

```

1  // aho-corasick
2
3  const int K = 26;
4
5  struct Vertex {
6      int next[K];
7      bool leaf = false;
8      int p = -1;
9      char pch;
10     int link = -1;
11     int go[K];
12     int exit_link = -1;
13
14     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
15         fill(begin(next), end(next), -1);
16         fill(begin(go), end(go), -1);
17     }
18 };
19
20 vector<Vertex> t(1);
21
22 void add_string(string const& s) {
23     int v = 0;
24     for (char ch : s) {
25         int c = ch - '0';
26         if (t[v].next[c] == -1) {
27             t[v].next[c] = t.size();
28             t.emplace_back(v, ch);
29         }
30         v = t[v].next[c];
31     }
32     t[v].leaf = true;
33 }
34
35 int go(int v, char ch);
36
37 int get_link(int v) {
38     if (t[v].link == -1) {
39         if (v == 0 || t[v].p == 0)
40             t[v].link = 0;
41         else
42             t[v].link = go(get_link(t[v].p), t[v].pch);
43     }
44     return t[v].link;
45 }
46
47 int go(int v, char ch) {
48     int c = ch - '0';
49     if (t[v].go[c] == -1) {
50         if (t[v].next[c] != -1)
51             t[v].go[c] = t[v].next[c];
52         else
53             t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
54     }
55     return t[v].go[c];
56 }
57
58
59 int exit_link(int v){
60     if(t[v].exit_link == -1){
61         if(t[v].leaf) t[v].exit_link = v;
62         else {
63             if(v == 0) t[v].exit_link = 0;
64             else t[v].exit_link = exit_link(get_link(v));
65         }
66     }
67     return t[v].exit_link;
68 }

```

```

1 // Balanced ternary (number system)
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6 using ll = long long;
7
8 string intToB3(int x){
9     string res = "";
10    int carry = 0;
11    while(x || carry){
12        int cur = x%3 + carry;
13        x /= 3;
14        carry = 0;
15        if(cur == 2){
16            cur = -1;
17            carry = 1;
18        } else if(cur == 3){
19            cur = 0;
20            carry = 1;
21        }
22        if(cur == -1) res += 'Z';
23        else res += '0' + cur;
24    }
25    reverse(res.begin(), res.end());
26    return res;
27 }
28
29 int main() {
30
31     int x, y;
32     cin >> x >> y;
33     x = abs(x);
34     y = abs(y);
35     if(x > y) swap(x, y);
36
37     bool res = 1;
38
39     string onlyX = intToB3(x);
40     string onlyY = intToB3(y);
41
42     string toAdd(onlyY.length() - onlyX.length(), '0');
43     onlyX = toAdd + onlyX;
44
45     for(int i = 0; i < onlyX.length(); i++){
46         if(onlyX[i] != '0' && onlyY[i] != '0') res = 0;
47         if(onlyX[i] == '0' && onlyY[i] == '0') res = 0;
48     }
49
50     if(res) puts("Possible");
51     else puts("Impossible");
52
53 }

```

```

1  // big int
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  using ll = long long;
7
8  string itos(int x){
9      if(x == 0) return "";
10     return itos(x/10) + char(x%10 + '0');
11 }
12
13 struct bigInt {
14
15     int base = 1000*1000*1000;
16     vector<int> a;
17
18     bigInt(){
19         a.push_back(0);
20     }
21     bigInt(string s){
22         read(s);
23     }
24     bigInt(ll x){
25         while(x){
26             a.push_back(x%base);
27             x /= base;
28         }
29     }
30
31     void print(){
32         printf ("%d", a.empty() ? 0 : a.back());
33         for (int i=(int)a.size()-2; i>=0; --i)
34             printf ("%09d", a[i]);
35         puts("");
36     }
37
38     void read(string s){
39         for (int i=(int)s.length(); i>0; i-=9)
40             if (i < 9)
41                 a.push_back (atoi (s.substr (0, i).c_str()));
42             else
43                 a.push_back (atoi (s.substr (i-9, 9).c_str()));
44
45         while (a.size() > 1 && a.back() == 0) a.pop_back();
46     }
47
48     int size(){
49         return a.size();
50     }
51
52     int& back(){
53         return a.back();
54     }
55
56     void pop_back(){
57         a.pop_back();
58     }
59
60     int& operator [] (int i){
61         return a[i];
62     }
63
64     bigInt operator = (const bigInt &another){
65         a = another.a;
66         return *this;
67     }
68
69     bigInt operator = (const string &x){

```

```

70         bigInt res(x);
71         return res;
72     }
73
74     bigInt operator + (ll x){
75         bigInt c = *this;
76         c += x;
77         return c;
78     }
79
80     bigInt operator += (ll x){
81         bigInt c(x);
82         *this += c;
83         return *this;
84     }
85
86     bigInt operator + (bigInt b){
87         bigInt c = *this;
88         c += b;
89         return c;
90     }
91
92     bigInt operator += (bigInt b){
93         int carry = 0;
94         for (int i = 0; i < max((int)a.size(), b.size()) || carry; ++i) {
95             if (i == a.size())
96                 a.push_back (0);
97             a[i] += carry + (i < b.size() ? b[i] : 0);
98             carry = a[i] >= base;
99             if (carry) a[i] -= base;
100         }
101         return *this;
102     }
103
104     bigInt operator - (bigInt b){
105         bigInt c = *this;
106         c -= b;
107         return c;
108     }
109
110     bigInt operator -= (bigInt b){
111         int carry = 0;
112         for (int i=0; i<b.size() || carry; ++i) {
113             a[i] -= carry + (i < b.size() ? b[i] : 0);
114             carry = a[i] < 0;
115             if (carry) a[i] += base;
116         }
117         while (a.size() > 1 && a.back() == 0)
118             a.pop_back();
119         return *this;
120     }
121
122     bigInt operator * (int b){
123         bigInt c = *this;
124         c *= b;
125         return c;
126     }
127
128     bigInt operator *= (int b){
129         int carry = 0;
130         for (int i=0; i<a.size() || carry; ++i) {
131             if (i == a.size())
132                 a.push_back (0);
133             long long cur = carry + a[i] * 1ll * b;
134             a[i] = int (cur % base); // x %= y is equal to x -= x/y*y;
135             carry = int (cur / base);
136         }
137         while (a.size() > 1 && a.back() == 0)
138             a.pop_back();

```

```

139         return *this;
140     }
141
142     bigInt operator * (bigInt b){ // O(n^2)
143         bigInt c = *this;
144         c *= b;
145         return c;
146     }
147
148     bigInt operator *= (bigInt b){ // O(n^2)
149         bigInt c;
150         c.a.resize((a.size()+b.size()));
151         for (int i=0; i<a.size(); ++i)
152             for (int j=0, carry=0; j<(int)b.size() || carry; ++j) {
153                 long long cur = c[i+j] + a[i] * 111 * (j < (int)b.size() ? b[j] : 0) +
154                     carry;
155                 c[i+j] = int (cur % base);
156                 carry = int (cur / base);
157             }
158         while (c.size() > 1 && c.back() == 0)
159             c.pop_back();
160         this->a = c.a;
161         return *this;
162     }
163
164     bigInt operator / (int b){
165         bigInt c = *this;
166         return c /= b;
167     }
168
169     bigInt operator /= (int b){
170         int carry = 0;
171         for (int i=(int)a.size()-1; i>=0; --i) {
172             long long cur = a[i] + carry * 111 * base;
173             a[i] = int (cur / b);
174             carry = int (cur % b);
175         }
176         while (a.size() > 1 && a.back() == 0)
177             a.pop_back();
178         return *this;
179     }
180
181     int operator % (int b){
182         bigInt c = *this;
183         int carry = 0;
184         for (int i=(int)c.size()-1; i>=0; --i) {
185             long long cur = c[i] + carry * 111 * base;
186             c[i] = int (cur / b);
187             carry = int (cur % b);
188         }
189         while (c.size() > 1 && c.back() == 0)
190             c.pop_back();
191         return carry;
192     }
193
194     bool operator == (const bigInt &b){
195         return b.a == a;
196     }
197
198     bool operator < (bigInt b){
199         if(a.size() < b.size()) return 1;
200         if(a.size() > b.size()) return 0;
201         for(int i = (int)a.size()-1; i >= 0; i--){
202             if(a[i] < b[i]) return 1;
203             if(b[i] < a[i]) return 0;
204         }
205         return 0;
206     }

```

```

207     bool operator <= (bigInt b){
208         bigInt temp = *this;
209         return temp < b || temp == b;
210     }
211
212 };
213
214 int main() {
215
216     string x = "123456789";
217     bigInt a = x;
218     bigInt b(x);
219     b = a/3;
220     a.print();
221     b.print();
222
223     return 0;
224 }
225
226 // 123456789987654
227

```

```

1 // centroid decomposition
2
3 set<int> G[N]; // adjacency list (note that this is stored in set, not vector)
4 int sz[N], pa[N];
5
6 int dfs(int u, int p) {
7     sz[u] = 1;
8     for(auto v : G[u]) if(v != p) {
9         sz[u] += dfs(v, u);
10    }
11    return sz[u];
12 }
13 int centroid(int u, int p, int n) {
14     for(auto v : G[u]) if(v != p) {
15         if(sz[v] > n / 2) return centroid(v, u, n);
16     }
17     return u;
18 }
19 void build(int u, int p) {
20     int n = dfs(u, p);
21     int c = centroid(u, p, n);
22     if(p == -1) p = c;
23     pa[c] = p;
24
25     vector<int> tmp(G[c].begin(), G[c].end());
26     for(auto v : tmp) {
27         G[c].erase(v); G[v].erase(c);
28         build(v, c);
29     }
30 }

```

```

1 // convex hull trick
2
3 #include "bits/stdc++.h"
4
5 using namespace std;
6 using ll = long long;
7
8 const int N = 1e5+10;
9 const double EPS = 1E-9;
10 int n;
11 ll a[N], b[N], dp[N];
12 // if to line are parallel then
13 // if the problem is max dp then then take the line with max c, else take the line with
   min c
14 struct Line {
15     ll m, c;
16     Line(ll m, ll c){
17         this->c = c;
18         this->m = m;
19     }
20
21     ll operator() (ll x){
22         return m*x + c;
23     }
24
25     double intersect(Line another){
26         return (double) (this->c - another.c) / (double) (another.m - this->m);
27     }
28 };
29
30
31 ll calc(vector<Line> &v, vector<double> &rng, double x){
32     int dist = upper_bound(rng.begin(), rng.end(), x) - rng.begin() - 1;
33     return v[dist](x);
34 }
35
36 int main () {
37
38     cin >> n;
39     for(int i = 0; i < n; i++) cin >> a[i];
40     for(int i = 0; i < n; i++) cin >> b[i];
41
42     dp[0] = 0;
43     vector<Line> v;
44     v.push_back({b[0], 0});
45     vector<double> rng;
46     rng.push_back(-1e17);
47
48     for(int i = 1; i < n; i++){
49         dp[i] = calc(v, rng, a[i]);
50         Line newLine(b[i], dp[i]);
51         while(v.size() >= 2 && v.end() [-2].intersect(newLine) + EPS <=
           v.end() [-2].intersect(v.end() [-1])){
52             v.pop_back();
53             rng.pop_back();
54         }
55         v.push_back(newLine);
56         rng.push_back(v.end() [-2].intersect(v.end() [-1]));
57         // when use this trick, slopes have to be sorted in increasing order
58         // if not sorted, then you can modify the logic and still correct
59     }
60
61     cout << dp[n-1] << endl;
62
63
64     return 0;
65 }

