```
// aho-corasick

const int K = 26;

struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];
    int exit_link = -1;

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - '0';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - '0';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}


int exit_link(int v){
    if(t[v].exit_link == -1){
        if(t[v].leaf) t[v].exit_link = v;
        else {
            if(v == 0) t[v].exit_link = 0;
            else t[v].exit_link = exit_link(get_link(v));
        }
    }
    return t[v].exit_link;
}
```

```
// Balanced ternary (number system)

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

string intToB3(int x){
    string res = "";
    int carry = 0;
    while(x || carry){
        int cur = x%3 + carry;
        x /= 3;
        carry = 0;
        if(cur == 2){
            cur = -1;
            carry = 1;
        } else if(cur == 3){
            cur = 0;
            carry = 1;
        }
        if(cur == -1) res += 'Z';
        else res += '0' + cur;
    }
    reverse(res.begin(), res.end());
    return res;
}

int main() {

    int x, y;
    cin >> x >> y;
    x = abs(x);
    y = abs(y);
    if(x > y) swap(x, y);

    bool res = 1;

    string onlyX = intToB3(x);
    string onlyY = intToB3(y);

    string toAdd(onlyY.length() - onlyX.length(), '0');
    onlyX = toAdd + onlyX;

    for(int i = 0; i < onlyX.length(); i++){
        if(onlyX[i] != '0' && onlyY[i] != '0') res = 0;
        if(onlyX[i] == '0' && onlyY[i] == '0') res = 0;
    }

    if(res) puts("Possible");
    else puts("Impossible");

}
```

```
// big int

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

string itos(int x){
    if(x == 0) return "";
    return itos(x/10) + char(x%10 + '0');
}

struct bigInt {

    int base = 1000*1000*1000;
    vector<int> a;

    bigInt(){
        a.push_back(0);
    }
    bigInt(string s){
        read(s);
    }
    bigInt(ll x){
        while(x){
            a.push_back(x%base);
            x /= base;
        }
    }

    void print(){
        printf ("%d", a.empty() ? 0 : a.back());
        for (int i=(int)a.size()-2; i>=0; --i)
            printf ("%09d", a[i]);
        puts("");
    }

    void read(string s){
        for (int i=(int)s.length(); i>0; i-=9)
            if (i < 9)
                a.push_back (atoi (s.substr (0, i).c_str()));
            else
                a.push_back (atoi (s.substr (i-9, 9).c_str()));

        while (a.size() > 1 && a.back() == 0) a.pop_back();
    }

    int size(){
        return a.size();
    }

    int& back(){
        return a.back();
    }

    void pop_back(){
        a.pop_back();
    }

    int& operator [](int i){
        return a[i];
    }

    bigInt operator = (const bigInt &another){
        a = another.a;
        return *this;
    }

    bigInt operator = (const string &x){
        bigInt res(x);
        return res;
    }

    bigInt operator + (ll x){
        bigInt c = *this;
        c += x;
        return c;
    }

    bigInt operator += (ll x){
        bigInt c(x);
        *this += c;
        return *this;
    }

    bigInt operator + (bigInt b){
        bigInt c = *this;
        c += b;
        return c;
    }

    bigInt operator += (bigInt b){
        int carry = 0;
        for (int i = 0; i < max((int)a.size(), b.size()) || carry; ++i) {
            if (i == a.size())
                a.push_back (0);
            a[i] += carry + (i < b.size() ? b[i] : 0);
            carry = a[i] >= base;
            if (carry)  a[i] -= base;
        }
        return *this;
    }

    bigInt operator - (bigInt b){
        bigInt c = *this;
        c -= b;
        return c;
    }

    bigInt operator -= (bigInt b){
        int carry = 0;
        for (int i=0; i<b.size() || carry; ++i) {
            a[i] -= carry + (i < b.size() ? b[i] : 0);
            carry = a[i] < 0;
            if (carry)  a[i] += base;
        }
        while (a.size() > 1 && a.back() == 0)
            a.pop_back();
        return *this;
    }

    bigInt operator * (int b){
        bigInt c = *this;
        c *= b;
        return c;
    }

    bigInt operator *= (int b){
        int carry = 0;
        for (int i=0; i<a.size() || carry; ++i) {
            if (i == a.size())
                a.push_back (0);
            long long cur = carry + a[i] * 1ll * b;
            a[i] = int (cur % base); // x %= y is equal to x -= x/y*y;
            carry = int (cur / base);
        }
        while (a.size() > 1 && a.back() == 0)
            a.pop_back();
        return *this;
    }

    bigInt operator * (bigInt b){ // O(n^2)
        bigInt c = *this;
        c *= b;
        return c;
    }

    bigInt operator *= (bigInt b){ // O(n^2)
        bigInt c;
        c.a.resize((a.size()+b.size()));
        for (int i=0; i<a.size(); ++i)
            for (int j=0, carry=0; j<(int)b.size() || carry; ++j) {
                long long cur = c[i+j] + a[i] * 1ll * (j < (int)b.size() ? b[j] : 0) +
                carry;
                c[i+j] = int (cur % base);
                carry = int (cur / base);
            }
        while (c.size() > 1 && c.back() == 0)
            c.pop_back();
        this->a = c.a;
        return *this;
    }

    bigInt operator / (int b){
        bigInt c = *this;
        return c /= b;
    }

    bigInt operator /= (int b){
        int carry = 0;
        for (int i=(int)a.size()-1; i>=0; --i) {
            long long cur = a[i] + carry * 1ll * base;
            a[i] = int (cur / b);
            carry = int (cur % b);
        }
        while (a.size() > 1 && a.back() == 0)
            a.pop_back();
        return *this;
    }

    int operator % (int b){
        bigInt c = *this;
        int carry = 0;
        for (int i=(int)c.size()-1; i>=0; --i) {
            long long cur = c[i] + carry * 1ll * base;
            c[i] = int (cur / b);
            carry = int (cur % b);
        }
        while (c.size() > 1 && c.back() == 0)
            c.pop_back();
        return carry;
    }

    bool operator == (const bigInt &b){
        return b.a == a;
    }

    bool operator < (bigInt b){
        if(a.size() < b.size()) return 1;
        if(a.size() > b.size()) return 0;
        for(int i = (int)a.size()-1; i >= 0; i--){
            if(a[i] < b[i]) return 1;
            if(b[i] < a[i]) return 0;
        }
        return 0;
    }

