lv.begin(), lv.end(), 0);
        queue<int> que; que.push(s); lv[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(const auto &e : g[v]) if(!lv[e.v] && e.c - e.f >= limit)
que.push(e.v), lv[e.v] = lv[v] + 1;
        }
        return lv[t] != 0;
    }
    flow_t dfs(int v, int t, flow_t fl=MAX_U){
        if(v == t || fl == flow_t(0)) return fl;
        for(int &i=idx[v]; i<g[v].size(); i++){
            auto &e = g[v][i];
            if(lv[e.v] != lv[v] + 1 || e.c - e.f == flow_t(0)) continue;
            flow_t now = dfs(e.v, t, min(fl, e.c - e.f));
            if(now == flow_t(0)) continue;
            e.f += now; g[e.v][e.r].f -= now;
            return now;
        }
        return 0;
    }
    flow_t maximum_flow(int s, int t){
        flow_t flow = 0, augment = 0;
        while(bfs(s, t)){
            fill(idx.begin(), idx.end(), 0);
```

```
            while((augment=dfs(s, t)) != flow_t(0)) flow += augment;
        }
        return flow;
    }
};

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M, K; cin >> N >> M;
    auto f = [](int v, int bit){ return v << 4 | bit; };

    vector<pair<int,int>> E(M);
    for(auto &[u,v] : E) cin >> u >> v;

    vector<int> V(N);
    for(auto &i : V) cin >> i;

    cin >> K;
    vector<tuple<int,int,int,int,int>> Q(K);
    for(auto &[t,u,i,v,j] : Q) cin >> t >> u >> i >> v >> j;

