O(bem facin) ACM-ICPC Team Notebook

Contents

1 Unlabeled

1.1 Template

1.1.1 Template

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sz(x) ((int) (x).size())
#define forn(i,n) for (int i = 0; i < int(n); ++i)
#define printArr(harry, tam) forn(ii, tam) cout << harry[i] << " \n"[i == tam -1]
#define maxn "100000"
#define mod "1000000007"
#define md(x) ((((x) % mod) + mod) % mod)
#define sc scanf
#define pr printf
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define EPS 1e-9
const int inf = int(1e9) + int(1e5); //Usado em Push_relabel

typedef long long ll;
typedef long double ld;
typedef pair<int, int> ii;
typedef vector<int> vi;
```

1.1.2 Debug functions

```cpp
// Trace dinâmico
template<typename T>
void trace(T a) { cout << a << "\n"; }
template<typename T, typename... Args>
void trace(T a, Args... args) { cout << a << " "; trace(args...); }

// Debugador de recursão
int recNum = 0;
void prTabs() { forn(ii, recNum) cout << "    "; }
template<typename... Args>
void recInit(Args... args) { prTabs(); trace("rec(", args..., ")"); recNum++; }
template<typename T>
void recEnd(T a) { prTabs(); trace("->", a); recNum--;}
```

2 Math

2.1 Number Theory

2.1.1 Extended GCD

```cpp
// Retorna x e y tal que ax + by = d e d = gcd(a, b);
pair<ll, ll> extended_gcd(ll a, ll b) {
    if (!b) return {1, 0};
    pair<ll, ll> t = extended_gcd(b, a % b);
    return {t.se, t.fi - t.se * (a / b)};
}
```

2.1.2 Multiplicative inverse

```cpp
ll modinverse(ll a, ll m) {
    return (extended_gcd(a, m).fi % m + m) % m;
}

// If m is prime
ll modinverse(ll a, ll m) {
    return pow_whith_mod(a, m-2, m);
}
```

2.1.3 Remainder Chinese Theorem

```cpp
// k is size of num[] and rem[].  Returns the smallest
// number x such that:
//  x % num[0] = rem[0],
//  x % num[1] = rem[1],
//  ..................
//  x % num[k-1] = rem[k-1]
// Assumption: Numbers in num[] are pairwise coprime
// (gcd for every pair is 1)
ll findMinX(vector<ll> &num, vector<ll> &rem) {
    ll prod = 1;
    for (int i = 0; i < (int)num.size(); i++)
        prod *= num[i];

    ll result = 0;
    for (int i = 0; i < (int)num.size(); i++) {
        ll pp = prod / num[i];
        result += rem[i] * modinverse(pp, num[i]) * pp;
    }
    return result % prod;
}
```

2.1.4 Totient function

```cpp
ll phi(ll n) {
    ll result = n;
    for (ll i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            while(n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if(n > 1)
        result -= result / n;
    return result;
}
```

2.1.5 Crivo linear

```cpp
#include <bits/stdc++.h>
#define MAX 1000007

using namespace std;

vector <int> primes,divisores(MAX), power(MAX);
bitset<MAX> is_prime;

void crivo(){
    is_prime.set();
    is_prime[0] = is_prime[1] = 0;
    divisores[1] = 1;

    for(int i = 2; i < MAX; i++){
        if(is_prime[i]){
            primes.push_back(i);
            power[i] = 1;
            divisores[i] = 2;
        }

        for(int  j = 0; j < primes.size() && i*primes[j] < MAX; j++){
            is_prime[i*primes[j]] = 0;
            if(i%primes[j] == 0){
                power[i*primes[j]] = power[i] + 1;
                divisores[i*primes[j]] = divisores[i]/(power[i]+1) *(power[i*primes[j]]+1);
                break;
            }
            else{
                power[i*primes[j]] = 1;
                divisores[i*primes[j]] = divisores[i]*2;
            }
        }
    }
}

int main(){
    crivo();
    int n;

    cin >> n;
    printf("%d\n", divisores[n]);
    return 0;
}
```

2.1.6 Miller-Rabin primality test

```cpp
using u64 = uint64_t;
//In case of u128 not works, use modular multiplication from the other notebook
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n) { // returns true if n is prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }
    // For 32 bit integer it is only necessary to check the first 4 prime bases
```

```
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

2.2 Binomial

```
ll fact(ll a, ll m){
    ll ans = a;
    while(--a) ans = ans * a % m;
    return ans;
}

ll binomial(ll a, ll b, ll m){
    ll ans = 1;

    for(int i = max(a-b,b)+1; i <= a; i++)
        ans = ans * i % m;

    return ans * expo(fact(min(a-b,b), m), m-2, m) % m;
}
```

2.3 Catalan Numbers

The Catalan numbers on nonnegative integers n are a set of numbers that arise in tree enumeration problems of the type, 'In how many ways can a regular n-gon be divided into n-2 triangles if different orientations are counted separately?' (Euler's polygon division problem). The solution is the Catalan number C(n-2)

```
// Catalan Generics

ll trp(ll x, ll y, ll m){
    if(0 <= y && y < m)
        return binomial(x+y, y);

    if(m <= y && y <= x+m-1)
        return binomial(x+y, y) - binomial(x+y, y-m);

    return 0;
}

ll triangle(ll x, ll y){
    return trp(x,y,1);
}

ll square(ll x){
    return trp(x,x,1);
}
```

2.4 Fast exponentiation

```
ll expo(ll a, ll b, ll m){
    if(!b) return 1;
    if(b&1) return a*expo(a, b-1, m) % m;
    return expo((a*a)%m, b>>1, m) % m;
}
```

2.5 Gaussian Elimination

this implementation can solve systems o linear equations in $O(N^3)$. Do never... NEVER, implment sub(vi, vi) and mul(vi, vi) routines with collateral effect as this can lead to imprecision and incorrectness of solution.

```
#include <bits/stdc++.h>
#define mod 1009
#define vi vector<int>
#define mtrx vector<vi>
#define pb push_back
using namespace std;

int expo(int a, int b){
        if(!b)return 1;
        if(b & 1) return a * expo(a, b-1) % mod;
        return expo(a*a%mod, b>>1);
}
vi mul(vi a, int b){
        vi c;
        for(int i = 0; i < a.size(); i++)
                c.pb(a[i] * b % mod);
        return c;
}
vi sub(vi a, vi b){
        vi c;
        for(int i = 0; i < a.size(); i++)
                c.pb((a[i] - b[i] + mod) % mod);
        return c;
}
vi gauss_jordan(mtrx &m){
        int sz = m[0].size();
        for (int i = 0; i < m.size(); i++){
                int divby = expo(m[i][i], mod-2);
                for (int k = i; k < sz; k++)
                        m[i][k] = m[i][k] * divby % mod;

                for (int k = i+1; k < m.size(); k++){
                        vi sb = mul(m[i], m[k][i]);
                        m[k] = sub(m[k], sb);
                }
        }
        vi ans;

        for (int i = m.size()-1; i+1; i--){
                ans.pb(m[i][sz-1]);
                for (int j = i-1; j+1; j--)
                        m[j][sz-1] = (m[j][sz-1] - m[j][i] * m[i][sz-1] % mod + mod) % mod;
        }
        reverse(ans.begin(), ans.end());
        return ans;
}

int main(){
        string s;
        cin >> s;
        const int maxn = 1 << 8;

        int ans[6];
        memset(ans, -1, sizeof ans);
        for (int i = 0; i < maxn; i++){
                int tmp[5];
                for (int j = 0; j < 4; j++)
                        tmp[j+1] = 1 + i / (1<<(j<<1)) % 4;

                for (int j = 0; j < 4; j++) {
                        int x = j+1;
                        int k = 0;
                        while(k < 15){
                                if (s[k] == '?' || s[k]-'0' == x)
                                        x = tmp[x];
                                else
                                        break;
                                k++;
                        }
                        if (k == 15){
                                mtrx m;
                                for(int i = 0; i < 4; i++){
                                        m.pb(vi());
                                        for(int j = 3; j+1; j--){
                                                m[i].pb(expo(i+1, j));
                                        }
                                        m[i].pb(tmp[i+1]);
                                }

                                vi solution = gauss_jordan(m);

                                for(int i = 0; i < 4; i++)
                                        ans[i] = solution[i];

                                ans[4] = mod;
                                ans[5] = j+1;
                        }
                }
```

```
        }

        for (int i = 0; i < 6; i++)
                cout << ans[i] << " \n"[i==5];

        return 0;
}
```

2.6 Matrix exponentiation

```
vector<vi> multiply(vector<vi>& A, vector<vi>& B) {
  int n = A.size(), m = A[0].size(), k = B[0].size();
  vector<vi> C(n, vi(k, 0));

  for(int i = 0; i < n; i++)
    for(int j = 0; j < k; j++)
      for(int l = 0; l < m; l++)
        C[i][j] += A[i][l] * B[l][j];

  return C;
}

vector<vi> power(vector<vi>& A, int k) {
  int n = A.size();
  vector<vi> resp(n, vi(n)), B = A;

  for(int i = 0; i < n; i++) resp[i][i]=1;

  while(k) {
    if(k & 1) resp = multiply(resp, B);
    B = multiply(B, B); k >>= 1;
  }
  return resp;
}

vector<vi> resp = power(A, k);
```

2.7 Bitmasks

2.7.1 Submasks iteration

```
vi subs;
for (int sub=mask; sub; sub=(sub-1)&mask)
    subs.pb(sub);
subs.pb(0);
```

2.8 FFT

2.8.1 FFT comum

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

using namespace std;

typedef complex<double> cd;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    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(cd & x : a) x /= n;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        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;
}


int main(){
    return 0;
}
```

2.8.2 FST

```
#include <bits/stdc++.h>
#define LOGN 22
#define MAX (1 << LOGN)

using namespace std;
typedef long long ll;

void FST(vector<ll> &a) {
    for (int b = 0; b < LOGN; b++){
        for (int i = 0; i < MAX; i++){
            if((i & (1 << b)) == 0){
                a[i + (1 << b)] = (a[i] != MAX)? min(a[i], a[i + (1 << b)]): a[i + (1 << b)];
            }
        }
    }
}

int arr[MAX];
int main(){
    int n;scanf("%d", &n);

    vector<ll> a(MAX, MAX);
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
        a[arr[i]] = arr[i];
    }
    FST(a);
    return 0;
}
```

2.8.3 Fast Walsh Hadamard Transform

```
#include <bits/stdc++.h>
#define MAX 1 << 17
#define LOGN 17
#define MOD 1000000007

using namespace std;
typedef long long ll;

ll inverse_m(ll a, ll b, ll s0 = 1LL, ll s1 = 0LL){
    return b == 0? s0: inverse_m(b, a%b, s1, s0-s1*(a/b));
}

// OR em que o AND é 0
```

```
vector<ll> ORANDO(vector<ll> &a) {
    vector<ll> ans(MAX, 0LL);
    for (int i = 0; i < MAX; i++){
        for (int j = i; j >= 0; j = (j-1)&i){
            ll mult = a[i^j]*a[j];
            ans[i] = (ans[i] + mult)%MOD;
            if(j == 0) break;
        }
    }

    return ans;
}

int T[3][2][2][2] = {
    { { {1,  1}, {1, -1} }, { {1,  1}, {1, -1} } }, /// xor
    { { {0,  1}, {1,  1} }, { {-1, 1}, {1,  0} } }, /// and
    { { {1,  1}, {1,  0} }, { {0,  1}, {1, -1} } }  /// or
};

void FFT(vector<ll> &a, int op, bool inverse = false) {
    ll u1 = T[op][inverse][0][0], v1 = T[op][inverse][0][1];
    ll u2 = T[op][inverse][1][0], v2 = T[op][inverse][1][1];

    for(int b = 0; b < LOGN; b++)
        for(int i = 0; i < MAX; i++)
            if((i & (1 << b)) == 0) {
                ll u = a[i], v   = a[i + (1 << b)];
                a[i] = (u*u1) + (v*v1);
                if(a[i] >= MOD) a[i] -= MOD;
                if(a[i] < 0) a[i] += MOD;

                a[i + (1 << b)] = (u*u2) + (v*v2);
                if(a[i + (1 << b)] >= MOD) a[i + (1 << b)] -= MOD;
                if(a[i + (1 << b)] < 0) a[i + (1 << b)] += MOD;
            }

        if (op == 0 && inverse){
        ll inv = inverse_m(1 << 17, MOD);
                for (int i=0; i<a.size(); i++) a[i] = (a[i]*inv)%MOD;
    }
}

/// op is 0 for XOR, 1 for AND, 2 for OR
vector<ll> convolution(vector<ll> a, vector<ll> b, int op) {
    FFT(a, op, false);
    FFT(b, op, false);
    for(int i=0; i<a.size(); i++)
        a[i] = (a[i] * b[i])%MOD;
    FFT(a, op, true);
    return a;
}

void print(vector<ll> &a, int sz){
    for(int i = 0; i < sz; i++) cout << a[i] << " ";
    cout << endl;
}
int main(){

    return 0;
}
```

3 Data Structures

3.1 BIT

3.1.1 BIT and example of use

```
int query(int i) {
    int s = 0;
    while(i) {
        s += bit[i];
        i -= i & (-i);
    }
    return s;
}
void update(int i, int v) {
    while(i <= MAXBIT) {
        bit[i] += v;
        i += i & (-i);
    }
}

//Contar a quantidade de inversões em arr
int getInvCount(int n){
    int inv = 0;
    for (int i=n-1; i>=0; i--) {
        inv += query(arr[i]-1);
        update(arr[i], 1);
    }
    return inv;
}
```

3.1.2 BIT 2D

```
int bit[MAXBITN][MAXBITM];

int query(int x, int y){
    int s = 0, i = x, j;
    while(i){
        j = y;
        while(j){
            s += bit[i][j];
            j -= j & (-j);
        }
        i -= i & (-i);
    }
    return s;
}

void update(int x, int y, int v){
    int i = x, j;
    while(i <= MAXBITN){
        j = y;
        while(j <= MAXBITM){
            bit[i][j] += v;
            j += j & (-j);
        }
        i += i & (-i);
    }
}
```

3.1.3 BIT 1D and 2D with map

```
map<int, int> bit;
int query(int i) {
    int s = 0;
    while(i) {
        s += (bit.count(i)) ? bit[i] : 0;
        i -= i & (-i);
    }
    return s;
}
void update(int i, int v) {
    while(i <= MAXBIT)
        if(bit.count(i)) bit[i] += v;
        else bit[i] = v;
        i += i & (-i);
    }
}
```

```
map<ii, int> bit;

int query(int x, int y){
    if(x < 0 || y < 0) {return 0;}
    int s = 0, i = x, j;
    while(i){
        j = y;
        while(j){
            s += (bit.count({i, j})) ? bit[{i, j}] : 0;
            j -= j & (-j);
        }
        i -= i & (-i);
    }
    return s;
}

void update(int x, int y, int v){
    int i = x, j;
    while(i <= MAXBITN){
        j = y;
        while(j <= MAXBITM){
            if(bit.count({i, j})) bit[{i, j}] += v;
            else bit[{i, j}] = v;
            j += j & (-j);
        }
        i += i & (-i);
    }
}
```

3.2 Sparse Table 1D e 2D

```
int val[maxn];
int sp[lgmaxn][maxn];
int lg[maxn+1];
void initLg() {
    lg[0] = -1;
    for(int i = 1; i <= maxn; i++) lg[i] = lg[i>>1]+1;
}

void build(int n){
    for(int i=0; i<n; i++) sp[0][i] = val[i];
    for(int i=1; i <= lg[n]; i++){
        for(int j=0; j + (1<<i) <= n; j++){
            sp[i][j] = max(sp[i-1][j], sp[i-1][j+(1<<(i-1))]);
        }
    }
}

int query(int l, int r){
    int k = lg[r-l+1];
    return max(sp[k][l], sp[k][r-(1<<k)+1]);
}

//Sparse Table 2D para Range Maximum/Minimum Query
int val2d[maxn][maxm];
int sp2d[maxn][maxn][lgmaxm][lgmaxm];
void build2d(int n, int m)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) sp2d[i][j][0][0] = val2d[i][j];
    }
    for (int p = 0; (1 << p) <= n; p++) {
        for (int q = 0; (1 << q) <= m ; q++) {
            if (p + q == 0) continue;
            for (int i = 0; i + (1<<p)-1 < n; i++) {
                for (int j = 0; j + (1 << q) - 1 < m; j++) {
                    if (p) sp2d[i][j][p][q] = max(sp2d[i][j][p - 1][q], sp2d[i + (1 << (p-1))][j][p -
                          1][q]);
                    else sp2d[i][j][p][q] = max(sp2d[i][j][p][q-1], sp2d[i][j+(1<<(q-1))][p][q-1]);
                }
            }
        }
    }
}

int query2d(int x1, int y1, int x2, int y2) {
    int x = 0, y = 0;
    while ((1<<(x+1)) <= x2 - x1 + 1) x++;
    while ((1<<(y+1)) <= y2 - y1 + 1) y++;
    x2 = x2 - (1<<x) + 1;
    y2 = y2 - (1<<y) + 1;

    return max( max(sp2d[x1][y1][x][y], sp2d[x2][y1][x][y]), max(sp2d[x1][y2][x][y], sp2d[x2][y2][x][y
          ]));
}
```

3.3 Segment Tree

3.3.1 Segment Tree

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

struct node{
        int valor;
        int qtd;
        int acumulado;

        node(){
                valor=0;
                qtd=0;
        }

        node(int v, int q){
                valor=v;
                qtd=q;
                this->acumulado = 0;
        }

        void update(int x){
                if (x==1) valor=qtd;
                this->acumulado|=x;
        }

        void shift(int l, int r,node* tree){
                tree[l].update(acumulado);
                tree[r].update(acumulado);
                acumulado=0;
        }
};

node merge(node a, node b){
        return node(a.valor+b.valor, a.qtd+b.qtd);
}

void update(int id, int l,int r,int s,int e, int v, node* tree){
        if(l>e || r<s) return;
        if (l>=s && r<=e){
                tree[id].update(v);return;
        }
        int mid=(l+r)/2;
        tree[id].shift(2*id+1,2*id+2,tree);
        update(2*id+1,l,mid,s,e,v,tree);
        update(2*id+2,mid+1,r,s,e,v,tree);
        tree[id]=merge(tree[2*id+1],tree[2*id+2]);
}
node query(int id, int l,int r,int x,int y, node* tree){
        if(r<x || l>y) return node();
        if(l>=x && r<=y) return tree[id];
        int mid=(l+r)/2;
    tree[id].shift(2*id+1,2*id+2,tree);
    return merge(query(2*id+1,l,mid,x,y,tree),query(2*id+2,mid+1,r,x,y,tree));
}
void build(int id, int l,int r, node* tree,node* base){
    if(l==r) tree[id]=base[l];
    else{
        int mid=(l+r)/2;
        build(2*id+1,l,mid,tree,base);
        build(2*id+2,mid+1,r,tree,base);
        tree[id]=merge(tree[2*id+1],tree[2*id+2]);
    }
}

node tree[400010];
node base[100010];

int main(){}
```

3.3.2 Segment Tree Dynamic Implementação 1

```
struct Node {
        ll v;
        Node *l, *r; //Nós filhos
        Node(ll _v = 0LL):
                v(_v), l(NULL), r(NULL) {}
};

typedef Node *pNode;

void merge(pNode node) {
        pNode l = node->l, r = node->r;
        if (!l) node->v = r->v;
        else if (!r) node->v = l->v;
        else node->v = min(l->v, r->v);
}

void update(pNode &node, ll l, ll r, ll i, ll v) {
        if (!node) node = new Node();
        ll m = (l+r)>>1;
        if (l == r) {
                node->v = v; return;
        } else if (i <= m) {
                update(node->l, l, m, i, v);
        } else {
                update(node->r, m+1, r, i, v);
        }
        merge(node);
}

ll query(pNode node, ll l, ll r, ll i, ll j) {
        if (!node || r < i || l > j) return LLONG_MAX;
        if (l >= i && r <= j) return node->v;
        int m = (l+r)>>1;
        return min(query(node->l, l, m, i, j), query(node->r, m + 1, r, i, j));
}

pNode root = NULL;
```

3.3.3 Segment Tree Dynamic Implementação 2

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

using namespace std;
typedef long long ll;

struct node{
    ll val, acum;
    node *l, *r;

    node(){
        val = acum = 0;
        l = r = NULL;
    }
} *root;

void init(){
    root = new node();
}
void add(node* no,ll l, ll r, ll v){
    no->val += (r-l+1)*v;
    no->acum += v;
}

void update(node* no, ll l, ll r, ll x, ll y, ll v){
    if(r < x || y < l) return;
    else if(x <= l && r <= y) add(no, l, r,  v);
    else{
        ll mid = (l+r)/2;

        no->l = (no->l == NULL)? new node(): no->l;
        no->r = (no->r == NULL)? new node(): no->r;
        add(no->l, l, mid, no->acum);
        add(no->r, mid+1, r,no->acum);
        no->acum = 0;

        update(no->l, l, mid, x, y, v);
        update(no->r, mid+1, r, x, y, v);

        no->val = no->l->val + no->r->val;
    }
}

ll query(node* no, ll l, ll r, ll x, ll y){
    if(r < x || y < l) return 0;
    else if(x <= l && r <= y) return no->val;
    else{
        ll mid = (l+r)/2;

        no->l = (no->l == NULL)? new node(): no->l;
        no->r = (no->r == NULL)? new node(): no->r;
        add(no->l, l, mid, no->acum);
        add(no->r, mid+1, r,no->acum);
        no->acum = 0;

        return query(no->l, l, mid, x, y) +
               query(no->r, mid+1, r, x, y);

    }
}

int main(){
    ll n, q; cin >> n >> q;

    init();
    while(q--){
        ll t; cin >> t;

        if(t == 1){
            ll l, r, v; cin >> l >> r >> v;
            update(root, 0, n-1, l, r, v);
        }else{
            ll l, r; cin >> l >> r;
            cout << query(root, 0, n-1, l, r) << endl;
        }
    }
    return 0;
}
```

3.3.4 Segment Tree Dynamic (With Lazy)

```cpp
typedef long long ll;

struct Node {
        ll v;
        ll acum;
        Node *l, *r; //Nós filhos
        Node(ll _v = 0LL):
                v(_v), acum(0LL), l(NULL), r(NULL) {}
};

typedef Node *pNode;


void propagate(pNode node, int l, int r){
        if (node->acum) {
                node->v += (r-l+1) * node->acum;
                if (l != r) {
                        node->l->acum += node->acum;
                        node->r->acum += node->acum;
                }
                node->acum = 0LL;
        }
}


ll merge(pNode node) {
        pNode l = node->l, r = node->r;
        if (!l) node->v = r->v;
        else if (!r) node->v = l->v;
        else node->v = l->v + r->v;
}

void update(pNode &node, ll l, ll r, ll i, ll j, ll v) {
        if (r < i || l > j) return;
        if (!node) node = new Node();
        if(l >= i && r <= j) {
                node->acum += v;
                return;
        }
        propagate(node, l, r);
        ll m = (l+r)>>1;
        update(node->l, l, m, i, j, v);
        update(node->r, m+1, r, i, j, v);
        merge(node);
}

ll query(pNode node, ll l, ll r, ll i, ll j) {
        if (!node || r < i || l > j) return 0LL;
        if (l >= i && r <= j) return node->v;
        ll m = (l+r)>>1;
        return (
                query(node->l, l, m, i, j) +
                query(node->r, m + 1, r, i, j)
                );
}

pNode root = NULL;
```

3.3.5 Segment Tree Persistente

```cpp
#include "bits/stdc++.h"
#define MAX 100007
#define MID 10007
#define INF 1e9 + 7

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<int, pii> piii;

struct node {
    ll val;
    node* l, *r;
    node() {}
    node(node* _l, node* _r, ll v) {
        l = _l;
        r = _r;
        val = v;
    }
};

node* version[2*MID];

ll check(node* no){
    return no? no->val: 0;
}

node* update(node* no, int l, int r, int x, ll value) {
    if (l == r) return new node(NULL, NULL, no->val+value);

    int mid = (l+r) / 2;

    node* ret = new node(no->l, no->r, 0);
    if(x <= mid) {
        if(no->l == NULL) ret->l = update(new node(NULL, NULL, 0), l, mid, x, value);
        else ret->l = update(no->l, l, mid, x, value);
    }else {
        if(no->r == NULL) ret->r = update(new node(NULL, NULL, 0), mid+1, r, x, value);
        else ret->r = update(no->r, mid+1, r, x, value);
    }
```

```
    ret->val = check(ret->l) + check(ret->r);
    //cout << ret->val << " " << l << " " << r << endl;
    return ret;
}


ll query(node* no, int l, int r, int x, int y) {
    if (x > r || y < l || l > r) return 0;
    if (x <= l && r <= y) return no->val;
    int mid = (l+r) >> 1;

    int q1 = (no->l != NULL)? query(no->l,l, mid,x,y): 0;
    int q2 = (no->r != NULL)? query(no->r,mid+1,r,x,y): 0;
    return q1 + q2;
}

int main(){
        return 0;
}
```

3.4 SQRT Decomposition

3.4.1 Implementação 1

```
//SQRT Decomposition to find sum of any interval in O(sqrt(n))

int block[sqrt_maxn];
int values[maxn];
int n, groups;

int query(int l, int r){
    int sum = 0;
    for(int i=l; i<=r;){
        if(i%groups==0 && i+groups-1 <= r){
            sum += block[i/groups];
            i += groups;
        }
        else sum += values[i++];
    }
    return sum;
}

void build() {
    groups = sqrt(n);
    for(int i=0; i<n; i++){
        block[i/groups] += values[i];
    }
}

int update(int i, int v) {
    block[i/groups] += (v - values[i]);
    values[i] = v;
}
```

3.4.2 Implementação 2

```
#include <bits/stdc++.h>
#define MAX 100007
#define SQ 327
#define  mp make_pair

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

struct SQRTT {
    int bucket[MAX], freq[SQ][MAX];
    pii interval[SQ][SQ];
    int ini[MAX], fim[MAX], a[MAX], n;

    int actual[MAX];
    void build(){
        memset(ini, -1, sizeof ini);
            memset(fim, -1, sizeof fim);
        // Delimitando intervalos dos buckets
        bucket[0] = 0; ini[0] = 0;
        for(int i = 1; i < n; i++){
            bucket[i] = i/SQ;
            ini[bucket[i]] = (bucket[i] != bucket[i-1])? i: ini[bucket[i]];
            fim[bucket[i-1]] = (bucket[i] != bucket[i-1])? i-1: fim[bucket[i-1]];
        }
        fim[bucket[n-1]] = n-1;

        // Acumulando
        for(int i = 0; i < n; i++) freq[bucket[i]][a[i]]++;

        for(int i = 1; i < SQ && ini[i] != -1; i++){
            for(int j = 0; j < MAX; j++) freq[i][j] += freq[i-1][j];
        }

        // pegando a melhor resposta entre os buckets
        for(int i = 0; i < SQ && ini[i] != -1; i++){
            memset(actual, 0, sizeof actual);
            int best = a[ini[i]];
            for(int j = ini[i]; j < n; j++){
                actual[a[j]]++;

                if(actual[a[j]] == actual[best]) best = min(a[j], best);
                else if(actual[a[j]] > actual[best]) best = a[j];

                if(fim[bucket[j]] == j) {interval[i][bucket[j]].first = best; interval[i][bucket[j]].
                        second = actual[best];}
            }
        }
        memset(actual, 0, sizeof actual);
    }

    int query(int l, int r){
        if(bucket[l] == bucket[r]){
            int best = l;
            for(int i = l; i <= r; i++){
                actual[a[i]]++;
                if(actual[a[i]] == actual[best]) best = min(a[i], best);
                else if(actual[a[i]] > actual[best]) best = a[i];
            }

            // limpando
            for(int i = l; i <= r; i++) actual[a[i]] = 0;
            return best;
        }

        int bl = bucket[l]+1, br = bucket[r]-1;
        pii best;
        if(bl <= br) best = interval[bl][br];
        else best = mp(-1, -1);

        for(int i = l; i <= fim[bucket[l]]; i++) actual[a[i]]++;
        for(int i = ini[bucket[r]]; i <= r; i++) actual[a[i]]++;

        for(int i = l; i <= fim[bucket[l]]; i++) {
            int total = freq[br][a[i]];
            total -= freq[bl-1][a[i]];
            total = total + actual[a[i]];
            if(total == best.second) best.first = min(best.first, a[i]);
            else if(total > best.second) {best.first = a[i]; best.second = total;}
        }
        for(int i = ini[bucket[r]]; i <= r; i++) {
            int total = freq[br][a[i]];
            total -= freq[bl-1][a[i]];
            total = total + actual[a[i]];
            if(total == best.second) best.first = min(best.first, a[i]);
            else if(total > best.second) {best.first = a[i]; best.second = total;}
        }

        // limpando
        for(int i = l; i <= fim[bucket[l]]; i++) actual[a[i]] = 0;
        for(int i = ini[bucket[r]]; i <= r; i++) actual[a[i]] = 0;
        return best.first;
    }
} ds;

int main(){
    int m; scanf("%d %d", &ds.n, &m);

    for(int i = 0; i < ds.n; i++) scanf("%d", &ds.a[i]);

    ds.build();
    while((m--)){
        int l, r; scanf("%d %d", &l, &r);
        printf("%d\n", ds.query(l, r));
    }

    return 0;
}
```

3.5 MO's

3.5.1 Mo's comum

```
//Mo's algorithm to find frequency of x on any interval
```

```
struct query{
    int id, left, right, x;
};

int values[maxn], freq[maxn], answer[maxn];
vector<query> qry;
int n, q, groups; //groups = sqrt(n)

bool comp(query a, query b){
    if (a.left/groups != b.left/groups) return (a.left/groups < b.left/groups);
    else return (a.right < b.right);
}

void build() {
    groups = sqrt(n);
    int l, r, x;
    for(int i=0; i<q; i++){
        sc("%d %d %d", &l, &r, &x);
        //Diminua 1 de l e r caso a questão seja 1-indexada
        qry.pb({i, l, r, x});
    }
}

void add(int x){ freq[values[x]]++; }

void remove(int x){ freq[values[x]]--; }

void mos(){
    sort(qry.begin(), qry.end(), comp);
    int L = 0, R = -1;
    int l, r;
    for(int i=0; i<q; i++){
        l = qry[i].left;
        r = qry[i].right;

        while(r > R) add(++R);
        while(r < R) remove(R--);

        while(l > L) remove(L++);
        while(l < L) add(--L);
        answer[qry[i].id] = freq[qry[i].x];
    }
}
```

3.5.2 Mo's com update

```
#include <bits/stdc++.h>
#define MAX 200007
#define BUCKET 2500

using namespace std;

typedef long long ll;

struct upd{int pos,velho,novo;};
struct que{int l,r,t,id;};

bool comp(que A,que B){

        if(A.l/BUCKET != B.l/BUCKET ) return A.l/BUCKET < B.l/BUCKET;
        if(A.r/BUCKET != B.r/BUCKET) return A.r/BUCKET < B.r/BUCKET;
        return A.t < B.t;
}

int mleft,mright,mtime;
int freq[MAX],arr[MAX],angry[MAX];
ll exibir[MAX],resp;
vector<que> Q;
vector<upd> U;

void add(int val){
        freq[val]++;
        resp += (freq[val] == 1);
    resp -= (freq[val] == 2);
}
void remove(int val){
        resp -= (freq[val] == 1);
    resp += (freq[val] == 2);
        freq[val]--;
}
void update(int idx,int novo){
        if(mleft <= idx && idx <= mright){
                remove(arr[idx]);
                arr[idx] = novo;
                add(arr[idx]);
        }else arr[idx] = novo;
}
void query(int idx){
    // Tempo
        for(int t = mtime+1;t<=Q[idx].t; t++) update(U[t-1].pos,U[t-1].novo);
        for(int t = mtime;t>Q[idx].t; t--) update(U[t-1].pos,U[t-1].velho);

    // Add
        for(int i = mright + 1;i <= Q[idx].r; i++) add(arr[i]);
        for(int i = mleft - 1;i >= Q[idx].l; i--) add(arr[i]);

    // Remove
        for(int i = mright;i > Q[idx].r; i--) remove(arr[i]);
        for(int i = mleft;i<Q[idx].l; i++) remove(arr[i]);
        mleft = Q[idx].l; mright = Q[idx].r; mtime = Q[idx].t;
        exibir[Q[idx].id] = resp;
}

void mos(int n){
    sort(Q.begin(),Q.end(),comp);
        for(int i = 0;i < n;i++) add(arr[i]);
        mleft = 0; mright = n-1;
        for(int i = 0; i < Q.size();i++) query(i);
}
int main(){
        int n, q; scanf("%d %d",&n, &q);
        for(int i = 0; i< n; i++) scanf("%d", &arr[i]);

    mtime = 0;
        for(int i = 0; i < q;i++){
                int tipo,l ,r; scanf("%d %d %d",&tipo,&l,&r);
                if(tipo == 2){
                        que aux;
                        aux.l = l; aux.r = r; aux.t = mtime; aux.id = i;
                        Q.push_back(aux);
                }else{
                        mtime++;
                        upd aux;
                        aux.pos = l; aux.velho = arr[l]; aux.novo = r;
                        arr[l] = r;
                        U.push_back(aux);
                        exibir[i] = -1;
                }
        }
        mos(n);
        for(int i = 0;i < q; i++) {
        if(exibir[i] != -1) printf("%lld\n",exibir[i]);
    }
        return 0;
}
```

3.6 Link-Cut Tree

```
#define INF 0x3f3f3f3f

/*
 * Link cut tree
 */

struct node {
        int sw, id, w;
        node *par, *ppar, *ls, *rs;
        node() {
                par = ppar = ls = rs = NULL;
                w = sw = INF;
        }
};

class LinkCutTree {
        vector<node> lct;
        int sum(node *p) { return p ? p->sw : 0; }
        void refresh(node* p) {
                p->sw = p->w + sum(p->ls) + sum(p->rs);
        }
        void rotateright(node* p) {
                node *q, *r;
                q = p->par, r = q->par;
                if (q->ls = p->rs) q->ls->par = q;
                p->rs = q, q->par = p;
                if (p->par = r) {
                        if (q == r->ls) r->ls = p;
                        else r->rs = p;
                }
                p->ppar = q->ppar;
                q->ppar = NULL;
                refresh(q);
        }
        void rotateleft(node* p) {
                node *q, *r;
                q = p->par, r = q->par;
                if (q->rs = p->ls) q->rs->par = q;
                p->ls = q, q->par = p;
```

```
        if (p->par = r) {
            if (q == r->ls) r->ls = p;
            else r->rs = p;
        }
        p->ppar = q->ppar;
        q->ppar = 0;
        refresh(q);
    }
    void splay(node* p) {
        node *q, *r;
        while (p->par != NULL) {
            q = p->par;
            if (q->par == NULL) {
                if (p == q->ls) rotateright(p);
                else rotateleft(p);
                continue;
            }
            r = q->par;
            if (q == r->ls) {
                if (p == q->ls) rotateright(q), rotateright(p);
                else rotateleft(p), rotateright(p);
            }
            else {
                if (p == q->rs) rotateleft(q), rotateleft(p);
                else rotateright(p), rotateleft(p);
            }
        }
        refresh(p);
    }
    node* access(node* p) {
        splay(p);
        if (p->rs != NULL) {
            p->rs->ppar = p; p->rs->par = NULL;
            p->rs = NULL; refresh(p);
        }
        node* last = p;
        while (p->ppar != NULL) {
            node* q = last = p->ppar;
            splay(q);
            if (q->rs != NULL) {
                q->rs->ppar = q;
                q->rs->par = NULL;
            }
            q->rs = p; p->par = q;
            p->ppar = NULL;
            refresh(q); splay(p);
        }
        return last;
    }
public:
    LinkCutTree(int n = 0) {
        lct.resize(n + 1);
        for(int i = 0; i <= n; i++) {
            lct[i].id = i;
            refresh(&lct[i]);
        }
    }
    void link(int u, int v, int w = 1) {
        //u becomes child of v
        node *p = &lct[u], *q = &lct[v];
        access(p); access(q);
        p->ls = q; q->par = p; p->w = w;
        refresh(p);
    }
    void cut(int u) {
        node* p = &lct[u]; access(p);
        p->ls->par = NULL; p->ls = NULL;
        refresh(p);
    }
    int findroot(int u) {
        node* p = &lct[u]; access(p);
        while (p->ls) p = p->ls;
        splay(p);
        return p->id;
    }
    bool IsSameTree(int u, int v) {
        return findroot(u) == findroot(v);
    }
    int depth(int u) {
        access(&lct[u]);
        return lct[u].sw - lct[u].w;
    }
    int LCA(int u, int v) {
        access(&lct[u]);
        return access(&lct[v])->id;
    }
    int dist(int u, int v) {
        if (!IsSameTree(u, v)) return INF;
        node* lca = &lct[LCA(u, v)];
        int ans = 0;
        access(&lct[u]); splay(lca);
        ans += sum(lca->rs);
        access(&lct[v]); splay(lca);
        ans += sum(lca->rs);
        return ans;
    }
};

/*
 * Undirected link cut tree
 */

class UndirectedLinkCutTree {
    LinkCutTree lct;
    vector<int> par;

    void invert(int u) {
        if (par[u] == -1) return;
        int v = par[u];
        invert(v);
        lct.cut(u); par[u] = -1;
        lct.link(v, u); par[v] = u;
    }
public:
    UndirectedLinkCutTree(int n = 0) {
        lct = LinkCutTree(n);
        par.assign(n+1, -1);
    }
    void link(int u, int v) {
        if (lct.depth(u) < lct.depth(v)) {
            invert(u);
            lct.link(u, v); par[u] = v;
        }
        else {
            invert(v);
            lct.link(v, u); par[v] = u;
        }
    }
    void cut(int u, int v) {
        if (par[v] == u) u = v;
        lct.cut(u); par[u] = -1;
    }
    bool IsSameTree(int u, int v) {
        return lct.IsSameTree(u, v);
    }
};

int main() {
    //Link-Cut com N vértices
        LinkCutTree lct(N+1);
    //u é filho de v
    lct.link(u, v);
    //u é cortado da árvore que está e passa a ser raís de sua própria árvore
    lct.cut(u);
    //pega o LCA entre u e v
    int lca = lct.LCA(u, v);
        return 0;
}
```

4 Graph

4.1 Centroid Decomposition

```
vi g[100001];
vi centroidG[100001];
int visited[100001];
bool marked[100001];
int subtree[100001];
int dfsId = 0;
int dfs(int i) {
    subtree[i] = 1; visited[i] = dfsId;
    for(int adj : g[i]) {
        if (visited[adj] != dfsId && !marked[adj])
            subtree[i] += dfs(adj);
    }
    return subtree[i];
}
int getCentroid(int node) {
    dfsId++;
    int sz = dfs(node);
    bool changed = true;
    while(changed) {
        changed = false;
        for(int adj : g[node]) {
            if(subtree[adj] < subtree[node] && subtree[adj] > sz / 2) {
                node = adj;
                changed = true; break;
```

```
                }
            }
        }
        marked[node] = true;
        for(int adj : g[node]) {
            if(!marked[adj]) {
                // Para fazer uma árvore não direcionada
                // int cent = getCentroid(adj);
                // centroidG[node].pb(cent);
                // centroidG[cent].pb(node);
                centroidG[node].pb(getCentroid(adj));
            }
        }
        return node;
}

// No main:
// memset(marked, false, sizeof marked);
// memset(visited, 0, sizeof visited);
// int root = getCentroid(1);
```

Tested on:
https://codeforces.com/contest/321/problem/C

4.2 Bridges Tree Decomposition

É uma forma de reduzir um grafo em seus componentes conectados formando uma árvore em que os vértices são os componentes e as arestas são as pontes do grafo original.
Pode ser utilizado em problemas que envolvem contar quantidade de pontes entre vértices ou que envolvem verificar se os vértices estão em componentes duplamente conectados

```
//Necessita de um Union-Find
map<int, int> corresp;
vector<int> g[maxn];
vector<int> tree[maxn];
int he[maxn];

void dfs1(int v, int anc) {
    int mini = he[v];
    for(int i : g[v]) {
        if (i != anc) {
            if (he[i] == -1) {
                he[i] = he[v]+1;
                dfs1(i, v);
            }
            if (he[i] <= he[v]) {
                Union(i, v);
            }
            mini = min(mini, he[i]);
        }
    }
    he[v] = mini;
}

void dfs2(int v) {
    he[v] = 1;
    for(int i : g[v]) {
        if (!he[i]) {
            he[i] = 1;
            dfs2(i);
            i = find(i); v = find(v);
            if (i != v) {
                tree[corresp[i]].pb(corresp[v]);
                tree[corresp[v]].pb(corresp[i]);
            }
        }
    }
}

//Retorna a quantidade de nós da árvore
int createBridgeTreeFromGraph() {
    memset(he, -1, sizeof he);
    for(int i = 0; i < n; i++) {
        if(he[i] == -1) {
            he[i] = 0;
            dfs1(i, -1);
        }
    }
    int j = 0;
    for(int i = 0; i < n; i++) {
        if (!corresp.count(find(i))) corresp[find(i)] = j++;
    }
    memset(he, 0, sizeof he);
    for(int i = 0; i < n; i++) {
        if(!he[i]) {
            dfs2(i);
        }
    }
    return j;
}
```

4.3 Max Flow

4.3.1 Push-Relabel $O(V^2 * E)$

```
namespace PushRelabel {
const int maxn = 200500;

struct Edge {
    int to, c, f;
};
vector<Edge> edge;

int n;
vector<int> g[maxn];
ll e[maxn];
int h[maxn];
int onH[maxn];
int S, T;
int ptr[maxn];
int relabelTimer;

void addEdge(int u, int v, int c) {
    g[u].push_back(sz(edge));
    edge.push_back({v, c, 0});
    g[v].push_back(sz(edge));
    edge.push_back({u, 0, 0});
}

void push(int id, int delta) {
    int u = edge[id ^ 1].to;
    int v = edge[id].to;
    edge[id].f += delta;
    edge[id ^ 1].f -= delta;
    e[u] -= delta;
    e[v] += delta;
}

void gap(int ch) {
    forn (u, n) {
        if (h[u] > ch)
            h[u] = max(h[u], n);
    }
}

int o[maxn];
void globalRelabeling() {
    int oc = 0;
    forn (i, n) {
        h[i] = n;
        onH[i] = 0;
    }
    onH[0] = 1;
    h[T] = 0;
    o[oc++] = T;
    forn (ii, oc) {
        int u = o[ii];
        for (int id: g[u]) {
            if (edge[id ^ 1].c == edge[id ^ 1].f)
                continue;
            int v = edge[id].to;
            if (h[v] != n)
                continue;
            h[v] = h[u] + 1;
            onH[h[v]]++;
            o[oc++] = v;
        }
    }
}

void relabel(int u) {
    int oldh = h[u];
```

```
    int newh = inf;
    for (int id: g[u]) {
        if (edge[id].c == edge[id].f)
            continue;
        newh = min(newh, h[edge[id].to] + 1);
    }
    h[u] = newh;
    onH[oldh]--;
    onH[newh]++;
    if (onH[oldh] == 0)
        gap(oldh);
    if (++relabelTimer == n)
        globalRelabeling(), relabelTimer = 0;
}

void discharge(int u) {
    while (e[u] > 0) {
        int &i = ptr[u];
        if (i == sz(g[u])) {
            i = 0;
            relabel(u);
            if (h[u] >= n)
                break;
            continue;
        } else {
            int id = g[u][i++];
            int v = edge[id].to;
            if (h[v] + 1 != h[u])
                continue;
            int delta = min(e[u], ll(edge[id].c - edge[id].f));
            push(id, delta);
        }
    }
}

ll flow(int _S, int _T) {
    S = _S, T = _T;
    forn (i, n)
        ptr[i] = 0, e[i] = 0;
    for (int id: g[S]) {
        int delta = edge[id].c;
        push(id, delta);
    }
    globalRelabeling();
    bool ok = false;
    while (!ok) {
        ok = true;
        forn (u, n) {
            if (h[u] < n && u != T && e[u] > 0)
                discharge(u), ok = false;
        }
    }
    return e[T];
}

} //PushRelabel
//END_CODE

int main() {
    int n, m;
    cin >> n >> m;
    PushRelabel::n = n;
    forn (i, m) {
        int u, v, c;
        cin >> u >> v >> c;
        --u, --v;
        PushRelabel::addEdge(u, v, c);
    }
    cout << PushRelabel::flow(0, n - 1) << '\n';
}
```

Tested on:
https://www.spoj.com/problems/FASTFLOW/

4.3.2 Dinic $O(V^2 * E)$

```
#include <bits/stdc++.h>
#define MAX 100007
#define INF 1e9 + 7

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};

int WALL = 50007, INI = 0, FIM = MAX-1;
int main(){
    Dinic graph = Dinic(MAX, INI, FIM);
        return 0;
}
```

4.3.3 Ford-Fulkerson $O(V * |F|)$

```
int cap[MAXN][MAXN];
int fnet[MAXN][MAXN];
int q[MAXN], qf, qb, pre[MAXN];
```

```
int fordFulkerson( int n, int s, int t )
{
    // init the flow network
    memset( fnet, 0, sizeof fnet);

    int flow = 0;

    while( true )
    {
        // find an augmenting path
        memset( pre, -1, sizeof pre);
        qf = 0; qb = 0;
        pre[q[qb++] = s] = -2;
        while( qb > qf && pre[t] == -1 )
            for( int u = q[qf++], v = 0; v < n; v++ )
                if( pre[v] == -1 && fnet[u][v] - fnet[v][u] < cap[u][v] )
                    pre[q[qb++] = v] = u;

        // see if we're done
        if( pre[t] == -1 ) break;

        // get the bottleneck capacity
        int bot = 1e9;
        for( int v = t, u = pre[v]; u >= 0; v = u, u = pre[v])
            bot = min(bot, cap[u][v] - fnet[u][v] + fnet[v][u]);
        // update the flow network
        for( int v = t, u = pre[v]; u >= 0; v = u, u = pre[v] )
            fnet[u][v] += bot;

        flow += bot;
    }

    return flow;
}
```

4.3.4 Min-Cost Max-Flow

```
#include <bits/stdc++.h>
#define MAX 10007
#define INF 1000000007

using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

struct edge{
    int to, cap, cost, rev;

    edge(int _to, int _cap, int _cost, int _rev): to(_to), cap(_cap), cost(_cost), rev(_rev){}
};

struct cost{
    vector<edge> graph[MAX];
    int pre[MAX], pre_e[MAX], inq[MAX], F, C;
    int dist[MAX];

    cost(){
        F = 0;
        C = 0;
        fill(graph, graph+MAX, vector<edge>());
    }

    void add(int from, int to, int cap, int cost){
        graph[from].push_back(edge(to, cap, cost, graph[to].size()));
        graph[to].push_back(edge(from, 0, -cost, graph[from].size()-1));
    }

    bool bfs(int ini, int fim){
        fill(dist, dist+MAX, INF);
        fill(inq, inq+MAX, 0);

        dist[ini] = 0;
        queue<int> q;
        q.push(ini);
        while(!q.empty()){
            int u = q.front();
            q.pop();

            inq[u] = 0;
            for(int i = 0; i < graph[u].size(); i++){
                int v = graph[u][i].to;
                int w = graph[u][i].cost;
                if(graph[u][i].cap > 0 && dist[v] > dist[u]+w){
                    pre[v] = u;
                    pre_e[v] = i;
                    dist[v] = dist[u] + w;
                    if(!inq[v]){
                        inq[v] = true;
                        q.push(v);
                    }
                }
            }
        }

        return dist[fim] != INF;
    }
    pii flow(int ini, int fim){
        while(bfs(ini, fim)){
            int tf = INF;
            for(int v = fim, u, l; v != ini; v = u){
                u = pre[v];
                l = pre_e[v];
                tf = min(tf, graph[u][l].cap);
            }

            for(int v = fim, u, l; v != ini; v = u){
                u = pre[v];
                l = pre_e[v];
                graph[u][l].cap -= tf;
                graph[v][graph[u][l].rev].cap += tf;
            }
            C += tf*dist[fim];
            F += tf;
        }
        return make_pair(F, C);
    }
};

int T = 0, P = MAX-1;
int main(){
    return 0;
}
```

4.3.5 Gomory Hu Tree

```
#define N 10100
#define LOGN 14
#define INF 2000000000

int n, m;
vector <Edge> graph[N]; //graph used to calculate_flow
vector <Edge> backupGraph[N]; //backup for graph

/*
 * Gomory Hu Tree / Gusfield's Algorithm - O((V-1)*(Flow Algorithm))
 * After preprocessing, queries about min-cost / max flow from sink
 * to target are solved in log(V)
 */

typedef pair<int, int> ii;
int par[N], fpar[N];    //parent in tree, flow to parent

bool inCut[N];
vector<ii> ghtree[N]; // pair defined as <next_node, edge_cost>

void copyGraph() {
    for(int i = 1; i <= n; i++) backupGraph[i] = graph[i];
}

void restoreGraph() {
    for(int i = 1; i <= n; i++) graph[i] = backupGraph[i];
}

void cutGraph(int s) {
        inCut[s] = true;
    for(Edge e : graph[s]) {
        int v = e.to;
        if (!inCut[v] && e.used_flow < e.cap) {
                    cutGraph(v);
                }
    }
}

int computeMinCut(int s, int t) {
    copyGraph();
        memset(inCut, false, sizeof inCut);
        int f = flow(s, t);
        cutGraph(s);
        restoreGraph();
        return f;
}

//1-indexed
void gomoryhu() {
        for (int i = 1; i <= n; i++) par[i] = 1;
        for (int s = 2; s <= n; s++) {
                int t = par[s];
                fpar[s] = computeMinCut(s, t);
                for (int u = s+1; u <= n; u++) {
```

```
                        if (par[u] == t && inCut[u]) {
                                par[u] = s;
                        }
                        if (par[t] == u && inCut[u]) {
                                par[s] = u; par[t] = s;
                                swap(fpar[s], fpar[t]);
                        }
                }
        }
        for (int i = 1; i <= n; i++) ghtree[i].clear();
        for (int i = 1; i <= n; i++) {
                if (par[i] == i) continue;
                ghtree[i].push_back(ii(fpar[i], par[i]));
                ghtree[par[i]].push_back(ii(fpar[i], i));
        }
}

int pa[LOGN][N];
int mini[LOGN][N];
int height[N];
void dfs(int node) {
        for (ii child: ghtree[node]) {
                if(child.second == pa[0][node]) continue;
                pa[0][child.second] = node;
                mini[0][child.second] = child.first;
                height[child.second] = height[node]+1;
                dfs(child.second);
        }
}
void buildLCA() {
        dfs(1);
        for(int i = 1; i < LOGN; i++) {
                for(int j = 1; j <= n; j++) {
                        pa[i][j] = pa[i-1][pa[i-1][j]];
                        mini[i][j] = min(mini[i-1][j], mini[i-1][pa[i-1][j]]);
                }
        }
}

int query(int a, int b) {
        int ret = INT_MAX;
        if (height[a] < height[b]) swap(a, b);
        int pot = 1<<13, i = 13;
        for(; i >= 0; i--) {
                if (height[a]-pot >= height[b]) {
                        ret = min(ret, mini[i][a]);
                        a = pa[i][a];
                }
                pot >>= 1;
        }

        if(a == b) return ret;

        pot = 1<<13, i = 13;
        for(; i >= 0; i--) {
                if (pa[i][a] != pa[i][b]) {
                        ret = min(ret, mini[i][a]);
                        ret = min(ret, mini[i][b]);
                        a = pa[i][a];
                        b = pa[i][b];
                }
                pot >>= 1;
        }

        ret = min(ret, mini[0][a]);
        ret = min(ret, mini[0][b]);
        return ret;
}

int main() {
        int u, v, cap;
        scanf("%d %d", &n, &m);
        for (int i = 0; i < m; i++) {
                scanf("%d %d", &u, &v);
                addEdge(u, v, 1);
                addEdge(v, u, 1);
        }

        gomoryhu();
        buildLCA();
        printf("%d\n", query(1, n));
}
```

4.4 Dominator Tree

```
struct Dom {
    int n;
    vector<vi> e, re;
    vi id, p, sdom, dom, dsu, best;
    vector<vi> bucket;
    int dtime = 0;

    Dom() {}
    Dom(int n) : n(n), e(n), re(n), id(n, -1), p(n),
    sdom(n), dom(n), dsu(n), best(n), bucket(n) {}

    void find(int v) {
        if (v != dsu[v]) {
            find(dsu[v]);
            if (sdom[best[dsu[v]]] <= sdom[best[v]]) {
                best[v] = best[dsu[v]];
            }
            dsu[v] = dsu[dsu[v]];
        }
    }

    void dfs1(int v) {
        id[v] = dtime++;
        for(int to: e[v]) {
            if (id[to] == -1) {
                dfs1(to);
                p[id[to]] = id[v];
            }
            re[id[to]].pb(id[v]);
        }
    }

    void pre(int root) {
        dfs1(root);
        //fils array from l to r with 0,1,2,3....n
        //iota(left iterator, right iterator, initial value);
        iota(best.begin(), best.end(), 0);
        iota(sdom.begin(), sdom.end(), 0);
        iota(dsu.begin(), dsu.end(), 0);
    }

    void run(int root) {
        pre(root);
        for(int v = n-1; v >= 0; v--) {
            for(int w: bucket[v]) {
                find(w);
                dom[w] = best[w];
            }
            for(int u: re[v]) {
                find(u);
                sdom[v] = min(sdom[v], sdom[best[u]]);
            }
            if (v){
                bucket[sdom[v]].pb(v);
                dsu[v] = p[v];
            }
        }

        for (int v = 1; v < n; v++) {
            if (dom[v] != sdom[v]) {
                dom[v] = dom[dom[v]];
            }
        }
        vi ndom(n), rev(n);

        // Caso não seja possível chegar em algum vértice no intervalo 0...n-1 a partir do root,
        // id[i] dos vértices inalcançáveis será -1, o que causaraá runtime error nessa parte do có
            digo.
        // Obs: para evitar isso, todos os nós entre 0 e n-1 devem realmente representar um nó alcançá
            vel
        // a partir do root
        for(int i = 0; i < n; i++) rev[id[i]] = i;
        for(int i = 0; i < n; i++) ndom[i] = rev[dom[id[i]]];
        dom = ndom;
    }
};

int n, m, root, a, b;
int main(){
    scanf("%d %d %d", &n, &m, &root);
    Dom d(n);
    for(int i = 0; i < m; i++) {
        sc("%d %d", &a, &b);
        d.e[a-1].pb(b-1);
    }
    d.run(root);
    // O dominator imediato de cada vértice v (nó mais próximo de v que está em todos os caminhos do
    // vértice root para o vértice v) pode ser acessado agora com d.dom[v]
    // Obs: O vértice root domina ele mesmo
        return 0;
}
```

Tested on:

Disjoint water supply (Latin America 2013)

4.5 Heavy-Light Decomposition

```
int parent[MAXN], depth[MAXN], heavy[MAXN], head[MAXN], pos[MAXN], segTree[4*MAXN], value[MAXN],
    nodeByPos[MAXN];
vector<vi> adj;
int cur_pos;

void mergeSegTree(int id);
void buildSegTree(int id, int l, int r);
void updateSegTree(int id, int l, int r, int i, int val);
int querySegTree(int id, int l, int r, int i, int j);

int dfs(int v) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

int decompose(int v, int h) {
    head[v] = h, pos[v] = cur_pos, nodeByPos[cur_pos++] = v;
    if (heavy[v] != -1) decompose(heavy[v], h);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c);
    }
}

void init() {
        memset(heavy, -1, sizeof heavy);
    cur_pos = 0;
    dfs(0);
    decompose(0, 0);
        buildSegTree(0, 0, n-1);
}

int merge(int a, int b) {
    //return union_of_partial_responses Ex: max(a,b)
    return max(a, b);
}

int query(int a, int b) {
    //res = NEUTRAL_ELEMENT
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path = querySegTree(0, 0, n-1, pos[head[b]], pos[b]);
        res = merge(res, cur_heavy_path);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path = querySegTree(0, 0, n-1, pos[a], pos[b]);
    res = merge(res, last_heavy_path);
    return res;
}

int main() {
    adj = vector<vi>(n, vi());
    // Pega entrada, preenche lista value[] e preenche grafo adj;
    init();
    // NÃOOOO precisa dividir queries (a, b) em (a, lca(a, b)) e (b, lca(a, b))
    // pois o método query(a, b) já resolve tudo
    return 0;
}
```

4.6 Djikstra

```
vector<int> g[MAX];
vector<int> p[MAX];
int vis[MAX];
int dis[MAX];
int parent[MAX];

void dijkstra(int no){

        priority_queue<pair<pair<int, int>, int>, vector<pair<pair<int, int>, int> >, greater<pair<
              pair<int, int>, int> > > fila;
        fila.push(make_pair(make_pair(0, no), -1));

        while (!fila.empty()){
                int d=fila.top().first.first;
                int v=fila.top().first.second;
                int pai=fila.top().second;

                fila.pop();

                if (vis[v]) continue;

                vis[v]=1;
                dis[v]=d;
                parent[v]=pai;

                for (int i=0; i<(int)g[v].size(); i++){
                        if (! vis[g[v][i]]){
                                fila.push(make_pair(make_pair(d+p[v][i], g[v][i]), v));
                        }
                }
        }
}

int main(){
        int n,m;
        scanf("%d %d", &n, &m);

        for (int i=0; i<m; i++){
                int a,b,w;
                scanf("%d %d %d", &a, &b, &w);

                g[a].push_back(b);
                g[b].push_back(a);
                p[a].push_back(w);
                p[b].push_back(w);
        }

        vector<int> res;
        dijkstra(1);

        if (dis[n]==0){
                printf("-1");
                return 0;
        }
        int r=n;

        while (r!=1){
                res.push_back(r);
                r=parent[r];
        }
        res.push_back(1);

        for (int i=res.size()-1; i>=0; i--){
                printf("%d ", res[i]);
        }
}
```

4.7 Matching

4.7.1 Blossom algorithm

```
#include <bits/stdc++.h>
using namespace std;
#define MAXN 2005
#define INF 1000000009

/* Codechef Problem: A game on a graph
// