```
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';
             else res += '0' + cur;
23
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);
         y = abs(y);
34
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++){</pre>
             if(onlyX[i] != '0' && onlyY[i] != '0') res = 0;
46
             if(onlyX[i] == '0' && onlyY[i] == '0') res = 0;
47
48
49
50
         if(res) puts("Possible");
51
         else puts("Impossible");
52
53
     }
```

```
// big int
 1
 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)
                  printf ("%09d", a[i]);
34
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) {</pre>
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) {</pre>
131
                   if (i == a.size())
132
                       a.push back (0);
133
                   long long cur = carry + a[i] * 111 * 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;
               return c;
145
146
          }
147
148
          bigInt operator \star= (bigInt b) { // O(n^2)
149
               biaInt c;
150
               c.a.resize((a.size()+b.size()));
1.51
               for (int i=0; i<a.size(); ++i)</pre>
                   for (int j=0, carry=0; j<(int)b.size() || carry; ++j) {</pre>
152
153
                       long long cur = c[i+j] + a[i] * 111 * (j < (int)b.size() ? b[j] : 0) +
                       carry;
154
                       c[i+j] = int (cur % base);
155
                       carry = int (cur / base);
156
                   }
157
               while (c.size() > 1 && c.back() == 0)
158
                   c.pop back();
159
               this->a = c.a;
               return *this;
160
161
          }
162
163
          bigInt operator / (int b){
164
               bigInt c = *this;
               return c /= b;
165
166
          }
167
168
          bigInt operator /= (int b){
169
               int carry = 0;
170
               for (int i=(int)a.size()-1; i>=0; --i) {
171
                   long long cur = a[i] + carry * 111 * base;
                   a[i] = int (cur / b);
172
173
                   carry = int (cur % b);
174
175
               while (a.size() > 1 && a.back() == 0)
176
                   a.pop back();
177
               return *this;
178
          }
179
180
           int operator % (int b) {
181
               bigInt c = *this;
182
               int carry = 0;
183
               for (int i=(int)c.size()-1; i>=0; --i) {
                   long long cur = c[i] + carry * 111 * base;
184
185
                   c[i] = int (cur / b);
186
                   carry = int (cur % b);
187
188
               while (c.size() > 1 && c.back() == 0)
189
                   c.pop back();
190
               return carry;
191
           }
192
193
          bool operator == (const bigInt &b) {
194
               return b.a == a;
195
196
197
          bool operator < (bigInt b) {</pre>
198
               if(a.size() < b.size()) return 1;</pre>
199
               if(a.size() > b.size()) return 0;
200
               for (int i = (int)a.size()-1; i >= 0; i--){
201
                   if(a[i] < b[i]) return 1;</pre>
202
                   if(b[i] < a[i]) return 0;</pre>
203
               }
204
               return 0;
205
          }
```

```
207
          bool operator <= (bigInt b) {</pre>
               bigInt temp = *this;
208
209
               return temp < b || temp == b;</pre>
210
          }
211
212
      };
213
214
      int main() {
215
          string x = "123456789";
216
          bigInt a = x;
bigInt b(x);
217
218
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
     }
```

```
// convex hull trick
 1
 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 a[N], b[N], dp[N];
11
     // if to line are parallel then
12
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
         11 m, c;
         Line(ll m, ll c){
16
17
             this->c = c;
18
             this->m = m;
19
         }
20
21
         11 operator()(11 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
     11 calc(vector<Line> &v, vector<double> &rng, double x) {
32
         int dist = upper bound(rng.begin(), rng.end(), x) - rng.begin() - \frac{1}{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 <=</pre>
             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
     }
```

```
// fft
 1
 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
             { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
12
13
     };
14
15
     void fft(vector<base> &p, bool inv = 0) {
         int n = p.size(), i = 0;
16
17
         for (int j = 1; j < n - 1; ++j) {
18
             for (int k = n \gg 1; k > (i ^= k); k \gg 1);
19
             if(j < i) swap(p[i], p[j]);</pre>
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];
                      p[j + 1] = p[j] - t;
27
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;</pre>
33
     }
34
35
     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
36
         reverse in main when needed
37
         int n = 1;
38
         while (n < a.size() + b.size())</pre>
39
             n <<= 1;
40
         fa.resize(n);
41
         fb.resize(n);
42
43
         fft(fa, false);
         fft(fb, false);
44
45
         for (int i = 0; i < n; i++)
46
             fa[i] = fa[i] * fb[i];
47
         fft(fa, true);
48
49
         vector<int> result(n);
50
         for (int i = 0; i < n; i++)</pre>
51
             result[i] = round(fa[i].a);
52
         return result;
53
     }
```

```
1
     // gauss elemination xor