```

```
    bool operator <= (bigInt b){
        bigInt temp = *this;
        return temp < b || temp == b;
    }

};

int main() {

    string x = "123456789";
    bigInt a = x;
    bigInt b(x);
    b = a/3;
    a.print();
    b.print();

    return 0;
}

// 123456789987654

// centroid decomposition

set<int> G[N]; // adjacency list (note that this is stored in set, not vector)
int sz[N], pa[N];

int dfs(int u, int p) {
  sz[u] = 1;
  for(auto v : G[u]) if(v != p) {
    sz[u] += dfs(v, u);
  }
  return sz[u];
}
int centroid(int u, int p, int n) {
  for(auto v : G[u]) if(v != p) {
    if(sz[v] > n / 2) return centroid(v, u, n);
  }
  return u;
}
void build(int u, int p) {
  int n = dfs(u, p);
  int c = centroid(u, p, n);
  if(p == -1) p = c;
  pa[c] = p;

  vector<int> tmp(G[c].begin(), G[c].end());
  for(auto v : tmp) {
    G[c].erase(v); G[v].erase(c);
    build(v, c);
  }
}
```

```
// convex hull trick

#include "bits/stdc++.h"

using namespace std;
using ll = long long;

const int N = 1e5+10;
const double EPS = 1E-9;
int n;
ll a[N], b[N], dp[N];
// if to line are parallel then
// if the problem is max dp then then take the line with max c, else take the line with min c
struct Line {
    ll m, c;
    Line(ll m, ll c){
        this->c = c;
        this->m = m;
    }

    ll operator()(ll x){
        return m*x + c;
    }

    double intersect(Line another){
        return (double)(this->c - another.c)/(double)(another.m - this->m);
    }

};

ll calc(vector<Line> &v, vector<double> &rng, double x){
    int dist = upper_bound(rng.begin(), rng.end(), x) - rng.begin() - 1;
    return v[dist](x);
}

int main () {

    cin >> n;
    for(int i = 0; i < n; i++) cin >> a[i];
    for(int i = 0; i < n; i++) cin >> b[i];

    dp[0] = 0;
    vector<Line> v;
    v.push_back({b[0], 0});
    vector<double> rng;
    rng.push_back(-1e17);

    for(int i = 1; i < n; i++){
        dp[i] = calc(v, rng, a[i]);
        Line newLine(b[i], dp[i]);
        while(v.size() >= 2 && v.end()[-2].intersect(newLine) + EPS <= v.end()[-2].intersect(v.end()[-1])){
            v.pop_back();
            rng.pop_back();
        }
        v.push_back(newLine);
        rng.push_back(v.end()[-2].intersect(v.end()[-1]));
    // when use this trick, slopes have to be sorted in increasing order
    // if not sorted, then you can modify the logic and still correct
    }

    cout << dp[n-1] << endl;


    return 0;
}
```

```cpp
// dsu on tree

void add(int cur, int pre, int bg){
    cnt[color[cur]]++;
    for(int i : tree[cur]){
        if(i != bg) add(i, cur, bg);
    }
}

void del(int cur, int pre){
    cnt[color[cur]]--;
    for(int i : tree[cur]){
        del(i, cur);
    }
}

void dfs(int cur, int pre, int keep){
    int mx = 0, bg = -1;

    for(int i : tree[cur]){
    if(i == pre) continue;
        if(si[i] > mx){
            mx = si[i];
            bg = i;
        }
    }


    for(int i : tree[cur]){
    if(i == pre) continue;
        if(i != bg) dfs(i, cur, 0);
    }

    if(bg+1) dfs(bg, cur, 1);

    add(cur, pre, bg);

    for(auto [f, s] : qu[cur]){
        int odd = __builtin_popcount(cnt[f]);
        if(odd < 2) ok[s] = 1;
    }

    if(keep == 0) del(cur, pre);

}
```

```cpp
// fft

const double PI = acos(-1);
struct base {
    double a, b;
    base(double a = 0, double b = 0) : a(a), b(b) {}
    const base operator + (const base &c) const
        { return base(a + c.a, b + c.b); }
    const base operator - (const base &c) const
        { return base(a - c.a, b - c.b); }
    const base operator * (const base &c) const
        { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};

void fft(vector<base> &p, bool inv = 0) {
    int n = p.size(), i = 0;
    for(int j = 1; j < n - 1; ++j) {
        for(int k = n >> 1; k > (i ^= k); k >>= 1);
        if(j < i) swap(p[i], p[j]);
    }
    for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
        double ang = 2 * PI / m;
        base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
        for(int i = 0, j, k; i < n; i += m) {
            for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w * wn) {
                base t = w * p[j + l];
                p[j + l] = p[j] - t;
                p[j] = p[j] + t;
            }
        }
    }
    if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end()); // don't forget to reverse in main when needed
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] = fa[i] * fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].a);
    return result;
}
```

```
// gauss elemination xor

struct Basis {
    const static int N = 61; // N = log(MAX_A)

    vector<ll> a;
    int sz = 0;

    Basis() { a.resize(N, 0); }

    Basis(vector<ll> v) { a.resize(N, 0); for(ll x : v) add(x); }

    void add(Basis another) { for(ll x : another.a) add(x); }

    void add(ll x) { // O(N)
        for (int i = N - 1; ~i; --i) {
            if (x & (1LL << i)) {
                if (a[i]) {
                    x ^= a[i];
                } else {
                    a[i] = x;
                    sz++;
                    break;
                }
            }
        }
    }

    bool can(ll x) {
        for (int i = N - 1; ~i; --i) {
            if (x & (1 << i)) {
                x ^= a[i];
            }
        }
        return x == 0;
    }

    ll getMax() {
        ll result = 0;
        for (int i = N - 1; ~i; --i) {
            if ((result ^ a[i]) > result) {
                result ^= a[i];
            }
        }
        return result;
    }

};
```

```cpp
// graph matching kuhn algorithm

int n;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...
    // zero-base indexed
    // one graph, not explicitly divided into to parts

    mt.assign(n, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < n; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

```cpp
// heavy-light decomposition

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

struct segment_tree {
    int n;
    vector<int> st;

    segment_tree(){}

    segment_tree(const vector<int> &v){
        n = v.size();
        st.resize(n*4+5, 0);
        build(1, 0, n-1, v);
    }

    void init(const vector<int> &v){
        st.clear();
        n = v.size();
        st.resize(n*4+5);
        build(1, 0, n-1, v);
    }

    int mrg(int l, int r){
        return max(l, r);
    }

    void build(int v, int l, int r, const vector<int> &ve){
        if(l == r) st[v] = ve[l];
        else {
            int mid = (l+r)/2;
            build(2*v, l, mid, ve);
            build(2*v+1, mid+1, r, ve);
            st[v] = mrg(st[2*v+1], st[2*v]);
        }
    }

    int get(int v, int l, int r, int tl, int tr){
        if(tl > tr) return 0;
        if(l == tl && r == tr) return st[v];
        int mid = (l+r)/2;
        int f = get(2*v, l, mid, tl, min(tr, mid));
        int s = get(2*v+1, mid+1, r, max(mid+1, tl), tr);
        return mrg(f, s);
    }

    int query(int l, int r){
        // assert(l <= r);
        return get(1, 0, n-1, l, r);
    }

    void upd(int v, int l, int r, int idx, int val){
        if(l == r) st[v] = val;
        else {
            int mid = (l+r)/2;
            if(idx <= mid) upd(2*v, l, mid, idx, val);
            else upd(2*v+1, mid+1, r, idx, val);
            st[v] = mrg(st[2*v+1], st[2*v]);
        }
    }

    void upd(int idx, int val){
        upd(1, 0, n-1, idx, val);
    }

    void clear(){
        n = 0;
        st.clear();
    }

};

const int N = 2e5+10;
vector<pair<int, int> > tree[N];
vector<int> part[N];
segment_tree st[N];
int head[N], sz[N], depth[N], parent[N], pos[N];//, val[N];

void init(int v, int pre, int h){
    depth[v] = h;
    sz[v] = 1;
    parent[v] = pre;
    for(auto [i, s] : tree[v]){
        if(i == pre) continue;
        init(i, v, h+1);
        sz[v] += sz[i];
    }
}

void dfs(int v, int pre, int h, int val){ // decomposition
    int mx = -1;
    head[v] = h;
    pos[v] = part[h].size();
    part[h].push_back(val);
    for(auto [i, s] : tree[v]){
        if(i == pre) continue;
        if(mx == -1 || sz[i] > sz[mx]){
            mx = i;
            val = s;
        }
    }

    if(mx != -1){
        dfs(mx, v, h, val);
    }

    for(auto [i, s] : tree[v]){
        if(i == pre || i == mx) continue;
        dfs(i, v, i, s);
    }

}

void upd_edge(int v, int u, int val){ // edit something in edge between vertexes u and v
    if(parent[u] != v) swap(v, u);
    assert(parent[u] == v);
    // cout << "u " << u << " v " << v << " val " << val << endl;
    // cout << "head " << head[u] << " pos " << pos[u] << endl;
    st[head[u]].upd(pos[u], val);
    part[head[u]][pos[u]] = val;
}

int query(int v, int u){
    int res = 0;

    while(head[v] != head[u]){
        if(depth[head[u]] < depth[head[v]]) swap(u, v);
        assert(pos[head[u]] == 0);
        res = max(res, st[head[u]].query(0, pos[u]));
        u = parent[head[u]];
    }

    if(depth[u] < depth[v]) swap(u, v);

