Notebook - Maratona de Programação

My little paçoca do Chico Bento

19/07/2025

Contents

1 Algoritmos

1.1 Mo

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define ld long double
#define ll long long
#define pb push_back
#define ff first
#define ss second
#define vi vector<int>
#define pii pair<int, int>
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
const int MAXN = 101;
const int INF = INT64_MAX;
const int MOD = 1e9+7;
const int LOG = 60;
const ld PI = acos(-1);

struct QueryMo{

    int l, r, sec, ord;

    QueryMo(int inL, int inR, int inSec, int inOrd){

        l = inL, r = inR, sec = inL / inSec, ord = inOrd;
    }

    bool operator<(QueryMo &compa){

        return make_pair(sec, r) < make_pair(compa.sec, compa.r);
    }

};
void solve(){

    int n, q; cin >> n >> q;

    vi v(n);
    map<int, int> id;

    for(int i = 0; i < n; i++){

        int x; cin >> x;

        if(!id.count(x)) id[x] = i+1;

        v[i] = id[x];
    }

    vector<QueryMo> auxQueries;

    int rt = min((int)200, (int)(sqrt(q)));

    for(int i = 0; i < q; i++){

        int l, r; cin >> l >> r;

        QueryMo aux = QueryMo(--l, --r, rt, i);

        auxQueries.pb(aux);
    }

    vi resp(q);

    sort(all(auxQueries));

    int actL = 0, actR = 0, ans = 0;
    vector<int> freqs(2e5+2, 0);

    ans++;
    freqs[v[actL]]++;

    for(auto query : auxQueries){

        while(actR < query.r){

            actR++;
            if(!freqs[v[actR]]) ans++;
            freqs[v[actR]]++;
        }

        while(actL > query.l){

            actL--;
            if(!freqs[v[actL]]) ans++;
            freqs[v[actL]]++;
        }

        while(actL < query.l){

            freqs[v[actL]]--;
            if(!freqs[v[actL]]) ans--;
            actL++;
        }

        while(actR > query.r){

            freqs[v[actR]]--;
            if(!freqs[v[actR]]) ans--;
            actR--;
        }


        resp[query.ord] = ans;
    }

    for(int i = 0; i < q; i++) cout << resp[i] << '\n';

    return;
```

```
}
int32_t main(){
    sws;

    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

1.2 Ternary Search

```
// Uma busca em uma curva, avaliando dois pontos diferentes
// Complexidade: O(Nlog3N)

double check(vector<int> v, vector<int> t, double x){
  double ans = 0;
  for(int i=0; i<v.size(); i++){
    ans = max(ans, (double)(abs(v[i]-x) + t[i]));
  }
  return ans;
}

int32_t main(){ sws;

  int t; cin>>t;
  while(t--){
    int n; cin>>n;
    vector<int> v(n);
    vector<int> t(n);
    input(v);
    input(t);

    double ans = 0.0;
    double l=0.0, r=1e9;
    while(r-l >= EPS){

      double mid1 = (double) l + (r - l) / 3;
      double mid2 = (double) r - (r - l) / 3;

      double x1 = check(v, t, mid1);
      double x2 = check(v, t, mid2);

      if(x1 < x2){
        r = mid2;
      }else{
        l = mid1;
        ans = l;
      }
    }
    cout<<fixed<<setprecision(7);
    cout<<ans<<endl;
  }
    return 0;
}
```

2 DP

2.1 Dp

```
// DP - Dynamic Programming

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

typedef long long ll;
const int MAX = 110;

int n;
int tab[MAX];
vector<int> v;

ll dp(int i){
    if(i>=n) return 0;
    if(tab[i] != -1) return tab[i];

    int pega = v[i] + dp(i+2);
    int npega = dp(i+1);

    tab[i] = max(pega, npega);
    return tab[i];
}

int main(){
    memset(tab, -1, sizeof(tab));
    cin>>n;

    v.assign(n, 0);

    cout<<dp(0)<<endl;

    return 0;
}
```

2.2 Knapsack

```
int n, t;
int tab[N][N];
bool pegou[N][N];
vector<pair<int,int>> v;

vector<int> resposta;

int dp(int idx, int dias){
    if(idx >= n) return 0;
    if(tab[idx][dias] != -1) return tab[idx][dias];

    int pega=0;
    if(dias+v[idx].first <= t){
        pega = dp(idx+1, dias+v[idx].first)+v[idx].second;
    }

    int npega = dp(idx+1, dias);
```

```
    if(pega>npega) pegou[idx][dias] = true;

    return tab[idx][dias] = max(pega, npega);
}

int32_t main(){
    memset(tab, -1, sizeof(tab));
    cin>>n>>t;
    for(int i=0; i<n; i++){
        int ti, di;
        cin>>ti>>di;

        v.push_back({ti, di});
    }
    dp(0, 0);
    int i = 0, j =0;
    vector<int> ans;
    // retornar os valores
    while(i < n){
        if(pegou[i][j]){
            j += v[i].first;
            ans.push_back(i+1);
        }
        i++;
    }
    cout<<ans.size()<<endl;
    for(int i=0; i<ans.size(); i++){
        cout<<ans[i]<<" ";
    }

}
```

2.3 Lcsubseq

```
string lcs(string x, string y) {
    int n = x.size(), m = y.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for (int i=0;i<=n;i++) {
        for (int j=0;j<=m;j++) {
            if (i == 0 or j == 0) continue;
            if (x[i-1] == y[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }

    // int len = dp[n][m];
    string ans = "";
    int i = n-1, j = m-1;
    while (i >= 0 and j >= 0) { // recover string
        if (x[i] == y[j]) ans.pb(x[i]), i--, j--;
        else if (dp[i][j+1] > dp[i+1][j]) i--;
        else j--;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

2.4 Lis

```
// Longest increase sequence
// O(nlogn)
multiset<int> S;
for(int i=0;i<n;i++){
    auto it = S.upper_bound(vet[i]); // upper - longest strictly increase
    sequence
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
// size of the lis
int ans = S.size();

// return the elements in LIS
////////////////////////////// see that later
// https://codeforces.com/blog/entry/13225?#comment-180208

vi LIS(const vi &elements){
    auto compare = [&](int x, int y) {
        return elements[x] < elements[y];
    };
    set< int, decltype(compare) > S(compare);

    vi previous( elements.size(), -1 );
    for(int i=0; i<int( elements.size() ); ++i){
        auto it = S.insert(i).first;
        if(it != S.begin())
            previous[i] = *prev(it);
        if(*it == i and next(it) != S.end())
            S.erase(next(it));
    }

    vi answer;
    answer.push_back( *S.rbegin() );
    while ( previous[answer.back()] != -1 )
        answer.push_back( previous[answer.back()] );
    reverse( answer.begin(), answer.end() );
    return answer;
}
```

2.5 Sosdp

```
// SOS DP [nohash]
//
// O(n 2^n)

// soma de sub-conjunto
vector<ll> sos_dp(vector<ll> f) {
  int N = __builtin_ctz(f.size());
  assert((1<<N) == f.size());
```

```
  for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N); mask++)
    if (mask>>i&1) f[mask] += f[mask^(1<<i)];
  return f;
}

// soma de super-conjunto
vector<ll> sos_dp(vector<ll> f) {
  int N = __builtin_ctz(f.size());
  assert((1<<N) == f.size());

  for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N); mask++)
    if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
  return f;
}
```

3 ED

3.1 Dst

```
// Sparse Table Disjunta
//
// Resolve qualquer operacao associativa
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
  int m[MAX2][2*MAX], n, v[2*MAX];
  int op(int a, int b) { return min(a, b); }
  void build(int n2, int* v2) {
    n = n2;
    for (int i = 0; i < n; i++) v[i] = v2[i];
    while (n&(n-1)) n++;
    for (int j = 0; (1<<j) < n; j++) {
      int len = 1<<j;
      for (int c = len; c < n; c += 2*len) {
        m[j][c] = v[c], m[j][c-1] = v[c-1];
        for (int i = c+1; i <  c+len; i++) m[j][i] = op(m[j][i-1], v[i]);
        for (int i = c-2; i >= c-len; i--) m[j][i] = op(v[i], m[j][i+1]);
      }
    }
  }
  int query(int l, int r) {
    if (l == r) return v[l];
    int j = __builtin_clz(1) - __builtin_clz(l^r);
    return op(m[j][l], m[j][r]);
  }
}
```

3.2 Dsu

```
struct DSU {
    int n;
    vector<int> parent, size;

    DSU(int n): n(n) {
        parent.resize(n, 0);
        size.assign(n, 1);

        for(int i=0;i<n;i++)
            parent[i] = i;
    }

    int find(int a) {
        if(a == parent[a]) return a;
        return parent[a] = find(parent[a]);
    }

    void join(int a, int b) {
        a = find(a); b = find(b);
        if(a != b) {
            if(size[a] < size[b]) swap(a, b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};
```

3.3 Lis Dp

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

// permite que êvoc saiba se um únmero
// faz parte de alguma lis
// calcula pra direita e pra esquerda

// The problem of finding the maximum of a prefix of an array (which changes)
    is a
// standard problem that can be solved by many different data structures. For
    instance we can use a Segment tree or a Fenwick tree.

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long int
#define float long double
#define ld long double
#define ll long long
#define pb push_back
#define ff first
#define ss second
#define vi vector<int>
#define vpii vector<pair<int, int>>
#define vvi vector<vector<int>>
#define pii pair<int, int>
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define in(v) for(auto & x : v) cin >> x;
// #define out(v) for(auto x : v) cout << x << ' ';
#define tfii tuple<float, int, int>

const int MAXN = 4e5+1;
const int INF = INT32_MAX;
```

```cpp
const int MOD = 1e9+7;
const int LOG = 31;
const ld PI = acos(-1);
const int MINF = INT64_MIN;
vpii dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

// vc é bom basta calma
// rating is just a number
// leia a ãquesto inteira e com calma
// ãno olhe os standings
// çesquea o tempo
// é ós mais um virtual
// se divirta
// AAAAAAAAAAAAAAAAAAAAAAAAAAA

class SegTree{
    int n, elem_neutro = 0;
    vector<int> tree, lazy, v;

    int merge(int a, int b){
        return max(a, b); //seg de soma
    }

    void build(int l, int r, int no){
        if(l==r){
            tree[no] = v[l];
            return;
        }
        int mid = (l+r)/2;
        build(l, mid, 2*no);
        build(mid+1, r, 2*no+1);

        tree[no] = merge(tree[2*no], tree[2*no+1]);
    }

    void update(int A, int B, int x, int l, int r, int no){
        prop(l, r, no);
        if(B<l or r<A) return;
        if(A<=l and r<=B){
            lazy[no] = x; //update de soma
            prop(l, r, no);
            return;
        }
        int mid = (l+r)/2;

        update(A, B, x, l, mid, 2*no);
        update(A, B, x, mid+1, r, 2*no+1);

        tree[no] = merge(tree[2*no], tree[2*no+1]);
    }

    void prop(int l, int r, int no){
        if(lazy[no]!=0){
            tree[no] = lazy[no]; //update de soma
            if(l!=r){
                lazy[2*no] = lazy[no]; //update de soma
                lazy[2*no+1] = lazy[no]; //update de soma
            }
            lazy[no] = 0;
        }
    }

    int query(int A, int B, int l, int r, int no){
        prop(l, r, no);
        if(B<l or r<A) return elem_neutro;
        if(A<=l and r<=B) return tree[no];
        int mid = (l+r)/2;

        return merge(query(A, B, l, mid, 2*no),
                     query(A, B, mid+1, r, 2*no+1));
    }

    public:
        SegTree(vector<int> &v){
                this->n=v.size();
                this->v=v;
                tree.assign(4*n, 0);
                lazy.assign(4*n, 0);
                build(0, n-1, 1);
            }
        int query(int l, int r){return query(l, r, 0, n-1, 1);}
        void update(int l, int r, int val){update(l, r, val, 0, n-1, 1);}
        void out(){for(int i=0; i<n; i++){cout<<query(i, i)<<" ";cout<<endl;}}
};


void solve(){

    int n, last = 1; cin >> n;

    map<int, int> compr;

    vi ord(n);
    set<int> s;

    for(int i = 0; i < n; i++){

        cin >> ord[i];

        s.insert(ord[i]);
    }

    for(auto x : s){

        compr[x] = last;

        last++;
    }

    vi v(last+1, 0), dp(n);

    SegTree left(v), right(v);

    for(int i = 0; i < n; i++){

        int aux = left.query(0, compr[ord[i]]-1);
```

```
        dp[i] += aux;
        left.update(compr[ord[i]], compr[ord[i]], aux+1);
    }

    for(int i = n-1; i > -1; i--){

        int aux = right.query(compr[ord[i]]+1, last);
        dp[i] += aux;
        right.update(compr[ord[i]], compr[ord[i]], aux+1);
    }

    set<int> ans;

    int lis = right.query(0, last);

    for(int i = 0; i < n; i++){

        if(lis == (dp[i] + 1)) ans.insert(i+1);
    }

    cout << ans.size() << '\n';

    for(auto x : ans) cout << x << ' ';

    cout << '\n';

    return;
}
int32_t main(){
    sws;

    int t = 1;
    cin >> t;

    while(t--)
        solve();

    return 0;
}
```