```



```

1  // dsu on tree
2
3  void add(int cur, int pre, int bg){
4      cnt[color[cur]]++;
5      for(int i : tree[cur]){
6          if(i != bg) add(i, cur, bg);
7      }
8  }
9
10 void del(int cur, int pre){
11     cnt[color[cur]]--;
12     for(int i : tree[cur]){
13         del(i, cur);
14     }
15 }
16
17 void dfs(int cur, int pre, int keep){
18     int mx = 0, bg = -1;
19
20     for(int i : tree[cur]){
21         if(i == pre) continue;
22         if(si[i] > mx){
23             mx = si[i];
24             bg = i;
25         }
26     }
27
28
29     for(int i : tree[cur]){
30         if(i == pre) continue;
31         if(i != bg) dfs(i, cur, 0);
32     }
33
34     if(bg+1) dfs(bg, cur, 1);
35
36     add(cur, pre, bg);
37
38     for(auto [f, s] : qu[cur]){
39         int odd = __builtin_popcount(cnt[f]);
40         if(odd < 2) ok[s] = 1;
41     }
42
43     if(keep == 0) del(cur, pre);
44
45 }

```

```

1  // fft
2
3  const double PI = acos(-1);
4  struct base {
5      double a, b;
6      base(double a = 0, double b = 0) : a(a), b(b) {}
7      const base operator + (const base &c) const
8          { return base(a + c.a, b + c.b); }
9      const base operator - (const base &c) const
10         { return base(a - c.a, b - c.b); }
11     const base operator * (const base &c) const
12         { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
13 };
14
15 void fft(vector<base> &p, bool inv = 0) {
16     int n = p.size(), i = 0;
17     for(int j = 1; j < n - 1; ++j) {
18         for(int k = n >> 1; k > (i ^ k); k >>= 1);
19         if(j < i) swap(p[i], p[j]);
20     }
21     for(int l = 1, m; (m = l << 1) <= n; l <= 1) {
22         double ang = 2 * PI / m;
23         base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
24         for(int i = 0, j, k; i < n; i += m) {
25             for(w = base(1, 0), j = i, k = i + 1; j < k; ++j, w = w * wn) {
26                 base t = w * p[j + 1];
27                 p[j + 1] = p[j] - t;
28                 p[j] = p[j] + t;
29             }
30         }
31     }
32     if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
33 }
34
35 vector<int> multiply(vector<int> const& a, vector<int> const& b) {
36     vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end()); // don't forget to
37     reverse in main when needed
38     int n = 1;
39     while (n < a.size() + b.size())
40         n <<= 1;
41     fa.resize(n);
42     fb.resize(n);
43
44     fft(fa, false);
45     fft(fb, false);
46     for (int i = 0; i < n; i++)
47         fa[i] = fa[i] * fb[i];
48     fft(fa, true);
49
50     vector<int> result(n);
51     for (int i = 0; i < n; i++)
52         result[i] = round(fa[i].a);
53     return result;
54 }

```

```

1 // gauss elimination xor
2
3 struct Basis {
4     const static int N = 61; // N = log(MAX_A)
5
6     vector<ll> a;
7     int sz = 0;
8
9     Basis() { a.resize(N, 0); }
10
11     Basis(vector<ll> v) { a.resize(N, 0); for(ll x : v) add(x); }
12
13     void add(Basis another) { for(ll x : another.a) add(x); }
14
15     void add(ll x) { // O(N)
16         for (int i = N - 1; ~i; --i) {
17             if (x & (1LL << i)) {
18                 if (a[i]) {
19                     x ^= a[i];
20                 } else {
21                     a[i] = x;
22                     sz++;
23                     break;
24                 }
25             }
26         }
27     }
28
29     bool can(ll x) {
30         for (int i = N - 1; ~i; --i) {
31             if (x & (1 << i)) {
32                 x ^= a[i];
33             }
34         }
35         return x == 0;
36     }
37
38     ll getMax() {
39         ll result = 0;
40         for (int i = N - 1; ~i; --i) {
41             if ((result ^ a[i]) > result) {
42                 result ^= a[i];
43             }
44         }
45         return result;
46     }
47
48 };

```

```

1  // graph matching kuhn algorithm
2
3  int n;
4  vector<vector<int>> g;
5  vector<int> mt;
6  vector<bool> used;
7
8  bool try_kuhn(int v) {
9      if (used[v])
10         return false;
11     used[v] = true;
12     for (int to : g[v]) {
13         if (mt[to] == -1 || try_kuhn(mt[to])) {
14             mt[to] = v;
15             return true;
16         }
17     }
18     return false;
19 }
20
21 int main() {
22     //... reading the graph ...
23     // zero-base indexed
24     // one graph, not explicitly divided into to parts
25
26     mt.assign(n, -1);
27     for (int v = 0; v < n; ++v) {
28         used.assign(n, false);
29         try_kuhn(v);
30     }
31
32     for (int i = 0; i < n; ++i)
33         if (mt[i] != -1)
34             printf("%d %d\n", mt[i] + 1, i + 1);
35 }

```

```

1 // heavy-light decomposition
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6 using ll = long long;
7
8 struct segment_tree {
9     int n;
10    vector<int> st;
11
12    segment_tree() {}
13
14    segment_tree(const vector<int> &v) {
15        n = v.size();
16        st.resize(n*4+5, 0);
17        build(1, 0, n-1, v);
18    }
19
20    void init(const vector<int> &v) {
21        st.clear();
22        n = v.size();
23        st.resize(n*4+5);
24        build(1, 0, n-1, v);
25    }
26
27    int mrg(int l, int r) {
28        return max(l, r);
29    }
30
31    void build(int v, int l, int r, const vector<int> &ve) {
32        if(l == r) st[v] = ve[l];
33        else {
34            int mid = (l+r)/2;
35            build(2*v, l, mid, ve);
36            build(2*v+1, mid+1, r, ve);
37            st[v] = mrg(st[2*v+1], st[2*v]);
38        }
39    }
40
41    int get(int v, int l, int r, int tl, int tr) {
42        if(tl > tr) return 0;
43        if(l == tl && r == tr) return st[v];
44        int mid = (l+r)/2;
45        int f = get(2*v, l, mid, tl, min(tr, mid));
46        int s = get(2*v+1, mid+1, r, max(mid+1, tl), tr);
47        return mrg(f, s);
48    }
49
50    int query(int l, int r) {
51        // assert(l <= r);
52        return get(1, 0, n-1, l, r);
53    }
54
55    void upd(int v, int l, int r, int idx, int val) {
56        if(l == r) st[v] = val;
57        else {
58            int mid = (l+r)/2;
59            if(idx <= mid) upd(2*v, l, mid, idx, val);
60            else upd(2*v+1, mid+1, r, idx, val);
61            st[v] = mrg(st[2*v+1], st[2*v]);
62        }
63    }
64
65    void upd(int idx, int val) {
66        upd(1, 0, n-1, idx, val);
67    }
68
69    void clear() {

```

```

70         n = 0;
71         st.clear();
72     }
73
74 };
75
76 const int N = 2e5+10;
77 vector<pair<int, int> > tree[N];
78 vector<int> part[N];
79 segment_tree st[N];
80 int head[N], sz[N], depth[N], parent[N], pos[N]; //, val[N];
81
82 void init(int v, int pre, int h){
83     depth[v] = h;
84     sz[v] = 1;
85     parent[v] = pre;
86     for(auto [i, s] : tree[v]){
87         if(i == pre) continue;
88         init(i, v, h+1);
89         sz[v] += sz[i];
90     }
91 }
92
93 void dfs(int v, int pre, int h, int val){ // decomposition
94     int mx = -1;
95     head[v] = h;
96     pos[v] = part[h].size();
97     part[h].push_back(val);
98     for(auto [i, s] : tree[v]){
99         if(i == pre) continue;
100         if(mx == -1 || sz[i] > sz[mx]){
101             mx = i;
102             val = s;
103         }
104     }
105
106     if(mx != -1){
107         dfs(mx, v, h, val);
108     }
109
110     for(auto [i, s] : tree[v]){
111         if(i == pre || i == mx) continue;
112         dfs(i, v, i, s);
113     }
114 }
115
116 void upd_edge(int v, int u, int val){ // edit something in edge between vertexes u and v
117     if(parent[u] != v) swap(v, u);
118     assert(parent[u] == v);
119     // cout << "u " << u << " v " << v << " val " << val << endl;
120     // cout << "head " << head[u] << " pos " << pos[u] << endl;
121     st[head[u]].upd(pos[u], val);
122     part[head[u]][pos[u]] = val;
123 }
124
125 int query(int v, int u){
126     int res = 0;
127
128     while(head[v] != head[u]){
129         if(depth[head[u]] < depth[head[v]]) swap(u, v);
130         assert(pos[head[u]] == 0);
131         res = max(res, st[head[u]].query(0, pos[u]));
132         u = parent[head[u]];
133     }
134
135     if(depth[u] < depth[v]) swap(u, v);
136
137     // cout << "pos " << pos[v] << " and " << pos[u] << endl;

```

```

139     res = max(res, st[head[u]].query(pos[v]+1, pos[u]));
140     return res;
141 }
142
143 void clear(int n){
144     for(int i = 0; i < n+3; i++){
145         tree[i].clear();
146         part[i].clear();
147         st[i].clear();
148         head[i] = sz[i] = depth[i] = parent[i] = pos[i] = -1;
149     }
150 }
151
152 char buf[20];
153
154 string read(){
155     scanf("%s", buf);
156     return buf;
157 }
158
159 int main()
160 {
161
162     int t;
163     cin >> t;
164     while(t--){
165         char x;
166         // scanf("%c", &x);
167         int n;
168         scanf("%d", &n);
169         clear(n);
170         vector<pair<int, int>> edges;
171         for(int i = 0; i < n-1; i++){
172             int x, y, w;
173             scanf("%d%d%d", &x, &y, &w);
174             edges.emplace_back(x, y);
175             tree[x].push_back({y, w});
176             tree[y].push_back({x, w});
177         }
178
179         int root = 1;
180         init(root, root, 0);
181         // the val parameter in decomposition function depends on type of operations
182         // you do, for sum operations you pass 0, for max you pass -INF, ans so on
183         dfs(root, root, root, 0);
184
185         for(int i = 1; i <= n; i++){
186             // cout << "i " << i << " and " << head[i] << endl;
187             if(head[i] == i) st[i].init(part[i]);
188         }
189
190         string q = "";
191         while(true){
192             int u, v;
193             q = read();
194             if(q == "DONE") break;
195             scanf("%d%d", &v, &u);
196             if(q == "CHANGE"){
197                 // cout << "hi " << edges[v-1].first << " and " << edges[v-1].second <<
198                 // endl;
199                 upd_edge(edges[v-1].first, edges[v-1].second, u);
200             } else if(q == "QUERY"){
201                 printf("%d\n", query(u, v));
202             }
203         }
204     }
205     return 0;

