1 Basic

1.1 Default

```cpp
#include<bits/stdc++.h>
#define int long long int
#define all(x) (x).begin(), (x).end()
#define INF 1e15+9
#define DBG(x) cout<<(#x " = ") <<x<<endl;
#define pb push_back
#define fastio ios_base::sync_with_stdio(0);cin.
    tie(0);
using namespace std;
const int maxn = 2e5+5;
const int MOD = 1e9+7; // 998244353;
typedef pair<int,int> P;

void solve(){
    // Do Something...
}
signed main(){
    fastio
    int t=1;
    cin>>t;
    while(t--) solve();
}
```

2 DataStructure

2.1 01trie

```cpp
// for xor
const int N = 1e5+5;
int tot,trie[41*N][2],n; // need reset tot, trie
int find(int x){
    int p=0,sum=0;
    for(int i=40;i>=0;i--){
        int id=(x>>i)&1;
        if(trie[p][id^1]){ // here, choose id^1
            sum=sum*2+1;
            p=trie[p][id^1];
        }
        else{
            sum=sum*2; // here, choose id
            p=trie[p][id];
        }
    }
    return sum;
}
// fixed:
void insert(int x){
    int p=0;
    for(int i=40;i>=0;i--){
        int id=(x>>i)&1;
        if(!trie[p][id]) trie[p][id]=++tot;
        p=trie[p][id];
    }
}
```

2.2 DSU f

```cpp
struct DSU {
    vector<int> f, siz;

    DSU(){}
    DSU(int n){
        init(n);
    }

    void init(int n){
        f.resize(n);
        iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x){
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y){
        return find(x) == find(y);
    }

    bool merge(int x, int y){
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x){
        return siz[find(x)];
    }
};

// USE: DSU dsu(n);
```

2.3 DSU set

```cpp
/*
This DSU supports the following operations :
    1 x y, Union the sets containing x and q. If
       they are already in the same set, ignore.
    2 x y, Move x to the set containing y. If
       they are already in the same set, ignore.
    3 x, Return the number of elements and the
       sum of elements in the set containing x.
*/
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int maxn = 2e5+5;

int realIndex[maxn];
int p[maxn];
int ele[maxn];
int sum[maxn];
int cnt;

void init(int n){
    cnt = n + 1;
    for(int i=1;i<=n;i++){
        p[i] = i;
        ele[i] = 1;
        sum[i] = i;
        realIndex[i] = i;
    }
}
```

```cpp

int find(int x){
    if(p[x] == x)
        return x;
    else
        return p[x] = find(p[x]);
}

void merge(int x, int y){
    x = realIndex[x];
    y = realIndex[y];
    int px = find(x), py = find(y);
    if(px == py)
        return;
    p[px] = py;
    sum[py] += sum[px];
    ele[py] += ele[px];
}

void moveXToY(int x, int y){
    if(find(realIndex[x]) == find(realIndex[y]))
        return;
    sum[find(realIndex[x])] -= x;
    ele[find(realIndex[x])] -= 1;
    realIndex[x] = cnt++;
    sum[realIndex[x]] = x;
    ele[realIndex[x]] = 1;
    p[realIndex[x]] = realIndex[x];
    merge(x, y);
}

void output(int x){
    x = realIndex[x];
    x = find(x);
    cout << ele[x] << ' ' << sum[x] << endl;
}

signed main(){
    int n, q;
    while(cin >> n >> q){
        init(n);
        while(q--){
            int type;
            cin >> type;
            if(type == 1){
                int x, y;
                cin >> x >> y;
                merge(x, y);
            }
            else if(type == 2){
                int x, y;
                cin >> x >> y;
                moveXToY(x, y);
            }
            else{
                int x;
                cin >> x;
                output(x);
            }
        }
    }
}
```