    long long res = 1 << 30;
    for(int bit=0; bit<(1<<K); bit++){
        int S = N * 16, T = S + 1;
        Dinic<long long> G(T + 1);
        for(int v=0; v<N; v++){
            if(V[v] == -1) continue;
            for(int i=0; i<16; i++){
                if(V[v] >> i & 1) G.add_edge(f(v,i), T, 1<<30);
                else G.add_edge(S, f(v,i), 1<<30);
            }
        }
        for(auto [u,v] : E) for(int i=0; i<16; i++) G.add_edge(f(u,i), f(v,i),
1);
        for(int q=0; q<K; q++){
            auto [t,u,i,v,j] = Q[q];
            if(t == 0) G.add_edge(f(u,i), f(v,j), 1<<30);
            else{
                if(bit >> q & 1) G.add_edge(S, f(u,i), 1<<30),
G.add_edge(f(v,j), T, 1<<30);
                else G.add_edge(f(u,i), T, 1<<30), G.add_edge(S, f(v,j), 1<<30);
            }
        }
        res = min(res, G.maximum_flow(S, T));
    }
    cout << (res < 1e9 ? res : -1);
}
```

I. ICPC Kingdom

아래 두 가지 조건을 만족하면서 크기가 $1, 2, \cdots, N-1$인 최대 가중치 간선 집합을 찾아야 합니다.

1. 선택한 간선에 사이클이 없어야 한다.
2. 각 작업자가 한 번씩만 일할 수 있도록 배정할 수 있어야 한다.

1번 조건은 graphic matroid, 2번 조건은 partition matroid이고, 두 매트로이드의 최대 가중치 공통 독립 집합을 구해야 하므로 matroid intersection을 구현하면 문제를 해결할 수 있습니다.

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

struct UnionFind{
    int n; vector<int> p;
    UnionFind(int n) : n(n), p(n) { iota(p.begin(), p.end(), 0); }
    int find(int v){ return v == p[v] ? v : p[v] = find(p[v]); }
    bool connected(int u, int v){ return find(u) == find(v); }
    bool merge(int u, int v){ return find(u) != find(v) && (p[p[u]]=p[v], true);
}
};

struct Matroid{
    virtual bool check(int i) = 0; // O(R^2N), O(R^2N)
    virtual void insert(int i) = 0; // O(R^3), O(R^2N)
    virtual void clear() = 0; // O(R^2), O(RN)
};

struct GraphicMatroid : public Matroid, public UnionFind {
    vector<array<int,2>> e;
    GraphicMatroid(int n, const vector<array<int,2>> &e) : UnionFind(n), e(e) {}
    bool check(int i){ return !connected(e[i][0], e[i][1]); }
    void insert(int i){ merge(e[i][0], e[i][1]); }
    void clear(){ iota(p.begin(), p.end(), 0); }
};

struct PartitionMatroid : public Matroid {
    vector<int> rem, group, limit;
    PartitionMatroid(const vector<int> &group, const vector<int> &limit)
        : rem(limit), group(group), limit(limit) {}
    bool check(int i){ return rem[group[i]] != 0; }
    void insert(int i){ --rem[group[i]]; }
    void clear(){ rem = limit; }
};

template<typename cost_t>
vector<cost_t> MI(const vector<cost_t> &cost, Matroid *m1, Matroid *m2){
    int n = cost.size();
    vector<pair<cost_t, int>> dist(n+1);
    vector<vector<pair<int, cost_t>>> adj(n+1);
    vector<int> pv(n+1), inq(n+1), flag(n); deque<int> dq;
    auto augment = [&]() -> bool {
        fill(dist.begin(), dist.end(), pair(numeric_limits<cost_t>::max()/2,
0));
        fill(adj.begin(), adj.end(), vector<pair<int, cost_t>>());
        fill(pv.begin(), pv.end(), -1);
        fill(inq.begin(), inq.end(), 0);
        dq.clear(); m1->clear(); m2->clear();
        for(int i=0; i<n; i++) if(flag[i]) m1->insert(i), m2->insert(i);
        for(int i=0; i<n; i++){
            if(flag[i]) continue;
            if(m1->check(i)) dist[pv[i]=i] = {cost[i], 0}, dq.push_back(i),
inq[i] = 1;
            if(m2->check(i)) adj[i].emplace_back(n, 0);
        }
        for(int i=0; i<n; i++){
```

```
            if(!flag[i]) continue;
            m1->clear(); m2->clear();
            for(int j=0; j<n; j++) if(i != j && flag[j]) m1->insert(j), m2-
>insert(j);
            for(int j=0; j<n; j++){
                if(flag[j]) continue;
                if(m1->check(j)) adj[i].emplace_back(j, cost[j]);
                if(m2->check(j)) adj[j].emplace_back(i, -cost[i]);
            }
        }
        while(dq.size()){
            int v = dq.front(); dq.pop_front(); inq[v] = 0;
            for(const auto &[i,w] : adj[v]){
                pair<cost_t, int> nxt{dist[v].first+w, dist[v].second+1};
                if(nxt < dist[i]){
                    dist[i] = nxt; pv[i] = v;
                    if(!inq[i]) dq.push_back(i), inq[i] = 1;
                }
            }
        }
        if(pv[n] == -1) return false;
        for(int i=pv[n]; ; i=pv[i]){
            flag[i] ^= 1; if(i == pv[i]) break;
        }
        return true;
    };
    vector<cost_t> res;
    while(augment()){
        int now = 0;
        for(int i=0; i<n; i++) if(flag[i]) now += cost[i];
        res.push_back(now);
    }
    return res;
}

ll FloorSqrt(ll n){
    ll l = 0, r = 2e9;
    while(l < r){
        ll m = (l + r + 1) / 2;
        if(m * m <= n) l = m;
        else r = m - 1;
    }
    return l;
}

int N, M, K, A[111], U[111], V[111], W[111];
vector<array<int,2>> Edge;
vector<ll> Weight;
vector<int> Group, Limit;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=0; i<N; i++) cin >> A[i];
    for(int i=0; i<M; i++) cin >> U[i] >> V[i];
    for(int i=0; i<M; i++) W[i] = -FloorSqrt(A[--U[i]] + A[--V[i]]);

    cin >> K; Limit.resize(K, 1);
```

```
    for(int i=0; i<K; i++){
        int c; cin >> c;
        for(int j=0; j<c; j++){
            int e; cin >> e; e--;
            Edge.push_back({U[e], V[e]});
            Weight.push_back(W[e]);
            Group.push_back(i);
        }
    }

    GraphicMatroid *m1 = new GraphicMatroid(N, Edge);
    PartitionMatroid *m2 = new PartitionMatroid(Group, Limit);
    auto res = MI<ll>(Weight, m1, m2);
    while(res.size() + 1 < N) res.push_back(1);
    for(auto i : res) cout << -i << " ";
}
```

J. JavaScript

단순 구현 문제입니다.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    string a, b; cin >> a >> b;
    for(auto i : a) if(!isdigit(i)) { cout << "NaN"; return 0; }
    for(auto i : b) if(!isdigit(i)) { cout << "NaN"; return 0; }
    cout << stoi(a) - stoi(b);
}
```