```

```
206     }
207     /*
208
209     1
210
211     3
212     1 2 1
213     2 3 2
214     QUERY 1 1
215     CHANGE 1 3
216     QUERY 1 2
217     QUERY 1 1
218     QUERY 2 2
219     QUERY 3 3
220     DONE
221
222     */
223
```



```

1 // hopcroft karp (graph matching)
2
3 // source: https://github.com/shah-deep/Graph-Theory-Algos
4 #include<bits/stdc++.h>
5
6 using namespace std;
7
8 const int N = 2e5 + 10;
9 vector<int> G[N];
10
11 bool bfs(vector<int> &setU, vector<int> &setV, vector<int> &dist, int m)
12 {
13     queue<int> que;
14
15     for (int u=1; u<=m; u++)
16     {
17         if (setU[u]==0)
18         {
19             dist[u] = 0;
20             que.push(u);
21         }
22
23         else dist[u] = INT_MAX;
24     }
25
26     dist[0] = INT_MAX;
27
28     while (!que.empty())
29     {
30         int u = que.front();
31         que.pop();
32
33         if (dist[u] < dist[0])
34         {
35             for (int i = 0; i < G[u].size(); ++i)
36             {
37                 int v = G[u][i];
38
39                 if (dist[setV[v]] == INT_MAX)
40                 {
41                     dist[setV[v]] = dist[u] + 1;
42                     que.push(setV[v]);
43                 }
44             }
45         }
46     }
47
48     return (dist[0] != INT_MAX);
49 }
50
51 bool dfs(int u, vector<int> &setU, vector<int> &setV, vector<int> &dist)
52 {
53     if (u != 0)
54     {
55
56         for (int i = 0; i < G[u].size(); ++i) {
57
58             int v = G[u][i];
59             if ((dist[setV[v]] == dist[u]+1) && dfs(setV[v], setU, setV, dist))
60             {
61                 setU[u] = v;
62                 setV[v] = u;
63                 return true;
64             }
65         }
66
67         dist[u] = INT_MAX;
68         return false;
69     }

```

```

70
71     return true;
72 }
73
74 void hopcroft_karp(int n1, int n2)
75 {
76     vector<int> setU(n1+1, 0), setV(n2+1, 0), dist(n1+1);
77
78     int cnt = 0;
79
80     while (bfs(setU, setV, dist, n1)) {
81         for (int u=1; u<=n1; u++) {
82             if (!setU[u] && dfs(u, setU, setV, dist))
83                 cnt++;
84         }
85     }
86 }
87
88 cout << "\nMaximum Matching:" << endl;
89 for(int i=1; i <= n1; i++){
90     if(setU[i] == 0) continue;
91     cout << "i " << i << " and " << setU[i] << endl;
92 }
93
94 cout << "\nCount: " << cnt << endl;
95 }
96
97 int main()
98 {
99     int n, m; // vertexes, edges
100     cin >> n >> m;
101     for(int i=0; i<m; i++){
102         int u, v; // one-base indexed
103         cin >> u >> v;
104         G[u].push_back(v);
105     }
106
107     hopcroft_karp(n, n);
108
109     return 0;
110 }
111

```

```

1 // hungarian algorithm
2
3 #include <bits/stdc++.h>
4
5 using namespace std ;
6 using ll = long long;
7
8 ///////////////////////////////////////////////////////////////////
9 // Hungarian.cpp: Implementation file for Class HungarianAlgorithm.
10 //
11 // This is a C++ wrapper with slight modification of a hungarian algorithm
12 // implementation by Markus Buehren.
13 // The original implementation is a few mex-functions for use in MATLAB, found here:
14 // http://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem
15 //
16 // Both this code and the original code are published under the BSD license.
17 // by Cong Ma, 2016
18 //
19 #ifndef HUNGARIAN_H
20 #define HUNGARIAN_H
21
22 using namespace std;
23
24
25 class HungarianAlgorithm
26 {
27 public:
28     HungarianAlgorithm();
29     ~HungarianAlgorithm();
30     double Solve(vector<vector<double>>& DistMatrix, vector<int>& Assignment);
31
32 private:
33     void assignmentoptimal(int *assignment, double *cost, double *distMatrix, int
34         nOfRows, int nOfColumns);
35     void buildassignmentvector(int *assignment, bool *starMatrix, int nOfRows, int
36         nOfColumns);
37     void computeassignmentcost(int *assignment, double *cost, double *distMatrix, int
38         nOfRows);
39     void step2a(int *assignment, double *distMatrix, bool *starMatrix, bool
40         *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
41         nOfRows, int nOfColumns, int minDim);
42     void step2b(int *assignment, double *distMatrix, bool *starMatrix, bool
43         *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
44         nOfRows, int nOfColumns, int minDim);
45     void step3(int *assignment, double *distMatrix, bool *starMatrix, bool
46         *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
47         nOfRows, int nOfColumns, int minDim);
48     void step4(int *assignment, double *distMatrix, bool *starMatrix, bool
49         *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
50         nOfRows, int nOfColumns, int minDim, int row, int col);
51     void step5(int *assignment, double *distMatrix, bool *starMatrix, bool
52         *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
53         nOfRows, int nOfColumns, int minDim);
54 };
55
56 #endif
57
58 HungarianAlgorithm::HungarianAlgorithm(){}
59 HungarianAlgorithm::~HungarianAlgorithm(){}
60
61 ///////////////////////////////////////////////////////////////////
62 // A single function wrapper for solving assignment problem.
63 ///////////////////////////////////////////////////////////////////

```

```

54 double HungarianAlgorithm::Solve(vector<vector<double>> & DistMatrix, vector<int> &
Assignment)
55 {
56     unsigned int nRows = DistMatrix.size();
57     unsigned int nCols = DistMatrix[0].size();
58
59     double *distMatrixIn = new double[nRows * nCols];
60     int *assignment = new int[nRows];
61     double cost = 0.0;
62
63     // Fill in the distMatrixIn. Mind the index is "i + nRows * j".
64     // Here the cost matrix of size MxN is defined as a double precision array of N*M
elements.
65     // In the solving functions matrices are seen to be saved MATLAB-internally in
row-order.
66     // (i.e. the matrix [1 2; 3 4] will be stored as a vector [1 3 2 4], NOT [1 2 3 4]).
67     for (unsigned int i = 0; i < nRows; i++)
68         for (unsigned int j = 0; j < nCols; j++)
69             distMatrixIn[i + nRows * j] = DistMatrix[i][j];
70
71     // call solving function
72     assignmentoptimal(assignment, &cost, distMatrixIn, nRows, nCols);
73
74     Assignment.clear();
75     for (unsigned int r = 0; r < nRows; r++)
76         Assignment.push_back(assignment[r]);
77
78     delete[] distMatrixIn;
79     delete[] assignment;
80     return cost;
81 }
82
83
84 //*****//
85 // Solve optimal solution for assignment problem using Munkres algorithm, also known as
Hungarian Algorithm.
86 //*****//
87 void HungarianAlgorithm::assignmentoptimal(int *assignment, double *cost, double
*distMatrixIn, int nOfRows, int nOfColumns)
88 {
89     double *distMatrix, *distMatrixTemp, *distMatrixEnd, *columnEnd, value, minValue;
90     bool *coveredColumns, *coveredRows, *starMatrix, *newStarMatrix, *primeMatrix;
91     int nOfElements, minDim, row, col;
92
93     /* initialization */
94     *cost = 0;
95     for (row = 0; row < nOfRows; row++)
96         assignment[row] = -1;
97
98     /* generate working copy of distance Matrix */
99     /* check if all matrix elements are positive */
100     nOfElements = nOfRows * nOfColumns;
101     distMatrix = (double *)malloc(nOfElements * sizeof(double));
102     distMatrixEnd = distMatrix + nOfElements;
103
104     for (row = 0; row < nOfElements; row++)
105     {
106         value = distMatrixIn[row];
107         if (value < 0)
108             cerr << "All matrix elements have to be non-negative." << endl;
109         distMatrix[row] = value;
110     }
111
112
113     /* memory allocation */
114     coveredColumns = (bool *)calloc(nOfColumns, sizeof(bool));
115     coveredRows = (bool *)calloc(nOfRows, sizeof(bool));
116     starMatrix = (bool *)calloc(nOfElements, sizeof(bool));
117     primeMatrix = (bool *)calloc(nOfElements, sizeof(bool));

```

```

118 newStarMatrix = (bool *)calloc(nOfElements, sizeof(bool)); /* used in step4 */
119
120 /* preliminary steps */
121 if (nOfRows <= nOfColumns)
122 {
123     minDim = nOfRows;
124
125     for (row = 0; row<nOfRows; row++)
126     {
127         /* find the smallest element in the row */
128         distMatrixTemp = distMatrix + row;
129         minValue = *distMatrixTemp;
130         distMatrixTemp += nOfRows;
131         while (distMatrixTemp < distMatrixEnd)
132         {
133             value = *distMatrixTemp;
134             if (value < minValue)
135                 minValue = value;
136             distMatrixTemp += nOfRows;
137         }
138
139         /* subtract the smallest element from each element of the row */
140         distMatrixTemp = distMatrix + row;
141         while (distMatrixTemp < distMatrixEnd)
142         {
143             *distMatrixTemp -= minValue;
144             distMatrixTemp += nOfRows;
145         }
146     }
147
148     /* Steps 1 and 2a */
149     for (row = 0; row<nOfRows; row++)
150         for (col = 0; col<nOfColumns; col++)
151             if (fabs(distMatrix[row + nOfRows*col]) < DBL_EPSILON)
152                 if (!coveredColumns[col])
153                 {
154                     starMatrix[row + nOfRows*col] = true;
155                     coveredColumns[col] = true;
156                     break;
157                 }
158 }
159 else /* if(nOfRows > nOfColumns) */
160 {
161     minDim = nOfColumns;
162
163     for (col = 0; col<nOfColumns; col++)
164     {
165         /* find the smallest element in the column */
166         distMatrixTemp = distMatrix + nOfRows*col;
167         columnEnd = distMatrixTemp + nOfRows;
168
169         minValue = *distMatrixTemp++;
170         while (distMatrixTemp < columnEnd)
171         {
172             value = *distMatrixTemp++;
173             if (value < minValue)
174                 minValue = value;
175         }
176
177         /* subtract the smallest element from each element of the column */
178         distMatrixTemp = distMatrix + nOfRows*col;
179         while (distMatrixTemp < columnEnd)
180             *distMatrixTemp++ -= minValue;
181     }
182
183     /* Steps 1 and 2a */
184     for (col = 0; col<nOfColumns; col++)
185         for (row = 0; row<nOfRows; row++)
186             if (fabs(distMatrix[row + nOfRows*col]) < DBL_EPSILON)

