

1 Basic

1.1 Default

```

1 #include <bits/stdc++.h>
2 #define int long long int
3 #define all(x) (x).begin(), (x).end()
4 #define INF 1e15+9
5 #define DBG(x) cout<<(#x " = ") <<x<<endl;
6 #define pb push_back
7 #define fastio ios_base::sync_with_stdio(0);cin.
8     tie(0);
9 using namespace std;
10 const int maxn = 2e5+5;
11 const int MOD = 1e9+7; // 998244353;
12 typedef pair<int,int> P;
13 void solve(){
14     // Do Something...
15 }
16 signed main(){
17     fastio
18     int t=1;
19     cin>>t;
20     while(t--) solve();
21 }

```

2 DataStructure

2.1 01trie

```

1 // for xor
2 const int N = 1e5+5;
3 int tot, trie[41*N][2], n; // need reset tot, trie
4 int find(int x){
5     int p=0, sum=0;
6     for(int i=40; i>=0; i--){
7         int id=(x>>i)&1;
8         if(trie[p][id^1]){ // here, choose id^1
9             sum=sum*2+1;
10            p=trie[p][id^1];
11        }
12        else{
13            sum=sum*2; // here, choose id
14            p=trie[p][id];
15        }
16    }
17    return sum;
18 }
19 // fixed:
20 void insert(int x){
21     int p=0;
22     for(int i=40; i>=0; i--){
23         int id=(x>>i)&1;
24         if(!trie[p][id]) trie[p][id]=++tot;
25         p=trie[p][id];
26     }
27 }

```

2.2 DSU f

```

1 struct DSU {
2     vector<int> f, siz;
3
4     DSU(){}
5     DSU(int n){

```

```

6         init(n);
7     }
8
9     void init(int n){
10         f.resize(n);
11         iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x){
16         while (x != f[x]) {
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y){
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y){
27         x = find(x);
28         y = find(y);
29         if (x == y) {
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36
37     int size(int x){
38         return siz[find(x)];
39     }
40 };
41
42 // USE: DSU dsu(n);

```

2.3 DSU set

```

1 /*
2 This DSU supports the following operations :
3 1 x y, Union the sets containing x and q. If
4   they are already in the same set, ignore.
5 2 x y, Move x to the set containing y. If
6   they are already in the same set, ignore.
7 3 x, Return the number of elements and the
8   sum of elements in the set containing x.
9 */
10 #include <bits/stdc++.h>
11 #define int long long
12 using namespace std;
13 const int maxn = 2e5+5;
14
15 int realIndex[maxn];
16 int p[maxn];
17 int ele[maxn];
18 int sum[maxn];
19 int cnt;
20
21 void init(int n){
22     cnt = n + 1;
23     for(int i=1; i<=n; i++){
24         p[i] = i;
25         ele[i] = 1;
26         sum[i] = i;
27         realIndex[i] = i;
28     }
29 }

```

```

27
28 int find(int x){
29     if(p[x] == x)
30         return x;
31     else
32         return p[x] = find(p[x]);
33 }
34
35 void merge(int x, int y){
36     x = realIndex[x];
37     y = realIndex[y];
38     int px = find(x), py = find(y);
39     if(px == py)
40         return;
41     p[px] = py;
42     sum[py] += sum[px];
43     ele[py] += ele[px];
44 }
45
46 void moveXToY(int x, int y){
47     if(find(realIndex[x]) == find(realIndex[y]))
48         return;
49     sum[find(realIndex[x])] -= x;
50     ele[find(realIndex[x])] -= 1;
51     realIndex[x] = cnt++;
52     sum[realIndex[x]] = x;
53     ele[realIndex[x]] = 1;
54     p[realIndex[x]] = realIndex[x];
55     merge(x, y);
56 }
57
58 void output(int x){
59     x = realIndex[x];
60     x = find(x);
61     cout << ele[x] << ' ' << sum[x] << endl;
62 }
63
64 signed main(){
65     int n, q;
66     while(cin >> n >> q){
67         init(n);
68         while(q--){
69             int type;
70             cin >> type;
71             if(type == 1){
72                 int x, y;
73                 cin >> x >> y;
74                 merge(x, y);
75             }
76             else if(type == 2){
77                 int x, y;
78                 cin >> x >> y;
79                 moveXToY(x, y);
80             }
81             else{
82                 int x;
83                 cin >> x;
84                 output(x);
85             }
86         }
87     }
88 }

```

2.4 Lazy Seg

```

1 // range upd (+k) and query sum
2 #include <bits/stdc++.h>
3 #define int long long int
4 using namespace std;