 2
 3
     struct Basis {
 4
         const static int N = 61; // N = log(MAX A)
 5
 6
         vector<11> 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) \}
              for (int i = N - 1; \sim i; --i) {
16
17
                  if (x & (1LL << i)) {</pre>
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; \sim i; --i) {
31
                  if (x & (1 << i)) {
32
                      x ^= a[i];
33
                  }
34
              }
35
              return x == 0;
36
         }
37
38
         11 getMax() {
39
              11 result = 0;
40
              for (int i = N - 1; \sim 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;
         for (int to : g[v]) {
12
             if (mt[to] == -1 || try_kuhn(mt[to])) {
13
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 1, int r){
28
             return max(l, r);
29
30
31
         void build(int v, int 1, int r, const vector<int> &ve) {
32
             if(l == r) st[v] = ve[l];
33
             else {
34
                  int mid = (1+r)/2;
35
                  build(2*v, 1, 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 = (1+r)/2;
45
             int f = get(2*v, 1, mid, tl, min(tr, mid));
             int s = get(2*v+1, mid+1, r, max(mid+1, tl), tr);
46
47
             return mrg(f, s);
48
         }
49
50
         int query(int 1, int r){
51
             // assert(l <= r);
52
             return get(1, 0, n-1, 1, 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 = (1+r)/2;
59
                  if (idx \leq mid) upd(2*v, 1, 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
      const int N = 2e5+10;
 76
 77
      vector<pair<int, int> > tree[N];
 78
      vector<int> part[N];
 79
      segment tree st[N];
      int head[N], sz[N], depth[N], parent[N], pos[N];//, val[N];
 80
 81
 82
      void init(int v, int pre, int h){
 83
          depth[v] = h;
 84
          sz[v] = 1;
          parent[v] = pre;
 85
 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
117
      void upd edge(int v, int u, int val) { // edit something in edge between vertexes u and v
118
          if(parent[u] != v) swap(v, u);
119
          assert(parent[u] == v);
          // cout << "u " << u << " v " << v << " val " << val << endl;
120
121
          // cout << "head " << head[u] << " pos " << pos[u] << endl;
122
          st[head[u]].upd(pos[u], val);
123
          part[head[u]][pos[u]] = val;
124
      }
125
126
      int query(int v, int u){
127
          int res = 0;
128
129
          while(head[v] != head[u]){
130
               if(depth[head[u]] < depth[head[v]]) swap(u, v);</pre>
131
              assert(pos[head[u]] == 0);
132
              res = max(res, st[head[u]].query(0, pos[u]));
133
               u = parent[head[u]];
134
          }
135
136
          if(depth[u] < depth[v]) swap(u, v);</pre>
137
138
          // 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;
               // scanf("%c", &x);
166
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++) {
                   int x, y, w;
172
                   scanf("%d%d%d", &x, &y, &w);
173
174
                   edges.emplace back(x, y);
                   tree[x].push_back({y, w});
175
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
              you do, for sum operations you pass 0, for max you pass -INF, ans so on
182
              dfs(root, root, root, 0);
183
184
              for(int i = 1; i <= n; i++){</pre>
                   // cout << "i " << i << " and " << head[i] << endl;
185
186
                   if(head[i] == i) st[i].init(part[i]);
187
               }
188
189
              string q = "";
190
              while(true){
191
                   int u, v;
192
                   q = read();
193
                   if(q == "DONE") break;
194
                   scanf("%d%d", &v, &u);
195
                   if(q == "CHANGE") {
196
                       // cout << "hi " << edges[v-1].first << " and " << edges[v-1].second <<
                       endl;
197
                       upd edge(edges[v-1].first, edges[v-1].second, u);
198
                   } else if(q == "QUERY"){
199
                       printf("%d\n", query(u, v));
200
                   }
201
               }
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
     // soruce: https://qithub.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++)</pre>
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])</pre>
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
 77
          vector\langle int \rangle setU(n1+1, 0), setV(n2+1, 0), dist(n1+1);
 78
 79
          int cnt = 0;
 80
 81
          while (bfs(setU, setV, dist, n1)) {
               for (int u=1; u<=n1; u++) {</pre>
 82
 83
                   if (!setU[u] && dfs(u, setU, setV, dist))
 84
                        cnt++;
 85
               }
 86
           }
 87
 88
           cout << "\nMaximum Matching:" << endl;</pre>
 89
           for(int i=1; i <= n1; i++){</pre>
 90
               if(setU[i] == 0) continue;
               cout << "i " << i << " and " << setU[i] << endl;</pre>
 91
 92
           }
 93
 94
          cout << "\nCount: " << cnt << endl;</pre>
 95
      }
 96
 97
     int main()
 98
99
          int n, m; // vertexes, edges
100
          cin >> n >> m;
           for(int i=0; i<m; i++){</pre>
101
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
```

```
// hungarian algorithm
1
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
    implementation by Markus Buehren.
12
     // The original implementation is a few mex-functions for use in MATLAB, found here:
13
    http://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-as
    signment-problem
14
    //
15
     // Both this code and the orignal code are published under the BSD license.
16
    // by Cong Ma, 2016
17
    //
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
        nOfRows, int nOfColumns);
34
        void buildassignmentvector (int *assignment, bool *starMatrix, int nOfRows, int
        nOfColumns);
35
        void computeassignmentcost (int *assignment, double *cost, double *distMatrix, int
        nOfRows);
36
        void step2a(int *assignment, double *distMatrix, bool *starMatrix, bool
        *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
        nOfRows, int nOfColumns, int minDim);
37
        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
38
        *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
        nOfRows, int nOfColumns, int minDim);
39
        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);
40
        void step5(int *assignment, double *distMatrix, bool *starMatrix, bool
        *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
        nOfRows, int nOfColumns, int minDim);
41
    };
42
43
     #endif
44
45
46
47
    HungarianAlgorithm::HungarianAlgorithm(){}
48
    HungarianAlgorithm::~HungarianAlgorithm(){}
49
50
51
    //****************//
52
    // A single function wrapper for solving assignment problem.
53
```

```
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".
          // Here the cost matrix of size MxN is defined as a double precision array of N*M
 64
          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
      void HungarianAlgorithm::assignmentoptimal (int *assignment, double *cost, double
 87
      *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++)</pre>
 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++)</pre>
105
          {
106
              value = distMatrixIn[row];
107
              if (value < 0)</pre>
108
                  cerr << "All matrix elements have to be non-negative." << endl;</pre>
109
              distMatrix[row] = value;
110
          }
111
112
          /* memory allocation */
113
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));
```

```
newStarMatrix = (bool *)calloc(nOfElements, sizeof(bool)); /* used in step4 */
118
119
120
           /* preliminary steps */
121
           if (nOfRows <= nOfColumns)</pre>
122
           {
123
               minDim = nOfRows;
124
125
               for (row = 0; row<nOfRows; row++)</pre>
126
127
                   /* find the smallest element in the row */
128
                   distMatrixTemp = distMatrix + row;
129
                   minValue = *distMatrixTemp;
130
                   distMatrixTemp += nOfRows;
131
                   while (distMatrixTemp < distMatrixEnd)</pre>
132
133
                        value = *distMatrixTemp;
134
                        if (value < minValue)</pre>
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)</pre>
142
143
                        *distMatrixTemp -= minValue;
144
                        distMatrixTemp += nOfRows;
145
                   }
146
               }
147
               /* Steps 1 and 2a */
148
149
               for (row = 0; row<nOfRows; row++)</pre>
150
                   for (col = 0; col<nOfColumns; col++)</pre>
151
                        if (fabs(distMatrix[row + nOfRows*col]) < DBL EPSILON)</pre>
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++)</pre>
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)</pre>
171
172
                        value = *distMatrixTemp++;
173
                        if (value < minValue)</pre>
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)</pre>
180
                        *distMatrixTemp++ -= minValue;
181
               }
182
183
               /* Steps 1 and 2a */
184
               for (col = 0; col<nOfColumns; col++)</pre>
185
                   for (row = 0; row<nOfRows; row++)</pre>
                        if (fabs(distMatrix[row + nOfRows*col]) < DBL EPSILON)</pre>
186
```

```
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++)</pre>
195
                  coveredRows[row] = false;
196
197
          }
198
199
          /* move to step 2b */
          step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
200
          coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
201
202
          /* compute cost and remove invalid assignments */
          computeassignmentcost(assignment, cost, distMatrixIn, nOfRows);
203
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
          for (row = 0; row<nOfRows; row++)</pre>
221
222
              for (col = 0; col<nOfColumns; col++)</pre>
223
                  if (starMatrix[row + nOfRows*col])
224
                  {
      #ifdef ONE INDEXING
225
                      assignment[row] = col + 1; /* MATLAB-Indexing */
226
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++)</pre>
240
241
              col = assignment[row];
242
              if (col >= 0)
243
                  *cost += distMatrix[row + nOfRows*col];
244
          }
245
      }
246
      /************************************
247
      void HungarianAlgorithm::step2a(int *assignment, double *distMatrix, bool *starMatrix,
248
      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++)</pre>
255
              starMatrixTemp = starMatrix + nOfRows*col;