```

```

187         if (!coveredRows[row])
188         {
189             starMatrix[row + nOfRows*col] = true;
190             coveredColumns[col] = true;
191             coveredRows[row] = true;
192             break;
193         }
194     for (row = 0; row<nOfRows; row++)
195         coveredRows[row] = false;
196
197 }
198
199 /* move to step 2b */
200 step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
201
202 /* compute cost and remove invalid assignments */
203 computeassignmentcost(assignment, cost, distMatrixIn, nOfRows);
204
205 /* free allocated memory */
206 free(distMatrix);
207 free(coveredColumns);
208 free(coveredRows);
209 free(starMatrix);
210 free(primeMatrix);
211 free(newStarMatrix);
212
213 return;
214 }
215
216 /*****/
217 void HungarianAlgorithm::buildassignmentvector(int *assignment, bool *starMatrix, int
nOfRows, int nOfColumns)
218 {
219     int row, col;
220
221     for (row = 0; row<nOfRows; row++)
222         for (col = 0; col<nOfColumns; col++)
223             if (starMatrix[row + nOfRows*col])
224             {
225 #ifdef ONE_INDEXING
226                 assignment[row] = col + 1; /* MATLAB-Indexing */
227 #else
228                 assignment[row] = col;
229 #endif
230                 break;
231             }
232 }
233
234 /*****/
235 void HungarianAlgorithm::computeassignmentcost(int *assignment, double *cost, double
*distMatrix, int nOfRows)
236 {
237     int row, col;
238
239     for (row = 0; row<nOfRows; row++)
240     {
241         col = assignment[row];
242         if (col >= 0)
243             *cost += distMatrix[row + nOfRows*col];
244     }
245 }
246
247 /*****/
248 void HungarianAlgorithm::step2a(int *assignment, double *distMatrix, bool *starMatrix,
bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
nOfRows, int nOfColumns, int minDim)
249 {
250     bool *starMatrixTemp, *columnEnd;

```

```

251     int col;
252
253     /* cover every column containing a starred zero */
254     for (col = 0; col<nOfColumns; col++)
255     {
256         starMatrixTemp = starMatrix + nOfRows*col;
257         columnEnd = starMatrixTemp + nOfRows;
258         while (starMatrixTemp < columnEnd){
259             if (*starMatrixTemp++)
260             {
261                 coveredColumns[col] = true;
262                 break;
263             }
264         }
265     }
266
267     /* move to step 3 */
268     step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
269         coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
270
271     /*****
272 void HungarianAlgorithm::step2b(int *assignment, double *distMatrix, bool *starMatrix,
273 bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
274 nOfRows, int nOfColumns, int minDim)
275 {
276     int col, nOfCoveredColumns;
277
278     /* count covered columns */
279     nOfCoveredColumns = 0;
280     for (col = 0; col<nOfColumns; col++)
281         if (coveredColumns[col])
282             nOfCoveredColumns++;
283
284     if (nOfCoveredColumns == minDim)
285     {
286         /* algorithm finished */
287         buildassignmentvector(assignment, starMatrix, nOfRows, nOfColumns);
288     }
289     else
290     {
291         /* move to step 3 */
292         step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
293             coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
294     }
295 }
296
297 /*****
298 void HungarianAlgorithm::step3(int *assignment, double *distMatrix, bool *starMatrix,
299 bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
300 nOfRows, int nOfColumns, int minDim)
301 {
302     bool zerosFound;
303     int row, col, starCol;
304
305     zerosFound = true;
306     while (zerosFound)
307     {
308         zerosFound = false;
309         for (col = 0; col<nOfColumns; col++)
310             if (!coveredColumns[col])
311                 for (row = 0; row<nOfRows; row++)
312                     if ((!coveredRows[row]) && (fabs(distMatrix[row + nOfRows*col]) <
313                         DBL_EPSILON))
314                     {
315                         /* prime zero */
316                         primeMatrix[row + nOfRows*col] = true;

```

```

313         /* find starred zero in current row */
314         for (starCol = 0; starCol<nOfColumns; starCol++)
315             if (starMatrix[row + nOfRows*starCol])
316                 break;
317
318         if (starCol == nOfColumns) /* no starred zero found */
319         {
320             /* move to step 4 */
321             step4(assignment, distMatrix, starMatrix, newStarMatrix,
322                 primeMatrix, coveredColumns, coveredRows, nOfRows,
323                 nOfColumns, minDim, row, col);
324             return;
325         }
326         else
327         {
328             coveredRows[row] = true;
329             coveredColumns[starCol] = false;
330             zerosFound = true;
331             break;
332         }
333     }
334 }
335
336 /* move to step 5 */
337 step5(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
338     coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
339 }
340
341 /*****
342 void HungarianAlgorithm::step4(int *assignment, double *distMatrix, bool *starMatrix,
343     bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
344     nOfRows, int nOfColumns, int minDim, int row, int col)
345 {
346     int n, starRow, starCol, primeRow, primeCol;
347     int nOfElements = nOfRows*nOfColumns;
348
349     /* generate temporary copy of starMatrix */
350     for (n = 0; n<nOfElements; n++)
351         newStarMatrix[n] = starMatrix[n];
352
353     /* star current zero */
354     newStarMatrix[row + nOfRows*col] = true;
355
356     /* find starred zero in current column */
357     starCol = col;
358     for (starRow = 0; starRow<nOfRows; starRow++)
359         if (starMatrix[starRow + nOfRows*starCol])
360             break;
361
362     while (starRow<nOfRows)
363     {
364         /* unstar the starred zero */
365         newStarMatrix[starRow + nOfRows*starCol] = false;
366
367         /* find primed zero in current row */
368         primeRow = starRow;
369         for (primeCol = 0; primeCol<nOfColumns; primeCol++)
370             if (primeMatrix[primeRow + nOfRows*primeCol])
371                 break;
372
373         /* star the primed zero */
374         newStarMatrix[primeRow + nOfRows*primeCol] = true;
375
376         /* find starred zero in current column */
377         starCol = primeCol;
378         for (starRow = 0; starRow<nOfRows; starRow++)
379             if (starMatrix[starRow + nOfRows*starCol])
380                 break;
381     }
382 }

```



```

377
378     /* use temporary copy as new starMatrix */
379     /* delete all primes, uncover all rows */
380     for (n = 0; n<nOfElements; n++)
381     {
382         primeMatrix[n] = false;
383         starMatrix[n] = newStarMatrix[n];
384     }
385     for (n = 0; n<nOfRows; n++)
386         coveredRows[n] = false;
387
388     /* move to step 2a */
389     step2a(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
390         coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
391 }
392
393 /*****
394 void HungarianAlgorithm::step5(int *assignment, double *distMatrix, bool *starMatrix,
395 bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
396 nOfRows, int nOfColumns, int minDim)
397 {
398     double h, value;
399     int row, col;
400
401     /* find smallest uncovered element h */
402     h = DBL_MAX;
403     for (row = 0; row<nOfRows; row++)
404         if (!coveredRows[row])
405             for (col = 0; col<nOfColumns; col++)
406                 if (!coveredColumns[col])
407                     {
408                         value = distMatrix[row + nOfRows*col];
409                         if (value < h)
410                             h = value;
411                     }
412
413     /* add h to each covered row */
414     for (row = 0; row<nOfRows; row++)
415         if (coveredRows[row])
416             for (col = 0; col<nOfColumns; col++)
417                 distMatrix[row + nOfRows*col] += h;
418
419     /* subtract h from each uncovered column */
420     for (col = 0; col<nOfColumns; col++)
421         if (!coveredColumns[col])
422             for (row = 0; row<nOfRows; row++)
423                 distMatrix[row + nOfRows*col] -= h;
424
425     /* move to step 3 */
426     step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
427         coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
428 }
429
430 int main()
431 {
432     int n;
433     scanf("%d", &n);
434
435     vector<vector<double>> a(n, vector<double>(n));
436
437     for(int i = 0; i < n; i++){
438         for(int j = 0; j < n; j++){
439             int x;
440             scanf("%d", &x);
441             a[i][j] = log2(x);
442             a[i][j] = 20.0 - a[i][j];
443         }
444     }
445 }

```

```
442     vector<int> ans(n, -1), ans1 = ans;
443     HungarianAlgorithm var;
444     double cost = var.Solve(a, ans);
445
446     for(int i = 0; i < n; i++){
447         ans1[ans[i]] = i;
448     }
449
450     for(int i : ans1) cout << i+1 << " ";
451
452     return 0 ;
453 }
454
455
456
```

```

1 // kth element log(n) (sparse segment tree)
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6 using ll = long long;
7
8 struct node {
9     int sum = 0;
10    node *left, *right;
11    node(){
12        left = right = nullptr;
13    }
14 } *root = new node();
15
16 void add(int num, node* root, int l, int r){ // add num to the tree
17     if(l == r){
18         root->sum++;
19         return;
20     }
21     int mid = (l+r)/2;
22     if(num <= mid){
23         if(root->left == nullptr) root->left = new node();
24         add(num, root->left, l, mid);
25     } else {
26         if(root->right == nullptr) root->right = new node();
27         add(num, root->right, mid+1, r);
28     }
29
30     root->sum = 0;
31     if(root->left) root->sum += root->left->sum;
32     if(root->right) root->sum += root->right->sum;
33 }
34
35 void del(int num, node* root, int l, int r){ // delete num-th element (need to check if
the num-th element is exist)
36     if(l == r){ // need to check if the
37         root->sum--;
38         return;
39     }
40
41     int mid = (l+r)/2;
42     if(root->left != nullptr && root->left->sum >= num){
43         del(num, root->left, l, mid);
44     } else {
45         if(root->left != nullptr) num -= root->left->sum;
46         del(num, root->right, mid+1, r);
47     }
48
49     root->sum = 0;
50     if(root->left) root->sum += root->left->sum;
51     if(root->right) root->sum += root->right->sum;
52 }
53
54 int get(int num, node* root, int l, int r){ // get k-th element
55     if(l == r){
56         return r;
57     }
58
59     int mid = (l+r)/2;
60     if(root->left != nullptr && root->left->sum >= num){
61         return get(num, root->left, l, mid);
62     } else {
63         if(root->left != nullptr) num -= root->left->sum;
64         return get(num, root->right, mid+1, r);
65     }
66 }
67
68 int getSum(node* root, int l, int r, int tl, int tr){

```

```

69     if(l > r) return 0;
70     if(l == tl && r == tr) return root->sum;
71     int mid = (l+r)/2;
72     int f = 0, s = 0;
73     if(root->left != nullptr) f = getSum(root->left, l, mid, tl, min(mid, tr));
74     if(root->right != nullptr) s = getSum(root->right, mid+1, r, max(mid+1, tl), tr);
75     return f + s;
76 }
77
78 void prtAny(node* root, int l, int r){
79     assert(root != nullptr);
80     if(l == r){
81         assert(root->sum > 0);
82         printf("%d", l);
83         return;
84     }
85     int mid = (l+r)/2;
86     if(root->left != nullptr && root->left->sum != 0) prtAny(root->left, l, mid);
87     else prtAny(root->right, mid+1, r);
88 }
89
90 int main()
91 {
92     int n, q;
93
94     scanf("%d%d", &n, &q);
95
96     for(int i = 0; i < n; i++){
97         int x;
98         scanf("%d", &x);
99         add(x, root, 1, n);
100     }
101
102     for(int i = 0; i < q; i++){
103         int x;
104         scanf("%d", &x);
105         if(x < 0){
106             del(-x, root, 1, n);
107         } else {
108             add(x, root, 1, n);
109         }
110     }
111
112     if(root->sum == 0) printf("0");
113     else {
114         prtAny(root, 1, n);
115     }
116
117     return 0;
118 }
119
120
121
122
123
124