```

```

5 const int N=1e5+10;
6 struct node{
7     int sum;
8     int l,r;
9     int tag;
10 }tr[N*4];
11 int a[N];
12 inline void pushup(int x){
13     tr[x].sum=tr[2*x].sum+tr[2*x+1].sum; //pushup
14     操作
15 }
16 inline void pushdown(int x){
17     if(tr[x].tag){
18         tr[2*x].tag+=tr[x].tag, tr[2*x+1].tag+=tr[x].tag;
19         tr[2*x].sum+=tr[x].tag*(tr[2*x].r-tr[2*x].l+1);
20         tr[2*x+1].sum+=tr[x].tag*(tr[2*x+1].r-tr[2*x+1].l+1);
21         tr[x].tag=0;
22     }
23 }
24 void build(int x,int l,int r){
25     tr[x].l=l, tr[x].r=r, tr[x].tag=0;
26     if(l==r){
27         tr[x].sum=a[l];
28         return;
29     }
30     int mid=(l+r)/2;
31     build(2*x,l,mid), build(2*x+1,mid+1,r);
32     pushup(x);
33 }
34 int query(int x,int l,int r){
35     if(l<=tr[x].l&&r>=tr[x].r) return tr[x].sum;
36     pushdown(x);
37     int mid=(tr[x].l+tr[x].r)/2, sum=0;
38     if(l<=mid) sum+=query(x*2,l,r);
39     if(r>mid) sum+=query(x*2+1,l,r);
40     return sum;
41 }
42 void update(int now,int l,int r,int k){
43     if(l<=tr[now].l&&r>=tr[now].r){
44         tr[now].sum+=k*(tr[now].r-tr[now].l+1);
45         tr[now].tag+=k; // 先改再標記
46     }
47     else{
48         pushdown(now);
49         int mid=(tr[now].l+tr[now].r)/2;
50         if(l<=mid) update(now*2,l,r,k);
51         if(r>mid) update(now*2+1,l,r,k);
52         pushup(now);
53     }
54 }
55 int n,q;
56 signed main(){
57     cin>>n>>q;
58     for(int i=1;i<=n;i++) cin>>a[i];
59     build(1,1,n);
60     while(q--){
61         int l,r,k,c;
62         cin>>c>>l>>r;
63         if(c==1){
64             cin>>k;
65             update(1,l,r,k);
66         }
67         else cout<<query(1,l,r)<<endl;
68     }
69 }

```

2.5 PBDS

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template<typename T> using rbt = tree<T,
    null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
5 int main(){
6     rbt<int> t; //declare
7 }
8
9 /*
10 不支援重複值 (需要的話可用左推+值來處理)
11 支援set, map之操作
12 find_by_order(k) : 像陣列一樣回傳第 k 個值。(0-
    based, pointer)
13 order_of_key(k) : 回傳 k 是集合裡第幾大。(0-based
    )
14
15 T 資料型別
16 null_type //當作 map 使用的時候要對應什麼資料型
    態。
17 //要當作 set 就用 null_type
18 less<T> // key value 要用什麼方式比較
19 */

```

2.6 Per Seg

```

1 struct Per_seg{
2     int l, r, m;
3     int v = 0;
4     Per_seg *ln = nullptr, *rn = nullptr;
5     Per_seg(int _l, int _r) : l(_l), r(_r), m((_l
        + _r) >> 1) {}
6     void build(){
7         if (l != r - 1)
8         {
9             ln = new Per_seg(l, m);
10            rn = new Per_seg(m, r);
11            ln->build();
12            rn->build();
13        }
14    }
15    void upd(int tar, int value){
16        if (tar == l && tar == r - 1){
17            v = value;
18            return;
19        }
20        else{
21            int m = (l + r) >> 1;
22            if (tar < m){
23                ln = new Per_seg(*ln);
24                ln->upd(tar, value);
25            }
26            else{
27                rn = new Per_seg(*rn);
28                rn->upd(tar, value);
29            }
30            v = ln->v + rn->v;
31        }
32    }
33    int query(int ll, int rr){
34        if (l == ll && r == rr){
35            return v;
36        }
37        else{
38            if (m >= rr){
39                return ln->query(ll, rr);

```

```

40            }
41            else if (m <= ll){
42                return rn->query(ll, rr);
43            }
44            else{
45                return ln->query(ll, m) + rn->
                    query(m, rr);
46            }
47        }
48    }
49 };
50
51 signed main(){
52     int n, q; // n = array size, q = query times
53     cin >> n >> q;
54     vector<Per_seg> *tr;
55     tr.push_back(new Per_seg(0, n));
56     tr[0]->build();
57     for (int i = 0; i < n; i++){
58         int a;
59         cin >> a;
60         tr[0]->upd(i, a); // init ver.0, 0-based!
61     } // build done
62     // Set the value a in array k to x: tr[k]->
        upd(a, x);
63     // Sum of values in range [a,b] in array k:
        tr[k]->query(l, r)
64     // []? []?
65     // Create a copy of array k: tr.push_back(new
        Per_seg(*tr[k]))
66 }

```

2.7 SparseTable

```

1 const int MAXN = 5e5 + 5;
2 const int lgN = 20;
3
4 struct SP{
5     vector<int> Sp[lgN];
6     void build(int n, int *a){
7         for(int i=0; i<n; i++){
8             Sp[0].push_back(a[i]);
9         }
10        for(int h=1; h<lgN; h++){
11            int len = (1 << (h - 1)), i = 0;
12            for(; i + len < n; i++){
13                Sp[h].push_back(max(Sp[h-1][i],
14                    Sp[h-1][i+len]));
15            }
16        }
17        int query(int l, int r){
18            int lg = __lg(r - l + 1);
19            int len = (1 << lg);
20            return max(Sp[lg][l], Sp[lg][r - len +
21                1]);
22        }
23    };

```

2.8 Treap

```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 struct node{ // support range reverse, range sum
    query
6     node *l, *r;

```