256
257
              columnEnd = starMatrixTemp + nOfRows;
258
              while (starMatrixTemp < columnEnd) {</pre>
259
                  if (*starMatrixTemp++)
260
261
                      coveredColumns[col] = true;
262
                      break:
263
                  }
264
              }
265
          }
266
267
          /* move to step 3 */
          step2b(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
268
          coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
269
270
      271
      void HungarianAlgorithm::step2b(int *assignment, double *distMatrix, bool *starMatrix,
272
      bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
      nOfRows, int nOfColumns, int minDim)
273
274
          int col, nOfCoveredColumns;
275
276
          /* count covered columns */
277
          nOfCoveredColumns = 0;
          for (col = 0; col<nOfColumns; col++)</pre>
278
279
              if (coveredColumns[col])
280
                  nOfCoveredColumns++;
281
282
          if (nOfCoveredColumns == minDim)
283
284
              /* algorithm finished */
285
              buildassignmentvector(assignment, starMatrix, nOfRows, nOfColumns);
286
          1
287
          else
288
289
              /* move to step 3 */
290
              step3(assignment, distMatrix, starMatrix, newStarMatrix, primeMatrix,
              coveredColumns, coveredRows, nOfRows, nOfColumns, minDim);
291
          }
292
293
294
      /**********************
295
      void HungarianAlgorithm::step3(int *assignment, double *distMatrix, bool *starMatrix,
296
      bool *newStarMatrix, bool *primeMatrix, bool *coveredColumns, bool *coveredRows, int
      nOfRows, int nOfColumns, int minDim)
297
      {
298
          bool zerosFound;
299
          int row, col, starCol;
300
301
          zerosFound = true;
302
          while (zerosFound)
303
304
              zerosFound = false;
305
              for (col = 0; col<nOfColumns; col++)</pre>
306
                  if (!coveredColumns[col])
307
                      for (row = 0; row<nOfRows; row++)</pre>
308
                          if ((!coveredRows[row]) && (fabs(distMatrix[row + nOfRows*col]) <</pre>
                          DBL EPSILON))
309
                          {
310
                              /* prime zero */
311
                              primeMatrix[row + nOfRows*col] = true;
```

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

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

}

```
442
443
          vector\langle int \rangle ans(n, -1), ans1 = ans;
444
          HungarianAlgorithm var;
445
          double cost = var.Solve(a, ans);
446
447
          for(int i = 0; i < n; i++){</pre>
448
              ans1[ans[i]] = i;
449
          }
450
          for(int i : ans1) cout << i+1 << " ";</pre>
451
452
453
          return 0 ;
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 1, int r){ // add num to the tree
17
         if(1 == r){
18
             root->sum++;
19
             return;
20
         }
21
         int mid = (1+r)/2;
22
         if(num <= mid) {</pre>
23
             if(root->left == nullptr) root->left = new node();
24
             add(num, root->left, 1, mid);
25
         } else {
26
             if(root->right == nullptr) root->right = new node();
27
             add(num, root->right, mid+1, r);
28
         }
29
         root->sum = 0;
30
31
         if(root->left) root->sum += root->left->sum;
32
         if(root->right) root->sum += root->right->sum;
33
34
     }
35
36
     void del(int num, node* root, int 1, int r){ // delete num-th element (need to check if
     the num-th element is exist)
37
         if(l == r) { // need to check if the}
38
             root->sum--;
39
             return;
40
         }
41
         int mid = (1+r)/2;
42
         if(root->left != nullptr && root->left->sum >= num) {
43
             del(num, root->left, 1, 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
55
     int get(int num, node* root, int 1, int r){ // get k-th element
56
         if(1 == r){
57
             return r;
58
59
         int mid = (1+r)/2;
60
         if(root->left != nullptr && root->left->sum >= num) {
61
             return get(num, root->left, 1, 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 1, int r, int tl, int tr){
```

```
69
 70
          if(1 > r) return 0;
 71
          if(l == tl && r == tr) return root->sum;
 72
          int mid = (1+r)/2;
 73
          int f = 0, s = 0;
 74
          if(root->left != nullptr) f = getSum(root->left, 1, mid, t1, min(mid, tr));
 75
          if(root->right != nullptr) s = getSum(root->right, mid+1, r, max(mid+1, t1), tr);
 76
          return f + s;
 77
 78
      }
 79
 80
      void prtAny(node* root, int 1, int r){
 81
          assert(root != nullptr);
 82
          if(1 == r){
 83
              assert(root->sum > 0);
 84
              printf("%d", 1);
 85
              return;
 86
          }
 87
 88
          int mid = (1+r)/2;
 89
          if(root->left != nullptr && root->left->sum != 0) prtAny(root->left, 1, mid);
 90
          else prtAny(root->right, mid+1, r);
 91
 92
      }
 93
 94
      int main()
 95
 96
 97
          int n, q;
 98
 99
          scanf("%d%d", &n, &q);
100
101
          for (int i = 0; i < n; i++) {
102
              int x;
              scanf("%d", &x);
103
104
              add(x, root, 1, n);
105
106
107
          for (int i = 0; i < q; i++) {