    // cout << "pos " << pos[v] << " and " << pos[u] << endl;
    res = max(res, st[head[u]].query(pos[v]+1, pos[u]));
    return res;
}

void clear(int n){
    for(int i = 0; i < n+3; i++){
        tree[i].clear();
        part[i].clear();
        st[i].clear();
        head[i] = sz[i] = depth[i] = parent[i] = pos[i] = -1;
    }
}

char buf[20];

string read(){
    scanf("%s", buf);
    return buf;
}

int main()
{

    int t;
    cin >> t;
    while(t--){
        char x;
        // scanf("%c", &x);
        int n;
        scanf("%d", &n);
        clear(n);
        vector<pair<int, int>> edges;
        for(int i = 0; i < n-1; i++){
            int x, y, w;
            scanf("%d%d%d", &x, &y, &w);
            edges.emplace_back(x, y);
            tree[x].push_back({y, w});
            tree[y].push_back({x, w});
        }

        int root = 1;
        init(root, root, 0);
        // the val parameter in decomposition function depends on type of operations you do, for sum operations you pass 0, for max you pass -INF, ans so on
        dfs(root, root, root, 0);

        for(int i = 1; i <= n; i++){
            // cout << "i " << i << " and " << head[i] << endl;
            if(head[i] == i) st[i].init(part[i]);
        }

        string q = "";
        while(true){
            int u, v;
            q = read();
            if(q == "DONE") break;
            scanf("%d%d", &v, &u);
            if(q == "CHANGE"){
                // cout << "hi " << edges[v-1].first << " and " << edges[v-1].second << endl;
                upd_edge(edges[v-1].first, edges[v-1].second, u);
            } else if(q == "QUERY"){
                printf("%d\n",  query(u, v));
            }
        }

    }

    return 0;
```

```
}
/*

1

3
1 2 1
2 3 2
QUERY 1 1
CHANGE 1 3
QUERY 1 2
QUERY 1 1
QUERY 2 2
QUERY 3 3
DONE

*/

```

```cpp
// hopcroft karp (graph matching)

// soruce: https://github.com/shah-deep/Graph-Theory-Algos
#include<bits/stdc++.h>

using namespace std;

const int N = 2e5 + 10;
vector<int> G[N];

bool bfs(vector<int> &setU, vector<int> &setV, vector<int> &dist, int m)
{
    queue<int> que;

    for (int u=1; u<=m; u++)
    {
        if (setU[u]==0)
        {
            dist[u] = 0;
            que.push(u);
        }

        else dist[u] = INT_MAX;
    }

    dist[0] = INT_MAX;

    while (!que.empty())
    {
        int u = que.front();
        que.pop();

        if (dist[u] < dist[0])
        {
            for (int i = 0; i < G[u].size(); ++i)
            {
                int v = G[u][i];

                if (dist[setV[v]] == INT_MAX)
                {
                    dist[setV[v]] = dist[u] + 1;
                    que.push(setV[v]);
                }
            }
        }
    }

    return (dist[0] != INT_MAX);
}

bool dfs(int u, vector<int> &setU, vector<int> &setV, vector<int> &dist)
{
    if (u != 0)
    {

        for (int i = 0; i < G[u].size(); ++i) {

            int v = G[u][i];
            if ((dist[setV[v]] == dist[u]+1) && dfs(setV[v], setU, setV, dist))
            {
                setU[u] = v;
                setV[v] = u;
                return true;
            }
        }

        dist[u] = INT_MAX;
        return false;
    }

    return true;
}

void hopcroft_karp(int n1, int n2)
{

    vector<int> setU(n1+1, 0), setV(n2+1, 0), dist(n1+1);

    int cnt = 0;

    while (bfs(setU, setV, dist, n1)) {
        for (int u=1; u<=n1; u++) {
            if (!setU[u] && dfs(u, setU, setV, dist))
                cnt++;
        }
    }

    cout << "\nMaximum Matching:" << endl;
    for(int i=1; i <= n1; i++){
        if(setU[i] == 0) continue;
        cout << "i " << i << " and " << setU[i] << endl;
    }

    cout << "\nCount: " << cnt << endl;
}

int main()
{
    int n, m; // vertexes, edges
    cin >> n >> m;
    for(int i=0; i<m; i++){
        int u, v; // one-base indexed
        cin >> u >> v;
        G[u].push_back(v);
    }

    hopcroft_karp(n, n);

    return 0;
}

```

```cpp
// hungarian algorithm

#include <bits/stdc++.h>

using namespace std ;
using ll = long long;

/////////////////////////////////////////////////////////////////////////////
// Hungarian.cpp: Implementation file for Class HungarianAlgorithm.
//
// This is a C++ wrapper with slight modification of a hungarian algorithm implementation by Markus Buehren.
// The original implementation is a few mex-functions for use in MATLAB, found here:
// http://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem
//
// Both this code and the orignal code are published under the BSD license.
// by Cong Ma, 2016
//

#ifndef HUNGARIAN_H
#define HUNGARIAN_H

using namespace std;


class HungarianAlgorithm
{
public:
    HungarianAlgorithm();
    ~HungarianAlgorithm();
    double Solve(vector <vector<double> >& DistMatrix, vector<int>& Assignment);

private:
    void assignmentoptimal(int *assignment, double *cost, double *distMatrix, int nOfRows, int nOfColumns);
    void buildassignmentvector(int *assignment, bool *starMatrix, int nOfRows, int nOfColumns);
    void computeassignmentcost(int *assignment, double *cost, double *distMatrix, int nOfRows);
    void step2a(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim);
    void step2b(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim);
    void step3(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim);
    void step4(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim, int row, int col);
    void step5(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim);
};


#endif


HungarianAlgorithm::HungarianAlgorithm(){}
HungarianAlgorithm::~HungarianAlgorithm(){}


//********************************************************//
// A single function wrapper for solving assignment problem.
//********************************************************//
```

```
double HungarianAlgorithm::Solve(vector <vector<double> >& DistMatrix, vector<int>& Assignment)
{
    unsigned int nRows = DistMatrix.size();
    unsigned int nCols = DistMatrix[0].size();

    double *distMatrixIn = new double[nRows * nCols];
    int *assignment = new int[nRows];
    double cost = 0.0;

    // Fill in the distMatrixIn. Mind the index is "i + nRows * j".
    // Here the cost matrix of size MxN is defined as a double precision array of N*M elements.
    // In the solving functions matrices are seen to be saved MATLAB-internally in row-order.
    // (i.e. the matrix [1 2; 3 4] will be stored as a vector [1 3 2 4], NOT [1 2 3 4]).
    for (unsigned int i = 0; i < nRows; i++)
        for (unsigned int j = 0; j < nCols; j++)
            distMatrixIn[i + nRows * j] = DistMatrix[i][j];

    // call solving function
    assignmentoptimal(assignment, &cost, distMatrixIn, nRows, nCols);

    Assignment.clear();
    for (unsigned int r = 0; r < nRows; r++)
        Assignment.push_back(assignment[r]);

    delete[] distMatrixIn;
    delete[] assignment;
    return cost;
}


//********************************************************//
// Solve optimal solution for assignment problem using Munkres algorithm, also known as Hungarian Algorithm.
//********************************************************//
void HungarianAlgorithm::assignmentoptimal(int *assignment, double *cost, double *distMatrixIn, int nOfRows, int nOfColumns)
{
    double *distMatrix, *distMatrixTemp, *distMatrixEnd, *columnEnd, value, minValue;
    bool *coveredColumns, *coveredRows, *starMatrix, *newStarMatrix, *primeMatrix;
    int nOfElements, minDim, row, col;

    /* initialization */
    *cost = 0;
    for (row = 0; row<nOfRows; row++)
        assignment[row] = -1;

    /* generate working copy of distance Matrix */
    /* check if all matrix elements are positive */
    nOfElements = nOfRows * nOfColumns;
    distMatrix = (double *)malloc(nOfElements * sizeof(double));
    distMatrixEnd = distMatrix + nOfElements;

    for (row = 0; row<nOfElements; row++)
    {
        value = distMatrixIn[row];
        if (value < 0)
            cerr << "All matrix elements have to be non-negative." << endl;
        distMatrix[row] = value;
    }


    /* memory allocation */
    coveredColumns = (bool *)calloc(nOfColumns, sizeof(bool));
    coveredRows = (bool *)calloc(nOfRows, sizeof(bool));
    starMatrix = (bool *)calloc(nOfElements, sizeof(bool));
    primeMatrix = (bool *)calloc(nOfElements, sizeof(bool));
    newStarMatrix = (bool *)calloc(nOfElements, sizeof(bool)); /* used in step4 */

    /* preliminary steps */
    if (nOfRows <= nOfColumns)
    {
        minDim = nOfRows;

        for (row = 0; row<nOfRows; row++)
        {
            /* find the smallest element in the row */
            distMatrixTemp = distMatrix + row;
            minValue = *distMatrixTemp;
            distMatrixTemp += nOfRows;
            while (distMatrixTemp < distMatrixEnd)
            {
                value = *distMatrixTemp;
                if (value < minValue)
                    minValue = value;
                distMatrixTemp += nOfRows;
            }