3.4 Min Queue

```
struct MinQ {
    stack<pair<ll,ll>> in;
    stack<pair<ll,ll>> out;

    void add(ll val) {
        ll minimum = in.empty() ? val : min(val, in.top().ss);
        in.push({val, minimum});
    }

    ll pop() {
        if(out.empty()) {
            while(!in.empty()) {
                ll val = in.top().ff;
                in.pop();
                ll minimum = out.empty() ? val : min(val, out.top().ss);
                out.push({val, minimum});
            }
        }
        ll res = out.top().ff;
        out.pop();
        return res;
    }

    ll minn() {
        ll minimum = LLINF;
        if(in.empty() || out.empty())
            minimum = in.empty() ? (ll)out.top().ss : (ll)in.top().ss;
        else
            minimum = min((ll)in.top().ss, (ll)out.top().ss);

        return minimum;
    }

    ll size() {
        return in.size() + out.size();
    }
};
```

3.5 Mo

```
//Distinct values queries
#include <bits/stdc++.h>

using namespace std;
//#define int long long
#define pii pair<int,int>
#define ll long long
#define vi vector<int>
#define pb push_back
#define endl "\n"
#define input(x) for (auto &it : x) cin >> it;
#define output(x) for (auto &it : x) cout << it << ' ';
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define ff first
#define ss second

const long double PI = acos(-1);
int atual = 1;
int freq[200002];
struct Q
{
    int l,r, idx, block;
    Q(int p1, int p2, int i, int b)
    {
        l = p1;
        r = p2;
        idx = i;
        block = l/b;
    }
    bool operator < (Q& query2)
    {
        if(block == query2.block) return r < query2.r;
        return block < query2.block;
```

```
    }

};

void add(int x)
{
    if(!freq[x])
    {
        atual++;
    }
    freq[x]++;
}
void rem(int x)
{
    freq[x]--;
    if(!freq[x])
    {
        atual--;
    }
}
void solve()
{
    int n,q;
    cin >> n >> q;
    int b = sqrt(q) + 1;
    b = n/b + 1;
    vector <int> v(n);
    map<int,int> compress;
    vector<Q> queries;
    int aux = 1;
    for(int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        if(compress.count(x))
        {
            v[i] = compress[x];
        }
        else
        {
            compress[x] = aux;
            v[i] = aux;
            aux++;
        }
    }

    for(int i = 0; i < q; i++)
    {
        int x,y;
        cin >> x >> y;
        queries.pb(Q(x-1,y-1,i,b));
    }

    sort(queries.begin(), queries.end());
    vi ans(q,0);
    int curl = 0, curr = 0;
    freq[v[0]]++;
    for(auto query : queries)
    {
        //cout << query.l << ' ' << query.r << '\n';
        while(curl > query.l)
        {
            curl--;
            add(v[curl]);
        }
        while(curr < query.r)
        {
            curr++;
            add(v[curr]);
        }
        while(curl < query.l)
        {
            rem(v[curl]);
            curl++;
        }
        while(curr > query.r)
        {
            rem(v[curr]);
            curr--;
        }
        ans[query.idx] = atual;
    }

    for(auto resp : ans) cout << resp << '\n';


    return;
}
int32_t main()
{   sws
    int t = 1;
    while (t--)
    {
        solve();
    }

    return 0;
}
```

3.6 Ordered Set

```
// disable define int long long
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
  using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
  tree_order_statistics_node_update>;

// k-th maior elemento - O(logN) - idx em 0
s.find_by_order(k)

// qtd elementos < k - O(logN)
s.order_of_key(k)
```

```
ord_set<int> s;
```

3.7 Prefixsum 2d

```
ll find_sum(vector<vi> &mat, int x1, int y1, int x2, int y2){
    // superior-esq(x1,y1) (x2,y2)inferior-dir
    return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+mat[x1-1][y1-1];
}

int main(){

    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            mat[i][j]+=mat[i-1][j]+mat[i][j-1]-mat[i-1][j-1];

}
```

3.8 Rmq

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define endl "\n"
#define pb push_back
#define ff first
#define ss second
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
const int MAXN = 1e7+1;
const int INF = INT64_MAX;
const int MOD = 1e9+7;

void solve(){

    int n; cin >> n;

    int aux = n, log = 0;

    while(aux / 2){

        log++;
        aux /= 2;
    }

    vector<vector<int>> spt(n, vector<int> (log+1));
    vector<int> v(n);

    for(int i = 0; i < n; i++) {cin >> v[i]; spt[i][0] = v[i];}

    for(int j = 1; j < log+1; j++){
        for(int i = 0; i + (1 << j) -1 < n; i++){

            spt[i][j] = min(spt[i][j-1], spt[i + (1 << j-1)][j-1]);
        }
    }

    int q; cin >> q;

    while(q--){

        int l, r; cin >> l >> r;

        aux = r-l+1; log = 0;

        while(aux / 2){log++; aux /= 2;}

        cout << min(spt[l][log], spt[r - (1 << log) + 1][log]) << '\n';
    }
    return;
}
int32_t main(){
    sws;

    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

3.9 Segtree Lazy

```
class SegTree{
    int n, elem_neutro = -1;
    vector<int> tree, lazy, v;

    int merge(int a, int b){
        return max(a, b); //seg de soma
    }

    void build(int l, int r, int no){
        if(l==r){
            tree[no] = v[l];
            return;
        }
        int mid = (l+r)/2;
        build(l, mid, 2*no);
        build(mid+1, r, 2*no+1);

        tree[no] = merge(tree[2*no], tree[2*no+1]);
    }

    void update(int A, int B, int x, int l, int r, int no){
        prop(l, r, no);
        if(B<l or r<A) return;
        if(A<=l and r<=B){
            lazy[no] = x; //update de soma
            prop(l, r, no);
            return;
        }
        int mid = (l+r)/2;
```

```
        update(A, B, x, l, mid, 2*no);
        update(A, B, x, mid+1, r, 2*no+1);

        tree[no] = merge(tree[2*no], tree[2*no+1]);
    }

    void prop(int l, int r, int no){
        if(lazy[no]!=0){
            tree[no] = lazy[no]; //update de soma
            if(l!=r){
                lazy[2*no] = lazy[no]; //update de soma
                lazy[2*no+1] = lazy[no]; //update de soma
            }
            lazy[no] = 0;
        }
    }

    int query(int A, int B, int l, int r, int no){
        prop(l, r, no);
        if(B<l or r<A) return elem_neutro;
        if(A<=l and r<=B) return tree[no];
        int mid = (l+r)/2;

        return merge(query(A, B, l, mid, 2*no),
                     query(A, B, mid+1, r, 2*no+1));
    }

    public:
        SegTree(vector<int> &v){
                this->n=v.size();
                this->v=v;
                tree.assign(4*n, 0);
                lazy.assign(4*n, 0);
                build(0, n-1, 1);
            }
        int query(int l, int r){return query(l, r, 0, n-1, 1);}
        void update(int l, int r, int val){update(l, r, val, 0, n-1, 1);}
        void out(){for(int i=0; i<n; i++){cout<<query(i, i)<<" ";cout<<endl;}}
};
```

4 Geometria

4.1 Convex Hull

```
#include <bits/stdc++.h>

using namespace std;
#define int long long
typedef int cod;

struct point
{
    cod x,y;
    point(cod x = 0, cod y = 0): x(x), y(y)
    {}
    double modulo()
    {
        return sqrt(x*x + y*y);
    }

    point operator+(point o)
    {
        return point(x+o.x, y+o.y);
    }
    point operator-(point o)
    {
        return point(x - o.x , y - o.y);
    }
    point operator*(cod t)
    {
        return point(x*t, y*t);
    }
    point operator/(cod t)
    {
        return point(x/t, y/t);
    }

    cod operator*(point o)
    {
        return x*o.x + y*o.y;
    }
    cod operator^(point o)
    {
        return x*o.y - y * o.x;
    }
    bool operator<(point o)
    {
        if( x != o.x) return x < o.x;
        return y < o.y;
    }

};

int ccw(point p1, point p2, point p3)
{
    cod cross = (p2-p1) ^ (p3-p1);
    if(cross == 0) return 0;
    else if(cross < 0) return -1;
    else return 1;
}

vector <point> convex_hull(vector<point> p)
{
    sort(p.begin(), p.end());
    vector<point> L,U;

    //Lower
    for(auto pp : p)
    {
        while(L.size() >= 2 and ccw(L[L.size() - 2], L.back(), pp) == -1)
        {
            // é -1 pq eu ãno quero excluir os colineares
            L.pop_back();
```

```
        }
        L.push_back(pp);
    }

    reverse(p.begin(), p.end());

    //Upper
    for(auto pp : p)
    {
        while(U.size() >= 2 and ccw(U[U.size()-2], U .back(), pp) == -1)
        {
            U.pop_back();
        }
        U.push_back(pp);
    }

    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}

cod area(vector<point> v)
{
    int ans = 0;
    int aux = (int)v.size();
    for(int i = 2; i < aux; i++)
    {
        ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
    }
    ans = abs(ans);
    return ans;
}

int bound(point p1 , point p2)
{
    return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
}
//teorema de pick [pontos = A - (bound+points)/2 + 1]

int32_t main()
{

    int n;
    cin >> n;

    vector<point> v(n);
    for(int i = 0; i < n; i++)
    {
        cin >> v[i].x >> v[i].y;
    }

    vector <point> ch = convex_hull(v);

    cout << ch.size() <<  '\n';
    for(auto p : ch) cout << p.x << " " << p.y << "\n";

    return 0;
}
```

4.2 Inside Polygon

```
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

4.3 Minkowski

```
// Minkowski Sum
//
// Computa A+B = {a+b : a \in A, b \in B}, em que
// A e B sao poligonos convexos
```

```cpp
// A+B eh um poligono convexo com no max |A|+|B| pontos
//
// O(|A|+|B|)

vector<pt> minkowski(vector<pt> p, vector<pt> q) {
  auto fix = [](vector<pt>& P) {
    rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
    P.push_back(P[0]), P.push_back(P[1]);
  };
  fix(p), fix(q);
  vector<pt> ret;
  int i = 0, j = 0;
  while (i < p.size()-2 or j < q.size()-2) {
    ret.push_back(p[i] + q[j]);
    auto c = ((p[i+1] - p[i]) ^ (q[j+1] - q[j]));
    if (c >= 0) i = min<int>(i+1, p.size()-2);
    if (c <= 0) j = min<int>(j+1, q.size()-2);
  }
  return ret;
}

ld dist_convex(vector<pt> p, vector<pt> q) {
  for (pt& i : p) i = i * -1;
  auto s = minkowski(p, q);
  if (inpol(s, pt(0, 0))) return 0;
  ld ans = DINF;
  for (int i = 0; i < s.size(); i++) ans = min(ans,
      disttoseg(pt(0, 0), line(s[(i+1)%s.size()], s[i])));
  return ans;
}
```

4.4 Misc

```cpp
// PI = acos(-1)

// area de um poligono, retorna 2*area
int area(vector<point> &polygon) {
    int a = 0;

    rep(i, 0, tam(polygon)) {
        a += polygon[i] ^ polygon[(i+1)%tam(polygon)];
    }

    return abs(a);
}

// numero de lattice points em um segmento de reta (a,b) -> (c,d):
// gcd(c-a, d-b) + 1