108
              int x;
109
              scanf("%d", &x);
110
              if(x < 0){
111
                   del(-x, root, 1, n);
112
              } else {
113
                   add(x, root, 1, n);
114
              }
115
          }
116
117
          if(root->sum == 0) printf("0");
118
          else {
119
              prtAny(root, 1, n);
120
          }
121
122
          return 0;
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 = (11)1e18;
8
9
    struct Line {
10
      ll m, b;
11
       11 operator()(11 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 1, int r, Line seg, Node* o) {
19
       if(1 + 1 == r) {
20
         if(seg(1) < o->seg(1)) o->seg = seg;
21
         return;
22
       }
23
       int mid = (1 + 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);
         else o->rson = new Node(seg);
28
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 1, int r, int x, Node* o) {
36
       if(l + 1 == r) return o->seg(x);
37
       int mid = (1 + r) >> 1;
38
       if(x < mid && o->lson) return min(o->seg(x), query(1, 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 / q;
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)</pre>
46
             shift_solution(x, y, a, b, sign_b);
47
         if (x > maxx)
48
             return 0;
49
         int 1x1 = 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)</pre>
58
             shift solution(x, y, a, b, -sign a);
59
         if (y > maxy)
60
             return 0;
61
         int 1x2 = 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 (1x2 > rx2)
69
             swap(1x2, 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) {
         int n = s.size();
int i = 0;
4
5
         vector<string> factorization;
6
 7
         while (i < n) {</pre>
8
              int j = i + 1, k = i;
9
              while (j < n && s[k] \le s[j]) {
10
                  if (s[k] < s[j])
11
                       k = i;
12
                  else
13
                       k++;
14
                  j++;
15
              while (i <= k) {</pre>
16
17
                  factorization.push_back(s.substr(i, j - k));
18
                  i += j - k;
19
              }
20
21
         return factorization;
22
     }
```

```
// manacher algorithm
 1
 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();
         s = "$" + s + "^";
10
11
         vector<int> p(n + 2);
12
         int l = 0, r = -1;
13
         for(int i = 1; i <= n; i++) {</pre>
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;
         for(auto c: s) {
27
28
              t += string("#") + c;
29
30
         auto res = manacher odd(t + "#");
31
         return vector<int>(begin(res) + 1, end(res) - 1);
32
     }
33
     char buf[1000010];
34
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++){}
              s1 += s[i];
45
              s1 += "#";
46
47
         }
48
49
         s = s1;
50
         s.pop back();
51
         cout << s << endl;</pre>
52
         int ans = 0; // number of subpalindromes
53
         for(int i = 0; i < palindorm.size(); i += 1){</pre>
54
              cout << palindorm[i] << " ";</pre>
55
              ans += palindorm[i]/2;
56
              // if(s[i] == "#") even palindrome with (s[i]-1)/2 length, centers in s[i] and
              s[i+1]
57
              // else odd palindrome with s[i]/2 length, centers in s[i]
58
59
         cout << endl;</pre>
60
61
         cout << "ans " << ans << endl;</pre>
62
63
         return 0;
64
     }
65
```

```
// max flow dinic
 1
 2
 3
     struct FlowEdge {
         int v, u;
 4
 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);
              edges.emplace back(u, v, 0);
26
             adj[v].push_back(m);
27
28
             adj[u].push back(m + 1);
29
             m += 2;
30
         }
31
32
         bool bfs() {
33
             while (!q.empty()) {
                  int v = q.front();
34
35
                  q.pop();
36
                  for (int id : adj[v]) {
37
                      if (edges[id].cap - edges[id].flow < 1)</pre>
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++) {</pre>
                  int id = adj[v][cid];
54
55
                  int u = edges[id].u;
56
                  if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)</pre>
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
     vector<vector<int>> capacity, flow;
 6
     vector<int> height, excess;
 7
8
9
    void push(int u, int v)
10
         int d = min(excess[u], capacity[u][v] - flow[u][v]);
11
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)</pre>
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]])
                      max height.clear();
34
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++) {</pre>
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
         }
```

```
1
     // min cost max flow dijekstra + johnson's algorithm
 2
 3
     // soruce: https://codeforces.com/blog/entry/95823
 4
 5
 6
 7
     If there are negative weights then we should calcalute potential by bellman-ford
     algorithm for the first time
 8
     * /
 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>> q;
21
         vector<Cost> dist, pot;
22
         vector<Cap> f;
23
         vector<bool> vis;
24
         vector<int> par;
25
         bool n2dijkstra = false;
         mcmf(int n) : n(n), g(n), dist(n), pot(n), f(n), vis(n), par(n) {}
26
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