```

7   int key, val, sum, pri, size, rev;
8   node(int k, int v) : l(0), r(0), key(k), val(
      v), sum(v), pri(rand()), size(1), rev(0)
      {}
9   void up();
10  void down();
11 };
12
13 node *merge(node *a, node *b){
14     if(!a || !b) return a ? a : b;
15     if(a -> pri < b -> pri){
16         a -> down();
17         a -> r = merge(a -> r, b);
18         a -> up();
19         return a;
20     }
21     else{
22         b -> down();
23         b -> l = merge(a, b -> l);
24         b -> up();
25         return b;
26     }
27 }
28
29 void split(node *o, node *&a, node *&b, int k){
30     // split by key
31     if(!o)
32         a = b = 0;
33     else{
34         o -> down();
35         if(o -> key < k){
36             a = o;
37             split(o -> r, a -> r, b, k);
38         }
39         else{
40             b = o;
41             split(o -> l, a, b -> l, k);
42         }
43         o -> up();
44     }
45 }
46
47 void insert(node *&root, int k, int v){
48     node *a, *b;
49     split(root, a, b, k);
50     root = merge(a, merge(new node(k, v), b));
51 }
52
53 bool erase(node *&o, int k){ // erase T[k]
54     if(!o)
55         return 0;
56     if(o -> key == k){
57         node *t = o;
58         o = merge(o -> l, o -> r);
59         delete t;
60         return 1;
61     }
62     node *&t = k < o -> key ? o -> l : o -> r;
63     if(erase(t, k)) return o -> up(), 1;
64     else return 0;
65 }
66
67 void node :: up(){
68     size = 1;
69     sum = val;
70     if(l) {
71         //L -> down();
72         size += l -> size;
73         sum += l -> sum;

```

```

74     }
75     if(r) {
76         //r -> down();
77         size += r -> size;
78         sum += r -> sum;
79     }
80 }
81
82 void node :: down(){
83     if(rev){
84         swap(l, r);
85         if(l) l -> rev ^= 1;
86         if(r) r -> rev ^= 1;
87         rev = 0;
88     }
89 }
90
91 inline int size(node *o){
92     return o ? o -> size : 0;
93 }
94
95 int Rank(node *& root, int val){// Number of
96     // elements smaller than val.
97     node *a, *b;
98     split(root, a, b, val);
99     int res = size(a);
100    root = merge(a, b);
101    return res;
102 }
103
104 void split2(node *o, node *&a, node *&b, int k){
105     // split by size
106     if(!o)
107         a = b = 0;
108     else{
109         o -> down();
110         if(k >= size(o -> l) + 1){
111             a = o;
112             int nk = k - (size(o -> l) + 1);
113             split2(o -> r, a -> r, b, nk);
114         }
115         else{
116             b = o;
117             split2(o -> l, a, b -> l, k);
118         }
119         o -> up();
120     }
121 }
122
123 node *kth(node *&root, int k){ // find T[k]
124     node *a, *b, *c;
125     split2(root, a, c, k);
126     split2(a, a, b, k - 1);
127     root = merge(a, merge(b, c));
128     return b;
129 }
130
131 void reverse(node *&root, int l, int r){
132     node *a, *b, *c;
133     split2(root, a, b, l - 1);
134     split2(b, b, c, r - l + 1);
135     b -> rev ^= 1;
136     root = merge(a, merge(b, c));
137 }
138
139 int query(node *&root, int l, int r){
140     node *a, *b, *c;
141     split2(root, a, b, l - 1);
142     split2(b, b, c, r - l + 1);
143     b -> down();

```

```

142     int res = b -> sum;
143     root = merge(a, merge(b, c));
144     return res;
145 }
146
147 void update(node *&root, int pos, int x){ // Let
148     T[pos] = x
149     erase(root, pos);
150     insert(root, pos, x);
151 }
152
153 signed main(){
154     node *T(nullptr);
155     int n, q;
156     cin >> n >> q;
157     for(int i=1; i<=n; i++){
158         int tmp;
159         cin >> tmp;
160         insert(T, i, tmp);
161     }
162     while(q--){
163         int op;
164         cin >> op;
165         if(op == 1){ // update
166             int pos, x;
167             cin >> pos >> x;
168             update(T, pos, x);
169         }
170         else{ // query
171             int l, r;
172             cin >> l >> r;
173             cout << query(T, l, r) << '\n';
174         }
175     }

```

3 Geometric

3.1 Closetpair

```

1 typedef pair<ll, ll> pii;
2 #define x first
3 #define y second
4 ll dd(const pii& a, const pii& b) {
5     ll dx = a.x - b.x, dy = a.y - b.y;
6     return dx * dx + dy * dy;
7 }
8 const ll inf = 1e18;
9 //在一點對陣列的[l, r]間找最近點對
10 ll dac(vector<pii>& p, int l, int r) {
11     if (l >= r) return inf;
12     int m = (l + r) / 2;
13     ll d = min(dac(p, l, m), dac(p, m + 1, r));
14     vector<pii> t;
15     for (int i = m; i >= l && p[m].x - p[i].x < d; i--)
16         t.push_back(p[i]);
17     for (int i = m + 1; i <= r && p[i].x - p[m].x < d; i++)
18         t.push_back(p[i]);
19     sort(t.begin(), t.end(),
20         [](pii& a, pii& b) { return a.y < b.y; });
21     int n = t.size();
22     for (int i = 0; i < n - 1; i++)
23         for (int j = i + 1; j < n; j++)
24             // 這裡可以知道是哪兩點是最小點對
25             d = min(d, dd(t[i], t[j]));

```

```

26     return d;
27 }
28 // 給一堆點，求最近點對的距離「的平方」。
29 ll closest_pair(vector<pii>& pp) {
30     sort(pp.begin(), pp.end());
31     return dac(pp, 0, pp.size() - 1);
32 }

```

3.2 Gramh

```

1 //define pdd (double/int)
2 int cross(pdd a, pdd b){
3     return a.first*b.second - a.second*b.first;
4 }
5
6 pdd operator-(pdd a, pdd b){
7     return {a.first - b.first, a.second - b.second};
8 }
9
10 double operator*(pdd a, pdd b){
11     return a.first * b.second - a.second * b.first;
12 }
13
14 // ps是所有的點，要去重!!!
15 vector<pdd> convexHull(vector<pdd>& ps) {
16     sort(all(ps));
17     vector<pdd> hull;
18     if (ps.size() <= 2) {
19         return ps;
20     }
21     for (int i = 0; i < ps.size(); i++) {
22         int s = hull.size();
23         for (pdd p : ps) {
24             while (hull.size() - s >= 2 && cross(
25                 hull.back() - hull[hull.size() - 2],
26                 p - hull[hull.size() - 2]) < 1e-10) {
27                 hull.pop_back();
28             }
29             hull.pb(p); //push_back
30         }
31         hull.pop_back();
32         reverse(all(ps));
33     }
34     return hull;
35 }

```

3.3 Rectangle Union Area