```

```

1 // li chao tree
2
3 typedef long long ll;
4
5 const int C = (int)1e5 + 5;
6 const int N = (int)1e5 + 5;
7 const ll inf = (ll)1e18;
8
9 struct Line {
10     ll m, b;
11     ll operator()(ll x) { return m * x + b; }
12 };
13 struct Node {
14     Line seg;
15     Node *lson, *rson;
16     Node(Line _seg): seg(_seg), lson(0), rson(0) {}
17 };
18 void insert(int l, int r, Line seg, Node* o) {
19     if(l + 1 == r) {
20         if(seg(l) < o->seg(l)) o->seg = seg;
21         return;
22     }
23     int mid = (l + r) >> 1;
24     if(seg.m < o->seg.m) swap(seg, o->seg);
25     if(o->seg(mid) > seg(mid)) {
26         swap(seg, o->seg);
27         if(o->rson) insert(mid, r, seg, o->rson);
28         else o->rson = new Node(seg);
29     }
30     else {
31         if(o->lson) insert(l, mid, seg, o->lson);
32         else o->lson = new Node(seg);
33     }
34 }
35 ll query(int l, int r, int x, Node* o) {
36     if(l + 1 == r) return o->seg(x);
37     int mid = (l + r) >> 1;
38     if(x < mid && o->lson) return min(o->seg(x), query(l, mid, x, o->lson));
39     else if(o->rson) return min(o->seg(x), query(mid, r, x, o->rson));
40     return o->seg(x);
41 }
42 void del(Node* o) {
43     if(o->lson) del(o->lson);
44     if(o->rson) del(o->rson);
45     delete o;
46 }

```

```

1 // Linear Diophantine Equation
2
3 int gcd(int a, int b, int& x, int& y) {
4     if (b == 0) {
5         x = 1;
6         y = 0;
7         return a;
8     }
9     int x1, y1;
10    int d = gcd(b, a % b, x1, y1);
11    x = y1;
12    y = x1 - y1 * (a / b);
13    return d;
14 }
15
16 bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
17     g = gcd(abs(a), abs(b), x0, y0);
18     if (c % g) {
19         return false;
20     }
21
22     x0 *= c / g;
23     y0 *= c / g;
24     if (a < 0) x0 = -x0;
25     if (b < 0) y0 = -y0;
26     return true;
27 }
28
29 void shift_solution(int &x, int &y, int a, int b, int cnt) {
30     x += cnt * b;
31     y -= cnt * a;
32 }
33
34 int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
35     int x, y, g;
36     if (!find_any_solution(a, b, c, x, y, g))
37         return 0;
38     a /= g;
39     b /= g;
40
41     int sign_a = a > 0 ? +1 : -1;
42     int sign_b = b > 0 ? +1 : -1;
43
44     shift_solution(x, y, a, b, (minx - x) / b);
45     if (x < minx)
46         shift_solution(x, y, a, b, sign_b);
47     if (x > maxx)
48         return 0;
49     int lx1 = x;
50
51     shift_solution(x, y, a, b, (maxx - x) / b);
52     if (x > maxx)
53         shift_solution(x, y, a, b, -sign_b);
54     int rx1 = x;
55
56     shift_solution(x, y, a, b, -(miny - y) / a);
57     if (y < miny)
58         shift_solution(x, y, a, b, -sign_a);
59     if (y > maxy)
60         return 0;
61     int lx2 = x;
62
63     shift_solution(x, y, a, b, -(maxy - y) / a);
64     if (y > maxy)
65         shift_solution(x, y, a, b, sign_a);
66     int rx2 = x;
67
68     if (lx2 > rx2)
69         swap(lx2, rx2);

```

```
70     int lx = max(lx1, lx2);
71     int rx = min(rx1, rx2);
72
73     if (lx > rx)
74         return 0;
75     return (rx - lx) / abs(b) + 1;
76 }
```

```

1 // lyndon factorization duval algorithm
2
3 vector<string> duval(string const& s) {
4     int n = s.size();
5     int i = 0;
6     vector<string> factorization;
7     while (i < n) {
8         int j = i + 1, k = i;
9         while (j < n && s[k] <= s[j]) {
10             if (s[k] < s[j])
11                 k = i;
12             else
13                 k++;
14             j++;
15         }
16         while (i <= k) {
17             factorization.push_back(s.substr(i, j - k));
18             i += j - k;
19         }
20     }
21     return factorization;
22 }

```



```

1 // manacher algorithm
2
3 #include <bits/stdc++.h>
4
5 using namespace std;
6 using ll = long long;
7
8 vector<int> manacher_odd(string s) {
9     int n = s.size();
10    s = "$" + s + "^";
11    vector<int> p(n + 2);
12    int l = 0, r = -1;
13    for(int i = 1; i <= n; i++) {
14        p[i] = max(0, min(r - i, p[l + (r - i)]));
15        while(s[i - p[i]] == s[i + p[i]]) {
16            p[i]++;
17        }
18        if(i + p[i] > r) {
19            l = i - p[i], r = i + p[i];
20        }
21    }
22    return vector<int>(begin(p) + 1, end(p) - 1);
23 }
24
25 vector<int> manacher(string s) {
26     string t;
27     for(auto c: s) {
28         t += string("#") + c;
29     }
30     auto res = manacher_odd(t + "#");
31     return vector<int>(begin(res) + 1, end(res) - 1);
32 }
33
34 char buf[1000010];
35
36 int main()
37 {
38
39     scanf("%s", buf);
40     string s = buf;
41     vector<int> palindorm = manacher(s);
42
43     string s1 = "";
44     for(int i = 0; i < s.length(); i++){
45         s1 += s[i];
46         s1 += "#";
47     }
48
49     s = s1;
50     s.pop_back();
51     cout << s << endl;
52     int ans = 0; // number of subpalindromes
53     for(int i = 0; i < palindorm.size(); i += 1){
54         cout << palindorm[i] << " ";
55         ans += palindorm[i]/2;
56         // if(s[i] == "#") even palindrome with (s[i]-1)/2 length, centers in s[i] and
57         // else odd palindrome with s[i]/2 length, centers in s[i]
58     }
59     cout << endl;
60
61     cout << "ans " << ans << endl;
62
63     return 0;
64 }
65

```

```

1 // max flow dinic
2
3 struct FlowEdge {
4     int v, u;
5     long long cap, flow = 0;
6     FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
7 };
8
9 struct Dinic {
10     const long long flow_inf = 1e18;
11     vector<FlowEdge> edges;
12     vector<vector<int>>> adj;
13     int n, m = 0;
14     int s, t;
15     vector<int> level, ptr;
16     queue<int> q;
17
18     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
19         adj.resize(n);
20         level.resize(n);
21         ptr.resize(n);
22     }
23
24     void add_edge(int v, int u, long long cap) {
25         edges.emplace_back(v, u, cap);
26         edges.emplace_back(u, v, 0);
27         adj[v].push_back(m);
28         adj[u].push_back(m + 1);
29         m += 2;
30     }
31
32     bool bfs() {
33         while (!q.empty()) {
34             int v = q.front();
35             q.pop();
36             for (int id : adj[v]) {
37                 if (edges[id].cap - edges[id].flow < 1)
38                     continue;
39                 if (level[edges[id].u] != -1)
40                     continue;
41                 level[edges[id].u] = level[v] + 1;
42                 q.push(edges[id].u);
43             }
44         }
45         return level[t] != -1;
46     }
47
48     long long dfs(int v, long long pushed) {
49         if (pushed == 0)
50             return 0;
51         if (v == t)
52             return pushed;
53         for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
54             int id = adj[v][cid];
55             int u = edges[id].u;
56             if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
57                 continue;
58             long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
59             if (tr == 0)
60                 continue;
61             edges[id].flow += tr;
62             edges[id ^ 1].flow -= tr;
63             return tr;
64         }
65         return 0;
66     }
67
68     long long flow() {
69         long long f = 0;

```

```

70     while (true) {
71         fill(level.begin(), level.end(), -1);
72         level[s] = 0;
73         q.push(s);
74         if (!bfs())
75             break;
76         fill(ptr.begin(), ptr.end(), 0);
77         while (long long pushed = dfs(s, flow_inf)) {
78             f += pushed;
79         }
80     }
81     return f;
82 }
83 };
84

```

```

1  // max flow VE complexity, (improved push preflow)
2
3  const int inf = 1000000000;
4
5  int n;
6  vector<vector<int>> capacity, flow;
7  vector<int> height, excess;
8
9  void push(int u, int v)
10 {
11     int d = min(excess[u], capacity[u][v] - flow[u][v]);
12     flow[u][v] += d;
13     flow[v][u] -= d;
14     excess[u] -= d;
15     excess[v] += d;
16 }
17
18 void relabel(int u)
19 {
20     int d = inf;
21     for (int i = 0; i < n; i++) {
22         if (capacity[u][i] - flow[u][i] > 0)
23             d = min(d, height[i]);
24     }
25     if (d < inf)
26         height[u] = d + 1;
27 }
28
29 vector<int> find_max_height_vertices(int s, int t) {
30     vector<int> max_height;
31     for (int i = 0; i < n; i++) {
32         if (i != s && i != t && excess[i] > 0) {
33             if (!max_height.empty() && height[i] > height[max_height[0]])
34                 max_height.clear();
35             if (max_height.empty() || height[i] == height[max_height[0]])
36                 max_height.push_back(i);
37         }
38     }
39     return max_height;
40 }
41
42 int max_flow(int s, int t)
43 {
44     height.assign(n, 0);
45     height[s] = n;
46     flow.assign(n, vector<int>(n, 0));
47     excess.assign(n, 0);
48     excess[s] = inf;
49     for (int i = 0; i < n; i++) {
50         if (i != s)
51             push(s, i);
52     }
53
54     vector<int> current;
55     while (!(current = find_max_height_vertices(s, t)).empty()) {
56         for (int i : current) {
57             bool pushed = false;
58             for (int j = 0; j < n && excess[i]; j++) {
59                 if (capacity[i][j] - flow[i][j] > 0 && height[i] == height[j] + 1) {
60                     push(i, j);
61                     pushed = true;
62                 }
63             }
64             if (!pushed) {
65                 relabel(i);
66                 break;
67             }
68         }
69     }

```

```
70
71     int max_flow = 0;
72     for (int i = 0; i < n; i++)
73         max_flow += flow[i][t];
74     return max_flow;
75 }
```

```

1 // min cost max flow dijkstra + johnson's algorithm
2
3 // source: https://codeforces.com/blog/entry/95823
4
5 /*
6
7 If there are negative weights then we should calculate potential by bellman-ford
8 algorithm for the first time
9
10 */
11 template<typename Cap, typename Cost>
12 struct mcmf {
13     struct edge {
14         int v;
15         Cap cap, flow;
16         Cost cost;
17     };
18     int n;
19     vector<edge> e;
20     vector<vector<int>>> g;
21     vector<Cost> dist, pot;
22     vector<Cap> f;
23     vector<bool> vis;
24     vector<int> par;
25     bool n2dijkstra = false;
26     mcmf(int n) : n(n), g(n), dist(n), pot(n), f(n), vis(n), par(n) {}
27     void add_edge(int u, int v, Cap cap, Cost cost) {
28         int k = e.size();
29         e.push_back({v, cap, 0, cost});
30         e.push_back({u, cap, cap, -cost});
31         g[u].push_back(k);
32         g[v].push_back(k ^ 1);
33     }
34     pair<Cap, Cost> solve(int s, int t) {
35         Cap flow = 0;
36         Cost cost = 0;
37         while(true) {
38             fill(dist.begin(), dist.end(), numeric_limits<Cost>::max());
39             fill(vis.begin(), vis.end(), false);
40             dist[s] = 0;
41             f[s] = numeric_limits<Cap>::max();
42             if(n2dijkstra) {
43                 while(true) {
44                     int x = -1; Cost d = numeric_limits<Cost>::max();
45                     for(int i = 0; i < n; i++) {
46                         if(!vis[i] && dist[i] < d) {
47                             x = i;
48                             d = dist[x];
49                         }
50                     }
51                     if(x == -1) break;
52                     vis[x] = true;
53                     for(int i : g[x]) {
54                         Cost d2 = d + e[i].cost + pot[x] - pot[e[i].v];
55                         if(!vis[e[i].v] && e[i].flow < e[i].cap && d2 < dist[e[i].v]) {
56                             dist[e[i].v] = d2;
57                             f[e[i].v] = min(f[x], e[i].cap - e[i].flow);
58                             par[e[i].v] = i;
59                         }
60                     }
61                 }
62             } else {
63                 priority_queue<pair<Cost, int>, vector<pair<Cost, int>>, greater<>> Q;
64                 Q.push({0, s});
65                 while(!Q.empty()) {
66                     Cost d; int x;
67                     tie(d, x) = Q.top(); Q.pop();
68                     if(vis[x]) continue;

```

```

69         vis[x] = true;
70         for(int i : g[x]) {
71             Cost d2 = d + e[i].cost + pot[x] - pot[e[i].v];
72             if(!vis[e[i].v] && e[i].flow < e[i].cap && d2 < dist[e[i].v]) {
73                 dist[e[i].v] = d2;
74                 f[e[i].v] = min(f[x], e[i].cap - e[i].flow);
75                 par[e[i].v] = i;
76                 Q.push({d2, e[i].v});
77             }
78         }
79     }
80 }
81 if(!vis[t]) break;
82 for(int i = 0; i < n; i++) {
83     dist[i] += pot[i] - pot[s];
84 }
85 cost += dist[t] * f[t];
86 flow += f[t];
87 int x = t;
88 while(x != s) {
89     e[par[x]].flow += f[t];
90     e[par[x] ^ 1].flow -= f[t];
91     x = e[par[x] ^ 1].v;
92 }
93 dist.swap(pot);
94 }
95 return {flow, cost};
96 }
97 };

```

```

1  // mo's algorithm
2
3  void remove(idx); // TODO: remove value at idx from data structure
4  void add(idx);    // TODO: add value at idx from data structure
5  int get_answer(); // TODO: extract the current answer of the data structure
6
7  const int block_size; // 700 or 800 may run better than 750
8
9  struct Query {
10     int l, r, idx;
11     bool operator<(Query other) const
12     {
13         return make_pair(l / block_size, r) <
14                make_pair(other.l / block_size, other.r);
15     }
16 };
17
18 vector<int> mo_s_algorithm(vector<Query> queries) {
19     vector<int> answers(queries.size());
20     sort(queries.begin(), queries.end());
21
22     // TODO: initialize data structure
23
24     int cur_l = 0;
25     int cur_r = -1;
26     // invariant: data structure will always reflect the range [cur_l, cur_r]
27     for (Query q : queries) {
28         while (cur_l > q.l) {
29             cur_l--;
30             add(cur_l);
31         }
32         while (cur_r < q.r) {
33             cur_r++;
34             add(cur_r);
35         }
36         while (cur_l < q.l) {
37             remove(cur_l);
38             cur_l++;
39         }
40         while (cur_r > q.r) {
41             remove(cur_r);
42             cur_r--;
43         }
44         answers[q.idx] = get_answer();
45     }
46     return answers;
47 }
48

```



```

1  // persistent segment tree
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  using ll = long long;
7
8  const int MaxNum = 2e5+10;
9
10 struct Vertex {
11     Vertex *l, *r;
12     ll sum;
13
14     Vertex(ll val) : l(nullptr), r(nullptr), sum(val) {}
15     Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
16         if (l) sum += l->sum;
17         if (r) sum += r->sum;
18     }
19 };
20
21 Vertex* build(int tl, int tr) {
22     if (tl == tr)
23         return new Vertex(0);
24     int tm = (tl + tr) / 2;
25     return new Vertex(build(tl, tm), build(tm+1, tr));
26 }
27
28 Vertex* update(Vertex* v, int tl, int tr, int pos, int val) {
29     if (tl == tr)
30         return new Vertex(0LL + val + v->sum);
31     int tm = (tl + tr) / 2;
32     if (pos <= tm)
33         return new Vertex(update(v->l, tl, tm, pos, val), v->r);
34     else
35         return new Vertex(v->l, update(v->r, tm+1, tr, pos, val));
36 }
37
38 ll get_sum(Vertex* v, Vertex *left, int tl, int tr, int l, int r) {
39     if (l > r)
40         return 0;
41     if (l == tl && tr == r){
42         //cout << "l " << l << " r " << r << " sum " << v->sum << " left " << left->sum
43         << endl;
44         return v->sum - left->sum;
45     }
46     int tm = (tl + tr) / 2;
47     return get_sum(v->l, left->l, tl, tm, l, min(r, tm))
48         + get_sum(v->r, left->r, tm+1, tr, max(l, tm+1), r);
49 }
50
51 void prt(Vertex *v, int tl, int tr){
52     if(tl == tr){
53         if(v->sum) cout << "tl " << tl << " sum " << v->sum << endl;
54         return;
55     }
56     int mid = (tl + tr)/2;
57     prt(v->l, tl, mid);
58     prt(v->r, mid+1, tr);
59 }
60
61 int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
62     if (tl == tr)
63         return tl;
64     int tm = (tl + tr) / 2, left_count = vr->l->sum - vl->l->sum;
65     if (left_count >= k)
66         return find_kth(vl->l, vr->l, tl, tm, k);
67     return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
68 }

```

```

69  int main()
70  {
71
72      int tl = 0, tr = MaxNum + 1;
73      vector<Vertex*> roots;
74      roots.push_back(build(tl, tr));
75      vector<Vertex*> roots1;
76      roots1.push_back(build(tl, tr));
77      vector<Vertex*> roots2;
78      roots2.push_back(build(tl, tr));
79      vector<Vertex*> roots3;
80      roots3.push_back(build(tl, tr));
81      vector<Vertex*> roots11;
82      roots11.push_back(build(tl, tr));
83      vector<Vertex*> roots21;
84      roots21.push_back(build(tl, tr));
85
86
87      int n;
88      scanf("%d", &n);
89
90      for(int i = 0; i < n; i++){
91          int x1, x2, y1, a, b, y2;
92          scanf("%d%d%d%d%d", &x1, &x2, &y1, &a, &b, &y2);
93          roots.push_back(update(roots.back(), tl, tr, x1, y1));
94          roots1.push_back(update(roots1.back(), tl, tr, x1, -b));
95          roots11.push_back(update(roots11.back(), tl, tr, x2, b));
96          roots2.push_back(update(roots2.back(), tl, tr, x1, -a));
97          roots21.push_back(update(roots21.back(), tl, tr, x2, a));
98          roots3.push_back(update(roots3.back(), tl, tr, x2+1, y2));
99
100     }
101
102     //prt(roots3.back(), tl, tr);
103
104     int m;
105     scanf("%d", &m);
106     ll last = 0;
107     ll l, r, x;
108     const ll mod = 1e9;
109     while(m--){
110         scanf("%lld%lld%lld", &l, &r, &x);
111         x = (x + last)%mod;
112         if(x <= MaxNum){
113             last = get_sum(roots[r], roots[l-1], tl, tr, x, tr);
114             last += get_sum(roots1[r], roots1[l-1], tl, tr, x, tr);
115             last += get_sum(roots11[r], roots11[l-1], tl, tr, x, tr);
116             last += x*get_sum(roots2[r], roots2[l-1], tl, tr, x, tr);
117             last += x*get_sum(roots21[r], roots21[l-1], tl, tr, x, tr);
118             last += get_sum(roots3[r], roots3[l-1], tl, tr, tl, x);
119         } else {
120             x = MaxNum;
121             last = get_sum(roots3[r], roots3[l-1], tl, tr, tl, x);
122         }
123         printf("%lld\n", last);
124     }
125
126     return 0;
127 }
128

```

```

1  // prefix function + automata
2
3  vector<int> prefix_function(string s) {
4      int n = (int)s.length();
5      vector<int> pi(n);
6      for (int i = 1; i < n; i++) {
7          int j = pi[i-1];
8          while (j > 0 && s[i] != s[j])
9              j = pi[j-1];
10         if (s[i] == s[j])
11             j++;
12         pi[i] = j;
13     }
14     return pi;
15 }
16 void compute_automaton(string s, vector<vector<int>>& aut) {
17     s += '#';
18     int n = s.size();
19     vector<int> pi = prefix_function(s);
20     aut.assign(n, vector<int>(26));
21     for (int i = 0; i < n; i++) {
22         for (int c = 0; c < 26; c++) {
23             if (i > 0 && 'a' + c != s[i])
24                 aut[i][c] = aut[pi[i-1]][c];
25             else
26                 aut[i][c] = i + ('a' + c == s[i]);
27         }
28     }
29 }

```

```

1  // primality test and integer factorization
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  using ll = uint64_t;
7
8  const int N = 100001;
9
10 bool prime[N];
11 vector<int> pr;
12
13 void si(){
14     prime[0] = prime[1] = 1;
15     for(int i = 2; i < N; i++){
16         if(prime[i] == 0){
17             pr.push_back(i);
18             for(ll j = 1LL*i*i; j < N; j += i){
19                 prime[j] = 1;
20             }
21         }
22     }
23 }
24
25 ll gcd(ll x, ll y){
26     while(y){
27         x = x%y;
28         swap(x, y);
29     }
30     return x;
31 }
32
33 ll mul(ll a, ll b, ll mod) {
34     long long result = 0;
35     while (b) {
36         if (b & 1)
37             result = (result + a) % mod;
38         a = (a + a) % mod;
39         b >>= 1;
40     }
41     return result;
42 }
43
44 ll binpower(ll base, ll e, ll mod)
45 {
46     ll result = 1;
47     base %= mod;
48     while(e){
49         if(e & 1)
50             result = mul(result, base, mod);
51         base = mul(base, base, mod);
52         e >>= 1;
53     }
54     return result;
55 }
56
57 bool check_compsite(ll n, ll a, ll d, int s)
58 {
59     ll x = binpower(a, d, n);
60     if(x == 1 || x == n - 1)
61         return false;
62     for(int r = 1; r < s; r++){
63         x = mul(x, x, n);
64         if(x == n - 1)
65             return false;
66     }
67     return true;
68 };
69

```

```

70 bool MillerRabin(ll n) // returns true if n is probably prime, else returns false.
71 {
72     if(n < 2)
73         return false;
74     int s = 0;
75     ll d = n - 1;
76     while((d & 1) == 0){
77         d >>= 1;
78         s++;
79     }
80     for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
81         if (n == a)
82             return true;
83         if (check_composite(n, a, d, s))
84             return false;
85     }
86     return true;
87 }
88
89 bool isPrime(ll n){
90     return MillerRabin(n);
91 }
92
93 ll f(ll x, ll c, ll mod) {
94     return (mul(x, x, mod) + c) % mod;
95 }
96
97 ll rho(ll n, ll x0 = 2, ll c = 1) {
98     ll x = x0;
99     ll y = x0;
100    ll g = 1;
101    while (g == 1) {
102        x = f(x, c, n);
103        y = f(y, c, n);
104        y = f(y, c, n);
105        g = gcd(max(x, y) - min(x, y), n);
106    }
107    return g;
108 }
109
110 vector<ll> factorize_small(ll n){
111     vector<ll> res;
112     for(int i : pr){
113         if(1LL*i*i > n) break;
114         while(n%i == 0){
115             n /= i;
116             res.push_back(i);
117         }
118     }
119     if(n > 1) res.push_back(n);
120     return res;
121 }
122
123 vector<ll> factorize(ll n){
124     if(n <= 100000) return factorize_small(n);
125     vector<ll> res;
126     if(n == 1) return res;
127     if(isPrime(n)){
128         res.push_back(n);
129         return res;
130     }
131     int x0 = 2, c = 1;
132     bool turn = 0;
133     ll g = rho(n, x0, c);
134     while(g == n){
135         if(turn) x0++;
136         else c++;
137         turn = 1 - turn;
138         g = rho(n, x0, c);

```

```

139         //cout << "n " << n << " g " << g << endl;
140     }
141     vector<ll> cur = factorize(g);
142     for(ll x : cur) res.push_back(x);
143     n /= g;
144     cur = factorize(n);
145     for(ll x : cur) res.push_back(x);
146     return res;
147 }
148
149 int main() {
150
151     si();
152     int t = 1;
153     //cin >> t;
154
155     while(t){
156
157         ll n;
158         cin >> n;
159         if(n == 0) break;
160
161
162         vector<ll> factors = factorize(n);
163         n = factors.size();
164         sort(factors.begin(), factors.end());
165         for(int i = 0; i < n;){
166             int cnt = 0, j = i;
167             while(i < n && factors[j] == factors[i]) cnt++, i++;
168             cout << factors[j] << "^" << cnt << " \n"[i == n];
169         }
170
171     }
172
173     return 0;
174
175 }
176

```

```
1 // random number generator
2
3 mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
4 return value between 0 and  $2^{32} - 1$  which is unsigned integer
5
6 mt19937_64 rng64(chrono::steady_clock::now().time_since_epoch().count());
7 the same problematic
8
9 random shuffle:
10 for (int i = 1; i < N; i++)
11     swap(permutation[i], permutation[uniform_int_distribution<int>(0, i)(rng32)]);
```

```

1  // randomized heap
2
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  using ll = long long;
7
8  struct Tree {
9      int value;
10     Tree* l = nullptr;
11     Tree* r = nullptr;
12     Tree(int val){
13         this->value = val;
14     }
15 };
16
17 typedef Tree* pt;
18
19 Tree* merge(Tree* t1, Tree* t2) {
20     if (!t1 || !t2)
21         return t1 ? t1 : t2;
22     if (t2->value < t1->value)
23         swap(t1, t2);
24     if (rand() & 1)
25         swap(t1->l, t1->r);
26     t1->l = merge(t1->l, t2);
27     return t1;
28 }
29
30 void output(pt t){
31     if(t == nullptr) return;
32     cout << t->value << " ";
33     output(t->l);
34     output(t->r);
35 }
36
37 void erase(pt& t){
38     if(t == nullptr) assert(false);
39     pt temp = t;
40     t = merge(t->l, t->r);
41     delete temp;
42 }
43
44 void insert(pt& t, int val){
45     pt node = new Tree(val);
46     t = merge(t, node);
47 }
48
49 int front(pt t){
50     if(t == nullptr) assert(false);
51     return t->value;
52 }
53
54 int main() {
55
56     int t;
57     cin >> t;
58     pt root = nullptr;
59     while(t--){
60         int x;
61         cin >> x;
62         if(x == 1){
63             int y;
64             cin >> y;
65             insert(root, y);
66         } else if(x==2){
67             erase(root);
68         } else if(x == 3){
69             cout << "front " << front(root) << endl;

```



```
70     }
71     output(root);
72     puts("");
73 }
74
75 }
76
```

```

1  // sparse table
2
3  struct sparse_table {
4      int N, K;
5      vector<vector<int>> st; // don't forget to replace int with long long when needed
6
7
8      // for min query
9      int log[N+1];
10     log[1] = 0;
11     for (int i = 2; i <= MAXN; i++)
12         log[i] = log[i/2] + 1;
13
14
15     int f(int x){return x;}
16     int f(int x, int y){return x + y;}
17
18     sparse_table(vector<int> a){
19         N = a.size();
20         K = ceil(log2(N)) + 1;
21         st.resize(N, vector<int>(K+1));
22
23         for (int i = 0; i < N; i++) st[i][0] = f(a[i]);
24
25         for (int j = 1; j <= K; j++)
26             for (int i = 0; i + (1 << j) <= N; i++)
27                 st[i][j] = f(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
28
29     }
30
31     ll sumQuery(int L, int R){
32         long long sum = 0;
33         for (int j = K; j >= 0; j--) {
34             if ((1 << j) <= R - L + 1) {
35                 sum += st[L][j];
36                 L += 1 << j;
37             }
38         }
39         return sum;
40     }
41
42     int minQuery(int L, int R){ // or maxQuery
43         int j = log[R - L + 1];
44         int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
45         return minimum;
46     }
47
48 };

```

```

1  // suffix array with comparing substring
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  using ll = long long;
7
8  #include <bits/stdc++.h>
9
10 using namespace std;
11 using ll = long long;
12
13 struct suffix_array {
14     string s;
15     int n;
16     vector<int> p, cnt, c;
17     vector<vector<int>> c1;
18     suffix_array(string s){
19         this->s = s;
20         build();
21     }
22     void build(){
23         s += '$';
24         n = s.size();
25         const int alphabet = 256;
26
27         p.resize(n, 0);
28         c = p;
29         cnt.resize(max(n, alphabet), 0);
30
31         for(auto x : s) cnt[x]++;
32
33         for(int i = 1; i < alphabet; i++) cnt[i] += cnt[i-1];
34
35         for(int i = 0; i < n; i++){
36             p[--cnt[s[i]]] = i;
37         }
38
39         c[p[0]] = 0;
40         int classes = 1;
41
42         for(int i = 1; i < n; i++){
43             if(s[p[i]] != s[p[i-1]]) classes++;
44             c[p[i]] = classes-1;
45         }
46         c1.push_back(c);
47         vector<int> pn(n), cn(n);
48         for (int h = 0; (1 << h) < n; ++h) {
49             for (int i = 0; i < n; i++) {
50                 pn[i] = p[i] - (1 << h);
51                 if (pn[i] < 0)
52                     pn[i] += n;
53             }
54             fill(cnt.begin(), cnt.begin() + classes, 0);
55             for (int i = 0; i < n; i++)
56                 cnt[c[pn[i]]]++;
57             for (int i = 1; i < classes; i++)
58                 cnt[i] += cnt[i-1];
59             for (int i = n-1; i >= 0; i--)
60                 p[--cnt[c[pn[i]]]] = pn[i];
61             cn[p[0]] = 0;
62             classes = 1;
63             for (int i = 1; i < n; i++) {
64                 pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
65                 pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
66                 if (cur != prev)
67                     ++classes;
68                 cn[p[i]] = classes - 1;
69             }

```

```

70         c.swap(cn);
71         cl.push_back(c);
72     }
73     c.pop_back();
74     for(int &i : c) i--;
75     p.erase(p.begin());
76     n--;
77     s.pop_back();
78 }
79
80 bool in_s(string pattern){
81     int l = 0, r = n-1;
82     while(l <= r){
83         int mid = (l+r)/2;
84         int sub = n-p[mid];
85         string x = s.substr(p[mid], min((int)pattern.length(), sub));
86         if(x == pattern) return true;
87         else if(x > pattern){
88             r = mid-1;
89         } else l = mid+1;
90     }
91     return false;
92 }
93
94 int compare(int i, int j, int l, int k) {
95     pair<int, int> a = {cl[k][i], cl[k][(i+l-(1 << k))%n]};
96     pair<int, int> b = {cl[k][j], cl[k][(j+l-(1 << k))%n]};
97     return a == b ? 0 : a < b ? -1 : 1;
98 }
99
100 int compare_substrings(int l, int r, int l2, int r2){
101     if(r-l == r2-l2){
102         return compare(l, l2, r-l+1, log2(r-l+1));
103     } else if(r-l < r2-l2){
104         r2 -= (r2-l2+1)-(r-l+1);
105         int res = compare(l, l2, r-l+1, log2(r-l+1));
106         if(res == 0) return -1;
107         return res;
108     } else {
109         r -= (r-l+1)-(r2-l2+1);
110         int res = compare(l, l2, r-l+1, log2(r-l+1));
111         if(res == 0) return 1;
112         return res;
113     }
114 }
115
116 vector<int> lcp_construction(string const& s, vector<int> const& p) {
117     int n = s.size();
118     vector<int> rank(n, 0);
119     for (int i = 0; i < n; i++)
120         rank[p[i]] = i;
121
122     int k = 0;
123     vector<int> lcp(n-1, 0);
124     for (int i = 0; i < n; i++) {
125         if (rank[i] == n - 1) {
126             k = 0;
127             continue;
128         }
129         int j = p[rank[i] + 1];
130         while (i + k < n && j + k < n && s[i+k] == s[j+k])
131             k++;
132         lcp[rank[i]] = k;
133         if (k)
134             k--;
135     }
136     return lcp;
137 }
138

```

```

139     };
140
141     int main() {
142
143         string s = "ahmadlaghadban";
144         int n = s.length();
145         suffix_array suf(s);
146
147         cout << s << endl;
148         for(int i = 0; i < n; i++){
149             int x = suf.p[i];
150             cout << x << " ";
151         }
152         cout << endl;
153
154         for(int i = 0; i < n; i++){
155             int x = suf.p[i];
156             for(int j = x; j < n; j++) cout << s[j];
157             cout << endl;
158         }
159
160         while(true){
161             int l, r, l2, r2;
162             cin >> l >> r >> l2 >> r2;
163             cout << suf.compare_substrings(l, r, l2, r2) << endl;
164         }
165
166         return 0;
167     }
168