// lattice points em um poligono
// com base no teorema de Pick
// retorna um pair onde ff == numero de pontos dentro e ss == numero de pontos
    na borda
pii lattice_points(vector<point> &polygon) {
    int in, sobre = 0;
    rep(i, 0, tam(polygon)) {
        point p = polygon[i], q = polygon[(i+1)%tam(polygon)];
        sobre += abs(gcd(q.x-p.x, q.y-p.y)) + 1;
    }
    sobre -= tam(polygon);

    in = (area(polygon) - sobre + 2)/2;
    return {in, sobre};
}

// numero de lattice points  na circunferencia
// r > 0
int lattice_circ(int r) {
    int ans = 4;

    rep(x, 1, r) {
        int ySquare = r*r - x*x;
        int y = sqrt(ySquare);

        if (y*y == ySquare) ans += 4;
    }

    return ans;
}

// numero de triangulos de lado 1 dentro de um hexagono regular
lado_tri = sides[0] + sides[1] + sides[2];
ans = lado_tri * lado_tri - sides[0] * sides[0] - sides[2] * sides[2] - sides
    [4] * sides[4];

// produto vetorial de 3 pontos (uso pra saber a pos relativa de um ponto em
    relacao a um segmento de reta)
int cross3(point s1, point s2, point p) {
    point vector1, vector2;

    vector1.x = s2.x - s1.x;
    vector1.y = s2.y - s1.y;

    vector2.x = p.x - s1.x;
    vector2.y = p.y - s1.y;

    return vector1 ^ vector2;
}

// verifica se um ponto esta sobre um segmento de reta, ja sabendo que esse
    ponto eh colinear
bool sobreSegmento(point s1, point s2, point p) {
    return p.x >= min(s1.x, s2.x) && p.x <= max(s1.x, s2.x) && p.y >= min(s1.y
    , s2.y) && p.y <= max(s1.y, s2.y);
}

// verifica se 2 segmentos de reta intersectam
bool intersect(point s1, point s2, point t1, point t2) {
    int t1_rel_s = cross3(s1, s2, t1);
    int t2_rel_s = cross3(s1, s2, t2);
    int s1_rel_t = cross3(t1, t2, s1);
    int s2_rel_t = cross3(t1, t2, s2);

    if (t1_rel_s != t2_rel_s && s1_rel_t != s2_rel_t)
        return true;
```

```
    if (t1_rel_s == 0 && sobreSegmento(s1, s2, t1))
        return true;

    if (t2_rel_s == 0 && sobreSegmento(s1, s2, t2))
        return true;

    if (s1_rel_t == 0 && sobreSegmento(t1, t2, s1))
        return true;

    if (s2_rel_t == 0 && sobreSegmento(t1, t2, s2))
        return true;

    return false;
}
```

4.5 Point Location

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long

#define pb push_back
#define ff first
#define ss second

const int MOD = 1e9+7;
const int MAX = 2e5+1;

int32_t main(){
    sws;

    int t; cin >> t;

    while(t--){

        int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

        int deltax1 = (x1-x2), deltay1 = (y1-y2);

        int compx = (x1-x3), compy = (y1-y3);

        int ans = (deltax1*compy) - (compx*deltay1);

        if(ans == 0){cout << "TOUCH\n"; continue;}
        if(ans < 0){cout << "RIGHT\n"; continue;}
        if(ans > 0){cout << "LEFT\n"; continue;}
    }
    return 0;
}
```

5 Grafos

5.1 Articulation Point

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define endl "\n"
#define int long long
#define ld long double
#define pb push_back
#define ff first
#define ss second
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
const int MAXN = 1e5+1;
const int INF = INT32_MAX;
const int MOD = 998244353;
const int LOG = 18;

vector<bool> vis(MAXN);
vector<vector<int>> g(MAXN);
vector<int> tin(MAXN, -1), low(MAXN, -1);
int t = 0;
set<int> ans;

void AP(int u, int p = -1){

    int qtdfilhos = 0;
    low[u] = tin[u] = t++;
    vis[u] = true;

    for(auto v: g[u]){

        if(v == p) continue;

        if(!vis[v]){

            qtdfilhos++;

            AP(v, u);

            low[u] = min(low[u], low[v]);

            if(low[v] >= tin[u] && u != 1) ans.insert(u);
        } else{

            low[u] = min(low[u], tin[v]);
        }
    }

    if(u == 1 && qtdfilhos >= 2) ans.insert(u);
}
void solve(){

    int n, m; cin >> n >> m;

    for(int i = 0; i < m; i++){

        int u, v; cin >> u >> v;
```

```
        g[u].pb(v);
        g[v].pb(u);
    }

    AP(1);

    cout << ans.size() << '\n';

    for(auto x : ans) cout << x << ' ';

    cout << '\n';

    return;
}
int32_t main(){
    sws;
    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

5.2 Bellman Ford

```
#include <bits/stdc++.h>

 // Calcula distancias com arestas negativas
//  Serve pra achar ciclo étambm
// Vetor de distancias da pripri
using namespace std;
#define int long long
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

const int MAXN = 2e5 + 1;
const int INF = 1e18+1;

vector<int> d(MAXN, -INF), points(MAXN, -1);
vector<bool> vis(MAXN);
vector<vector<int>> g(MAXN);
int n;

struct edge{

    int x, y, c;
};

vector<edge> edges;
void bf(int u){

    d[u] = 0;

    for(int i = 0; i < n - 1; i++){

        for(auto e : edges){

            if(d[e.x] > -INF){
                if(d[e.y] < d[e.x] + e.c){

                    d[e.y] = d[e.x] + e.c;
                }
            }
        }
    }
}

int dfs(int u){

    if(u == n) return points[u] = 1;

    vis[u] = true;

    int aux = -1;
    for(auto v : g[u]){

        if(!vis[v]) aux = dfs(v);
    }

    return points[u] = aux;
}

bool find(){

   for(int i = 0; i <= n - 1; i++){

        for(auto e : edges){

                if(d[e.y] < d[e.x] + e.c){

                    if(points[e.y] == 1) return true;
                    d[e.y] = d[e.x] + e.c;
                }
        }
    }

    return false;
}


int32_t main(){
    sws;

    int m; cin >> n >> m;

    for(int i = 0; i < m; i++){

        edge e; cin >> e.x >> e.y >> e.c;

        g[e.x].push_back(e.y);
        edges.push_back(e);
    }

    bf(1);
    dfs(1);
```

```
    if(find()){cout << -1 << '\n'; return 0;}

    cout << d[n] << '\n';
    return 0;
}
```

5.3 Bridgetree

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 2e5+1;
const int MOD = 1e9+7;
const int LOG = 30;

vector<bool> vis;
vector<int> tin, low, comp;
vector<vector<int>> g(MAX), bt(MAX);
map<pair<int, int>, bool> ponteh;
int time = 0;

void buildBt(int u, int c){

    comp[u] = c;
    vis[u] = true;

    for(auto v : g[u]){

        if(vis[v] || ponteh[{u, v}]) continue;

        buildBt(v, c);
    }
}

void findBridge(int u, int p = -1){

    vis[u] = true;
    tin[u] = low[u] = time++;

    for(auto v : g[u]){

        if(v == p) continue;

        if(vis[v]){

            low[u] = min(low[u], tin[u]);
        } else{

            dfs(v, u);
            low[u] = min(low[u], low[v]);

            if(low[v] > tin[u])
                éPonte(u, v);
        }
    }
}

int32_t main(){
    sws;

    for(auto[u, v]: ponteh){

        if(v){

            bt[comp[u.ff]].pb(comp[u.ss]);
            bt[comp[u.ss]].pb(comp[u.ff]);
        }
    }
    return 0;
}
```

5.4 Dfs Tree

```
int desce[N], sobe[N], vis[N], h[N];
int backedges[N], pai[N];

// backedges[u] = backedges que comecam embaixo de (ou =) u e sobem pra cima
    de u; backedges[u] == 0 => u eh ponte
void dfs(int u, int p) {
    if(vis[u]) return;
    pai[u] = p;
    h[u] = h[p]+1;
    vis[u] = 1;

    for(auto v : g[u]) {
        if(p == v or vis[v]) continue;
        dfs(v, u);
        backedges[u] += backedges[v];
    }
    for(auto v : g[u]) {
        if(h[v] > h[u]+1)
            desce[u]++;
        else if(h[v] < h[u]-1)
            sobe[u]++;
    }
    backedges[u] += sobe[u] - desce[u];
}
```

5.5 Dijktra

```
#define pii pair<int, int>
vector<vector<pii>> g(N);
vector<bool> used(N);
vector<ll> d(N, LLINF);
priority_queue< pii, vector<pii>, greater<pii> > fila;

void dijkstra(int k) {
    d[k] = 0;
    fila.push({0, k});

    while (!fila.empty()) {
```

```
        auto [w, u] = fila.top();
        fila.pop();
        if (used[u]) continue;
        used[u] = true;

        for (auto [v, w]: g[u]) {
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                fila.push({d[v], v});
            }
        }
    }
}
```

5.6 Dinic

```
// Dinitz do Bruno Maletta
struct dinitz {
  const bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
  int lim;                    // com constante alta
  struct edge {
    int to, cap, rev, flow;
    bool res;
    edge(int to_, int cap_, int rev_, bool res_)
      : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
  };

  vector<vector<edge>> g;
  vector<int> lev, beg;
  ll F;
  dinitz(int n) : g(n), F(0) {}

  void add(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size(), false);
    g[b].emplace_back(a, 0, g[a].size()-1, true);
  }
  bool bfs(int s, int t) {
    lev = vector<int>(g.size(), -1); lev[s] = 0;
    beg = vector<int>(g.size(), 0);
    queue<int> q; q.push(s);
    while (q.size()) {
      int u = q.front(); q.pop();
      for (auto& i : g[u]) {
        if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
        if (scaling and i.cap - i.flow < lim) continue;
        lev[i.to] = lev[u] + 1;
        q.push(i.to);
      }
    }
    return lev[t] != -1;
  }
  int dfs(int v, int s, int f = INF) {
    if (!f or v == s) return f;
    for (int& i = beg[v]; i < g[v].size(); i++) {
      auto& e = g[v][i];
      if (lev[e.to] != lev[v] + 1) continue;
      int foi = dfs(e.to, s, min(f, e.cap - e.flow));
      if (!foi) continue;
      e.flow += foi, g[e.to][e.rev].flow -= foi;
      return foi;
    }
    return 0;
  }
  ll max_flow(int s, int t) {
    for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
      while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
    return F;
  }
};

// Recupera as arestas do corte s-t
vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
  g.max_flow(s, t);
  vector<pair<int, int>> cut;
  vector<int> vis(g.g.size(), 0), st = {s};
  vis[s] = 1;
  while (st.size()) {
    int u = st.back(); st.pop_back();
    for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
      vis[e.to] = 1, st.push_back(e.to);
  }
  for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
    if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
  return cut;
}
```

5.7 Floyd

```
// Floyd Warshall

int dist[N][N];

for(int k = 1; k <= n; k++)
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

5.8 Ford Fulkerson Isa

```
#include <bits/stdc++.h>

using namespace std;
#define int long long
#define pb push_back

// Description:
// Obtains the maximum possible flow rate given a network. A network is a
    graph with a single source vertex and a single sink vertex in which each
    edge has a capacity

// Complexity:
// O(V * E^2) where V is the number of vertex and E is the number of edges
const int MAXN = 501;
const int MAXE = 1001;
const int INF = INT64_MAX;
```

```cpp
// represents the capacities of the edges
int capacity[MAXN][MAXE];
// represents the graph and it may contain negative edges
vector<int> adj[MAXN];
int n, e;


int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur])
        {
            //cout << "cur next " << cur << ' ' << next <<  ' ' << parent[next
    ] << ' ' << capacity[cur][next] << endl;
            if (parent[next] == -1 && capacity[cur][next])
            {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                {
                    //cout << new_flow << endl;
                    return new_flow;
                }
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n+1);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

```cpp
int32_t main()
{
    cin>>n>>e;
    int s = 1, t = n;
    //cin>>s>>t;

    for(int i = 0; i < e; i++)
    {
        int from, to, cap;
        cin>>from>>to>>cap;

        capacity[from][to] += cap;
        adj[from].push_back(to);
        //adding the negative edges
        adj[to].push_back(from);
    }

    // for(int i = 1; i <= n; i++)
    // {   cout << i << " : ";
    //     for(auto x : graph[i]) cout << x << ' ';
    //     cout << endl;
    // }

    int maxFlow = maxflow(s, t);

    cout<<maxFlow<<endl;

    return 0;
}
```

5.9 Kosaraju

```cpp
vector<int> g[N], gi[N]; // grafo invertido
int vis[N], comp[N]; // componente conexo de cada vertice
stack<int> S;

void dfs(int u){
    vis[u] = 1;
    for(auto v: g[u]) if(!vis[v]) dfs(v);
    S.push(u);
}

void scc(int u, int c){
    vis[u] = 1; comp[u] = c;
    for(auto v: gi[u]) if(!vis[v]) scc(v, c);
}

void kosaraju(int n){
    for(int i=0;i<n;i++) vis[i] = 0;
    for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
    for(int i=0;i<n;i++) vis[i] = 0;
    while(S.size()){
        int u = S.top();
        S.pop();
        if(!vis[u]) scc(u, u);
    }
}
```

5.10 Mcmf