```

1 const int maxn = 1e5 + 10;
2 struct rec{
3     int t, b, l, r;
4     //t頂·b底·l左·r右邊界點
5 } r[maxn];
6 int n, cnt[maxn << 2];
7 long long st[maxn << 2], ans = 0;
8 vector<int> x, y;
9 vector<pair<pair<int, int>, pair<int, int>>> v;
10 void modify(int t, int l, int r, int ql, int qr, int v) {
11     if (ql <= l && r <= qr) cnt[t] += v;
12     else {
13         int m = (l + r) >> 1;
14         if (qr <= m) modify(t << 1, l, m, ql, qr, v);
15         else if (ql <= m) modify(t << 1, m, r, ql, qr, v);
16         else modify(t << 1, l, m, ql, qr, v);
17         modify(t << 1, m, r, ql, qr, v);
18     }
19 }

```

```

15     else if (ql >= m) modify(t << 1 | 1, m, r
16         , ql, qr, v);
17     else modify(t << 1, l, m, ql, m, v),
18         modify(t << 1 | 1, m, r, m, qr, v);
19 }
20 if (cnt[t]) st[t] = y[r] - y[l];
21 else if (r - l == 1) st[t] = 0;
22 else st[t] = st[t << 1] + st[t << 1 | 1];
23 }
24 int main() {
25     cin >> n;
26     for (int i = 0; i < n; i++) {
27         //輸入個個長方形的上下左右界
28         cin >> r[i].l >> r[i].r >> r[i].b >> r[i]
29             .t;
30         if (r[i].l > r[i].r) swap(r[i].l, r[i].r)
31         ;
32         if (r[i].b > r[i].t) swap(r[i].b, r[i].t)
33         ;
34         x.push_back(r[i].l);
35         x.push_back(r[i].r);
36         y.push_back(r[i].b);
37         y.push_back(r[i].t);
38     }
39     sort(x.begin(), x.end());
40     sort(y.begin(), y.end());
41     x.erase(unique(x.begin(), x.end()), x.end());
42     y.erase(unique(y.begin(), y.end()), y.end());
43     for (int i = 0; i < n; i++) {
44         r[i].l = lower_bound(x.begin(), x.end(),
45             r[i].l) - x.begin();
46         r[i].r = lower_bound(x.begin(), x.end(),
47             r[i].r) - x.begin();
48         r[i].b = lower_bound(y.begin(), y.end(),
49             r[i].b) - y.begin();
50         r[i].t = lower_bound(y.begin(), y.end(),
51             r[i].t) - y.begin();
52         v.emplace_back(make_pair(r[i].l, 1),
53             make_pair(r[i].b, r[i].t));
54         v.emplace_back(make_pair(r[i].r, -1),
55             make_pair(r[i].b, r[i].t));
56     }
57     sort(v.begin(), v.end(), [](pair<pair<int,
58         int>, pair<int, int>> a, pair<pair<int,
59         int>, pair<int, int>> b){
60         if (a.first.first != b.first.first)
61             return a.first.first < b.first.first;
62         return a.first.second > b.first.second;
63     });
64     for (int i = 0; i < v.size(); i++) {
65         if (i) ans += (x[v[i].first.first] - x[v[
66             i - 1].first.first]) * st[1];
67         modify(1, 0, y.size(), v[i].second.first,
68             v[i].second.second, v[i].first.second
69             );
70     }
71     cout << ans << '\n';
72     return 0;
73 }

```

```

8 pdd operator+(pdd a, pdd b){
9     return {a.F + b.F, a.S + b.S};
10 }
11 pdd operator-(pdd a, pdd b){
12     return {a.F - b.F, a.S - b.S};
13 }
14 pdd operator*(pdd a, double b){
15     return {a.F * b, a.S * b};
16 }
17 pdd operator/(pdd a, double b){
18     return {a.F / b, a.S / b};
19 }
20 double operator*(pdd a, pdd b){
21     return a.F * b.S - a.S * b.F;
22 }
23 int judge(double a, double b){
24     if (fabs(a-b) < eps) return 0;
25     if (a < b) return -1;
26     return 1;
27 }
28 pdd rotate(pdd a, double b){
29     return {a.F*cos(b)+a.S*sin(b), -a.F*sin(b)+a.
30         S*cos(b)};
31 }
32 double lens(pdd a, pdd b){
33     double dx = b.F - a.F, dy = b.S - a.S;
34     return sqrt(dx*dx + dy*dy);
35 }
36 pdd intersection(pdd p, pdd v, pdd q, pdd w){//求
37     交點
38     pdd u = p - q;
39     double t = w*u/(v*w);
40     return p + v * t;
41 }
42 pair<pdd, pdd> bisector(pdd a, pdd b){//求中垂線
43     pdd p = (a + b) / 2.0;
44     pdd v = rotate(b - a, pi / 2.0);
45     return {p, v};
46 }
47 Circle circle(pdd a, pdd b, pdd c){ //三點求圓
48     auto n = bisector(a, b), m = bisector(a, c);
49     pdd o = intersection(n.F, n.S, m.F, m.S);
50     double r = lens(o, a);
51     return {o, r};
52 }
53 void solve(){
54     p.clear();
55     cin >> n;
56     p.resize(n);
57     //輸入所有的點
58     for (int i = 1; i <= n; i++){
59         cin >> p[i].F >> p[i].S;
60     }
61     random_shuffle(all(p));
62     c = {p[0], 0};
63     for(int i = 1; i <= n; i++){
64         if (judge(c.r, lens(c.o, p[i])) == -1){
65             c = {p[i], 0};
66             for (int j = 0; j < i; j++){
67                 if (judge(c.r, lens(c.o, p[j]))
68                     == -1){
69                     c = {(p[i] + p[j]) / 2.0,
70                         lens(p[i], p[j]) / 2.0};
71                     for (int k = 0; k < j; k++){
72                         if (judge(c.r, lens(c.o,
73                             p[k])) == -1){
74                             c = circle(p[i], p[j]
75                                 , p[k]);
76                         }
77                     }
78                 }
79             }
80         }
81     }

```

3.4 TheLeastCoverCircle

```

1 const double eps = 1e-10, pi = acos(-1);
2 struct Circle{
3     pdd o;
4     double r;
5 }c;
6 vector<pdd> p;
7 int R, n, r;

```

```

67         for (int k = 0; k < j; k++){
68             if (judge(c.r, lens(c.o,
69                 p[k])) == -1){
70                 c = circle(p[i], p[j]
71                     , p[k]);
72             }
73         }
74     }

```

```

72         }
73     }
74 }
75 //c: {圓心, 半徑}
76 if (n == 1) c = {(p[0] + p[1]) / 2.0, 0};
77 cout << setprecision(9) << fixed;
78 cout << -c.o.F << " " << -c.o.S << endl;
79 }
80

```

4 Graph

4.1 2e cc

```

1 // i.e. bridge tree
2 // Remember to reset vis[]
3 map<int,int> compId;
4 vector<int> g2[N];
5 void dfs(int now,int p,int iid){
6     vis[now]=1;
7     compId[now]=iid;
8     for(auto [nxt,id]:g[now]){
9         if(bridge[id]) continue;
10        if(nxt==p) continue;
11        if(!vis[nxt]) dfs(nxt,now,iid);
12    }
13 }
14 //Then, in main()
15 int iid=0;
16 for(int i=0;i<n;i++) vis[i]=0;
17 for(int i=0;i<n;i++){
18     if(!vis[i]){
19         dfs(i,-1,iid);
20         iid+=1;
21     }
22 }
23 for(int i=0;i<m;i++){
24     if(bridge[i]){
25         auto [u,v] = edge[i];
26         g2[compId[u]].pb(compId[v]);
27         g2[compId[v]].pb(compId[u]);
28     }
29 }

```

4.2 2sat tarjan

```

1 const int N = 2005;
2 int low[N],dfn[N],color[N],ins[N]; //要開兩倍大
3 // color[x] 是 x 所在的 scc 的topo逆序。
4 vector<int> g[N];
5 int dfsClock,sccCnt;
6 stack<int> stk;
7 void tarjan(int u) {
8     low[u] = dfn[u] = ++dfsClock;
9     stk.push(u); ins[u] = true;
10    for (const auto &v : g[u]) {
11        if (!dfn[v]) tarjan(v), low[u] = std::min(
12            low[u], low[v]);
13        else if (ins[v]) low[u] = std::min(low[u],
14            dfn[v]);
15    }
16    if (low[u] == dfn[u]) {
17        ++sccCnt;
18        do {
19            color[u] = sccCnt;
20            u = stk.top(); stk.pop(); ins[u] =
21                false;
22        } while (u != u);
23    }
24 }

```

```

19         } while (low[u] != dfn[u]);
20     }
21 }
22 signed main(){
23     g[i].pb(j); // i->j
24
25     for (int i = 1; i <= (n << 1); ++i) if (!dfn[
26         i]) tarjan(i); // run tarjan, 注意0~2n-1
27         or 1~2n
28
29     for(int i=1;i<=n;i++){
30         if(color[i] == color[i+n]){
31             cout<<"NO"<<endl;
32             return;
33         }
34     }
35     cout<<"YES"<<endl;
36     // 找環 注意建邊方法是(i,i+1) or (i,i+n)
37
38     for(int i=1;i<=n;i++){
39         if(color[i] < color[i+n]){
40             cout<<1<<" ";
41         }
42         else cout<<0<<" ";
43     }
44     // 構造解 (注意是0~n-1還是1~n)

```

4.3 Bridge

```

1 //for undirected graph, find bridge
2 const int N = 1e6+5;
3
4 vector<pair<int,int> > edge(N); //{u,v} ->
5     remember to input
6 vector<pair<int,int> > g[N]; //{nxt,edge_id}
7 vector<int> bridge(N);
8 int dfn[N],vis[N],low[N],id;
9 void tarjan(int now,int p){
10     dfn[now]=id++;
11     vis[now]=1;
12     low[now]=dfn[now];
13     for(auto [nxt,id]:g[now]){
14         if(nxt==p) continue;
15         if(vis[nxt]){
16             low[now]=min(low[now],dfn[nxt]); //
17                 back edge!
18         }
19         else{
20             tarjan(nxt,now);
21             low[now]=min(low[now],low[nxt]);
22             if(low[nxt]>dfn[now]){
23                 bridge[id]=1;
24             }
25         }
26     }
27 }
28 signed main(){
29     // construct
30     for(int i=0;i<n;i++){
31         dfn[i]=1e9; //reset
32         low[i]=1e9;
33         vis[i]=0;
34     }
35     id=0;
36     for(int i=0;i<n;i++){
37         if(!vis[i]){
38             tarjan(i,-1);
39         }
40     }

```