            /* subtract the smallest element from each element of the row */
            distMatrixTemp = distMatrix + row;
            while (distMatrixTemp < distMatrixEnd)
            {
                *distMatrixTemp -= minValue;
                distMatrixTemp += nOfRows;
            }
        }

        /* Steps 1 and 2a */
        for (row = 0; row<nOfRows; row++)
            for (col = 0; col<nOfColumns; col++)
                if (fabs(distMatrix[row + nOfRows*col]) < DBL_EPSILON)
                    if (!coveredColumns[col])
                    {
                        starMatrix[row + nOfRows*col] = true;
                        coveredColumns[col] = true;
                        break;
                    }
    }
    else /* if(nOfRows > nOfColumns) */
    {
        minDim = nOfColumns;

        for (col = 0; col<nOfColumns; col++)
        {
            /* find the smallest element in the column */
            distMatrixTemp = distMatrix + nOfRows*col;
            columnEnd = distMatrixTemp + nOfRows;

            minValue = *distMatrixTemp++;
            while (distMatrixTemp < columnEnd)
            {
                value = *distMatrixTemp++;
                if (value < minValue)
                    minValue = value;
            }

            /* subtract the smallest element from each element of the column */
            distMatrixTemp = distMatrix + nOfRows*col;
            while (distMatrixTemp < columnEnd)
                *distMatrixTemp++ -= minValue;
        }

        /* Steps 1 and 2a */
        for (col = 0; col<nOfColumns; col++)
            for (row = 0; row<nOfRows; row++)
                if (fabs(distMatrix[row + nOfRows*col]) < DBL_EPSILON)
```

```
                    if (!coveredRows[row])
                    {
                        starMatrix[row + nOfRows*col] = true;
                        coveredColumns[col] = true;
                        coveredRows[row] = true;
                        break;
                    }
        for (row = 0; row<nOfRows; row++)
            coveredRows[row] = false;

    }

    /* move to step 2b */
    step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);

    /* compute cost and remove invalid assignments */
    computeassignmentcost(assignment, cost, distMatrixIn, nOfRows);

    /* free allocated memory */
    free(distMatrix);
    free(coveredColumns);
    free(coveredRows);
    free(starMatrix);
    free(primeMatrix);
    free(newStarMatrix);

    return;
}

/********************************************************/
void HungarianAlgorithm::buildassignmentvector(int *assignment, bool *starMatrix, int nOfRows, int nOfColumns)
{
    int row, col;

    for (row = 0; row<nOfRows; row++)
        for (col = 0; col<nOfColumns; col++)
            if (starMatrix[row + nOfRows*col])
            {
#ifdef ONE_INDEXING
                assignment[row] = col + 1; /* MATLAB-Indexing */
#else
                assignment[row] = col;
#endif
                break;
            }
}

/********************************************************/
void HungarianAlgorithm::computeassignmentcost(int *assignment, double *cost, double *distMatrix, int nOfRows)
{
    int row, col;

    for (row = 0; row<nOfRows; row++)
    {
        col = assignment[row];
        if (col >= 0)
            *cost += distMatrix[row + nOfRows*col];
    }
}

/********************************************************/
void HungarianAlgorithm::step2a(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim)
{
    bool *starMatrixTemp, *columnEnd;
    int col;

    /* cover every column containing a starred zero */
    for (col = 0; col<nOfColumns; col++)
    {
        starMatrixTemp = starMatrix + nOfRows*col;
        columnEnd = starMatrixTemp + nOfRows;
        while (starMatrixTemp < columnEnd){
            if (*starMatrixTemp++)
            {
                coveredColumns[col] = true;
                break;
            }
        }
    }

    /* move to step 3 */
    step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
    coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
}

/********************************************************/
void HungarianAlgorithm::step2b(int *assignment, double *distMatrix, bool *starMatrix,
bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
nOfRows, int nOfColumns, int minDim)
{
    int col, nOfCoveredColumns;

    /* count covered columns */
    nOfCoveredColumns = 0;
    for (col = 0; col<nOfColumns; col++)
        if (coveredColumns[col])
            nOfCoveredColumns++;

    if (nOfCoveredColumns == minDim)
    {
        /* algorithm finished */
        buildassignmentvector(assignment, starMatrix, nOfRows, nOfColumns);
    }
    else
    {
        /* move to step 3 */
        step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
        coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
    }

}

/********************************************************/
void HungarianAlgorithm::step3(int *assignment, double *distMatrix, bool *starMatrix,
bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
nOfRows, int nOfColumns, int minDim)
{
    bool zerosFound;
    int row, col, starCol;

    zerosFound = true;
    while (zerosFound)
    {
        zerosFound = false;
        for (col = 0; col<nOfColumns; col++)
            if (!coveredColumns[col])
                for (row = 0; row<nOfRows; row++)
                    if ((!coveredRows[row]) && (fabs(distMatrix[row + nOfRows*col]) <
                    DBL_EPSILON))
                    {
                        /* prime zero */
                        primeMatrix[row + nOfRows*col] = true;

                        /* find starred zero in current row */
                        for (starCol = 0; starCol<nOfColumns; starCol++)
                            if (starMatrix[row + nOfRows*starCol])
                                break;

                        if (starCol == nOfColumns) /* no starred zero found */
                        {
                            /* move to step 4 */
                            step4(assignment, distMatrix, starMatrix, newStarMatrix,
                            primeMatrix, coveredColumns, coveredRows, nOfRows,
                            nOfColumns, minDim, row, col);
                            return;
                        }
                        else
                        {
                            coveredRows[row] = true;
                            coveredColumns[starCol] = false;
                            zerosFound = true;
                            break;
                        }
                    }
    }

    /* move to step 5 */
    step5(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
    coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
}

/********************************************************/
void HungarianAlgorithm::step4(int *assignment, double *distMatrix, bool *starMatrix,
bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
nOfRows, int nOfColumns, int minDim, int row, int col)
{
    int n, starRow, starCol, primeRow, primeCol;
    int nOfElements = nOfRows*nOfColumns;

    /* generate temporary copy of starMatrix */
    for (n = 0; n<nOfElements; n++)
        newStarMatrix[n] = starMatrix[n];

    /* star current zero */
    newStarMatrix[row + nOfRows*col] = true;

    /* find starred zero in current column */
    starCol = col;
    for (starRow = 0; starRow<nOfRows; starRow++)
        if (starMatrix[starRow + nOfRows*starCol])
            break;

    while (starRow<nOfRows)
    {
        /* unstar the starred zero */
        newStarMatrix[starRow + nOfRows*starCol] = false;

        /* find primed zero in current row */
        primeRow = starRow;
        for (primeCol = 0; primeCol<nOfColumns; primeCol++)
            if (primeMatrix[primeRow + nOfRows*primeCol])
                break;

        /* star the primed zero */
        newStarMatrix[primeRow + nOfRows*primeCol] = true;

        /* find starred zero in current column */
        starCol = primeCol;
        for (starRow = 0; starRow<nOfRows; starRow++)
            if (starMatrix[starRow + nOfRows*starCol])
                break;
    }
    /* use temporary copy as new starMatrix */
    /* delete all primes, uncover all rows */
    for (n = 0; n<nOfElements; n++)
    {
        primeMatrix[n] = false;
        starMatrix[n] = newStarMatrix[n];
    }
    for (n = 0; n<nOfRows; n++)
        coveredRows[n] = false;

    /* move to step 2a */
    step2a(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
}

/**********************************************************/
void HungarianAlgorithm::step5(int *assignment, double *distMatrix, bool *starMatrix, bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int nOfRows, int nOfColumns, int minDim)
{
    double h, value;
    int row, col;

    /* find smallest uncovered element h */
    h = DBL_MAX;
    for (row = 0; row<nOfRows; row++)
        if (!coveredRows[row])
            for (col = 0; col<nOfColumns; col++)
                if (!coveredColumns[col])
                {
                    value = distMatrix[row + nOfRows*col];
                    if (value < h)
                        h = value;
                }

    /* add h to each covered row */
    for (row = 0; row<nOfRows; row++)
        if (coveredRows[row])
            for (col = 0; col<nOfColumns; col++)
                distMatrix[row + nOfRows*col] += h;

    /* subtract h from each uncovered column */
    for (col = 0; col<nOfColumns; col++)
        if (!coveredColumns[col])
            for (row = 0; row<nOfRows; row++)
                distMatrix[row + nOfRows*col] -= h;