2.4 Lazy Seg

```cpp
// range upd (+k) and query sum
#include <bits/stdc++.h>
#define int long long int
using namespace std;
const int N=1e5+10;
struct node{
    int sum;
    int l,r;
    int tag;
}tr[N*4];
int a[N];
inline void pushup(int x){
    tr[x].sum=tr[2*x].sum+tr[2*x+1].sum;//pushup 操作
}
inline void pushudown(int x){
    if(tr[x].tag){
        tr[2*x].tag+=tr[x].tag,tr[2*x+1].tag+=tr[x].tag;
        tr[2*x].sum+=tr[x].tag*(tr[2*x].r-tr[2*x].l+1);
        tr[2*x+1].sum+=tr[x].tag*(tr[2*x+1].r-tr[2*x+1].l+1);
        tr[x].tag=0;
    }
}
void build(int x,int l,int r){
    tr[x].l=l,tr[x].r=r,tr[x].tag=0;
    if(l==r){
        tr[x].sum=a[l];
        return;
    }
    int mid=(l+r)/2;
    build(2*x,l,mid),build(2*x+1,mid+1,r);
    pushup(x);
}
int query(int x,int l,int r){
    if(l<=tr[x].l&&r>=tr[x].r) return tr[x].sum;
    pushudown(x);
    int mid=(tr[x].l+tr[x].r)/2,sum=0;
    if(l<=mid) sum+=query(x*2,l,r);
    if(r>mid) sum+=query(x*2+1,l,r);
    return sum;
}
void update(int now,int l,int r,int k){
    if(l<=tr[now].l&&r>=tr[now].r){
        tr[now].sum+=k*(tr[now].r-tr[now].l+1);
        tr[now].tag+=k; // 先改再標記
    }
    else{
        pushudown(now);
        int mid=(tr[now].l+tr[now].r)/2;
        if(l<=mid) update(now*2,l,r,k);
        if(r>mid) update(now*2+1,l,r,k);
        pushup(now);
    }
}
int n,q;
signed main(){
    cin>>n>>q;
    for(int i=1;i<=n;i++) cin>>a[i];
    build(1,1,n);
    while(q--){
        int l,r,k,c;
        cin>>c>>l>>r;
        if(c==1){
            cin>>k;
            update(1,l,r,k);
        }
        else cout<<query(1,l,r)<<endl;
    }
}
```

2.5 PBDS

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T> using rbt = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
int main(){
    rbt<int> t; //declare
}

/*
不支援重複值 (需要的話可用左推+值來處理)
支援set, map之操作
find_by_order(k) ：像陣列一樣回傳第 k 個值。(0-based, pointer)
order_of_key(k) ：回傳 k 是集合裡第幾大。(0-based)

T 資料型別
null_type //當作 map 使用的時候要對應什麼資料型態，
     //要當作 set 就用 null_type
less<T> // key value 要用什麼方式比較
*/
```

2.6 Per Seg

```cpp
struct Per_seg{
    int l, r, m;
    int v = 0;
    Per_seg *ln = nullptr, *rn = nullptr;
    Per_seg(int _l, int _r) : l(_l), r(_r), m((_l + _r) >> 1) {}
    void build(){
        if (l != r - 1)
        {
            ln = new Per_seg(l, m);
            rn = new Per_seg(m, r);
            ln->build();
            rn->build();
        }
    }
    void upd(int tar, int value){
        if (tar == l && tar == r - 1){
            v = value;
            return;
        }
        else{
            int m = (l + r) >> 1;
            if (tar < m){
                ln = new Per_seg(*ln);
                ln->upd(tar, value);
            }
            else{
                rn = new Per_seg(*rn);
                rn->upd(tar, value);
            }
            v = ln->v + rn->v;
        }
    }
    int query(int ll, int rr){
        if (l == ll && r == rr){
            return v;
        }
        else{
            if (m >= rr){
                return ln->query(ll, rr);
            }
            else if (m <= ll){
                return rn->query(ll, rr);
            }
            else{
                return ln->query(ll, m) + rn->query(m, rr);
            }
        }
    }
};

signed main(){
    int n, q; // n = array size, q = query times
    cin>>n>>q;
    vector<Per_seg *> tr;
    tr.push_back(new Per_seg(0, n));
    tr[0]->build();
    for (int i = 0; i < n; i++){
        int a;
        cin >> a;
        tr[0]->upd(i, a); // init ver.0, 0-based!
    } // build done
    // Set the value a in array k to x: tr[k]->upd(a, x);
    // Sum of values in range [a,b) in array k: tr[k]->query(l, r)
    // []? []?
    // Create a copy of array k: tr.push_back(new Per_seg(*tr[k]))
}
```

2.7 SparseTable

```cpp
const int MAXN = 5e5 + 5;
const int lgN = 20;