```cpp
template<typename T> struct mcmf {
  struct edge {
    int to, rev, flow, cap; // para, id da reversa, fluxo, capacidade
    bool res; // se eh reversa
    T cost; // custo da unidade de fluxo
    edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
    edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
      : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_) {}
  };

  vector<vector<edge>> g;
  vector<int> par_idx, par;
  T inf;
  vector<T> dist;

  mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max()/3) {}

  void add(int u, int v, int w, T cost) { // de u pra v com cap w e custo cost
    edge a = edge(v, g[v].size(), 0, w, cost, false);
    edge b = edge(u, g[u].size(), 0, 0, -cost, true);

    g[u].push_back(a);
    g[v].push_back(b);
  }

  vector<T> spfa(int s) { // nao precisa se nao tiver custo negativo
    deque<int> q;
    vector<bool> is_inside(g.size(), 0);
    dist = vector<T>(g.size(), inf);

    dist[s] = 0;
    q.push_back(s);
    is_inside[s] = true;

    while (!q.empty()) {
      int v = q.front();
      q.pop_front();
      is_inside[v] = false;

      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;

          if (is_inside[to]) continue;
          if (!q.empty() and dist[to] > dist[q.front()]) q.push_back(to);
          else q.push_front(to);
          is_inside[to] = true;
        }
      }
    }
    return dist;
  }
  bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
      auto [d, v] = q.top();
      q.pop();
      if (dist[v] < d) continue;
      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        cost += pot[v] - pot[to];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;
          q.emplace(dist[to], to);
          par_idx[to] = i, par[to] = v;
        }
      }
    }
    return dist[t] < inf;
  }

  pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
    vector<T> pot(g.size(), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
      for (int i = 0; i < g.size(); i++)
        if (dist[i] < inf) pot[i] += dist[i];

      int mn_flow = flow - f, u = t;
      while (u != s){
        mn_flow = min(mn_flow,
          g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].flow);
        u = par[u];
      }

      ret += pot[t] * mn_flow;

      u = t;
      while (u != s) {
        g[par[u]][par_idx[u]].flow += mn_flow;
        g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
        u = par[u];
      }

      f += mn_flow;
    }

    return make_pair(f, ret);
  }

  // Opcional: retorna as arestas originais por onde passa flow = cap
  vector<pair<int,int>> recover() {
    vector<pair<int,int>> used;
    for (int i = 0; i < g.size(); i++) for (edge e : g[i])
      if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
    return used;
  }
```

```
};
```

5.11 Two Sat

```
// 2-SAT
//
// solve() retorna um par, o first fala se eh possivel
// atribuir, o second fala se cada variavel eh verdadeira
//
// O(|V|+|E|) = O(#variaveis + #restricoes)

struct sat {
  int n, tot;
  vector<vector<int>> g;
  vector<int> vis, comp, id, ans;
  stack<int> s;

  sat() {}
  sat(int n_) : n(n_), tot(n), g(2*n) {}

  int dfs(int i, int& t) {
    int lo = id[i] = t++;
    s.push(i), vis[i] = 2;
    for (int j : g[i]) {
      if (!vis[j]) lo = min(lo, dfs(j, t));
      else if (vis[j] == 2) lo = min(lo, id[j]);
    }
    if (lo == id[i]) while (1) {
      int u = s.top(); s.pop();
      vis[u] = 1, comp[u] = i;
      if ((u>>1) < n and ans[u>>1] == -1) ans[u>>1] = ~u&1;
      if (u == i) break;
    }
    return lo;
  }

  void add_impl(int x, int y) { // x -> y = !x ou y
    x = x >= 0 ? 2*x : -2*x-1;
    y = y >= 0 ? 2*y : -2*y-1;
    g[x].push_back(y);
    g[y^1].push_back(x^1);
  }
  void add_cl(int x, int y) { // x ou y
    add_impl(~x, y);
  }
  void add_xor(int x, int y) { // x xor y
    add_cl(x, y), add_cl(~x, ~y);
  }
  void add_eq(int x, int y) { // x = y
    add_xor(~x, y);
  }
  void add_true(int x) { // x = T
    add_impl(~x, x);
  }
  void at_most_one(vector<int> v) { // no max um verdadeiro
    g.resize(2*(tot+v.size()));
    for (int i = 0; i < v.size(); i++) {
      add_impl(tot+i, ~v[i]);
      if (i) {
        add_impl(tot+i, tot+i-1);
        add_impl(v[i], tot+i-1);
      }
    }
    tot += v.size();
  }

  pair<bool, vector<int>> solve() {
    ans = vector<int>(n, -1);
    int t = 0;
    vis = comp = id = vector<int>(2*tot, 0);
    for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i, t);
    for (int i = 0; i < tot; i++)
      if (comp[2*i] == comp[2*i+1]) return {false, {}};
    return {true, ans};
  }
};
```

6 Math

6.1 Fastexp

```
// recursivo
int fast_exp(int base, int e, int m){
    if(!e) return 1;
    int ans = fast_exp(base * base % m, e/2, m);
    if(e % 2) return base * ans % m;
    else return ans;
}
//iterativo
int fast_exp(int base, int e, int m) {
  int ret = 1;
  while (e) {
    if (e & 1) ret = (ret * base) % m;
    e >>= 1;
    base = (base * base) % m;
  }
  return ret;
}
```

6.2 Fft

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define pii pair<int,int>
#define ll long long
#define vi vector<int>
#define vvi vector<vector<int>>
#define pb push_back
#define all(x) x.begin(), x.end()
#define endl "\n"
#define ff first
#define ss second
#define input(x) for (auto &it : x) cin >> it;
```

```cpp
#define output(x) for (auto &it : x) cout << it << ' ';
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

const int INF = INT64_MAX;
const long double PI = acos(-1);
const int MAX = (1e6) + 1;
const int MOD = 998244353;
const int LOG = 30;

using cd = complex<double>;

// FFT (usei na H da mineira de 2024 de contar os quadrados)

void fft(vector<cd> &A, bool invert) {
  int N = size(A);

  for (int i = 1, j = 0; i < N; i++) {
    int bit = N >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(A[i], A[j]);
  }

  for (int len = 2; len <= N; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < N; i += len) {
      cd w(1);
      for (int j = 0; j < len/2; j++) {
        cd u = A[i+j], v = A[i+j+len/2] * w;
        A[i+j] = u + v;
        A[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }

  if (invert) {
    for (auto &x : A)
      x /= N;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const& B) {
  vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < size(A) + size(B))
    N <<= 1;
  fa.resize(N);
  fb.resize(N);

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

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



void solve()
{
  vector<int> A(MAX,0);
  vector<int> B(MAX,0);

  int n;
  cin >> n;

  for(int i = 0; i < n; i++)
  {
    int x;
    cin >> x;
    // A com os expoentes positivos e B com os expoentes negativos
    A[x] = 1;
    //A[i] é o coeficiente de z^i
    B[MAX-1-x] = 1;
  }

  // MAX-1 é o novo "0"

  //multiply me da o resultado da çãmultiplicao desses dois ôpolinmios
  // C[i] é o coeficiente de x^i

  vector<int> C = multiply(A,B);

  return;
}



int32_t main()
{
    sws

    int t = 1;
    //cin >> t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

6.3 Inverso Mult

```cpp
// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
```

```
ll inv(ll a, ll m) { // com gcd
  ll x, y;
  gcd(a, m, x, y);
  return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo entao phi(m) = p-1
  ll e = phim-1;
  return fexp(a, e);
}
```

6.4 Matrix Exp

```
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident) {
            for(int i = 0; i < min(r, c); i++) {
                m[i][i] = 1;
            }
        }
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++) {
            for(int k = 0; k < c; k++) {
                for(int j = 0; j < o.c; j++) {
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % MOD;
                }
            }
        }

        return Matrix(res);
    }
};

Matrix fexp(Matrix b, int e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}
```

6.5 Mulmod

```
ll mulmod(ll a, ll b) {
    if(a == 0) {
        return 0LL;
    }
    if(a%2 == 0) {
        ll val = mulmod(a/2, b);
        return (val + val) % MOD;
    }
    else {
        ll val = mulmod((a-1)/2, b);
        val = (val + val) % MOD;
        return (val + b) % MOD;
    }
}
```

6.6 Mult Matriz

```
    for(int i=0; i<n; i++) {
        aux_ab=0, aux_ba=0;
        for (int j=0; j<n; j++){
            aux_ab+= A[i][j]*B[j][i];
            aux_ba+= B[i][j]*A[j][i];
        }
        if (aux_ab!=aux_ba){
            val = false;
            break;
        }
    }
```

7 Misc

7.1 Bitwise

```
// Least significant bit (lsb)
    int lsb(int x) { return x&-x; }
    int lsb(int x) { return __builtin_ctz(x); } // bit position
// Most significant bit (msb)
    int msb(int x) { return 32-1-__builtin_clz(x); } // bit position

// Power of two
    bool isPowerOfTwo(int x){ return x && (!(x&(x-1))); }

// floor(log2(x))
int flog2(int x)  { return 32-1-__builtin_clz(x); }
int flog2ll(ll x) { return 64-1-__builtin_clzll(x); }

// Built-in functions
// Number of bits 1
__builtin_popcount()
__builtin_popcountll()

// Number of leading zeros
__builtin_clz()
__builtin_clzll()
```

```
// Number of trailing zeros
__builtin_ctz()
__builtin_ctzll()
```

7.2 Template

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long int
#define float long double
#define ld long double
#define ll long long
#define pb push_back
#define ff first
#define ss second
#define vi vector<int>
#define vpii vector<pair<int, int>>
#define vvi vector<vector<int>>
#define pii pair<int, int>
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define in(v) for(auto & x : v) cin >> x;
#define out(v) for(auto x : v) cout << x << ' ';
#define tfii tuple<float, int, int>
#define rep(a,b,c) for(int a = (int)b ; a < (int)c ; a++)
#define repi(a,b,c) for(int a = (int)b ; a >= (int)c ; a--)
#define tam(x) ((int)x.size())
#define endl '\n'

const int MAXN = 31700;
const int INF = INT64_MAX;
const int MOD = 1e9+7;
const int LOG = 31;
const ld PI = acos(-1);
const int MINF = INT64_MIN;
vpii dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

void solve(){


    return;
}
int32_t main(){
    sws;

    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

7.3 Test

```
#!/bin/bash

# to color the output text in different colours
green=$(tput setaf 71);
red=$(tput setaf 1);
blue=$(tput setaf 32);
orange=$(tput setaf 178);
bold=$(tput bold);
reset=$(tput sgr0);

// You can change the version of C++ or add the compiler flags you wish
g++ -std=c++17 gen.cpp -o generator || { echo ${bold}${orange}Compilation
    Error in ${reset} gen.cpp; exit 1; }
g++ -std=c++17 $1.cpp -o original || { echo ${bold}${orange}Compilation Error$
    {reset} in $1.cpp; exit 1; }
g++ -std=c++17 $2.cpp -o brute || { echo ${bold}${orange}Compilation Error${
    reset} in $2.cpp; exit 1; }

if [ $# -eq 2 ]
    then
        max_tests=10 # default number of test cases | change it accordingly
    else
        max_tests=$3
fi

diff_found=0
i=1

while [ $i -le $max_tests ]
do
    # Generate test_case and save it in input1.txt
    ./generator > input1.txt

    # run original solution, take input from above generated test case i.e.
    from input1.txt
    # and save it in original_output.txt
    ./original < input1.txt > original_output.txt #|| {echo failed; exit 1;}

    # run brute force solution, take input from above generated test case i.e.
     from input1.txt
    # and save it in brute_output.txt
    ./brute < input1.txt > brute_output.txt
    #python3 brute.py < input1.txt > brute_output.txt

    # check if files original_output and brute_output
    # differs(we are ignoring spaces and then comparing files)
    if diff --tabsize=1 -F --label --side-by-side --ignore-space-change
    original_output.txt brute_output.txt > dont_show_on_terminal.txt; then
        echo "${orange}test_case #$i: ${bold}${green}passed${reset}"
    else
        echo "${orange}test_case #$i: ${bold}${red}failed${reset}"
        diff_found=1
        break
    fi
    i=$((i+1))
done
```

```
if [ $diff_found -eq 1 ]
then
    echo "${blue}Input: ${reset}"
    cat input1.txt
    echo ""

    echo "${blue}Output: ${reset}"
    cat original_output.txt
    echo ""

    echo "${blue}Expected: ${reset}"
    cat brute_output.txt
    echo ""

    notify-send "Wrong Answer"
else
    notify-send "Accepted"
fi

#rm input1.txt
rm generator
rm original
rm brute
rm original_output.txt
rm brute_output.txt
rm dont_show_on_terminal.txt

# to run do: bash test.sh <brute file> <solution file>
# flags to compile code: g++ <file name>.cpp -fsanitize=address,undefined -fno
    -omit-frame-pointer -g -Wall -Wshadow -std=c++17 -Wno-unused-result -Wno-
    sign-compare -Wno-char-subscripts -o <file name>
```

7.4 Xorbasis

```
/**
 * Author: Wallace
 * Date: 11/08/2024
 * Description: Xor Basis
 * Time: O(size(base)) = O(\log{mx\_val});
 * Status: Tested with lots of problems
*/

struct XorBasis {
    vector<ll> B; // basis

    ll reduce(ll vec) {
        for(auto b : B) vec = min(vec, vec^b);
        return vec;
    }

    void add(ll vec) {
        ll val = reduce(vec);
        if (val) B.pb(val);
    }
};

// Extended //
struct XorBasis {
    vector<ll> B; // Basis

    ll dim() { return B.size(); } // O(1)

    ll reduce(ll vec) { // O(log(a_max))
        for(auto b : B) vec = min(vec, vec^b);
        return vec;
    }

    bool add(ll vec) { // O(log(a_max))
        ll val = reduce(vec);
        if (val) {
            B.pb(val);
            return true;
        }
        return false;
    }

    ll mxVal() { // O(log(a_max))
        ll mx = 0;
        for(auto b : B) mx = max(mx, mx^b);
        return mx;
    }

    void gaussJordan() { // O(log(a_max)^2)
        sort(B.begin(), B.end(), greater<ll>());
        for(ll i=1; i<(ll)B.size(); i++) {
            for(ll j=0; j<i; j++) {
                B[j] = min(B[j], B[j]^B[i]);
            }
        }
    }
};

// Problem description: (Ivan and Burgers)
// given a static array x[1, N]
// for each query, answer then max xor-sum of any subset in subarray [L, R]
// Contrains: 1 <= L <= R <= N, N <= 5e5, Q <= 5e5

// Probably the complexity is O(N log^2(N) + Q)
// Similarly, we can answer other type of queries related to xor-basis,
// because we will have it computed (Atcoder: H - Xor Query)
int32_t main(){ sws;
    ll n; cin >> n;

    vector<ll> x(n+1);
    for(ll i=1; i<=n; i++) {
        cin >> x[i];
    }

    vector<vector<pll>> queries(n+1);
    ll q; cin >> q;
    for(ll i=1; i<=q; i++) {
        ll l, r; cin >> l >> r;
        queries[r].pb({l, i});
    }
```

```
    vector<XorBasis> xb(n+1); // extended version of XorBasis
    vector<ll> ans(q+1);

    for(ll r=1; r<=n; r++) {

        // O(de bom), maybe log?
        for(ll l=r; l>=1; l--) {
            if (!xb[l].add(x[r])) break;
            // We can break here, because this xor-basis of L already contains
     a basis that doesn't need x[r]
            // Therefore, the xor-basis of L-1, L-2, ..., which contains the
    xor-basis of L, also doesn't need x[r]
        }

        // solve all queries ending in r,
        // knowing that all xor-basis are computed up to r.
        for(auto [left, i] : queries[r]) {
            ans[i] = xb[left].mx;
        }
    }

    for(ll i=1; i<=q; i++) {
        cout << ans[i] << endl;
    }
}
```

8 QuestoesCSES

8.1 Alex And Complicated Task

```
#include <bits/stdc++.h>
using namespace std;
// o problema tinha como objetivo encontrar quatro uplas alternadas
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define endl "\n"
#define ld long double
#define pb push_back
#define ff first
#define ss second
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
const int MAXN = 5e5+2;
const int INF = INT32_MAX;
const int MOD = 1e9+7;
const int LOG = 60;

vector<int> v(MAXN, INF);
vector<int> Ar(4*MAXN);
vector<int> best_4upla(MAXN, INF);
int n;

void build(int id, int il, int ir){ //Por ser uma arvore os filhos ãso
    tratados com 2*pai+1 ou somente 2*pai
// Os intervalos ãso tratados pela ãdiviso na metade dos intervalos
    if(il == ir){ //Ent chegamos a um intervalo unico

        Ar[id] = v[il];
        return;
    }

    int im = (il + ir) / 2; // çTraa-se o meio para definir o novo intervalo
    de çãconstruo

    build(2*id, il, im);
    build(2*id + 1, im + 1, ir);      //desce na arvore para definir o valor
    dos filhos

    Ar[id] = min(Ar[2*id], Ar[2*id+1]);     //define o nodo atual

    return;
}

int query(int l, int r, int id, int il, int ir){

    if(il >= l && ir <= r) return Ar[id];  //Esse il e ir se encontram dentro
    do meu alvo
    if(r < il || l > ir) return INF;      //ãNo se encontram

    int im = (ir + il) / 2;

    int esq = query(l, r, 2*id, il, im);
    int dir = query(l, r, 2*id+1, im+1, ir);

    return min(esq, dir);
}

void update(int idx, int x, int id, int il, int ir){

    if(il == ir){

        Ar[id] = x;
        v[idx] = x;
        return;
    }

    int im = (il + ir) / 2;

    if(im < idx){

        update(idx, x, 2*id+1, im + 1, ir);
    } else{

        update(idx, x, 2*id, il, im);
    }

    Ar[id] = min(Ar[2*id + 1], Ar[2*id]);

    return;
}

vector<int> memo(MAXN, -1);

int dp(int x){

    if(x > n) return memo[x] = 0;
```

```
    if(memo[x] != -1) return memo[x];

    int aux = 0;

    aux = max(aux, dp(x+1));
    if(best_4upla[x] != INF) aux = max(aux, dp(best_4upla[x]+1)+1);

    return memo[x] = aux;
}

void solve(){

    build(1, 1, MAXN-1);

    cin >> n;

    vector<int> nums(n+2), ante(n+2, -1), poste(n+2, -1);

    map<int, vector<int>> ids;

    set<int> aux;

    map<int, int> last;

    for(int i = 1; i <= n; i++){

        int x; cin >> x;
        nums[i] = x;
        if(last.count(x)){

            ante[i] = last[x];
            poste[last[x]] = i;
        }

        aux.insert(x);
        ids[x].pb(i);

        last[x] = i;
    }


    for(auto i : aux){

        for(int j = 0; j + 3 < (int)(ids[i].size()); j++){

            best_4upla[ids[i][j]] = ids[i][j+3];
        }
    }

    for(int i = n; i > 0; i--){

        if(ante[i] == -1) continue;

        best_4upla[ante[i]] = min(best_4upla[ante[i]], query(ante[i]+1, i-1,
    1, 1, MAXN-1));

        update(ante[i], i, 1, 1, MAXN-1);
    }

    int ans = dp(1);

    cout << 4*ans << '\n';

    vector<int> idxs;

    int i = 1;
    while(i <= n && ans){
        if(best_4upla[i] != INF && memo[best_4upla[i]+1] == ans-1){

            idxs.pb(i);
            idxs.pb(best_4upla[i]);
            ans--;
            i = best_4upla[i]+1;

        } else if(memo[i+1] == ans){

            i++;
        }
    }

    for(int i = 0; i < (int)(idxs.size())-1; i+=2){

        cout << nums[idxs[i]] << ' ' << nums[idxs[i+1]] << ' ' << nums[idxs[i
    ]] << ' ' << nums[idxs[i+1]] << ' ';
    }

    cout << '\n';

    return;
}
int32_t main(){
    sws;

    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

8.2 Bracketsequence

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define esp  ' '
#define int long long int
#define pii pair<int, int>
#define pb push_back
#define ff first
#define ss second
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);cout.tie(nullptr);
const string YES = "YES";
const string NO = "NO";
```

```
const int MAX= 2e6+5;
const int MOD= 1e9+7;
const int INF = 0x3f3f3f3f3f3f3f3f;
int fat[MAX], C[MAX];

int fexp(int b, int e){
    if (e==0) return 1;

    int ans = fexp(b, e/2);
    if(e%2) return (((ans*ans)%MOD)*b)%MOD;
    else return (ans*ans)%MOD;
}

void fluminense(){
    int  n; cin >> n;
    int ans = 0;
    if(n%2==1) ans=0;
    else{
        n = n>>1;
        ans = C[n];
    }

    cout << ans <<endl;


}

int32_t main(){
    sws;
    fat[0]=1;
    for(int i=1; i<MAX; i++) fat[i] = (i*fat[i-1])%MOD;
    for(int i=0; i<(MAX>>1)-1; i++){
        C[i] = ((((fat[2*i]*(fexp(fat[i], MOD-2)%MOD))%MOD*(fexp(fat[i], MOD
    -2)%MOD)))%MOD)*(fexp(i+1, MOD-2)%MOD);
        C[i]%=MOD;
    }


    int T=1;
    //cin >> T;
    while(T--)fluminense();

}
```

8.3 Editdistance

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long int
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 5e3+1;

vector<vector<int>> memo(MAX, vector<int> (MAX, -1));
string s, t;

int dp(int i, int j){

    if(i == -1) return j+1;
    if(j == -1) return i+1;

    if(memo[i][j] != -1) return memo[i][j];

    int ins = dp(i-1, j) + 1;
    int del = dp(i, j-1) + 1;
    int mod = dp(i-1, j-1) + (s[i] != t[j]);

    int aux = min(del, mod);
    return memo[i][j] = min(ins, aux);
}
int32_t main(){
    sws;

    cin >> s >> t;

    cout << dp(s.size()-1, t.size()-1) << '\n';
    return 0;
}
```

8.4 Generate Strings

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define endl "\n"
#define pb push_back
#define ff first
#define ss second
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
const int MAXN = 1e7+1;
const int INF = INT64_MAX;
const int MOD = 1e9+7;

void solve(){

    int n, x, y; cin >> n >> x >> y;

    vector<int> dp(2*(n+1), INF);

    dp[0] = 0;

    for(int i = 1; i <= n; i++){

        dp[i] = min(dp[i-1]+x, dp[i]);
        if(!(i%2)) dp[i] = min(dp[i / 2]+y, dp[i]);
        else dp[i] = min(dp[(i+1)/2]+x+y, dp[i]);
    }
    cout << dp[n] << '\n';

    return;
}
```

```
int32_t main(){
    sws;

    int t = 1;
    // cin >> t;

    while(t--)
        solve();

    return 0;
}
```

8.5 Multtable

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

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define endl "\n"
#define pb push_back
#define all(x) x.begin(), x.end()

typedef long long ll;
typedef long double ld;
const ll MOD = 1e9+7;
const int MAX = 1e6+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const int LOG = 21;

int k, n;

bool check(int x){

    int aux = 0;

    for(int i = 1; i <= n; i++){

        aux += min(n, x / i);
    }

    return (aux < k);
}

void solve(){

    cin >> n;

    k = (n*n+1)/2;

    int l = 1, r = n*n, ans = 0;

    while(l <= r){

        int mid = (l+r) / 2;

        if(check(mid)){

            l = mid+1;
        } else{

            ans = mid;
            r = mid-1;
        }
    }

    cout << ans << '\n';
    return;
}
int32_t main(){
    sws;

     int t = 1;
    //  cin >> t;

    while(t--)
        solve();

    return 0;
}
```

8.6 Prefixsumqueries

```
#include <bits/stdc++.h>

using namespace std;
#define int long long
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

const int MAXN = 2e5 + 1;
const int INF = 1e18+1;
vector<int> v(MAXN, 0), t(4*MAXN), lazy(4*MAXN), aux(MAXN);

int merge(int x, int y){

    return max(x, y);
}
void prop(int id, int il, int ir){

    if(!lazy[id]) return;

    if(il != ir){

        lazy[2*id] += lazy[id];
        lazy[2*id+1] += lazy[id];
    }

    t[id] += lazy[id];
    lazy[id] = 0;

    return;
}

void build(int id, int il, int ir){

    if(il == ir){
```

```
        t[id] = v[il];
        return;
    }

    int im = (il + ir) >> 1;

    build(2*id, il, im);
    build(2*id+1, im+1, ir);

    t[id] = merge(t[2*id], t[2*id+1]);

    return;
}

void update(int id, int il, int ir, int l, int r, int x){

    prop(id, il, ir);
    if(l <= il && ir <= r){

        lazy[id] += x;
        prop(id, il, ir);
        return;
    }
    if(l > ir || il > r) return;

    int im = (ir+il) >> 1;

    update(2*id, il, im, l, r, x);
    update(2*id+1, im+1, ir, l, r, x);

    t[id] = merge(t[2*id+1], t[2*id]);
}

int query(int id, int il, int ir, int l, int r){

    prop(id, il, ir);
    if(l <= il && ir <= r) return t[id];
    if(l > ir || il > r) return -INF;

    int im = (ir+il) >> 1;

    int esq = query(2*id, il, im, l, r);
    int dir = query(2*id+1, im+1, ir, l, r);

    return merge(esq, dir);
}

int32_t main(){
    sws;

    int n, q; cin >> n >> q;

    for(int i = 1; i <= n; i++){

        cin >> aux[i];

        v[i] = v[i-1] + aux[i];
    }

    build(1, 0, n);

    while(q--){

        int t, l, r; cin >> t >> l >> r;

        if(t == 2){

            cout << query(1, 0, n, l-1, r) - query(1, 0, n, l-1, l-1) << '\n';
        } else {

            update(1, 0, n, l, n, r-aux[l]);

            aux[l] = r;
        }
    }

    return 0;
}
```

8.7 Reachabilityqueries

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 5e4+1;
const int MOD = 1e9+7;

vector<int> g[MAX], ginv[MAX], gscc[MAX];
vector<int> scc(MAX);
vector<bool> vis(MAX), visscc(MAX);
vector<bitset<MAX>> ans(MAX);

stack<int> s;

void topo(int u){

    vis[u] = true;

    for(int v : g[u]) if(!vis[v]) topo(v);

    s.push(u);

    return;
}

void gsscc(int u, int c){

    scc[u] = c;

    for(auto v : ginv[u]){
```

```
        if(!scc[v]) gsscc(v, c);
        else gscc[scc[v]].push_back(scc[u]);
    }
}

bitset<MAX> dfs(int u){

    visscc[u] = true;
    ans[u].set(u);

    for(auto v : gscc[u]){

        if(!visscc[v]) dfs(v);

        ans[u] |= ans[v];
    }

    return ans[u];
}
int32_t main(){
    sws;

    int n, m, q; cin >> n >> m >> q;

    for(int i = 0; i < m; i++){

        int u, v; cin >> u >> v;

        g[u].push_back(v);
        ginv[v].push_back(u);
    }

    int comp = 1;

    for(int i = 1; i <= n; i++)
        if(!vis[i]) topo(i);

    while(!s.empty()){
        int u;
        tie(u) = s.top(); s.pop();

        if(!scc[u]){

            // cout << comp << '\n';
            gsscc(u, comp);
            comp++;
        }
    }

    for(int i = 1; i <= n; i++)
        if(!visscc[i]) dfs(i);

    while(q--){

        int u, v; cin >> u >> v;

        if(ans[scc[u]][scc[v]]) cout << "YES\n";
        else cout << "NO\n";

    }

    return 0;
}
```

8.8 Rectanglecutting

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long int
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 5e3+1;

vector<vector<int>> memo(MAX, vector<int> (MAX, -1));
int l, c;

int dp(int x, int y){

    if(x == y) return 0;

    if(memo[x][y] != -1) return memo[x][y];

    int aux = INF, aux1 = -INF, aux2 = -INF;

    for(int i = 1; i <= x/2 ; i++){

        aux1 = dp(i, y) + dp(x-i, y)+1;
        aux = min(aux, aux1);
    }

    for(int i = 1; i <= y/2 ; i++){

        aux2 = dp(x, i) + dp(x, y-i)+1;
        aux = min(aux, aux2);
    }

    return memo[x][y] = aux;
}
int32_t main(){
    sws;

    cin >> l >> c;

    int ans = dp(l, c);
    cout << ans << '\n';
    return 0;
}
```

8.9 Removalgame

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

```cpp
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 5e3+1;

int memo[MAX][MAX][2];
vector<int> v(MAX);

int dp(int l, int r, bool w){

    if(l > r) return 0;
    if(memo[l][r][w] != -1) return memo[l][r][w];

    if(w){

        int aux = dp(l+1, r, !w);
        int aux1 = dp(l, r-1, !w);
        return memo[l][r][w] = min(aux, aux1);
    } else{

        int aux = dp(l+1, r, !w) + v[l];
        int aux1 = dp(l, r-1, !w) + v[r];
        return memo[l][r][w] = max(aux, aux1);
    }
}
int32_t main(){
    sws;

    int n; cin >> n;

    memset(memo, -1, sizeof(memo));
    for(int i = 0; i < n; i++) cin >> v[i];

    cout << dp(0, n-1, 0) << '\n';

    return 0;
}
```

8.10 Sintaxenextperm

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
int32_t main(){

    string s; cin >> s;

    vector<char> c;

    for(int i = 0; i < s.size(); i++) c.push_back(s[i]);

    set<string> se;
    sort(c.begin(), c.end());

    string resp = "";
    for(int i = 0; i < s.size(); i++) resp += c[i];

    se.insert(resp);

    while(next_permutation(c.begin(), c.end())){

        string resp = "";
        for(int i = 0; i < s.size(); i++) resp += c[i];

        se.insert(resp);
    }

    cout << se.size() << '\n';

    for(auto t: se) cout << t << '\n';
    return 0;
}
```

9 Strings

9.1 Aho

```cpp
// Trie + KMP

struct Aho {
    public:
        int n;
        vector<vector<int>> adj, nxt;
        vector<int> lnk, lnkt;
        vector<int> terms;

        Aho(): n(1) {
            adj.emplace_back(26, -1);
            terms.pb(0);
        }

        int add_str(string &s) {
            int cur = 0;
            for (char c : s) {
                int& prox = adj[cur][c-'a'];

                if (prox == -1) {
                    prox = n;
                    cur = n;

                    adj.emplace_back(26, -1);
                    terms.pb(0);
                    n++;
                } else
                    cur = prox;
            }

            terms[cur]++;
            return cur;
        }

        void build() {
            int v, prox;
            lnkt.assign(n, 0);
            lnk.assign(n, 0);
```

```
            nxt.assign(n, vector<int>(26, 0));

            queue<int> q;
            q.push(0);
            while (!q.empty()) {
                v = q.front();
                q.pop();

                rep(i, 0, 26) {
                    prox = adj[v][i];
                    if (prox != -1) {
                        lnk[prox] = v ? nxt[lnk[v]][i] : 0;
                        lnkt[prox] = terms[lnk[prox]] ? lnk[prox] : lnkt[lnk[
    prox]];
                        q.push(prox);
                        nxt[v][i] = prox;
                    } else
                        nxt[v][i] = nxt[lnk[v]][i];
                }
            }
        }
};
```

9.2 Hash

```
inline int add(int a, int b, int mod){a+=b;if(a>=mod)a-=mod;return a;}
inline int sub(int a, int b, int mod){a-=b;if(a<0)a+=mod;return a;}
inline int mul(int a, int b, int mod){return (a*b)%mod;}

const int mod1 = 1000015187;
const int mod2 = 1000027957;
// outros primos: 1000041323, 1000015553, 1000028537, 50331653
// mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());
    // random number
// const int base = uniform_int_distribution<int>(356, mod1-1)(rng); //
    alphabet < base < lowest_mod
const int base = 31;
struct HS {
    int n;
    vector<int> s, t, h1, h2, hi1, hi2, p1, p2;
    template<typename T>
    HS(T x) : n(x.size()), s(x.begin(), x.end()), t(x.rbegin(), x.rend()),
              h1(n), h2(n), hi1(n), hi2(n), p1(n), p2(n) {
        // evita ter 0 no vetor
        for (auto& it : s) it += 1;
        for (auto& it : t) it += 1;
        p1[0] = p2[0] = 1;
        h1[0] = h2[0] = s[0];
        hi1[0] = hi2[0] = t[0];
        for (int i = 1; i < n; ++i) {
            p1[i] = mul(base, p1[i-1], mod1);
            p2[i] = mul(base, p2[i-1], mod2);

            h1[i] = add(mul(base, h1[i-1], mod1), s[i], mod1);
            h2[i] = add(mul(base, h2[i-1], mod2), s[i], mod2);

            hi1[i] = add(mul(base, hi1[i-1], mod1), t[i], mod1);
            hi2[i] = add(mul(base, hi2[i-1], mod2), t[i], mod2);
        }
    }

    int query(int l, int r, bool inv = false) const {
        const auto& hs1 = inv ? hi1 : h1;
        const auto& hs2 = inv ? hi2 : h2;
        int h1_val = (l == 0) ? hs1[r] : sub(hs1[r], mul(hs1[l-1], p1[r-l+1],
    mod1), mod1);
        int h2_val = (l == 0) ? hs2[r] : sub(hs2[r], mul(hs2[l-1], p2[r-l+1],
    mod2), mod2);
        return h1_val ^ (h2_val << 29);
    }

    int query_inv(int a, int b) const {
        return query(n - b - 1, n - a - 1, true);
    }
};


// ########################
// # úMltiplos MODS:
// # Aumenta a complexidade!!
// ########################
const int base = 31;
// pode adicionar mais mods
vector<int> mods = {1000015187, 1000027957, 1000041323};
struct HS {
    vector<int> s, t;
    int n, m;
    vector<vector<int>> h, hi, p;
    template<typename T>
    HS(T x) : n(x.size()), m(mods.size()),
              s(x.begin(), x.end()), t(x.rbegin(), x.rend()),
              h(m, vector<int>(n)), hi(m, vector<int>(n)), p(m, vector<
    int>(n)) {
        for (auto& it : s) it += 1;
        for (auto& it : t) it += 1;
        for (int i = 0; i < m; ++i) {
            p[i][0] = 1;
            h[i][0] = s[0];
            hi[i][0] = t[0];
            for (int j = 1; j < n; ++j) {
                p[i][j] = mul(base, p[i][j-1], mods[i]);
                h[i][j] = add(mul(base, h[i][j-1], mods[i]), s[j], mods[i]);
                hi[i][j] = add(mul(base, hi[i][j-1], mods[i]), t[j], mods[i]);
            }
        }
    }
    vector<int> query(int l, int r, bool inv = false) const {
        vector<int> result(m);
        for (int i = 0; i < m; ++i) {
            const auto& hs = inv ? hi[i] : h[i];
            result[i] = (l == 0) ? hs[r] : sub(hs[r], mul(hs[l-1], p[i][r-l
    +1], mods[i]), mods[i]);
        }
        return result;
    }
    vector<int> query_inv(int a, int b) const {
```

```
        return query(n - b - 1, n - a - 1, true);
    }
};

 // çãimplementao do Maxwell e do seu time, potencialmente util
 // pois consegui entender melhor

 struct Hash {
    ll MOD, P;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s, ll MOD, ll P = 31): s(s), MOD(MOD), P(P), n(s.size()), h(n)
    , hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};

struct DoubleHash {
    const ll MOD1 = 90264469;
    const ll MOD2 = 25699183;

    Hash hash1, hash2;

    DoubleHash();

    DoubleHash(string s) : hash1(s, MOD1), hash2(s, MOD2) {}

    pair<int, int> query(int l, int r) {
        return { hash1.query(l, r), hash2.query(l, r) };
    }

    pair<int, int> query_inv(int l, int r) {
        return { hash1.query_inv(l, r), hash2.query_inv(l, r) };
    }
};

struct TripleHash {
    const ll MOD1 = 90264469;
    const ll MOD2 = 25699183;
    const ll MOD3 = 81249169;

    Hash hash1, hash2, hash3;

    TripleHash();

    TripleHash(string s) : hash1(s, MOD1), hash2(s, MOD2), hash3(s, MOD3) {}

    tuple<int, int, int> query(int l, int r) {
        return { hash1.query(l, r), hash2.query(l, r), hash3.query(l, r) };
    }

    tuple<int, int, int> query_inv(int l, int r) {
        return { hash1.query_inv(l, r), hash2.query_inv(l, r), hash3.query_inv
    (l, r) };
    }
};

struct HashK {
    vector<ll> primes; // more primes = more hashes
    vector<Hash> hash;

    HashK();

    HashK(string s, vector<ll> primes): primes(primes) {
        for (auto p : primes) {
            hash.push_back(Hash(s, p));
        }
    }

    vector<int> query(int l, int r) {
        vector<int> ans;

        for (auto h : hash) {
            ans.push_back(h.query(l, r));
        }

        return ans;
    }

    vector<int> query_inv(int l, int r) {
        vector<int> ans;

        for (auto h : hash) {
            ans.push_back(h.query_inv(l, r));
        }

        return ans;
    }
};
```

9.3 Kmp

```
#include <bits/stdc++.h>

using namespace std;

// KMP computa online o nxt, importante para õquestes q usam
// a prefix function

const int MAX = 1e6+1;
string p;
vector<int> nbr(MAX);
```

```
int nxt(char c, int n){

    while(n != -1){
        if((n+1) < p.size() && p[n + 1] == c){

            n++;
            break;
        } else {

            n = nbr[n];
        }
    }

    if(n == -1 && p[0] == c) n++;

    return n;
}

void kmp(){

    int n = p.size();

    nbr[0] = -1;

    for(int i = 1; i < n; i++){

        nbr[i] = nbr[i-1];
        nbr[i] = nxt(p[i], nbr[i]);
    }
}

// KMP Botelho e Dudu onde rola a épr çãcomputao
// para cada letra do alfabeto


template<class T>
struct KMP{
  int n; vi p; T in; vector<vi> a;
  template<class S>
  KMP(S s, T ain, int asz):n(sz(s)),p(n,0),in(ain),a(n+1,vi(asz,0)){
    rep(i, 1, n){
      int j = p[i-1];
      while(j and s[j]!=s[i])j = p[j-1];
      p[i] = j + (s[i]==s[j]);

    }
    rep(i, 0, n+1) rep(c, 0, asz){
        if (i and (i==n or c+in!=s[i]))a[i][c] = a[p[i-1]][c];
        else a[i][c] = i + (c+in == s[i]);
      }
  }
  int nxt(int cur, T c){return a[cur][c-in];}
};
```

9.4 Lcs

```
string LCSubStr(string X, string Y)
{
    int m = X.size();
    int n = Y.size();

    int result = 0, end;
    int len[2][n];
    int currRow = 0;

    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0)
                len[currRow][j] = 0;
            else if(X[i-1] == Y[j-1]){
                len[currRow][j] = len[1-currRow][j-1] + 1;
                if(len[currRow][j] > result){
                    result = len[currRow][j];
                    end = i - 1;
                }
            }
            else
                len[currRow][j] = 0;
        }

        currRow = 1 - currRow;
    }

    if(result==0)
        return string();

    return X.substr(end - result + 1, result);
}
```

9.5 Lcs Especial

```
void recover(int i, int j){
     if (i>=s_size || j>=t_size) return ;
     if (s[i]==t[j]){ans.push_back(s[i]);recover(i+1, j+1);}
     else if(lcs_size[i+1][j]>lcs_size[i][j+1]) return recover(i+1, j);
     else return recover(i, j+1);

 }

 int main(){
     cin >> s >> t;
     s_size = s.size();
     t_size = t.size();


     for(int i=s_size-1; i>=0; i--){
         for(int j =t_size-1;j>=0; j--){
             if(s[i]==t[j]) lcs_size[i][j] = 1+lcs_size[i+1][j+1];
             else lcs_size[i][j] = max(lcs_size[i+1][j], lcs_size[i][j+1]);
         }
     }


     recover(0,0);

     cout << ans << endl;
```

```
}
```

9.6 Suffix Array

```
// A suffix array will contain integers that represent the
// starting indexes of the all the suffixes of a given string, after the
    aforementioned suffixes are sorted.

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for (int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for (int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i]]++;
        for (int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for (int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        for (int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] !=
            ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
        ra = nra;
        if (ra[sa[n-1]] == n-1) break;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}


int32_t main(){
    sws;
    string s;
    cin>>s;

    vector<int> suf = suffix_array(s);
    vector<int> lcp = kasai(s, suf);

    ll ans = 0;
    for(int i=0; i<s.size(); i++){
        if(islower(s[suf[i]])){
            int sz = s.size()-suf[i];
            ans += (sz - lcp[i]);
        }
    }
    cout<<ans<<endl;
}
```

9.7 Suffix Automata

```
// Suffix Automata
// estrutura que compacta todas as substrings de uma string
// todo caminho no automato eh uma substring, e toda substring tem um caminho
    associado a ela
// o link de um no eh o maior sufixo dela que possui um conjunto de
    ocorrencias maior do que o proprio no (ou o lnk eh a raiz)
// cada no representa varias substrings, as de tamanho len[lnk[no]]+1 ate as
    de len[no] com mesmo sufixo

struct SuffixAutomata {
    int n, id = 1, last = 1;
    vector<vector<int>> to;
    vector<int> len, lnk, occ, states, fpos;

    SuffixAutomata(string &s, char a='a'): n(s.size()), to(2*n+2, vector<int
    >(26, 0)), len(2*n+2), lnk(2*n+2), occ(2*n+2, 0), fpos(2*n+2) {
        len[1] = lnk[1] = 0, states.pb(1);
        for (char c : s) push(c-a);

        sort(all(states), [&](int a, int b) {return len[a] > len[b];});
        for (int st : states)
            occ[lnk[st]] += occ[st];
    }

    void push(int c) {
        int curr = ++id;
        int prev = last;
        last = curr;
        states.pb(curr);

        len[curr] = len[prev]+1;
        fpos[curr] = len[curr]-1;
        occ[curr] = 1;

        for (; prev && !to[prev][c] ; prev = lnk[prev]) to[prev][c] = curr;

        int q = to[prev][c];
        if (!prev)
            lnk[curr] = 1;
        else if (len[prev] + 1 == len[q])
            lnk[curr] = q;
        else {
            int clone = ++id;
            states.pb(clone);

            lnk[clone] = lnk[q];
            to[clone] = to[q];
            fpos[clone] = fpos[q];
            len[clone] = len[prev]+1;

            lnk[q] = lnk[curr] = clone;
```

```
            for (; to[prev][c] == q ; prev = lnk[prev]) to[prev][c] = clone;
        }
    }
};
```

9.8 Trie

```
struct Trie{

    int trie[MAX][26];
    bool finish[MAX];
    int nxt = 1, len = 0;

    void add(string s){
        int node = 0;
        for(auto c: s){
            if(trie[node][c-'a'] == 0)
                node = trie[node][c-'a'] = nxt++;
            else
                node = trie[node][c-'a'];
        }
        if(!finish[node]){
            finish[node] = true;
            len++;
        }
    }

    bool find(string s, bool remove=false){
        int node = 0;
        for(auto c: s)
            if(trie[node][c-'a'] == 0)
                return false;
            else
                node = trie[node][c-'a'];
        if(remove and finish[node]){
            finish[node]=false;
            len--;
        }
        return finish[node];
    }
};
```

9.9 Trie Xor

```
// TrieXOR
//
// adiciona, remove e verifica se existe strings binarias
// max_xor(x) = maximiza o xor de x com algum valor da trie
//
// raiz = 0
//
// https://codeforces.com/problemset/problem/706/D
//
// O(|s|) adicionar, remover e buscar

struct TrieXOR {
    int n, alph_sz, nxt;
    vector<vector<int>> trie;
    vector<int> finish, paths;

    TrieXOR() {}

    TrieXOR(int n, int alph_sz = 2) : n(n), alph_sz(alph_sz) {
        nxt = 1;
        trie.assign(n, vector<int>(alph_sz));
        finish.assign(n * alph_sz, 0);
        paths.assign(n * alph_sz, 0);
    }

    void add(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0)
                trie[curr][b] = nxt++;

            paths[curr]++;
            curr = trie[curr][b];
        }

        paths[curr]++;
        finish[curr]++;
    }

    void rem(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            paths[curr]--;
            curr = trie[curr][b];
        }

        paths[curr]--;
        finish[curr]--;
    }

    int search(int x) {
        int curr = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);

            if (trie[curr][b] == 0) return false;

            curr = trie[curr][b];
        }

        return (finish[curr] > 0);
    }

    int max_xor(int x) { // maximum xor with x and any number of trie
```

```
        int curr = 0, ans = 0;

        for (int i = 31; i >= 0; i--) {
            int b = ((x&(1 << i)) > 0);
            int want = b^1;

            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) want ^=
     1;
            if (trie[curr][want] == 0 || paths[trie[curr][want]] == 0) break;
            if (want != b) ans |= (1 << i);

            curr = trie[curr][want];
        }

        return ans;
    }
};
```

9.10 Z Func

```
vector<int> Z(string s) {
    int n = s.size();
    vector<int> z(n);
    int x = 0, y = 0;
    for (int i = 1; i < n; i++) {
        z[i] = max(0, min(z[i - x], y - i + 1));
        while (i + z[i] < n and s[z[i]] == s[i + z[i]]) {
            x = i; y = i + z[i]; z[i]++;
        }
    }
    return z;
}
```

10 Tree

10.1 Binary Lifting

```
vector<int> adj[MAX];
const int LOG = 30;
int up[MAX][LOG], parent[MAX];

void process(int n){
    for(int v=1; v<=n; v++){
        up[v][0]= parent[v];
        for(int i=1; i<LOG; i++){
            up[v][i] = up[ up[v][i-1] ][i-1];
        }
    }
}

int jump(int n, int k){
    for(int i=0; i<LOG; i++){
    if(k & (1 << i)){
      n = up[n][i];
    }
  }
  if(n == 0) return -1;
  return n;
}

int32_t main(){

    int n, q; cin>>n>>q;

    parent[1] = 0;
    for(int i=1; i<=n-1; i++){
        int x;
        cin>>x;
        parent[i+1] = x;

        adj[i+1].pb(x);
        adj[x].pb(i+1);
    }
    process(n);
    for(int i=0; i<q; i++){
        int a, b;
        cin>>a>>b;

        cout<<(jump(a,b))<<endl;
    }
}

// binary lifting pra achar o max no caminho entre duas arestas

vector<vector<pii>> g;
vector<int> depth(MAX);
vector<pii> pai;
int p2k[MAX][LOG], bin_max[MAX][LOG];

int preprocess(int u, int p, int d) {
    depth[u] = d;

    for (auto [v, c] : g[u]) {
        if (v == p) continue;

        pai[v] = {u, c};
        preprocess(v, u, d+1);
    }
}

int max_cam(int u, int v) {
    int ans = -INF;
    if (depth[u] < depth[v]) swap(u, v);

    for (int i = LOG-1 ; i >= 0 ; i--) {
        if (depth[p2k[u][i]] >= depth[v]) {
            ans = max(ans, bin_max[u][i]);
            u = p2k[u][i];
        }
    }

    if (u == v) return ans;

    for (int i = LOG-1 ; i >= 0 ; i--) {
```

```
        if (p2k[v][i] != p2k[u][i]) {
            ans = max(ans, bin_max[v][i]);
            ans = max(ans, bin_max[u][i]);

            v = p2k[v][i];
            u = p2k[u][i];
        }
    }

    ans = max(ans, max(bin_max[u][0], bin_max[v][0]));
    return ans;
}

void solve() {
    // ...

    preprocess(1, 0, 0);
    rep(no, 1, n+1) {
        p2k[no][0] = pai[no].ff;
        bin_max[no][0] = pai[no].ss;
    }

    rep(i, 1, LOG) {
        rep(no, 1, n+1) {
            p2k[no][i] = p2k[p2k[no][i-1]][i-1];
            bin_max[no][i] = max(bin_max[no][i-1], bin_max[p2k[no][i-1]][i-1])
    ;
        }
    }

    // ...
}
```

10.2 Centroid

```
// acha a centroide, no da arvore em que nenhuma das subarvores de seus filhos
     possui tamanho maior que n/2
// usado na centroid decomposition, em que resolvemos o problema passando pela
     centroide e depois para cada subarvore por recursao
// achar a centroide => O(n)
// centroid decomposition => O(n*log(n))

vector<vector<int>> g;
vector<bool> inG;
vector<int> subtree_sz;

int find_centroid(int v) {
    int pai = 0;
    bool ok = false;

    while (!ok) {
        ok = true;

        for (int u : g[v]) {
            if (u == pai || !inG[u] || 2*subtree_sz[u] <= subtree_sz[v])
    continue;

            subtree_sz[v] -= subtree_sz[u];
            subtree_sz[u] += subtree_sz[v];
            pai = v;
            v = u;
            ok = false;
            break;
        }
    }

    return v;
}
```

10.3 Eulertour Segt

```
#include <bits/stdc++.h>

 /*

    Permite computar uma seg numa arvore
    éAlm de tornar a arvore em um array
 */
using namespace std;
#define int long long
const int MOD = 1e9+7;

const int MAX = 2e5+1;

vector<int> segt(8*MAX), euler(2*MAX), in(MAX), out(MAX), aux(MAX);
int tempo = 0;
vector<int> t[MAX];

void dfs(int u, int p){

    euler[tempo] = u;
    in[u] = tempo;

    tempo++;

    for(auto v : t[u]){

        if(v != p) dfs(v, u);
    }

    euler[tempo] = u;
    out[u] = tempo;

    tempo++;

    return;
}

void build(int id, int il, int ir){

    if(il == ir){

        segt[id] = aux[euler[il]];
        return;
    }
```

```
    int im = (il + ir) / 2;

    build(2*id, il, im);
    build(2*id+1, im+1, ir);


    segt[id] = segt[2*id] + segt[2*id+1];
}

void update(int id, int il, int ir, int idx, int x){

    if(il == ir){

        segt[id] = x;
        aux[euler[idx]] = x;
        return;
    }

    int im = (il + ir) / 2;

    if(im < idx){

        update(2*id+1, im+1, ir, idx, x);
    } else {

        update(2*id, il, im, idx, x);
    }

    segt[id] = segt[2*id] + segt[2*id+1];

    return;
}

int query(int id, int il, int ir, int l, int r){

    if(il >= l && ir <= r){

        return segt[id];
    }
    if(l > ir || r < il) return 0;

    int im = (il + ir) / 2;

    int esq = query(2*id, il, im, l, r);
    int dir = query(2*id+1, im+1, ir, l, r);

    return esq + dir;
}

int32_t main(){

    int n, q; cin >> n >> q;

    for(int i = 1; i <= n; i++) cin >> aux[i];

    for(int i = 2; i <= n; i++){

        int u, v; cin >> u >> v;

        t[u].push_back(v);
        t[v].push_back(u);
    }

    dfs(1, 0);

    build(1, 0, 2*(n));

    while(q--){

        int t; cin >> t;

        if(t == 1){

            int v, p; cin >> p >> v;

            update(1, 0, 2*(n), in[p], v);
            update(1, 0, 2*(n), out[p], v);
        } else {

            int p; cin >> p;

            cout << query(1, 0, 2*(n), in[p], out[p]) / 2 << '\n';
        }
    }

    return 0;
}
```

10.4 Hld

```
// Heavy-Light Decompostion
// ordena o euler tour de forma que fazer queries no caminho de um no ate a
    raiz fica O(log(n) * (complexidade da query em range))

struct HLD {
    int n;
    vector<vector<int>> g;
    vector<int> sz, ordem, head, idx, parent;
    SegTree seg;

    HLD(int n, vector<vector<int>> &g): n(n), g(g), sz(n+1), idx(n+1), head(n
    +1), parent(n+1) {
        subtree_sz(1, -1, 0);
        build(1, 1, -1);

        seg = SegTree(ordem.size(), ordem);
    }

    void update(int v, int x) {
        seg.update(idx[v], x);
    }

    int query(int u) {
        int ans = -INF;
```

```
        int anc = 1;        // colocando ate a raiz como exemplo, poderia ser
    ate o lca tbm
        while (head[u] != head[anc]) {
            ans = max(ans, seg.query(idx[head[u]], idx[u]));
            u = parent[head[u]];
        }
        ans = max(ans, seg.query(idx[anc], idx[u]));

        return ans;
    }

    int subtree_sz(int v, int pai, int d) {
        int sub = 1;
        parent[v] = pai;

        for (auto u : g[v]) {
            if (u == pai) continue;

            sub += subtree_sz(u, v, d+1);
        }

        return sz[v] = sub;
    }

    void build(int v, int h, int pai) {
        int mai = -1, to = -1;
        head[v] = h;
        idx[v] = ordem.size();
        ordem.pb(val[v]);

        for (auto u : g[v]) {
            if (u == pai) continue;

            if (sz[u] > mai) {
                mai = sz[u];
                to = u;
            }
        }

        if (to != -1) build(to, h, v);
        for (auto u : g[v]) {
            if (u == to || u == pai) continue;

            build(u, u, v);
        }
    }
};
```

10.5 Isomorftree

```
// Isomorfismo de arvores
//
// thash() retorna o hash da arvore (usando centroids como vertices especiais)
    .
// Duas arvores sao isomorfas sse seu hash eh o mesmo
//
// O(|V|.log(|V|))

map<vector<int>, int> mphash;

struct tree {
  int n;
  vector<vector<int>> g;
  vector<int> sz, cs;

  tree(int n_) : n(n_), g(n_), sz(n_) {}

  void dfs_centroid(int v, int p) {
    sz[v] = 1;
    bool cent = true;
    for (int u : g[v]) if (u != p) {
      dfs_centroid(u, v), sz[v] += sz[u];
      if(sz[u] > n/2) cent = false;
    }
    if (cent and n - sz[v] <= n/2) cs.push_back(v);
  }
  int fhash(int v, int p) {
    vector<int> h;
    for (int u : g[v]) if (u != p) h.push_back(fhash(u, v));
    sort(h.begin(), h.end());
    if (!mphash.count(h)) mphash[h] = mphash.size();
    return mphash[h];
  }
  ll thash() {
    cs.clear();
    dfs_centroid(0, -1);
    if (cs.size() == 1) return fhash(cs[0], -1);
    ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1], cs[0]);
    return (min(h1, h2) << 30) + max(h1, h2);
  }
};


// Versao mais rapida com hash, ideal para hash de floresta.
// subtree_hash(v, p) retorna o hash da subarvore enraizada em v com pai p.
// tree_hash() retorna o hash da arvore.
// forest_hash() retorna o hash da floresta.
// use o vetor forb[] para marcar vertices que nao podem ser visitados.
//
// O(|V|.log(|V|))

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int uniform(ll l, ll r) {
  uniform_int_distribution<ll> uid(l, r);
  return uid(rng);
}

const int MOD = 1e9 + 7;
const int H = 13;
const int P = uniform(1, MOD-1);
const int P2 = uniform(1, MOD-1);

struct tree {
  int fn;
  vector<vector<int>> g;
```

```
  vector<int> sz, cs;
  vector<bool> forb;

  tree(int n_) : fn(n_), g(n_), sz(n_), forb(n_) {}

  void dfs_size(int v, int p) {
    sz[v] = 1;
    for (int u : g[v]) if (u != p and !forb[u]) {
      dfs_size(u, v), sz[v] += sz[u];
    }
  }
  void dfs_centroid(int v, int p, int n) {
    bool cent = true;
    for (int u : g[v]) if (u != p and !forb[u]) {
      dfs_centroid(u, v, n);
      if(sz[u] > n/2) cent = false;
    }
    if (cent and n - sz[v] <= n/2) cs.push_back(v);
  }
  int subtree_hash(int v, int p) {
    int h = H;
    for (int u : g[v]) if (u != p and !forb[u]) {
      h = ll(h) * (P + subtree_hash(u, v)) %  MOD;
    }
    return h;
  }
  int tree_hash(int v=0) {
    cs.clear();
    dfs_size(v, -1);
    dfs_centroid(v, -1, sz[v]);
    if (cs.size() == 1) return subtree_hash(cs[0], -1);
    assert (cs.size() == 2);
    int h1 = subtree_hash(cs[0], cs[1]);
    int h2 = subtree_hash(cs[1], cs[0]);
    return ll(P + h1) * (P + h2) % MOD;
  }
  int forest_hash() {
    fill(sz.begin(), sz.end(), 0);
    int hash = 1;
    for (int v = 0; v < fn; v++) if (!sz[v] and !forb[v]) {
      hash = hash * ll(P2 + tree_hash(v)) % MOD;
    }
    return hash;
  }
};
```

10.6 Kruskall

```
// Arvore geradora minima (arvore conexa com peso minimo)
// O(MlogN)

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

int n;
class DSU{
    vector<int> parent, sz;
    public:
    void make(int v){
        parent[v] = v;
        sz[v] = 1;
    }

    int find(int v){
        if (v == parent[v]) return v;
        return parent[v] = find(parent[v]);
    }

    void union_(int a, int b){
        a = find(a), b = find(b);

        if(sz[b]>sz[a]) swap(a,b);
        if (a != b){
            sz[a] += sz[b];
            parent[b] = a;
        }
    }

    bool same(int a, int b){
        a = find(a), b = find(b);
        return a == b;
    }

    DSU(int n): parent(n+1), sz(n+1){
        for(int i=1; i<=n; i++) make(i);
    }
};

// {a, b, weight}
vector<tuple<int,int,int>> MST(vector<tuple<int,int,int>> &v){
    DSU dsu(n);
    sort(v.begin(), v.end());
    vector<tuple<int,int,int>> ans;
    for(int i=0; i<v.size(); i++){
        int w, a, b;
        tie(w, a, b) = v[i];
        if(!dsu.same(a, b)){
            dsu.union_(a, b);
            ans.push_back({a, b, w});
        }
    }
    return ans;
}

int32_t main(){
    int m;
    cin>>n>>m;
    DSU dsu(n);
    vector<tuple<int,int,int>> vt;
    for(int i=0; i<m; i++){
        int a, b, w;
        cin>>a>>b>>w;
        // {weight, a, b}
        vt.push_back({w, a, b});
    }
    vector<tuple<int,int,int>> ans = MST(vt);
```

```
    return 0;
}
```

10.7 Lca

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define sws ios::sync_with_stdio(false);cin.tie(nullptr);
typedef pair<int, int> ii;
#define INF INT64_MAX
const int MAX = 2e5+1;
const int MOD = 1e9+7;
const int LOG = 30;

int ances[MAX][LOG];
int depth[MAX];

int get_lca(int no, int no1){

    int k;

    if (depth[no1] > depth[no]) swap(no, no1);

    k = depth[no] - depth[no1];

    for(int i = LOG-1; i >= 0; i--){

        if(k & (1 << i)){

            no = ances[no][i];
        }
    }

    if(no == no1) return no;

    for(int i = LOG-1; i >= 0; i--){

        if(ances[no][i] != ances[no1][i]){

            no = ances[no][i];
            no1 = ances[no1][i];
        }
    }
    return ances[no][0];

}
int32_t main(){
    sws;

    int n, q; cin >> n >> q;

    vector<int> parents(n+1);

    for(int i = 2; i <= n; i++){

        int v; cin >> v;

        parents[i] = v;
    }

    for(int j = 1; j < LOG; j++){

        for(int i = 1; i <= n; i++){

            ances[i][0] = parents[i];

            if(i != 1) depth[i] = depth[parents[i]] + 1;

            ances[i][j] = ances[ances[i][j-1]][j-1];
        }
    }

    while(q--){

        int no, no1; cin >> no >> no1;

        int ans = get_lca(no, no1);

        cout << ans << '\n';
    }

    return 0;
}
```

10.8 Virtualtree

```
// Virtual Tree
//
// Comprime uma arvore dado um conjunto S de vertices, de forma que
// o conjunto de vertices da arvore comprimida contenha S e seja
// minimal e fechado sobre a operacao de LCA
// Se |S| = k, a arvore comprimida tem menos que 2k vertices
// As arestas de virt possuem a distancia do vertice ate o vizinho
// Retorna a raiz da virtual tree
//
// lca::pos deve ser a ordem de visitacao no dfs
// voce pode usar o LCAcomHLD, por exemplo
//
// O(k log(k))

vector<pair<int, int>> virt[MAX];

#warning lembrar de buildar o LCA antes
int build_virt(vector<int> v) {
  auto cmp = [&](int i, int j) { return lca::pos[i] < lca::pos[j]; };
  sort(v.begin(), v.end(), cmp);
  for (int i = v.size()-1; i; i--) v.push_back(lca::lca(v[i], v[i-1]));
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());
  for (int i = 0; i < v.size(); i++) virt[v[i]].clear();
  for (int i = 1; i < v.size(); i++) virt[lca::lca(v[i-1], v[i])].clear();
  for (int i = 1; i < v.size(); i++) {
    int parent = lca::lca(v[i-1], v[i]);
    int d = lca::dist(parent, v[i]);
```

```
#warning soh to colocando aresta descendo
    virt[parent].emplace_back(v[i], d);
  }
  return v[0];
}
```