    /* move to step 3 */
    step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix, coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
}

int main()
{

    int n;
    scanf("%d", &n);

    vector<vector<double>> a(n, vector<double>(n));

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            int x;
            scanf("%d", &x);
            a[i][j] = log2(x);
            a[i][j] = 20.0 - a[i][j];
        }
    }

    vector<int> ans(n, -1), ans1 = ans;
    HungarianAlgorithm var;
    double cost = var.Solve(a, ans);

    for(int i = 0; i < n; i++){
        ans1[ans[i]] = i;
    }

    for(int i : ans1) cout << i+1 << " ";

    return 0 ;
}


```

```
// kth element log(n) (sparse segment tree)

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

struct node {
    int sum = 0;
    node *left, *right;
    node(){
        left = right = nullptr;
    }
} *root = new node();

void add(int num, node* root, int l, int r){ // add num to the tree
    if(l == r){
        root->sum++;
        return;
    }
    int mid = (l+r)/2;
    if(num <= mid){
        if(root->left == nullptr) root->left = new node();
        add(num, root->left, l, mid);
    } else {
        if(root->right == nullptr) root->right = new node();
        add(num, root->right, mid+1, r);
    }

    root->sum = 0;
    if(root->left) root->sum += root->left->sum;
    if(root->right) root->sum += root->right->sum;

}

void del(int num, node* root, int l, int r){ // delete num-th element (need to check if the num-th element is exist)
    if(l == r){ // need to check if the
        root->sum--;
        return;
    }
    int mid = (l+r)/2;
    if(root->left != nullptr && root->left->sum >= num){
        del(num, root->left, l, mid);
    } else {
        if(root->left != nullptr) num -= root->left->sum;
        del(num, root->right, mid+1, r);
    }

    root->sum = 0;
    if(root->left) root->sum += root->left->sum;
    if(root->right) root->sum += root->right->sum;

}

int get(int num, node* root, int l, int r){ // get k-th element
    if(l == r){
        return r;
    }
    int mid = (l+r)/2;
    if(root->left != nullptr && root->left->sum >= num){
        return get(num, root->left, l, mid);
    } else {
        if(root->left != nullptr) num -= root->left->sum;
        return get(num, root->right, mid+1, r);
    }
}

int getSum(node* root, int l, int r, int tl, int tr){
    if(l > r) return 0;
    if(l == tl &&  r == tr) return root->sum;
    int mid = (l+r)/2;
    int f = 0, s = 0;
    if(root->left != nullptr) f = getSum(root->left, l, mid, tl, min(mid, tr));
    if(root->right != nullptr) s = getSum(root->right, mid+1, r, max(mid+1, tl), tr);
    return f + s;

}

void prtAny(node* root, int l, int r){
    assert(root != nullptr);
    if(l == r){
        assert(root->sum > 0);
        printf("%d", l);
        return;
    }

    int mid = (l+r)/2;
    if(root->left != nullptr && root->left->sum != 0) prtAny(root->left, l, mid);
    else prtAny(root->right, mid+1, r);

}

int main()
{

    int n, q;

    scanf("%d%d", &n, &q);

    for(int i = 0; i < n; i++){
        int x;
        scanf("%d", &x);
        add(x, root, 1, n);
    }

    for(int i = 0; i < q; i++){
        int x;
        scanf("%d", &x);
        if(x < 0){
            del(-x, root, 1, n);
        } else {
            add(x, root, 1, n);
        }
    }

    if(root->sum == 0) printf("0");
    else {
        prtAny(root, 1, n);
    }

    return 0;
}
```

```cpp
// li chao tree

typedef long long ll;

const int C = (int)1e5 + 5;
const int N = (int)1e5 + 5;
const ll inf = (ll)1e18;

struct Line {
  ll m, b;
  ll operator()(ll x) { return m * x + b; }
};
struct Node {
  Line seg;
  Node *lson, *rson;
  Node(Line _seg): seg(_seg), lson(0), rson(0) {}
};
void insert(int l, int r, Line seg, Node* o) {
  if(l + 1 == r) {
    if(seg(l) < o->seg(l)) o->seg = seg;
    return;
  }
  int mid = (l + r) >> 1;
  if(seg.m < o->seg.m) swap(seg, o->seg);
  if(o->seg(mid) > seg(mid)) {
    swap(seg, o->seg);
    if(o->rson) insert(mid, r, seg, o->rson);
    else o->rson = new Node(seg);
  }
  else {
    if(o->lson) insert(l, mid, seg, o->lson);
    else o->lson = new Node(seg);
  }
}
ll query(int l, int r, int x, Node* o) {
  if(l + 1 == r) return o->seg(x);
  int mid = (l + r) >> 1;
  if(x < mid && o->lson) return min(o->seg(x), query(l, mid, x, o->lson));
  else if(o->rson) return min(o->seg(x), query(mid, r, x, o->rson));
  return o->seg(x);
}
void del(Node* o) {
  if(o->lson) del(o->lson);
  if(o->rson) del(o->rson);
  delete o;
}
```

```
// Linear Diophantine Equation

int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

```cpp
// lyndon factorization duval algorithm

vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}
```

```cpp
// manacher algorithm

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 0, r = -1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}

char buf[1000010];

int main()
{

    scanf("%s", buf);
    string s = buf;
    vector<int> palindorm = manacher(s);

    string s1 = "";
    for(int i = 0; i < s.length(); i++){
        s1 += s[i];
        s1 += "#";
    }

    s = s1;
    s.pop_back();
    cout << s << endl;
    int ans = 0; // number of subpalindromes
    for(int i = 0; i < palindorm.size(); i += 1){
        cout << palindorm[i] << " ";
        ans += palindorm[i]/2;
        // if(s[i] == "#") even palindrome with (s[i]-1)/2 length, centers in s[i] and s[i+1]
        // else odd palindrome with s[i]/2 length, centers in s[i]
    }
    cout << endl;

    cout << "ans " << ans << endl;

    return 0;
}
```

```cpp
// max flow dinic

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

```
// max flow VE complexity, (improved push preflow)

const int inf = 1000000000;

int n;
vector<vector<int>> capacity, flow;
vector<int> height, excess;

void push(int u, int v)
{
    int d = min(excess[u], capacity[u][v] - flow[u][v]);
    flow[u][v] += d;
    flow[v][u] -= d;
    excess[u] -= d;
    excess[v] += d;
}

void relabel(int u)
{
    int d = inf;
    for (int i = 0; i < n; i++) {
        if (capacity[u][i] - flow[u][i] > 0)
            d = min(d, height[i]);
    }
    if (d < inf)
        height[u] = d + 1;
}

vector<int> find_max_height_vertices(int s, int t) {
    vector<int> max_height;
    for (int i = 0; i < n; i++) {
        if (i != s && i != t && excess[i] > 0) {
            if (!max_height.empty() && height[i] > height[max_height[0]])
                max_height.clear();
            if (max_height.empty() || height[i] == height[max_height[0]])
                max_height.push_back(i);
        }
    }
    return max_height;
}

int max_flow(int s, int t)
{
    height.assign(n, 0);
    height[s] = n;
    flow.assign(n, vector<int>(n, 0));
    excess.assign(n, 0);
    excess[s] = inf;
    for (int i = 0; i < n; i++) {
        if (i != s)
            push(s, i);
    }

    vector<int> current;
    while (!(current = find_max_height_vertices(s, t)).empty()) {
        for (int i : current) {
            bool pushed = false;
            for (int j = 0; j < n && excess[i]; j++) {
                if (capacity[i][j] - flow[i][j] > 0 && height[i] == height[j] + 1) {
                    push(i, j);
                    pushed = true;
                }
            }
            if (!pushed) {
                relabel(i);
                break;
            }
        }
    }

    int max_flow = 0;
    for (int i = 0; i < n; i++)
        max_flow += flow[i][t];
    return max_flow;
}
```

```cpp
// min cost max flow dijekstra + johnson's algorithm

// soruce: https://codeforces.com/blog/entry/95823

/*

If there are negative weights then we should calcalute potential by bellman-ford algorithm for the first time