struct SP{
    vector <int> Sp[lgN];
    void build(int n, int *a){
        for(int i=0;i<n;i++)
            Sp[0].push_back(a[i]);
        for(int h=1;h<lgN;h++){
            int len = (1 << (h - 1)), i = 0;
            for(; i + len < n; i++)
                Sp[h].push_back(max(Sp[h-1][i], Sp[h-1][i+len]));
            for(; i < n; i++)
                Sp[h].push_back(Sp[h-1][i]);
        }
    }
    int query(int l, int r){
        int lg = __lg(r - l + 1);
        int len = (1 << lg);
        return max(Sp[lg][l], Sp[lg][r - len + 1]);
    }
};
```

2.8 Treap

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

struct node{ // support range reverse, range sum query
    node *l, *r;
    int key, val, sum, pri, size, rev;
    node(int k, int v) : l(0), r(0), key(k), val(
        v), sum(v), pri(rand()), size(1), rev(0)
        {};
    void up();
    void down();
};

node *merge(node *a, node *b){
    if(!a || !b) return a ? a : b;
    if(a -> pri < b -> pri){
        a -> down();
        a -> r = merge(a -> r, b);
        a -> up();
        return a;
    }
    else{
        b -> down();
        b -> l = merge(a, b -> l);
        b -> up();
        return b;
    }
}

void split(node *o, node *&a, node *&b, int k){
    // split by key
    if(!o)
        a = b = 0;
    else{
        o -> down();
        if(o -> key < k){
            a = o;
            split(o -> r, a -> r, b, k);
        }
        else{
            b = o;
            split(o -> l, a, b -> l, k);
        }
        o -> up();
    }

}

void insert(node *&root, int k, int v){
    node *a, *b;
    split(root, a, b, k);
    root = merge(a, merge(new node(k, v), b));
}

bool erase(node *&o, int k){  // erase T[k]
    if(!o)
        return 0;
    if(o -> key == k){
        node *t = o;
        o = merge(o -> l, o -> r);
        delete t;
        return 1;
    }
    node *&t = k < o -> key ? o -> l : o -> r;
    if(erase(t, k)) return o -> up(), 1;
    else return 0;
}

void node :: up(){
    size = 1;
    sum = val;
    if(l) {
        //l -> down();
        size += l -> size;
        sum += l -> sum;
    }
    if(r) {
        //r -> down();
        size += r -> size;
        sum += r -> sum;
    }
}

void node :: down(){
    if(rev){
        swap(l, r);
        if(l) l -> rev ^= 1;
        if(r) r -> rev ^= 1;
        rev = 0;
    }
}

inline int size(node *o){
    return o ? o -> size : 0;
}

int Rank(node *& root, int val){// Number of
    elements smaller than val.
    node *a, *b;
    split(root, a, b, val);
    int res = size(a);
    root = merge(a, b);
    return res;
}

void split2(node *o, node *&a, node *&b, int k){
    // split by size
    if(!o)
        a = b = 0;
    else{
        o -> down();
        if(k >= size(o -> l) + 1){
            a = o;
            int nk = k - (size(o -> l) + 1);
            split2(o -> r, a -> r, b, nk);
        }
        else{
            b = o;
            split2(o -> l, a, b -> l, k);
        }
        o -> up();
    }
}

node *kth(node *&root, int k){ // find T[k]
    node *a, *b, *c;
    split2(root, a, c, k);
    split2(a, a, b, k - 1);
    root = merge(a, merge(b, c));
    return b;
}

void reverse(node *&root, int l, int r){
    node *a, *b, *c;
    split2(root, a, b, l - 1);
    split2(b, b, c, r - l + 1);
    b -> rev ^= 1;
    root = merge(a, merge(b, c));
}

int query(node *&root, int l, int r){
    node *a, *b, *c;
    split2(root, a, b, l - 1);
    split2(b, b, c, r - l + 1);
    b -> down();
    int res = b -> sum;
    root = merge(a, merge(b, c));
    return res;
}

void update(node *&root, int pos, int x){ // let
    T[pos] = x
    erase(root, pos);
    insert(root, pos, x);
}

signed main(){
    node *T(nullptr);
    int n, q;
    cin >> n >> q;
    for(int i=1;i<=n;i++){
        int tmp;
        cin >> tmp;
        insert(T, i, tmp);
    }
    while(q--){
        int op;
        cin >> op;
        if(op == 1){ // update
            int pos, x;
            cin >> pos >> x;
            update(T, pos, x);
        }
        else{ // query
            int l, r;
            cin >> l >> r;
            cout << query(T, l, r) << '\n';
        }
    }
}
```

3 Geometric

3.1 Closetpair

```
typedef pair<ll, ll> pii;
#define x first
#define y second
ll dd(const pii& a, const pii& b) {
    ll dx = a.x - b.x, dy = a.y - b.y;
    return dx * dx + dy * dy;
}
const ll inf = 1e18;
//在一點對陣列的[l, r]間找最近點對
ll dac(vector<pii>& p, int l, int r) {
    if (l >= r) return inf;
    int m = (l + r) / 2;
    ll d = min(dac(p, l, m), dac(p, m + 1, r));
    vector<pii> t;
    for (int i = m; i >= l && p[m].x - p[i].x < d
        ; i--)
        t.push_back(p[i]);
    for (int i = m + 1; i <= r && p[i].x - p[m].x
         < d; i++)
        t.push_back(p[i]);
    sort(t.begin(), t.end(),
         [](pii& a, pii& b) { return a.y < b.y;
            });
    int n = t.size();
    for (int i = 0; i < n - 1; i++)
        for (int j = 1; j < 4 && i + j < n; j++)
            // 這裡可以知道是哪兩點是最小點對
            d = min(d, dd(t[i], t[i + j]));
    return d;
}
// 給一堆點,求最近點對的距離「的平方」。
ll closest_pair(vector<pii>& pp) {
    sort(pp.begin(), pp.end());
    return dac(pp, 0, pp.size() - 1);
}
```