```

37     }
38 }
39 // use
40 }

```

4.4 Dinic

```

1 (a) Bounded Maxflow Construction:
2 1. add two node ss, tt
3 2. add_edge(ss, tt, INF)
4 3. for each edge u -> v with capacity [l, r]:
5     add_edge(u, tt, l)
6     add_edge(ss, v, l)
7     add_edge(u, v, r-l)
8 4. see (b), check if it is possible.
9 5. answer is maxflow(ss, tt) + maxflow(s, t)
10 -----
11 (b) Bounded Possible Flow:
12 1. same construction method as (a)
13 2. run maxflow(ss, tt)
14 3. for every edge connected with ss or tt:
15     rule: check if their rest flow is exactly
16         0
17 4. answer is possible if every edge do satisfy
18     the rule;
19 5. otherwise, it is NOT possible.
20 -----
21 (c) Bounded Minimum Flow:
22 1. same construction method as (a)
23 2. answer is maxflow(ss, tt)
24 -----
25 (d) Bounded Minimum Cost Flow:
26 * the concept is somewhat like bounded possible
27   flow.
28 1. same construction method as (a)
29 2. answer is maxflow(ss, tt) + ( $\sum l * cost$  for
30   every edge)
31 -----
32 (e) Minimum Cut:
33 1. run maxflow(s, t)
34 2. run cut(s)
35 3. ss[i] = 1: node i is at the same side with s.
36 -----
37 const long long INF = 1LL<<60;
38 struct Dinic { //O(VVE), with minimum cut
39     static const int MAXN = 5003;
40     struct Edge{
41         int u, v;
42         long long cap, rest;
43     };
44     int n, m, s, t, d[MAXN], cur[MAXN];
45     vector<Edge> edges;
46     vector<int> G[MAXN];
47     void init(){
48         edges.clear();
49         for ( int i = 0 ; i < n ; i++ ) G[i].
50             clear();
51         n = 0;
52     }
53     // min cut start

```

```

53 bool side[MAXN];
54 void cut(int u) {
55     side[u] = 1;
56     for ( int i : G[u] ) {
57         if ( !side[ edges[i].v ] && edges[i].
58             rest ) cut(edges[i].v);
59     }
60 }
61 // min cut end
62 int add_node(){
63     return n++;
64 }
65 void add_edge(int u, int v, long long cap){
66     edges.push_back( {u, v, cap, cap} );
67     edges.push_back( {v, u, 0, 0LL} );
68     m = edges.size();
69     G[u].push_back(m-2);
70     G[v].push_back(m-1);
71 }
72 bool bfs(){
73     fill(d,d+n,-1);
74     queue<int> que;
75     que.push(s); d[s]=0;
76     while (!que.empty()){
77         int u = que.front(); que.pop();
78         for (int ei : G[u]){
79             Edge &e = edges[ei];
80             if (d[e.v] < 0 && e.rest > 0){
81                 d[e.v] = d[u] + 1;
82                 que.push(e.v);
83             }
84         }
85     }
86     return d[t] >= 0;
87 }
88 long long dfs(int u, long long a){
89     if ( u == t || a == 0 ) return a;
90     long long flow = 0, f;
91     for ( int &i=cur[u]; i < (int)G[u].size()
92         ; i++ ) {
93         Edge &e = edges[ G[u][i] ];
94         if ( d[u] + 1 != d[e.v] ) continue;
95         f = dfs(e.v, min(a, e.rest) );
96         if ( f > 0 ) {
97             e.rest -= f;
98             edges[ G[u][i]^1 ].rest += f;
99             flow += f;
100             a -= f;
101             if ( a == 0 ) break;
102         }
103     }
104     return flow;
105 }
106 long long maxflow(int _s, int _t){
107     s = _s, t = _t;
108     long long flow = 0, mf;
109     while ( bfs() ){
110         fill(cur,cur+n,0);
111         while ( (mf = dfs(s, INF)) ) flow +=
112             mf;
113     }
114     return flow;
115 }
116 }
117 } dinic;

```


4.5 Hungarian

```

1 // Maximum Cardinality Bipartite Matching
2 // Worst case O(nm)
3
4 struct Graph{
5     static const int MAXN = 5003;
6     vector<int> G[MAXN];
7     int n, match[MAXN], vis[MAXN];
8
9     void init(int _n){
10         n = _n;
11         for (int i=0; i<n; i++) G[i].clear();
12     }
13
14     bool dfs(int u){
15         for (int v:G[u]){
16             if (vis[v]) continue;
17             vis[v]=true;
18             if (match[v]==-1 || dfs(match[v])){
19                 match[v] = u;
20                 match[u] = v;
21                 return true;
22             }
23         }
24         return false;
25     }
26
27     int solve(){
28         int res = 0;
29         memset(match, -1, sizeof(match));
30         for (int i=0; i<n; i++){
31             if (match[i]==-1){
32                 memset(vis, 0, sizeof(vis));
33                 if (dfs(i)) res++;
34             }
35         }
36         return res;
37     }
38 } graph;

```

4.6 LCA fd

```

1 // online O(nlogn + mlogn)
2 const int N = 300005;
3 int d[N], f[N][20]; // f[i][j] = i's 2^j father, d[
4 // i] = depth of i
5 vector<int> g[N]; // graph
6
7 void dfs(int now, int p, int dep){
8     d[now] = dep+1;
9     for(int nxt:g[now]){
10         if(nxt==p) continue;
11         f[nxt][0] = now;
12         dfs(nxt, now, dep+1);
13     }
14     return;
15 }
16
17 int lca(int x, int y){
18     if(d[x]<d[y]) swap(x,y);
19     int k = d[x]-d[y];
20     for(int i=0; i<20; i++){
21         if(k&1) x = f[x][i];
22         k>>=1;
23     } // jump to the same depth/height
24     if(x==y) return x;
25     for(int i=19; i>=0; i--){
26         if(f[x][i]!=f[y][i]){

```