*/

template<typename Cap, typename Cost>
struct mcmf {
    struct edge {
        int v;
        Cap cap, flow;
        Cost cost;
    };
    int n;
    vector<edge> e;
    vector<vector<int>> g;
    vector<Cost> dist, pot;
    vector<Cap> f;
    vector<bool> vis;
    vector<int> par;
    bool n2dijkstra = false;
    mcmf(int n) : n(n), g(n), dist(n), pot(n), f(n), vis(n), par(n) {}
    void add_edge(int u, int v, Cap cap, Cost cost) {
        int k = e.size();
        e.push_back({v, cap, 0, cost});
        e.push_back({u, cap, cap, -cost});
        g[u].push_back(k);
        g[v].push_back(k ^ 1);
    }
    pair<Cap, Cost> solve(int s, int t) {
        Cap flow = 0;
        Cost cost = 0;
        while(true) {
            fill(dist.begin(), dist.end(), numeric_limits<Cost>::max());
            fill(vis.begin(), vis.end(), false);
            dist[s] = 0;
            f[s] = numeric_limits<Cap>::max();
            if(n2dijkstra) {
                while(true) {
                    int x = -1; Cost d = numeric_limits<Cost>::max();
                    for(int i = 0; i < n; i++) {
                        if(!vis[i] && dist[i] < d) {
                            x = i;
                            d = dist[x];
                        }
                    }
                    if(x == -1) break;
                    vis[x] = true;
                    for(int i : g[x]) {
                        Cost d2 = d + e[i].cost + pot[x] - pot[e[i].v];
                        if(!vis[e[i].v] && e[i].flow < e[i].cap && d2 < dist[e[i].v]) {
                            dist[e[i].v] = d2;
                            f[e[i].v] = min(f[x], e[i].cap - e[i].flow);
                            par[e[i].v] = i;
                        }
                    }
                }
            }else {
                priority_queue<pair<Cost, int>, vector<pair<Cost, int>>, greater<>> Q;
                Q.push({0, s});
                while(!Q.empty()) {
                    Cost d; int x;
                    tie(d, x) = Q.top(); Q.pop();
                    if(vis[x]) continue;
                    vis[x] = true;
                    for(int i : g[x]) {
                        Cost d2 = d + e[i].cost + pot[x] - pot[e[i].v];
                        if(!vis[e[i].v] && e[i].flow < e[i].cap && d2 < dist[e[i].v]) {
                            dist[e[i].v] = d2;
                            f[e[i].v] = min(f[x], e[i].cap - e[i].flow);
                            par[e[i].v] = i;
                            Q.push({d2, e[i].v});
                        }
                    }
                }
            }
            if(!vis[t]) break;
            for(int i = 0; i < n; i++) {
                dist[i] += pot[i] - pot[s];
            }
            cost += dist[t] * f[t];
            flow += f[t];
            int x = t;
            while(x != s) {
                e[par[x]].flow += f[t];
                e[par[x] ^ 1].flow -= f[t];
                x = e[par[x] ^ 1].v;
            }
            dist.swap(pot);
        }
        return {flow, cost};
    }
};
```

```
// mo's algorithm

void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure

const int block_size; // 700 or 800 may run better than 750

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

```cpp
// persistent segment tree

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MaxNum = 2e5+10;

struct Vertex {
    Vertex *l, *r;
    ll sum;

    Vertex(ll val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(int tl, int tr) {
    if (tl == tr)
        return new Vertex(0);
    int tm = (tl + tr) / 2;
    return new Vertex(build(tl, tm), build(tm+1, tr));
}

Vertex* update(Vertex* v, int tl, int tr, int pos, int val) {
    if (tl == tr)
        return new Vertex(0LL + val + v->sum);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, val), v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos, val));
}

ll get_sum(Vertex* v, Vertex *left, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r){
        //cout << "l " << l << " r " << r << " sum " << v->sum << " left " << left->sum << endl;
        return v->sum - left->sum;
    }
    int tm = (tl + tr) / 2;
    return get_sum(v->l, left->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, left->r, tm+1, tr, max(l, tm+1), r);
}

void prt(Vertex *v, int tl, int tr){
    if(tl == tr){
        if(v->sum) cout << "tl " << tl << " sum " << v->sum << endl;
        return;
    }
    int mid = (tl + tr)/2;
    prt(v->l, tl, mid);
    prt(v->r, mid+1, tr);
}

int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2, left_count = vr->l->sum - vl->l->sum;
    if (left_count >= k)
        return find_kth(vl->l, vr->l, tl, tm, k);
    return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
}

```

```
int main()
{

    int tl = 0, tr = MaxNum + 1;
    vector<Vertex*> roots;
    roots.push_back(build(tl, tr));
    vector<Vertex*> roots1;
    roots1.push_back(build(tl, tr));
    vector<Vertex*> roots2;
    roots2.push_back(build(tl, tr));
    vector<Vertex*> roots3;
    roots3.push_back(build(tl, tr));
    vector<Vertex*> roots11;
    roots11.push_back(build(tl, tr));
    vector<Vertex*> roots21;
    roots21.push_back(build(tl, tr));


    int n;
    scanf("%d", &n);

    for(int i = 0; i < n; i++){
        int x1, x2, y1, a, b, y2;
        scanf("%d%d%d%d%d%d", &x1, &x2, &y1, &a, &b, &y2);
        roots.push_back(update(roots.back(), tl, tr, x1, y1));
        roots1.push_back(update(roots1.back(), tl, tr, x1, -b));
        roots11.push_back(update(roots11.back(), tl, tr, x2, b));
        roots2.push_back(update(roots2.back(), tl, tr, x1, -a));
        roots21.push_back(update(roots21.back(), tl, tr, x2, a));
        roots3.push_back(update(roots3.back(), tl, tr, x2+1, y2));

    }

    //prt(roots3.back(), tl, tr);

    int m;
    scanf("%d", &m);
    ll last = 0;
    ll l, r, x;
    const ll mod = 1e9;
    while(m--){
        scanf("%lld%lld%lld", &l, &r, &x);
        x = (x + last)%mod;
        if(x <= MaxNum){
            last = get_sum(roots[r], roots[l-1], tl, tr, x, tr);
            last += get_sum(roots1[r], roots1[l-1], tl, tr, x, tr);
            last += get_sum(roots11[r], roots11[l-1], tl, tr, x, tr);
            last += x*get_sum(roots2[r], roots2[l-1], tl, tr, x, tr);
            last += x*get_sum(roots21[r], roots21[l-1], tl, tr, x, tr);
            last += get_sum(roots3[r], roots3[l-1], tl, tr, tl, x);
        } else {
            x = MaxNum;
            last = get_sum(roots3[r], roots3[l-1], tl, tr, tl, x);
        }
        printf("%lld\n", last);
    }

    return 0;
}
```

```
// prefix function + automata

vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
void compute_automaton(string s, vector<vector<int>>& aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}
```

```cpp
// primality test and integer factorization

#include <bits/stdc++.h>

using namespace std;
using ll = uint64_t;

const int N = 100001;

bool prime[N];
vector<int> pr;

void si(){
    prime[0] = prime[1] = 1;
    for(int i = 2; i < N; i++){
        if(prime[i] == 0){
            pr.push_back(i);
            for(ll j = 1LL*i*i; j < N; j += i){
                prime[j] = 1;
            }
        }
    }
}

ll gcd(ll x, ll y){
    while(y){
        x = x%y;
        swap(x, y);
    }
    return x;
}

ll mul(ll a, ll b, ll mod) {
    long long result = 0;
    while (b) {
        if (b & 1)
            result = (result + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return result;
}

ll binpower(ll base, ll e, ll mod)
{
    ll result = 1;
    base %= mod;
    while(e){
        if(e & 1)
            result = mul(result, base, mod);
        base = mul(base, base, mod);
        e >>= 1;
    }
    return result;
}

bool check_compsite(ll n, ll a, ll d, int s)
{
    ll x = binpower(a, d, n);
    if(x == 1 || x == n - 1)
        return false;
    for(int r = 1; r < s; r++){
        x = mul(x, x, n);
        if(x == n - 1)
            return false;
    }
    return true;
};
```

```
bool MillerRabin(ll n) // returns true if n is probably prime, else returns false.
{
    if(n < 2)
        return false;
    int s = 0;
    ll d = n - 1;
    while((d & 1) == 0){
        d >>= 1;
        s++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_compsite(n, a, d, s))
            return false;
    }
    return true;
}

bool isPrime(ll n){
    return MillerRabin(n);
}

ll f(ll x, ll c, ll mod) {
    return (mul(x, x, mod) + c) % mod;
}

ll rho(ll n, ll x0 = 2, ll c = 1) {
    ll x = x0;
    ll y = x0;
    ll g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(y, c, n);
        y = f(y, c, n);
        g = gcd(max(x, y)-min(x, y), n);
    }
    return g;
}

vector<ll> factorize_small(ll n){
    vector<ll> res;
    for(int i : pr){
        if(1LL*i*i > n) break;
        while(n%i == 0){
            n /= i;
            res.push_back(i);
        }
    }
    if(n > 1) res.push_back(n);
    return res;
}

vector<ll> factorize(ll n){
    if(n <= 100000) return factorize_small(n);
    vector<ll> res;
    if(n == 1) return res;
    if(isPrime(n)){
        res.push_back(n);
        return res;
    }
    int x0 = 2, c = 1;
    bool turn = 0;
    ll g = rho(n, x0, c);
    while(g == n){
        if(turn) x0++;
        else c++;
        turn = 1 - turn;
        g = rho(n, x0, c);
        //cout << "n " << n << " g " << g << endl;
    }
    vector<ll> cur = factorize(g);
    for(ll x : cur) res.push_back(x);
    n /= g;
    cur = factorize(n);
    for(ll x : cur) res.push_back(x);
    return res;
}

int main() {

    si();
    int t = 1;
    //cin >> t;

    while(t){

        ll n;
        cin >> n;
        if(n == 0) break;


        vector<ll> factors = factorize(n);
        n = factors.size();
        sort(factors.begin(), factors.end());
        for(int i = 0; i < n;){
            int cnt = 0, j = i;
            while(i < n && factors[j] == factors[i]) cnt++, i++;
            cout << factors[j] << "^" << cnt << " \n"[i == n];
        }

    }

    return 0;

}