3.2 Gramh

```
//#define pdd (double/int)
int cross(pdd a, pdd b){
    return a.first*b.second - a.second*b.first;
}

pdd operator-(pdd a, pdd b){
    return {a.first - b.first, a.second - b.
        second};
}

double operator*(pdd a, pdd b){
    return a.first * b.second -  a.second * b.
        first;
}

// ps是所有的點, 要去重!!!
vector<pdd> convexHull(vector<pdd>& ps) {
    sort(all(ps));
    vector<pdd> hull;
    if (ps.size() <= 2) {
        return ps;
    }
    for (int i = 0; i < 2; i++) {
        int s = hull.size();
        for (pdd p : ps) {
            while (hull.size() - s >= 2 && cross(
                hull.back() - hull[hull.size() -
                2], p - hull[hull.size() - 2]) < 1
                e-10) {
                hull.pop_back();
            }
            hull.pb(p); //push_back
        }
        hull.pop_back();
        reverse(all(ps));
    }
    return hull;
}
```

3.3 Rectangle Union Area

```
const int maxn = 1e5 + 10;
struct rec{
    int t, b, l, r;
    //t頂,b底,l左,r右邊界點
} r[maxn];
int n, cnt[maxn << 2];
long long st[maxn << 2], ans = 0;
vector<int> x, y;
vector<pair<pair<int, int>, pair<int, int>>> v;
void modify(int t, int l, int r, int ql, int qr,
    int v) {
    if (ql <= l && r <= qr) cnt[t] += v;
    else {
        int m = (l + r) >> 1;
        if (qr <= m) modify(t << 1, l, m, ql, qr,
            v);
        else if (ql >= m) modify(t << 1 | 1, m, r
            , ql, qr, v);
        else modify(t << 1, l, m, ql, m, v),
            modify(t << 1 | 1, m, r, m, qr, v);
    }
    if (cnt[t]) st[t] = y[r] - y[l];
    else if (r - l == 1) st[t] = 0;
    else st[t] = st[t << 1] + st[t << 1 | 1];
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        //輸入個個長方形的上下左右界
        cin >> r[i].l >> r[i].r >> r[i].b >> r[i
            ].t;
        if (r[i].l > r[i].r) swap(r[i].l, r[i].r)
            ;
        if (r[i].b > r[i].t) swap(r[i].b, r[i].t)
            ;
        x.push_back(r[i].l);
        x.push_back(r[i].r);
        y.push_back(r[i].b);
        y.push_back(r[i].t);
    }
    sort(x.begin(), x.end());
    sort(y.begin(), y.end());
    x.erase(unique(x.begin(), x.end()), x.end());
    y.erase(unique(y.begin(), y.end()), y.end());
    for (int i = 0; i < n; i++) {
        r[i].l = lower_bound(x.begin(), x.end(),
            r[i].l) - x.begin();
        r[i].r = lower_bound(x.begin(), x.end(),
            r[i].r) - x.begin();
        r[i].b = lower_bound(y.begin(), y.end(),
            r[i].b) - y.begin();
        r[i].t = lower_bound(y.begin(), y.end(),
            r[i].t) - y.begin();
        v.emplace_back(make_pair(r[i].l, 1),
            make_pair(r[i].b, r[i].t));
        v.emplace_back(make_pair(r[i].r, -1),
            make_pair(r[i].b, r[i].t));
    }
    sort(v.begin(), v.end(), [](pair<pair<int,
        int>, pair<int, int>> a, pair<pair<int,
        int>, pair<int, int>> b){
        if (a.first.first != b.first.first)
            return a.first.first < b.first.first;
        return a.first.second > b.first.second;
    });
    for (int i = 0; i < v.size(); i++) {
        if (i) ans += (x[v[i].first.first] - x[v[
            i - 1].first.first]) * st[1];
        modify(1, 0, y.size(), v[i].second.first,
             v[i].second.second, v[i].first.second
            );
    }
    cout << ans << '\n';
    return 0;
}
```