```

```

1  // suffix automaton
2
3  #include <bits/stdc++.h>
4
5  using namespace std;
6  using ll = long long;
7
8  struct state {
9      int len, link;
10     bool clone = 0;
11     map<char, int> next;
12 };
13
14 vector<state> st;
15 vector<ll> cnt, subs;
16
17 struct suffix_automaton {
18     int maxlen, sz, last;
19     string s;
20     void build_again(string s){
21         this->s = s;
22         maxlen = s.length()*2 + 10;
23         st.clear();
24         st.resize(maxlen);
25         cnt.clear();
26         cnt.resize(maxlen*2);
27         subs = cnt;
28         sz = last = 0;
29         build();
30     }
31
32     void build(){
33         sa_init();
34         for(char x : s) sa_extend(x);
35     }
36     suffix_automaton(){};
37     suffix_automaton(string s){
38         this->s = s;
39         maxlen = s.length()*2 + 10;
40         st.resize(maxlen);
41         cnt.resize(maxlen*2);
42         subs = cnt;
43         sz = last = 0;
44         build();
45     }
46
47     void sa_init() {
48         st[0].len = 0;
49         st[0].link = -1;
50         sz = 1;
51         last = 0;
52     }
53
54     void sa_extend(char c) {
55         int cur = sz++;
56         st[cur].len = st[last].len + 1;
57         int p = last;
58         while (p != -1 && !st[p].next.count(c)) {
59             st[p].next[c] = cur;
60             p = st[p].link;
61         }
62         if (p == -1) {
63             st[cur].link = 0;
64         } else {
65             int q = st[p].next[c];
66             if (st[p].len + 1 == st[q].len) {
67                 st[cur].link = q;
68             } else {
69                 int clone = sz++;

```

```

70         st[clone].len = st[p].len + 1;
71         st[clone].next = st[q].next;
72         st[clone].link = st[q].link;
73         st[clone].clone = 1;
74         while (p != -1 && st[p].next[c] == q) {
75             st[p].next[c] = clone;
76             p = st[p].link;
77         }
78         st[q].link = st[cur].link = clone;
79     }
80 }
81 last = cur;
82 }
83
84 void occ(){
85     vector<pair<int, int>> v;
86     for(int i = sz-1; i; i--){
87         cnt[i] = !st[i].clone;
88         v.push_back({st[i].len, i});
89     }
90     sort(v.begin(), v.end());
91     for(int i = v.size()-1; i >= 0; i--){
92         int suf = st[v[i].second].link;
93         cnt[suf] += cnt[v[i].second];
94     }
95     cnt[0] = 0;
96 }
97
98 void difSubs(int sta, int pre = -1){
99     if(subs[sta]) return;
100    if(sta != 0){
101        subs[sta] += cnt[sta]; // if you want only different substrings just
102        replace 'cnt[sta]' with '1';
103    }
104    for(auto [f, s] : st[sta].next){
105        difSubs(s, sta);
106        subs[sta] += subs[s];
107    }
108 }
109
110 int lcs(string t){
111     int v = 0, l = 0, ans = 0;
112     for(int i = 0; i < t.length(); i++){
113         while(v != 0 && st[v].next.count(t[i]) == 0){
114             v = st[v].link;
115             l = st[v].len;
116         }
117         if(st[v].next.count(t[i])) v = st[v].next[t[i]], l++;
118         ans = max(ans, l);
119     }
120     return ans;
121 }
122
123 void prt(int sta, string cur){
124     if(st[sta].next.empty()) cout << "cur " << cur << endl;
125     else {
126         for(auto [f,s] : st[sta].next){
127             string curl = cur + f;
128             prt(s, curl);
129         }
130     }
131 }
132
133 void prtWhole(int sta, string cur = ""){
134     cout << "cur " << cur << " sta " << sta << " " << cnt[sta] << endl;
135     for(auto [f,s] : st[sta].next){
136         string curl = cur + f;
137         prtWhole(s, curl);
138     }
139 }

```

```

138     }
139
140 };
141
142 char buf[100010];
143 bool vis[200010];
144
145 ll calc(int sta, suffix_automaton &suf){
146     if(vis[sta]) return 0;
147     vis[sta] = 1;
148     ll res = cnt[sta]*cnt[sta]*(st[sta].len - st[st[sta].link].len);
149     for(auto [f, s] : st[sta].next){
150         res += calc(s, suf);
151     }
152     return res;
153 }
154
155 string read(){
156     scanf("%s", buf);
157     return buf;
158 }
159
160 string s;
161 suffix_automaton suf;
162
163 int main() {
164
165     int t;
166     scanf("%d", &t);
167
168     while(t--){
169         s = read();
170         suf.build_again(s);
171         suf.occ();
172         for(int i = 0; i < s.length()*2+5; i++) vis[i] = 0;
173         ll ans = calc(0, suf);
174         cout << ans << endl;
175     }
176
177     return 0;
178 }
179

```



```

1 // treap implicit
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 using ll = long long;
7 mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
8 typedef struct item * pitem;
9 struct item { // best practice is to do all operations using split and merge operations
10     int prior, value, cnt;
11     bool rev;
12     // int f = 0; variable for range queries, do range queries by 2 splits, then merge
    parts again
13     // don't forget to edit it in upd_cnt function
14     // If you use it as an dynamic array, it's 1-base indexed
15     pitem l, r;
16     item(int val){
17         value = val;
18         prior = rand();
19         l = r = NULL;
20     }
21 };
22
23 int cnt (pitem it) {
24     return it ? it->cnt : 0;
25 }
26
27 void upd_cnt (pitem it) {
28     if (it)
29         it->cnt = cnt(it->l) + cnt(it->r) + 1;
30 }
31
32 void push (pitem it) {
33     if (it && it->rev) { // same principle as segment tree, call it in top of any other
        operation function
34         it->rev = false;
35         swap (it->l, it->r);
36         if (it->l) it->l->rev ^= true;
37         if (it->r) it->r->rev ^= true;
38     }
39 }
40
41 void merge (pitem & t, pitem l, pitem r) {
42     push (l);
43     push (r);
44     if (!l || !r)
45         t = l ? l : r;
46     else if (l->prior > r->prior)
47         merge (l->r, l->r, r), t = l;
48     else
49         merge (r->l, l, r->l), t = r;
50     upd_cnt (t);
51 }
52
53 void split (pitem t, pitem & l, pitem & r, int key, int add = 0) { // first key
    elements in left and remains in right
54     if (!t)
55         return void( l = r = 0 );
56     push (t);
57     int cur_key = add + cnt(t->l);
58     if (key <= cur_key)
59         split (t->l, l, t->l, key, add), r = t;
60     else
61         split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
62     upd_cnt (t);
63 }
64
65 void insert(pitem &t, int pos, int value){ // insert value in index = pos
    pitem t1, t2;

```

```

67     split(t, t1, t2, pos);
68     pitem newItem = new item(value);
69     merge(t1, t1, newItem);
70     merge(t, t1, t2);
71     upd_cnt(t);
72 }
73
74 void erase(pitem &t, int key, int add = 0){ // erase value in index = pos
75     pitem l, r, l1, r1;
76     split(t, l, r, key+1);
77     split(l, l1, r1, key);
78     merge(t, l1, r);
79     delete r1;
80     upd_cnt(t);
81 }
82
83 void reverse (pitem &t, int l, int r) {
84     pitem t1, t2, t3;
85     split (t, t1, t2, l);
86     split (t2, t2, t3, r-l+1);
87     t2->rev ^= true;
88     merge (t, t1, t2);
89     merge (t, t, t3);
90     upd_cnt(t);
91 }
92
93 int elementAt(pitem &t, int key){
94     pitem l, r, l1, r1;
95     split(t, l, r, key+1);
96     split(l, l1, r1, key);
97     int res = r1->value;
98     merge(l, l1, r1);
99     merge(t, l, r);
100     upd_cnt(t);
101     return res;
102 }
103
104 void output (pitem t) {
105     if (!t) return;
106     push (t);
107     output (t->l);
108     printf ("%d ", t->value);
109     output (t->r);
110 }
111
112 int main() {
113
114     pitem root = NULL;
115     int t;
116     scanf("%d", &t);
117     while(t--){
118
119         int x, y, z;
120         cin >> x;
121         if(x == 1){
122             cin >> y >> z;
123             insert(root, y, z);
124         } else if(x == 2) {
125             cin >> y;
126             erase(root, y);
127         } else if(x == 3){
128             cin >> y >> z;
129             reverse(root, y, z);
130         } else if(x == 4){
131             cin >> y;
132             cout << "element " << elementAt(root, y) << endl;
133         } else {
134             output(root);
135             puts("");

```

```
136         }
137
138     }
139
140 }
141
142 /*
143
144 100
145 1 0 7
146 1 0 6
147 1 0 5
148 1 0 4
149 1 0 3
150 1 0 2
151 1 0 1
152
153 */
154
```

```

1 // treap regular
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 using ll = long long;
7 mt19937 rng32(chrono::steady_clock::now().time_since_epoch().count());
8 struct item {
9     int key, prior, cnt = 0;
10     ll val = 0;
11     item *l, *r;
12     item () { }
13     item (int key) : key(key), prior(rng32()), l(NULL), r(NULL), val(0), cnt(1) { }
14 };
15 typedef item* pitem;
16
17 int cnt (pitem t) {
18     return t ? t->cnt : 0;
19 }
20
21 void upd_cnt (pitem t) {
22     if (t)
23         t->cnt = 1 + cnt(t->l) + cnt (t->r);
24 }
25
26 void split (pitem t, int key, pitem & l, pitem & r) {
27     if (!t)
28         l = r = NULL;
29     else if (t->key <= key)
30         split (t->r, key, t->r, r), l = t;
31     else
32         split (t->l, key, l, t->l), r = t;
33     upd_cnt(t);
34 }
35
36 void merge (pitem & t, pitem l, pitem r) {
37     if (!l || !r)
38         t = l ? l : r;
39     else if (l->prior > r->prior)
40         merge (l->r, l->r, r), t = l;
41     else
42         merge (r->l, l, r->l), t = r;
43     upd_cnt(t);
44 }
45
46 void insert (pitem & t, pitem it) {
47     pitem l = NULL, r = NULL;
48     split(t, it->key, l, r);
49     merge(t, l, it);
50     merge(t, it, r);
51 }
52
53 void erase (pitem & t, int key) {
54     pitem l = NULL, r = NULL, cur = NULL;
55     split(t, key, l, r);
56     split(l, key-1, l, cur);
57     merge(t, l, r);
58     delete cur;
59 }
60
61 int kth(pitem t, int k){
62     if(!t) return -1;
63     if(cnt(t->l) >= k) return kth(t->l, k);
64     k -= cnt(t->l);
65     if(k == 1) return t->key;
66     else return kth(t->r, k - 1);
67 }
68
69 int main() {
70
71     int n, q;
72     scanf("%d%d", &n, &q);
73

```

```

74     pitem root = NULL;
75
76     for(int i = 0; i < n; i++){
77         int x;
78         scanf("%d", &x);
79         pitem cur = new item(x);
80         insert(root, cur);
81     }
82
83     for(int i = 0; i < q; i++){
84         int x;
85         scanf("%d", &x);
86         if(x < 0){
87             x = -x;
88             int kth_e = kth(root, x);
89             assert(kth_e + 1);
90             erase(root, kth_e);
91         } else {
92             pitem cur = new item(x);
93             insert(root, cur);
94         }
95     }
96
97     if(cnt(root)) printf("%d", kth(root, 1));
98     else printf("%d", 0);
99
100 }

```