```

26         x = f[x][i];
27         y = f[y][i];
28     }
29     } // find the first different -> higher is LCA
30     return f[x][0];
31 }
32 void sol(){
33     int n,m;
34     cin>>n>>m;
35     for(int i=1; i<n; i++){
36         int u,v;
37         cin>>u>>v;
38         g[u].pb(v);
39         g[v].pb(u);
40     }
41     dfs(1,0,0); // arbitrarily choose a root,
42     // here choose 1 as root
43     for(int j=1; j<20; j++){
44         for(int i=1; i<=n; i++){
45             f[i][j] = f[f[i][j-1]][j-1];
46         } // get all f
47     }
48     // --- use lca(u,v) to get ---

```

4.7 LCA tarjan

```

1 // Tarjan (offline, O(n + m))
2 void dfs(int now, int p, int dep){
3     d[now] = dep; // d[i] = depth of i, be
4     // careful about "root should set to 0/1"
5     for(int nxt:g[now]){
6         if(nxt==p) continue;
7         dfs(nxt, now, dep+1);
8         connect(now, nxt); // connect son "to" its
9         // parent
10         vis[nxt] = 1;
11     }
12     // Deal with query
13     for(auto i:q[now]){
14         int nxt = i.first; // query has {now, nxt}
15         int id = i.second; // query_id
16         if(vis[nxt]){
17             qans[id] = find_root(nxt);
18         }
19     }
20 }

```

5 Math

5.1 FFT

```

1 const int N = 1e7+10;
2 const double Pi = acos(-1.0);
3 struct Complex{
4     double x,y;
5     Complex ( double xx=0, double yy=0){
6         x=xx;
7         y=yy;
8     }
9 };
10 Complex a[N], b[N];
11 Complex operator + (Complex a, Complex b) {
12     return Complex(a.x + b.x , a.y + b.y);}
13 Complex operator - (Complex a, Complex b) {
14     return Complex(a.x - b.x , a.y - b.y);}

```

```

13 Complex operator * (Complex a, Complex b) {
14     return Complex(a.x * b.x - a.y * b.y , a.x * b
15         .y + a.y * b.x);}
16
17 int limit=1, h=0, rev[N];
18
19 void fft(Complex *A, int flag){
20     for(int i=0; i<limit; i++){
21         if(i<rev[i]){
22             swap(A[i], A[rev[i]]);
23         }
24     }
25     for(int len=1; len<limit; len<=1){
26         // Len = 待合併區間的一半
27         Complex Wn(cos(Pi/len), flag*sin(Pi/len))
28         ;
29         for(int R=len<<1, j=0; j<limit; j+=R){
30             Complex w(1, 0);
31             for (int k = 0; k < len; k++, w = w *
32                 Wn) {
33                 Complex x = A[j + k], y = w * A[j
34                     + len + k];
35                 A[j + k] = x + y;
36                 A[j + len + k] = x - y;
37             }
38         }
39     }
40 }
41
42 int main(){
43     int n,m; // n,m次方
44     cin>>n>>m;
45     for(int i=0; i<=n; i++) cin>>a[i].x;
46     for(int i=0; i<=m; i++) cin>>b[i].x;
47     while(limit<=n+m){
48         limit=limit<<1;
49         h++;
50     }
51     for(int i=0; i<limit; i++){
52         rev[i] = (rev[i>>1]>>1) | ((i&1)<<(h-1));
53         // bit reverse
54     }
55     fft(a, 1);
56     fft(b, 1);//FFT
57     for(int i=0; i<=limit; i++) a[i]=a[i]*b[i];
58     fft(a, -1);//IFFT
59     for(int i=0; i<=n+m; i++) cout<<(int)(a[i].x
60         / limit + 0.5)<<" ";
61 }

```

5.2 LinearSieve

```

1 int LeastPrimeDivisor[maxn];
2 vector<int> pr;
3
4 void LinearSieve(){
5     for(int i = 2; i < maxn; i++){
6         if(!LeastPrimeDivisor[i]) pr.push_back(i),
7             LeastPrimeDivisor[i] = i;
8         for(int p : pr){
9             if(i * p >= maxn) break;
10            LeastPrimeDivisor[i * p] = p;
11            if(i % p == 0) break;
12        }
13    }
14 }

```

5.3 NTT

```

1 const int N = 1e7+10;
2 const int P = 998244353, G = 3, Gi = 332748118; //
3     primitive root = 3, Gi = mod inverse of 3
4
5 int fastpow(int x,int p){
6     int sum = 1;
7     while(p){
8         if(p&1) sum = sum*x%P;
9         x = x*x%P;
10        p = p>>1;
11    }
12    return sum;
13 }
14
15 int a[N], b[N], limit=1, h=0, rev[N];
16 inline void NTT(int *A, int flag) {
17     for(int i = 0; i < limit; i++)
18         if(i < rev[i]) swap(A[i], A[rev[i]]);
19     for(int len = 1; len < limit; len <= 1){
20         int Wn = fastpow( flag == 1 ? G : Gi , (P -
21             1) / (len << 1));
22         for(int j = 0; j < limit; j += (len << 1)){
23             int w = 1;
24             for(int k = 0; k < len; k++) {
25                 int x = A[j + k], y = w * A[j + k
26                     + len] % P;
27                 A[j + k] = (x + y) % P,
28                 A[j + k + len] = (x - y + P) % P;
29                 w = (w * Wn) % P;
30             }
31         }
32     }
33 }
34
35 signed main(){
36     int n,m;
37     cin>>n>>m; // n,m次方
38     for(int i=0; i<=n; i++){
39         cin>>a[i];
40         a[i]=(a[i] + P) % P;
41     }
42     for(int i=0; i<=m; i++){
43         cin>>b[i];
44         b[i]=(b[i] + P) % P;
45     }
46     while(limit<=n+m){
47         limit=limit<<1;
48         h++;
49     }
50     for(int i=0; i<limit; i++){
51         rev[i] = (rev[i>>1]>>1) | ((i&1)<<(h-1));
52     }
53     NTT(a, 1);
54     NTT(b, 1);
55     for(int i=0; i<=limit; i++) a[i]=a[i]*b[i]%P;
56     NTT(a, -1);
57
58     int inv = fastpow(limit, P - 2);
59     for(int i=0; i<=n+m; i++){
60         cout<<(a[i]*inv)%P << " ";
61     }
62 }

```

5.4 RabinMiller