3.4 TheLeastCoverCircle

```
const double eps = 1e-10, pi = acos(-1);
struct Circle{
    pdd o;
    double r;
}c;
vector<pdd> p;
int R, n, r;
pdd operator+(pdd a, pdd b){
    return {a.F + b.F, a.S + b.S};
}
pdd operator-(pdd a, pdd b){
    return {a.F - b.F, a.S - b.S};
}
pdd operator*(pdd a, double b){
    return {a.F * b, a.S * b};
}
pdd operator/(pdd a, double b){
    return {a.F / b, a.S / b};
}
double operator*(pdd a, pdd b){
    return a.F * b.S -  a.S * b.F;
}
int judge(double a, double b){
    if (fabs(a-b) < eps) return 0;
    if (a < b) return -1;
    return 1;
}
pdd rotate(pdd a, double b){
    return {a.F*cos(b)+a.S*sin(b), -a.F*sin(b)+a.
        S*cos(b)};
}
double lens(pdd a, pdd b){
    double dx = b.F - a.F, dy = b.S - a.S;
    return sqrt(dx*dx + dy*dy);
}
pdd intersection(pdd p, pdd v, pdd q, pdd w){//求
    交點
    pdd u = p - q;
    double t = w*u/(v*w);
    return p + v * t;
}
pair<pdd, pdd> bisector(pdd a, pdd b){//求中垂線
    pdd p = (a + b) / 2.0;
    pdd v = rotate(b - a, pi / 2.0);
    return {p, v};
}
Circle circle(pdd a, pdd b, pdd c){ //三點求圓
    auto n = bisector(a, b), m = bisector(a, c);
    pdd o = intersection(n.F, n.S, m.F, m.S);
    double r = lens(o, a);
    return {o, r};
}
void solve(){
    p.clear();
    cin >> n;
    p.resize(n);
    //輸入所有的點
    for (int i = 1; i <= n; i++){
        cin >> p[i].F >> p[i].S;
    }
    random_shuffle(all(p));
    c = {p[0], 0};
    for(int i = 1; i <= n; i++){
        if (judge(c.r, lens(c.o, p[i])) == -1){
            c = {p[i], 0};
            for (int j = 0; j < i; j++){
                if (judge(c.r, lens(c.o, p[j]))
                    == -1){
                    c = {(p[i] + p[j]) / 2.0,
                        lens(p[i], p[j]) / 2.0};
                    for (int k = 0; k < j; k++){
                        if (judge(c.r, lens(c.o,
                            p[k])) == -1){
                            c = circle(p[i], p[j
                                ], p[k]);
                        }
                    }
                }
            }
        }
    }
    //c: {圓心, 半徑}
    if (n == 1) c = {(p[0] + p[1]) / 2.0, 0};
    cout << setprecision(9) << fixed;
    cout << -c.o.F << " " << -c.o.S << endl;
}
```

4 Graph

4.1 2e cc

```
// i.e. bridge tree
// Remember to reset vis[]
map<int,int> compId;
vector<int> g2[N];
void dfs(int now,int p,int iid){
    vis[now]=1;
    compId[now]=iid;
    for(auto [nxt,id]:g[now]){
        if(bridge[id]) continue;
        if(nxt==p) continue;
        if(!vis[nxt]) dfs(nxt,now,iid);
    }
}
//////Then, in main()
    int iid=0;
    for(int i=0;i<n;i++) vis[i]=0;
    for(int i=0;i<n;i++){
        if(!vis[i]){
            dfs(i,-1,iid);
            iid+=1;
        }
    }
    for(int i=0;i<m;i++){
        if(bridge[i]){
            auto [u,v] = edge[i];
            g2[compId[u]].pb(compId[v]);
            g2[compId[v]].pb(compId[u]);
        }
    }
```

4.2 2sat tarjan

```
const int N = 2005;
int low[N],dfn[N],color[N],ins[N]; //要開兩倍大
// color[x] 是 x 所在的 scc 的topo逆序。
vector<int> g[N];
int dfsClock,sccCnt;
stack<int> stk;
void tarjan(int u) {
    low[u] = dfn[u] = ++dfsClock;
    stk.push(u); ins[u] = true;
    for (const auto &v : g[u]) {
        if (!dfn[v]) tarjan(v), low[u] = std::min(low[u], low[v]);
        else if (ins[v]) low[u] = std::min(low[u], dfn[v]);
    }
    if (low[u] == dfn[u]) {
        ++sccCnt;
        do {
            color[u] = sccCnt;
            u = stk.top(); stk.pop(); ins[u] = false;
        } while (low[u] != dfn[u]);
    }
}

signed main(){
    g[i].pb(j); // i->j

    for (int i = 1; i <= (n << 1); ++i) if (!dfn[i]) tarjan(i);  // run tarjan, 注意0~2n-1 or 1~2n

    for(int i=1;i<=n;i++){
        if(color[i] == color[i+n]){
            cout<<"NO"<<endl;
            return;
        }
    }
    cout<<"YES"<<endl;
    // 找環 注意建邊方法是(i,i+1) or (i,i+n)

    for(int i=1;i<=n;i++){
        if(color[i] < color[i+n]){
            cout<<1<<" ";
        }
        else cout<<0<<" ";
    }
    // 構造解 (注意是0~n-1還是1~n)
}
```

4.3 Bridge

```
//for undirected graph, find bridge
const int N = 1e6+5;

vector<pair<int,int> > edge(N); //{u,v} -> remember to input
vector<pair<int,int> > g[N]; //{nxt,edge_id}
vector<int> bridge(N);
int dfn[N],vis[N],low[N],id;
void tarjan(int now,int p){
    dfn[now]=id++;
    vis[now]=1;
    low[now]=dfn[now];
    for(auto [nxt,id]:g[now]){
        if(nxt==p) continue;
        if(vis[nxt]){
            low[now]=min(low[now],dfn[nxt]); // back edge!
        }
        else{
            tarjan(nxt,now);
            low[now]=min(low[now],low[nxt]);
            if(low[nxt]>dfn[now]){
                bridge[id]=1;
            }
        }
    }
}
signed main(){
    // construct
    for(int i=0;i<n;i++){
        dfn[i]=1e9;//reset
        low[i]=1e9;
        vis[i]=0;
    }
    id=0;
    for(int i=0;i<n;i++){
        if(!vis[i]){
            tarjan(i,-1);
        }
    }
    // use
}
```

4.4 Dinic

```
(a) Bounded Maxflow Construction:
1. add two node ss, tt
2. add_edge(ss, tt, INF)
3. for each edge u -> v with capacity [l, r]:
        add_edge(u, tt, l)
        add_edge(ss, v, l)
        add_edge(u, v, r-l)
4. see (b), check if it is possible.
5. answer is maxflow(ss, tt) + maxflow(s, t)
-------------------------------------------------------
(b) Bounded Possible Flow:
1. same construction method as (a)
2. run maxflow(ss, tt)
3. for every edge connected with ss or tt:
        rule: check if their rest flow is exactly
            0
4. answer is possible if every edge do satisfy
    the rule;
5. otherwise, it is NOT possible.
-------------------------------------------------------
(c) Bounded Minimum Flow:
1. same construction method as (a)
2. answer is maxflow(ss, tt)
-------------------------------------------------------
(d) Bounded Minimum Cost Flow:
* the concept is somewhat like bounded possible
    flow.
1. same construction method as (a)
2. answer is maxflow(ss, tt) + (∑ l * cost for
    every edge)
-------------------------------------------------------
(e) Minimum Cut:
1. run maxflow(s, t)
2. run cut(s)
3. ss[i] = 1: node i is at the same side with s.
-------------------------------------------------------


const long long INF = 1LL<<60;
struct Dinic {  //O(VVE), with minimum cut
    static const int MAXN = 5003;
    struct Edge{
        int u, v;
        long long cap, rest;
    };

    int n, m, s, t, d[MAXN], cur[MAXN];
    vector<Edge> edges;
    vector<int> G[MAXN];

    void init(){
        edges.clear();
        for ( int i = 0 ; i < n ; i++ ) G[i].
            clear();
        n = 0;
    }

    // min cut start
    bool side[MAXN];
    void cut(int u) {
        side[u] = 1;
        for ( int i : G[u] ) {
            if ( !side[ edges[i].v ] && edges[i].
                rest ) cut(edges[i].v);
        }
    }
    // min cut end

    int add_node(){
        return n++;
    }

    void add_edge(int u, int v, long long cap){
        edges.push_back( {u, v, cap, cap} );
        edges.push_back( {v, u, 0, 0LL} );
        m = edges.size();
        G[u].push_back(m-2);
        G[v].push_back(m-1);
    }

    bool bfs(){
        fill(d,d+n,-1);
        queue<int> que;
        que.push(s); d[s]=0;
        while (!que.empty()){
            int u = que.front(); que.pop();
            for (int ei : G[u]){
                Edge &e = edges[ei];
                if (d[e.v] < 0 && e.rest > 0){
                    d[e.v] = d[u] + 1;
                    que.push(e.v);
                }
            }
        }
        return d[t] >= 0;
    }

    long long dfs(int u, long long a){
        if ( u == t || a == 0 ) return a;
        long long flow = 0, f;
        for ( int &i=cur[u]; i < (int)G[u].size()
            ; i++ ) {
            Edge &e = edges[ G[u][i] ];
            if ( d[u] + 1 != d[e.v] ) continue;
            f = dfs(e.v, min(a, e.rest) );
            if ( f > 0 ) {
                e.rest -= f;
                edges[ G[u][i]^1 ].rest += f;
                flow += f;
                a -= f;
                if ( a == 0 )break;
            }
        }
        return flow;
    }

    long long maxflow(int _s, int _t){
        s = _s, t = _t;
        long long flow = 0, mf;
        while ( bfs() ){
            fill(cur,cur+n,0);
            while ( (mf = dfs(s, INF)) ) flow +=
                mf;
        }
        return flow;
    }
} dinic;
```

4.5 Hungarian

```cpp
// Maximum Cardinality Bipartite Matching
// Worst case O(nm)

struct Graph{
    static const int MAXN = 5003;
    vector<int> G[MAXN];
    int n, match[MAXN], vis[MAXN];

    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++) G[i].clear();
    }

    bool dfs(int u){
        for (int v:G[u]){
            if (vis[v]) continue;
            vis[v]=true;
            if (match[v]==-1 || dfs(match[v])){
                match[v] = u;
                match[u] = v;
                return true;
            }
        }
        return false;
    }

    int solve(){
        int res = 0;
        memset(match,-1,sizeof(match));
        for (int i=0; i<n; i++){
            if (match[i]==-1){
                memset(vis,0,sizeof(vis));
                if ( dfs(i) ) res++;
            }
        }
        return res;
    }
} graph;
```