```

```
// random number generator

mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
return value between 0 and 2^32 - 1 which is unsigned integer

mt19937_64 rng64(chrono::steady_clock::now().time_since_epoch().count());
the same problematic

random shuffle:
for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[uniform_int_distribution<int>(0, i)(rng32)]);
```

```
// randomized heap

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

using ll = long long;

struct Tree {
    int value;
    Tree* l = nullptr;
    Tree* r = nullptr;
    Tree(int val){
        this->value = val;
    }
};

typedef Tree* pt;

Tree* merge(Tree* t1, Tree* t2) {
    if (!t1 || !t2)
        return t1 ? t1 : t2;
    if (t2->value < t1->value)
        swap(t1, t2);
    if (rand() & 1)
        swap(t1->l, t1->r);
    t1->l = merge(t1->l, t2);
    return t1;
}

void output(pt t){
    if(t == nullptr) return;
    cout << t->value << " ";
    output(t->l);
    output(t->r);
}

void erase(pt& t){
    if(t == nullptr) assert(false);
    pt temp = t;
    t = merge(t->l, t->r);
    delete temp;
}

void insert(pt& t, int val){
    pt node = new Tree(val);
    t = merge(t, node);
}

int front(pt t){
    if(t == nullptr) assert(false);
    return t->value;
}

int main() {

    int t;
    cin >> t;
    pt root = nullptr;
    while(t--){
        int x;
        cin >> x;
        if(x == 1){
            int y;
            cin >> y;
            insert(root, y);
        } else if(x==2){
            erase(root);
        } else if(x == 3){
            cout << "front " << front(root) << endl;
        }
        output(root);
        puts("");
    }

}

```

```
// sparse table

struct sparse_table {
    int N, K;
    vector<vector<int>> st; // don't forget to replace int with long long when needed


    // for min query
    int log[N+1];
    log[1] = 0;
    for (int i = 2; i <= MAXN; i++)
        log[i] = log[i/2] + 1;


    int f(int x){return x;}
    int f(int x, int y){return x + y;}

    sparse_table(vector<int> a){
        N = a.size();
        K = ceil(log2(N)) + 1;
        st.resize(N, vector<int>(K+1));

        for (int i = 0; i < N; i++) st[i][0] = f(a[i]);

        for (int j = 1; j <= K; j++)
            for (int i = 0; i + (1 << j) <= N; i++)
                st[i][j] = f(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);

    }

    ll sumQuery(int L, int R){
        long long sum = 0;
        for (int j = K; j >= 0; j--) {
            if ((1 << j) <= R - L + 1) {
                sum += st[L][j];
                L += 1 << j;
            }
        }
        return sum;
    }

    int minQuery(int L, int R){ // or maxQuery
        int j = log[R - L + 1];
        int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
        return minimum;
    }

};
```

```
// suffix array with comparing substring

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

struct suffix_array {
    string s;
    int n;
    vector<int> p, cnt, c;
    vector<vector<int>> c1;
    suffix_array(string s){
        this->s = s;
        build();
    }
    void build(){
        s += '$';
        n = s.size();
        const int alphabet = 256;

        p.resize(n, 0);
        c = p;
        cnt.resize(max(n, alphabet), 0);

        for(auto x : s) cnt[x]++;

        for(int i = 1; i < alphabet; i++) cnt[i] += cnt[i-1];

        for(int i = 0; i < n; i++){
            p[--cnt[s[i]]] = i;
        }

        c[p[0]] = 0;
        int classes = 1;

        for(int i = 1; i < n; i++){
            if(s[p[i]] != s[p[i-1]]) classes++;
            c[p[i]] = classes-1;
        }
        c1.push_back(c);
        vector<int> pn(n), cn(n);
        for (int h = 0; (1 << h) < n; ++h) {
            for (int i = 0; i < n; i++) {
                pn[i] = p[i] - (1 << h);
                if (pn[i] < 0)
                    pn[i] += n;
            }
            fill(cnt.begin(), cnt.begin() + classes, 0);
            for (int i = 0; i < n; i++)
                cnt[c[pn[i]]]++;
            for (int i = 1; i < classes; i++)
                cnt[i] += cnt[i-1];
            for (int i = n-1; i >= 0; i--)
                p[--cnt[c[pn[i]]]] = pn[i];
            cn[p[0]] = 0;
            classes = 1;
            for (int i = 1; i < n; i++) {
                pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
                pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
                if (cur != prev)
                    ++classes;
                cn[p[i]] = classes - 1;
            }
            c.swap(cn);
            c1.push_back(c);
        }
        c.pop_back();
        for(int &i : c) i--;
        p.erase(p.begin());
        n--;
        s.pop_back();
    }

    bool in_s(string pattern){
        int l = 0, r = n-1;
        while(l <= r){
            int mid = (l+r)/2;
            int sub = n-p[mid];
            string x = s.substr(p[mid], min((int)pattern.length(), sub));
            if(x == pattern) return true;
            else if(x > pattern){
                r = mid-1;
            } else l = mid+1;
        }
        return false;
    }

    int compare(int i, int j, int l, int k) {
        pair<int, int> a = {c1[k][i], c1[k][(i+l-(1 << k))%n]};
        pair<int, int> b = {c1[k][j], c1[k][(j+l-(1 << k))%n]};
        return a == b ? 0 : a < b ? -1 : 1;
    }

    int compare_substrings(int l, int r, int l2, int r2){
        if(r-l == r2-l2){
            return compare(l, l2, r-l+1, log2(r-l+1));
        } else if(r-l < r2-l2){
            r2 -= (r2-l2+1)-(r-l+1);
            int res = compare(l, l2, r-l+1, log2(r-l+1));
            if(res == 0) return -1;
            return res;
        } else {
            r -= (r-l+1)-(r2-l2+1);
            int res = compare(l, l2, r-l+1, log2(r-l+1));
            if(res == 0) return 1;
            return res;
        }
    }

    vector<int> lcp_construction(string const& s, vector<int> const& p) {
        int n = s.size();
        vector<int> rank(n, 0);
        for (int i = 0; i < n; i++)
            rank[p[i]] = i;

        int k = 0;
        vector<int> lcp(n-1, 0);
        for (int i = 0; i < n; i++) {
            if (rank[i] == n - 1) {
                k = 0;
                continue;
            }
            int j = p[rank[i] + 1];
            while (i + k < n && j + k < n && s[i+k] == s[j+k])
                k++;
            lcp[rank[i]] = k;
            if (k)
                k--;
        }
        return lcp;
    }