              q[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++) {</pre>
46
                               if(!vis[i] && dist[i] < d) {</pre>
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]) {</pre>
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]) {</pre>
73
                                   dist[e[i].v] = d2;
74
                                   f[e[i].v] = min(f[x], e[i].cap - e[i].flow);
                                   par[e[i].v] = i;
75
76
                                   Q.push({d2, e[i].v});
77
                               }
78
                          }
79
                      }
80
                  }
81
                  if(!vis[t]) break;
82
                  for(int i = 0; i < n; i++) {</pre>
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
     };
```

```
// mo's algorithm
1
 2
 3
    void remove(idx); // TODO: remove value at idx from data structure
                        // TODO: add value at idx from data structure
     void add(idx);
 4
     int get answer(); // TODO: extract the current answer of the data structure
 5
 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
             return make pair(l / block_size, r) <</pre>
13
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]
         for (Query q : queries) {
27
28
             while (cur_l > q.l) {
29
                 cur 1--;
30
                 add(cur_l);
31
32
             while (cur r < q.r) {</pre>
33
                 cur r++;
34
                 add(cur r);
35
36
             while (cur 1 < q.1) {</pre>
37
                 remove(cur 1);
38
                 cur 1++;
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
```

```
// persistent segment tree
 2
 3
     #include <bits/stdc++.h>
 4
 5
     using namespace std;
 6
     using 11 = long long;
 7
8
    const int MaxNum = 2e5+10;
9
10
    struct Vertex {
11
         Vertex *1, *r;
12
         ll sum;
13
14
         Vertex(ll val) : l(nullptr), r(nullptr), sum(val) {}
15
         Vertex (Vertex *1, Vertex *r) : 1(1), r(r), sum(0) {
16
             if (1) sum += 1->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(OLL + val + v->sum);
31
         int tm = (tl + tr) / 2;
32
         if (pos <= tm)</pre>
33
             return new Vertex(update(v->1, t1, tm, pos, val), v->r);
34
         else
35
             return new Vertex(v->1, update(v->r, tm+1, tr, pos, val));
36
     }
37
38
     11 get sum(Vertex* v, Vertex *left, int tl, int tr, int l, int r) {
39
         if (1 > r)
40
             return 0;
41
         if (1 == tl && tr == r) {
             //cout << "l " << l << " r " << r << " sum " << v->sum << " left " << left->sum
42
             << endl;
43
             return v->sum - left->sum;
44
         1
45
         int tm = (tl + tr) / 2;
         return get sum(v->1, left->1, t1, tm, 1, min(r, tm))
46
47
              + get sum(v->r, left->r, tm+1, tr, max(1, tm+1), r);
48
     }
49
50
    void prt(Vertex *v, int tl, int tr){
51
         if(tl == tr){
52
             if(v->sum) cout << "tl " << tl << " sum " << v->sum << endl;</pre>
53
             return;
54
55
         int mid = (tl + tr)/2;
56
         prt(v->1, t1, mid);
57
         prt(v->r, mid+1, tr);
58
59
60
     int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
61
         if (tl == tr)
62
             return tl;
63
         int tm = (tl + tr) / 2, left count = vr - l - sum - vl - sum;
64
         if (left count >= k)
             return find kth(vl->l, vr->l, tl, tm, k);
65
66
         return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
67
     }
68
```

```
69
      int main()
 70
      {
 71
 72
          int tl = 0, tr = MaxNum + 1;
 73
          vector<Vertex*> roots;
          roots.push back(build(tl, tr));
 74
 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%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(), t1, tr, x1, -a));
 97
              roots21.push back(update(roots21.back(), tl, tr, x2, a));
 98
              roots3.push back(update(roots3.back(), t1, tr, x2+1, y2));
 99
100
          }
101
          //prt(roots3.back(), tl, tr);
102
103
104
          int m;
          scanf("%d", &m);
105
106
          ll last = 0;
107
          11 1, r, x;
108
          const 11 \mod = 1e9;
109
          while (m--) {
110
              scanf("%11d%11d%11d", &1, &r, &x);
111
              x = (x + last) % mod;
112
              if(x <= MaxNum) {</pre>
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);
              } else {
119
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
```

```
// 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++) {</pre>
 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++) {
                 if (i > 0 && 'a' + c != s[i])
23
24
                      aut[i][c] = aut[pi[i-1]][c];
25
                 else
26
                      aut[i][c] = i + ('a' + c == s[i]);
27
             }
28
         }
29
     }
```

```
// 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++){</pre>
              if(prime[i] == 0){
16
17
                  pr.push_back(i);
18
                  for(ll j = 1LL*i*i; j < N; j += i){</pre>
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) {
         long long result = 0;
34
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
         ll result = 1;
46
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
```

```
bool MillerRabin(11 n) // returns true if n is probably prime, else returns false.