4.6 LCA fd

```cpp
// online O(nlogn + mlogn)
const int N = 300005;
int d[N],f[N][20];// f[i][j] = i's 2^j father, d[
    i] = depth of i
vector<int> g[N]; // graph

void dfs(int now,int p,int dep){
    d[now] = dep+1;
    for(int nxt:g[now]){
        if(nxt==p) continue;
        f[nxt][0] = now;
        dfs(nxt,now,dep+1);
    }
    return;
}

int lca(int x, int y){
    if(d[x]<d[y]) swap(x,y);
    int k = d[x]-d[y];
    for(int i=0;i<20;i++){
        if(k&1) x = f[x][i];
        k>>=1;
    }// jump to the same depth/height
    if(x==y) return x;
    for(int i=19; i>=0;i--){
        if(f[x][i]!=f[y][i]){
            x = f[x][i];
            y = f[y][i];
        }
    }// find the first different -> higher is LCA
    return f[x][0];
}
void sol(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        g[u].pb(v);
        g[v].pb(u);
    }
    dfs(1,0,0); // arbitrarily choose a root,
        here choose 1 as root
    for(int j=1;j<20;j++){
        for(int i=1;i<=n;i++){
            f[i][j] = f[f[i][j-1]][j-1];
        }// get all f
    }
    // --- use lca(u,v) to get ---
}
```

4.7 LCA tarjan

```cpp
// Tarjan (offline, O(n + m))
void dfs(int now, int p, int dep){
    d[now] = dep; // d[i] = depth of i, be
        careful about "root should set to 0/1"
    for(int nxt:g[now]){
        if(nxt==p) continue;
        dfs(nxt, now, dep+1);
        connect(now,nxt); // connect son "to" its
            parent
        vis[nxt] = 1;
    }
    // Deal with query
    for(auto i:q[now]){
        int nxt = i.first; // query has {now,nxt}
        int id = i.second; // query_id
        if(vis[nxt]){
            qans[id] = find_root(nxt);
        }
    }
}
```

5 Math

5.1 FFT

```cpp
const int N = 1e7+10;
const double Pi = acos(-1.0);
struct Complex{
    double x,y;
    Complex ( double xx=0, double yy=0){
        x=xx;
        y=yy;
    }
};
Complex a[N], b[N];
Complex operator + (Complex a, Complex b) {
    return Complex(a.x + b.x , a.y + b.y);}
Complex operator - (Complex a, Complex b) {
    return Complex(a.x - b.x , a.y - b.y);}
```

```
Complex operator * (Complex a, Complex b) {
    return Complex(a.x * b.x - a.y * b.y , a.x * b
    .y + a.y * b.x);}

int limit=1, h=0, rev[N];

void fft(Complex *A, int flag){
    for(int i=0; i<limit; i++){
        if(i<rev[i]){
            swap(A[i], A[rev[i]]);
        }
    }
    for(int len=1; len<limit; len<<=1){
        // len = 待合併區間的一半
        Complex Wn(cos(Pi/len), flag*sin(Pi/len))
            ;
        for(int R=len<<1, j=0; j<limit; j+=R){
            Complex w(1, 0);
            for (int k = 0; k < len; k++, w = w *
                 Wn) {
                Complex x = A[j + k], y = w * A[j
                    + len + k];
                A[j + k] = x + y;
                A[j + len + k] = x - y;
            }
        }
    }
}

int main(){
  int n,m; // n,m次方
  cin>>n>>m;
    for(int i=0; i<=n; i++) cin>>a[i].x;
    for(int i=0; i<=m; i++) cin>>b[i].x;
    while(limit<=n+m){
        limit=limit<<1;
        h++;
    }
    for(int i=0; i<limit; i++){
        rev[i] = (rev[i>>1]>>1) | ((i&1)<<(h-1));
           // bit reverse
    }
    fft(a, 1);
    fft(b, 1);//FFT
    for(int i=0; i<=limit; i++) a[i]=a[i]*b[i];
    fft(a, -1);//IFFT
    for(int i=0; i<=n+m; i++) cout<<(int)(a[i].x
        / limit + 0.5)<<" ";
}
```

5.2 LinearSieve

```
int LeastPrimeDivisor[maxn];
vector<int> pr;

void LinearSieve(){
  for(int i = 2; i < maxn; i++){
    if(!LeastPrimeDivisor[i]) pr.push_back(i),
        LeastPrimeDivisor[i] = i;
    for(int p : pr){
      if(i * p >= maxn) break;
      LeastPrimeDivisor[i * p] = p;
      if(i % p == 0) break;
    }
  }
}
```