```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4

```

```

5 int QuickPow(int base, int exponent, int mod){
6     if(exponent == 0)
7         return 1;
8     if(exponent == 1)
9         return base;
10    if(exponent % 2)
11        return QuickPow(base, exponent - 1, mod)
12        * base % mod;
13    int tmp = QuickPow(base, exponent / 2, mod);
14    return tmp * tmp % mod;
15 }
16 bool RabinMiller(int d, int n){
17     int a = 2 + rand() % (n - 2);
18     if(QuickPow(a, n - 1, n) != 1)
19         return false;
20     int cur = QuickPow(a, d, n);
21     int nx;
22     while(d != n - 1){
23         nx = (cur * cur) % n;
24         d *= 2;
25         if(cur != 1 && cur != n - 1){
26             if(nx == 1)
27                 return false;
28         }
29         cur = nx;
30     }
31     return true;
32 }
33
34 bool isPrime(int n, int k){
35     if(n <= 1)
36         return false;
37     if(n <= 3)
38         return true;
39     if(n == 4)
40         return false;
41     if((n - 1) % 6 != 0 && (n + 1) % 6 != 0)
42         return false;
43     int d = n - 1;
44     int r = 0;
45     while(d % 2 == 0){
46         d /= 2;
47         r ++;
48     }
49     for(int i=0; i<k; i++){
50         if(!RabinMiller(d, n))
51             return false;
52     }
53     return true;
54 }
55
56 signed main(){
57     int n;
58     while(cin >> n){
59         if(isPrime(n, 5)) // 預設 k = 5
60             cout << "質數" << endl;
61         else
62             cout << "非質數" << endl;
63     }
64 }

```

6 Misc

6.1 Rand

```

1 mt19937_64 rnd(random_device{}());
2 uniform_int_distribution<int> dist(i, j);

```

```

3 /*
4 dist(rnd) -> 取 i~j範圍內的整數
5 */

```

6.2 SG

```

1 int f[100]; // state, reset to -1
2
3 int sg(int x) {
4     if (f[x] != -1) return f[x];
5     unordered_set<int> S;
6     if (x >= 1) S.insert(sg(x - 1)); // all sub-
7     states (suppose x can be x-1, x-2, 0)
8     if (x >= 2) S.insert(sg(x - 2));
9     if (x >= 3) S.insert(sg(0));
10    for (int i = 0; i++) {
11        if (!S.count(i)) return f[x] = i;
12    }
13 }

```

7 Strings

7.1 Rolling hash

```

1 const int N = 2000; // string size
2 int X = 1000000007, P = 4000000007;
3 // other primes : 8298176713 5119240589
4 // 3735751997 3218996237...
5 int s[N], p[N]; // if hash兩次 -> 開兩個s,p
6
7 void Hash(string& str){
8     s[0] = str[0];
9     p[0] = 1;
10    for(int i=1; i<str.size(); i++){
11        s[i] = (s[i-1]*X + str[i])%P;
12        p[i] = (p[i-1]*X) % P;
13    }
14    return;
15 }
16
17 int hash_i(int a, int b){
18     if(!a) return s[b];
19     int tmp = s[b] - s[a-1]*p[b-a+1]; //
20     calculate H[a:b]
21     tmp=tmp%P;
22     if(tmp<0) return tmp+P;
23     return tmp;
24 }
25
26 signed main(){
27     // Hash to generate s and p
28     // hash_i to calculate s[a:b]
29     string s;
30     cin>>s;
31     Hash(s);
32     cout<<hash_i(0,2)<<" "<<hash_i(3,5);
33 }

```

ACM ICPC Team - NYCU OverFlowers

Contents

1 Basic	1		
1.1 Default	1		
2 DataStructure	1		
2.1 01trie	1		
2.2 DSU f	1		
2.3 DSU set	1		
2.4 Lazy Seg	2		
2.5 PBDS	3		
2.6 Per Seg	3		
2.7 SparseTable	3		
2.8 Treap	3		
3 Geometric	5		
3.1 Closetpair	5		
		3.2 Gramh	5
		3.3 Rectangle Union Area	5
		3.4 TheLeastCoverCircle	6
		4 Graph	7
		4.1 2e cc	7
		4.2 2sat tarjan	7
		4.3 Bridge	7
		4.4 Dinic	8
		4.5 Hungarian	9
		4.6 LCA fd	9
		4.7 LCA tarjan	9
		5 Math	9
		5.1 FFT	9
		5.2 LinearSieve	10
		5.3 NTT	10
		5.4 RabinMiller	10
		6 Misc	11
		6.1 Rand	11
		6.2 SG	11
		7 Strings	11
		7.1 Rolling hash	11