```

```
};

int main() {

    string s = "ahmadlaghadban";
    int n = s.length();
    suffix_array suf(s);

    cout << s << endl;
    for(int i = 0; i < n; i++){
        int x = suf.p[i];
        cout << x << " ";
    }
    cout << endl;

    for(int i = 0; i < n; i++){
        int x = suf.p[i];
        for(int j = x; j < n; j++) cout << s[j];
        cout << endl;
    }

    while(true){
        int l, r, l2, r2;
        cin >> l >> r >> l2 >> r2;
        cout << suf.compare_substrings(l, r, l2, r2) << endl;
    }

    return 0;
}
```

```cpp
// suffix automaton

#include <bits/stdc++.h>

using namespace std;
using ll = long long;

struct state {
    int len, link;
    bool clone = 0;
    map<char, int> next;
};

vector<state> st;
vector<ll> cnt, subs;

struct suffix_automaton {
    int maxLen, sz, last;
    string s;
    void build_again(string s){
        this->s = s;
        maxLen = s.length()*2 + 10;
        st.clear();
        st.resize(maxLen);
        cnt.clear();
        cnt.resize(maxLen*2);
        subs = cnt;
        sz = last = 0;
        build();
    }

    void build(){
        sa_init();
        for(char x : s) sa_extend(x);
    }
    suffix_automaton(){};
    suffix_automaton(string s){
        this->s = s;
        maxLen = s.length()*2 + 10;
        st.resize(maxLen);
        cnt.resize(maxLen*2);
        subs = cnt;
        sz = last = 0;
        build();
    }

    void sa_init() {
        st[0].len = 0;
        st[0].link = -1;
        sz = 1;
        last = 0;
    }

    void sa_extend(char c) {
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next.count(c)) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                st[clone].next = st[q].next;
                st[clone].link = st[q].link;
                st[clone].clone = 1;
                while (p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }

    void occ(){
        vector<pair<int, int>> v;
        for(int i = sz-1; i; i--){
            cnt[i] = !st[i].clone;
            v.push_back({st[i].len, i});
        }
        sort(v.begin(), v.end());
        for(int i = v.size()-1; i >= 0; i--){
            int suf = st[v[i].second].link;
            cnt[suf] += cnt[v[i].second];
        }
        cnt[0] = 0;
    }

    void difSubs(int sta, int pre = -1){
        if(subs[sta]) return;
        if(sta != 0){
            subs[sta] += cnt[sta]; // if you want only different substrings just replace 'cnt[sta]' with '1';
        }
        for(auto [f, s] : st[sta].next){
            difSubs(s, sta);
            subs[sta] += subs[s];
        }
    }

    int lcs(string t){
        int v = 0, l = 0, ans = 0;
        for(int i = 0; i < t.length(); i++){
            while(v != 0 && st[v].next.count(t[i]) == 0){
                v = st[v].link;
                l = st[v].len;
            }
            if(st[v].next.count(t[i])) v = st[v].next[t[i]], l++;
            ans = max(ans, l);
        }
        return ans;
    }

    void prt(int sta, string cur){
        if(st[sta].next.empty()) cout << "cur " << cur << endl;
        else {
            for(auto [f,s] : st[sta].next){
                string cur1 = cur + f;
                prt(s, cur1);
            }
        }
    }

    void prtWhole(int sta, string cur = ""){
        cout << "cur " << cur << " sta " << sta << " " << cnt[sta] << endl;
        for(auto [f,s] : st[sta].next){
            string cur1 = cur + f;
            prtWhole(s, cur1);
        }
    }

};

char buf[100010];
bool vis[200010];

ll calc(int sta, suffix_automaton &suf){
    if(vis[sta]) return 0;
    vis[sta] = 1;
    ll res = cnt[sta]*cnt[sta]*(st[sta].len - st[st[sta].link].len);
    for(auto [f, s] : st[sta].next){
        res += calc(s, suf);
    }
    return res;
}

string read(){
    scanf("%s", buf);
    return buf;
}

string s;
suffix_automaton suf;

int main() {

    int t;
    scanf("%d", &t);

    while(t--){
        s = read();
        suf.build_again(s);
        suf.occ();
        for(int i = 0; i < s.length()*2+5; i++) vis[i] = 0;
        ll ans = calc(0, suf);
        cout << ans << endl;
    }

    return 0;
}

```

```cpp
// treap implicit

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

using ll = long long;
mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
typedef struct item * pitem;
struct item { // best practice is to do all operations using split and merge operations
    int prior, value, cnt;
    bool rev;
    // int f = 0; variable for range queries, do range queries by 2 splits, then merge parts again
    // don't forget to edit it in upd_cnt function
    // If you use it as an dynamic array, it's 1-base indexed
    pitem l, r;
    item(int val){
        value = val;
        prior = rand();
        l = r = NULL;
    }
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) { // same principle as segment tree, call it in top of any other operation function
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) { // first key elements in left and remains in right
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void insert(pitem &t, int pos, int value){ // insert value in index = pos
    pitem t1, t2;
    split(t, t1, t2, pos);
    pitem newItem = new item(value);
    merge(t1, t1, newItem);
    merge(t, t1, t2);
    upd_cnt(t);
}

void erase(pitem &t, int key, int add = 0){ // erase value in index = pos
    pitem l, r, l1, r1;
    split(t, l, r, key+1);
    split(l, l1, r1, key);
    merge(t, l1, r);
    delete r1;
    upd_cnt(t);
}

void reverse (pitem &t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
    upd_cnt(t);
}

int elementAt(pitem &t, int key){
    pitem l, r, l1, r1;
    split(t, l, r, key+1);
    split(l, l1, r1, key);
    int res = r1->value;
    merge(l, l1, r1);
    merge(t, l, r);
    upd_cnt(t);
    return res;
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}

int main() {

    pitem root = NULL;
    int t;
    scanf("%d", &t);
    while(t--){

        int x, y, z;
        cin >> x;
        if(x == 1){
            cin >> y >> z;
            insert(root, y, z);
        } else if(x == 2) {
            cin >> y;
            erase(root, y);
        } else if(x == 3){
            cin >> y >> z;
            reverse(root, y, z);
        } else if(x == 4){
            cin >> y;
            cout << "element " << elementAt(root, y) << endl;
        } else {
            output(root);
            puts("");
        }

    }

}

/*

100
1 0 7
1 0 6
1 0 5
1 0 4
1 0 3
1 0 2
1 0 1

*/

```

```cpp
// treap regular

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

using ll = long long;
mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
struct item {
    int key, prior, cnt = 0;
    ll val = 0;
    item *l, *r;
    item () { }
    item (int key) : key(key), prior(rng32()), l(NULL), r(NULL), val(0), cnt(1) { }
};
typedef item* pitem;

int cnt (pitem t) {
    return t ? t->cnt : 0;
}

void upd_cnt (pitem t) {
    if (t)
        t->cnt = 1 + cnt(t->l) + cnt (t->r);
}

void split (pitem t, int key, pitem & l, pitem & r) {
    if (!t)
        l = r = NULL;
    else if (t->key <= key)
        split (t->r, key, t->r, r),  l = t;
    else
        split (t->l, key, l, t->l),  r = t;
    upd_cnt(t);
}

void merge (pitem & t, pitem l, pitem r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt(t);
}

void insert (pitem & t, pitem it) {
    pitem l = NULL, r = NULL;
    split(t, it->key, l, r);
    merge(t, l, it);
    merge(t, it, r);
}

void erase (pitem & t, int key) {
    pitem l = NULL, r = NULL, cur = NULL;
    split(t, key, l, r);
    split(l, key-1, l, cur);
    merge(t, l, r);
    delete cur;
}

int kth(pitem t, int k){
    if(!t) return -1;
    if(cnt(t->l) >= k) return kth(t->l, k);
    k -= cnt(t->l);
    if(k == 1) return t->key;
    else return kth(t->r, k - 1);
}

int main() {

    int n, q;
    scanf("%d%d", &n, &q);

    pitem root = NULL;

    for(int i = 0; i < n; i++){
        int x;
        scanf("%d", &x);
        pitem cur = new item(x);
        insert(root, cur);
    }

    for(int i = 0; i < q; i++){
        int x;
        scanf("%d", &x);
        if(x < 0){
            x = -x;
            int kth_e = kth(root, x);
            assert(kth_e + 1);
            erase(root, kth_e);
        } else {
            pitem cur = new item(x);
            insert(root, cur);
        }
    }

    if(cnt(root)) printf("%d", kth(root, 1));
    else printf("%d", 0);

}
```