5.3 NTT

```
const int N = 1e7+10;
const int P = 998244353, G = 3, Gi = 332748118;//
    primitive root = 3, Gi = mod inverse of 3

int fastpow(int x,int p){
    int sum = 1;
    while(p){
        if(p&1) sum = sum*x%P;
        x = x*x%P;
        p = p>>1;
    }
    return sum;
}

int a[N], b[N], limit=1, h=0, rev[N];
inline void NTT(int *A, int flag) {
  for(int i = 0; i < limit; i++)
    if(i < rev[i]) swap(A[i], A[rev[i]]);
  for(int len = 1; len < limit; len <<= 1){
    int Wn = fastpow( flag == 1 ? G : Gi , (P -
        1) / (len << 1));
    for(int j = 0; j < limit; j += (len << 1)){
      int w = 1;
      for(int k = 0; k < len; k++) {
            int x = A[j + k], y = w * A[j + k
                + len] % P;
            A[j + k] = (x + y) % P,
            A[j + k + len] = (x - y + P) % P;
            w = (w * Wn) % P;
        }
    }
  }
}

signed main(){
    int n,m;
    cin>>n>>m; // n,m次方
    for(int i=0; i<=n; i++){
        cin>>a[i];
        a[i]=(a[i] + P) % P;
    }
    for(int i=0; i<=m; i++){
        cin>>b[i];
        b[i]=(b[i] + P) % P;
    }
    while(limit<=n+m){
        limit=limit<<1;
        h++;
    }
    for(int i=0; i<limit; i++){
        rev[i] = (rev[i>>1]>>1) | ((i&1)<<(h-1));
    }
    NTT(a, 1);
    NTT(b, 1);
    for(int i=0; i<=limit; i++) a[i]=a[i]*b[i]%P;
    NTT(a, -1);

    int inv = fastpow(limit, P - 2);
    for(int i=0; i<=n+m; i++){
        cout<<(a[i]*inv)%P << " ";
    }
}
```

5.4 RabinMiller

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

```

```cpp
int QuickPow(int base, int exponent, int mod){
    if(exponent == 0)
        return 1;
    if(exponent == 1)
        return base;
    if(exponent % 2)
        return QuickPow(base, exponent - 1, mod)
            * base % mod;
    int tmp = QuickPow(base, exponent / 2, mod);
    return tmp * tmp % mod;
}

bool RabinMiller(int d, int n){
    int a = 2 + rand() % (n - 2);
    if(QuickPow(a, n - 1, n) != 1)
        return false;
    int cur = QuickPow(a, d, n);
    int nx;
    while(d != n - 1){
        nx = (cur * cur) % n;
        d *= 2;
        if(cur != 1 && cur != n - 1){
            if(nx == 1)
                return false;
        }
        cur = nx;
    }
    return true;
}

bool isPrime(int n, int k){
    if(n <= 1)
        return false;
    if(n <= 3)
        return true;
    if(n == 4)
        return false;
    if((n - 1) % 6 != 0 && (n + 1) % 6 != 0)
        return false;
    int d = n - 1;
    int r = 0;
    while(d % 2 == 0){
        d /= 2;
        r ++;
    }
    for(int i=0;i<k;i++){
        if(!RabinMiller(d, n))
            return false;
    }
    return true;
}

signed main(){
    int n;
    while(cin >> n){
        if(isPrime(n, 5)) // 預設k = 5
            cout << "質數" << endl;
        else
            cout << "非質數" << endl;
    }
}
```

6 Misc

6.1 Rand

```cpp
mt19937_64 rnd(random_device{}());
uniform_int_distribution<int> dist(i, j);
/*
dist(rnd) -> 取 i~j範圍內的整數
*/
```

6.2 SG

```cpp
int f[100]; // state, reset to -1

int sg(int x) {
    if (f[x] != -1) return f[x];
    unordered_set<int> S;
    if (x >= 1) S.insert(sg(x - 1)); // all sub-
        states (suppose x can be x-1, x-2, 0)
    if (x >= 2) S.insert(sg(x - 2));
    if (x >= 3) S.insert(sg(0));
    for (int i = 0;; i++) {
        if (!S.count(i)) return f[x] = i;
    }
}
```

7 Strings

7.1 Rolling hash

```cpp
const int N = 2000; // string size
int X = 1000000007, P = 4000000007;
// other primes : 8298176713 5119240589
    3735751997 3218996237...
int s[N], p[N]; // if hash兩次 -> 開兩個s,p

void Hash(string& str){
    s[0] = str[0];
    p[0] = 1;
    for(int i=1; i<str.size(); i++){
        s[i] = (s[i-1]*X + str[i])%P;
        p[i] = (p[i-1]*X) % P;
    }
    return;
}
int hash_i(int a,int b){
    if(!a) return s[b];
    int tmp = s[b] - s[a-1]*p[b-a+1]; //
        calculate H[a:b]
    tmp=tmp%P;
    if(tmp<0) return tmp+P;
    return tmp;
}

signed main(){
    // Hash to generate s and p
    // hash_i to calculate s[a:b]
    string s;
    cin>>s;
    Hash(s);
    cout<<hash_i(0,2)<<" "<<hash_i(3,5);
}
```

ACM ICPC Team - NYCU OverFlowers

Contents