 70
 71
      {
 72
          if(n < 2)
 73
              return false;
 74
          int s = 0;
 75
          11 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}) {
              if (n == a)
 81
 82
                   return true;
 83
               if (check compsite(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
      11 f(11 x, 11 c, 11 mod) {
 94
          return (mul(x, x, mod) + c) % mod;
 95
 96
 97
      ll rho(ll n, ll x0 = 2, ll c = 1) {
 98
          11 x = x0;
 99
          11 y = x0;
100
          11 g = 1;
101
          while (q == 1) \{
              x = f(x, c, n);
102
              y = f(y, c, n);
103
              y = f(y, c, n);
104
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<11> res;
          for(int i : pr){
112
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);</pre>
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++;
              else c++;
136
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
              11 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;){</pre>
166
                   int cnt = 0, j = i;
167
                   while(i < n && factors[j] == factors[i]) cnt++, i++;</pre>
168
                   cout << factors[j] << "^" << cnt << " \n"[i == n];</pre>
169
              }
170
171
          }
172
173
          return 0;
174
175
176
```

```
// 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
1
 2
 3
    #include <bits/stdc++.h>
    using namespace std;
 4
 5
 6
    using ll = long long;
 7
8
    struct Tree {
9
        int value;
10
        Tree* 1 = nullptr;
11
         Tree* r = nullptr;
         Tree(int val){
12
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->1, t1->r);
26
         t1->1 = merge(t1->1, t2);
27
         return t1;
28
    }
29
30 void output(pt t){
31
        if(t == nullptr) return;
32
         cout << t->value << " ";
33
         output (t->1);
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->1, 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){
                 cout << "front " << front(root) << endl;</pre>
69
```

```
// sparse table
1
 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++)</pre>
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++)</pre>
                  for (int i = 0; i + (1 << j) <= N; i++)</pre>
26
27
                      st[i][j] = f(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
28
29
         }
30
31
         11 sumQuery(int L, int R){
32
             long long sum = 0;
33
             for (int j = K; j \ge 0; j--) {
                  if ((1 << j) <= R - L + 1) {
34
35
                      sum += st[L][j];
                      L += 1 << j;
36
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 \ll 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 11 = long long;
12
13
     struct suffix array {
14
         string s;
15
         int n;
         vector<int> p, cnt, c;
16
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)</pre>
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++)</pre>
                      cnt[i] += cnt[i-1];
58
59
                  for (int i = n-1; i >= 0; i--)
60
                      p[--cnt[c[pn[i]]]] = pn[i];
61
                  cn[p[0]] = 0;
                  classes = 1;
62
63
                  for (int i = 1; i < n; i++) {</pre>
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]};
                      if (cur != prev)
66
67
                           ++classes;
68
                      cn[p[i]] = classes - 1;
69
                  }
```

```
c.swap(cn);
 71
                   c1.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) {
               int l = 0, r = n-1;
 81
               while(1 <= r){</pre>
 82
 83
                   int mid = (1+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 = \{c1[k][i], c1[k][(i+l-(1 << k))%n]\};
               pair<int, int> b = {c1[k][j], c1[k][(j+l-(1 << k))%n]};
 96
 97
               return a == b ? 0 : a < b ? -1 : 1;
 98
          }
 99
100
          int compare substrings (int 1, int r, int 12, int r2) {
101
               if(r-1 == r2-12){
                   return compare(1, 12, r-1+1, log2(r-1+1));
102
103
               } else if(r-1 < r2-12){</pre>
104
                   r2 = (r2-12+1)-(r-1+1);
105
                   int res = compare(1, 12, r-1+1, log2(r-1+1));
106
                   if (res == 0) return -1;
107
                   return res;
108
               } else {
109
                   r = (r-1+1)-(r2-12+1);
110
                   int res = compare(1, 12, r-1+1, log2(r-1+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;</pre>
          for(int i = 0; i < n; i++){</pre>
148
149
               int x = suf.p[i];
150
               cout << x << " ";
151
           }
152
           cout << endl;</pre>
153
          for(int i = 0; i < n; i++){</pre>
154
155
               int x = suf.p[i];
156
               for(int j = x; j < n; j++) cout << s[j];</pre>
157
               cout << endl;</pre>
158
           }
159
160
          while(true) {
161
               int 1, r, 12, r2;
162
               cin >> 1 >> r >> 12 >> r2;
               cout << suf.compare_substrings(1, r, 12, r2) << endl;</pre>
163
164
165
166
          return 0;
167
     }
168
```

```
// suffix automaton
 1
 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++;
```

```
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
                   replace 'cnt[sta]' with '1';
102
103
               for(auto [f, s] : st[sta].next){
104
                   difSubs(s, sta);
105
                   subs[sta] += subs[s];
106
               }
107
           }
108
109
           int lcs(string t){
110
               int v = 0, l = 0, ans = 0;
111
               for (int i = 0; i < t.length(); i++){}
112
                   while (v != 0 \&\& st[v].next.count(t[i]) == 0){
113
                       v = st[v].link;
114
                       l = st[v].len;
115
116
                   if(st[v].next.count(t[i])) v = st[v].next[t[i]], l++;
117
                   ans = max(ans, 1);
118
               1
119
               return ans;
120
121
122
           void prt(int sta, string cur){
123
               if(st[sta].next.empty()) cout << "cur " << cur << endl;</pre>
124
               else {
125
                   for(auto [f,s] : st[sta].next){
                       string cur1 = cur + f;
126
127
                       prt(s, curl);
128
                   }
129
               }
130
           }
131
132
          void prtWhole(int sta, string cur = ""){
               cout << "cur " << cur << " sta " << sta << " " << cnt[sta] << endl;</pre>
133
134
               for(auto [f,s] : st[sta].next){
135
                   string cur1 = cur + f;
136
                   prtWhole(s, curl);
137
               }
```

```
138
         }
139
140
      };
141
      char buf[100010];
142
143
     bool vis[200010];
144
145
      ll calc(int sta, suffix automaton &suf){
146
          if(vis[sta]) return 0;
147
          vis[sta] = 1;
          11 res = cnt[sta]*cnt[sta]*(st[sta].len - st[st[sta].link].len);
148
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;</pre>
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
         // If you use it as an dynamic array, it's 1-base indexed
14
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\rightarrow cnt = cnt(it\rightarrow l) + cnt(it\rightarrow 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->1, it->r);
36
             if (it->1) it->1->rev ^= true;
37
             if (it->r) it->r->rev ^= true;
38
         }
39
     }
40
41
     void merge (pitem & t, pitem l, pitem r) {
42
         push (1);
43
         push (r);
44
         if (!l || !r)
45
             t = 1 ? 1 : r;
46
         else if (l->prior > r->prior)
47
             merge (1->r, 1->r, r), t = 1;
48
         else
49
             merge (r->1, 1, r->1), t = r;
50
         upd cnt (t);
51
     }
52
53
     void split (pitem t, pitem & 1, 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->1);
58
         if (key <= cur key)</pre>
59
             split (t->1, 1, t->1, 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
66
         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 1, r, 11, r1;
 76
          split(t, l, r, key+1);
 77
          split(1, 11, r1, key);
 78
          merge(t, 11, r);
 79
          delete r1;
 80
          upd cnt(t);
 81
 82
      void reverse (pitem &t, int 1, int r) {
 83
 84
          pitem t1, t2, t3;
 85
          split (t, t1, t2, 1);
 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 1, r, 11, r1;
 95
          split(t, 1, r, key+1);
 96
          split(1, 11, r1, key);
 97
          int res = r1->value;
 98
          merge(1, 11, r1);
 99
          merge(t, 1, 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->1);
108
          printf ("%d ", t->value);
109
          output (t->r);
110
      }
111
112
      int main() {
113
114
          pitem root = NULL;
115
          int t;
          scanf("%d", &t);
116
117
          while(t--){
118
119
               int x, y, z;
               cin >> x;
120
121
               if(x == 1){
122
                   cin >> y >> z;
123
                   insert(root, y, z);
124
               } else if(x == 2) {
125
                   cin >> y;
                   erase(root, y);
126
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;</pre>
133
               } else {
134
                   output(root);
135
                   puts("");
```

```
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
         11 \text{ val} = 0;
         item *1, *r;
11
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->1) + cnt (t->r);
24
     }
25
26
    void split (pitem t, int key, pitem & 1, pitem & r) {
27
         if (!t)
28
             1 = r = NULL;
29
         else if (t->key <= key)</pre>
30
             split (t->r, key, t->r, r),
31
32
             split (t->1, key, 1, t->1),
33
         upd cnt(t);
34
     }
35
36
    void merge (pitem & t, pitem l, pitem r) {
37
         if (!l || !r)
38
             t = 1 ? 1 : r;
39
         else if (l->prior > r->prior)
             merge (1->r, 1->r, r), t = 1;
40
41
         else
             merge (r->1, 1, r->1), t = r;
42
43
         upd cnt(t);
44
     }
45
46
     void insert (pitem & t, pitem it) {
47
         pitem l = NULL, r = NULL;
         split(t, it->key, l, r);
48
         merge(t, l, it);
49
50
         merge(t, it, r);
51
     }
52
53
     void erase (pitem & t, int key) {
54
         pitem l = NULL, r = NULL, cur = NULL;
         split(t, key, l, r);
55
56
         split(1, key-1, 1, cur);
57
         merge(t, 1, r);
58
         delete cur;
59
     }
60
61
     int kth(pitem t, int k){
62
         if(!t) return -1;
63
         if(cnt(t->1) \geq= k) return kth(t->1, k);
64
         k = cnt(t->1);
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;
         scanf("%d%d", &n, &q);
73
```

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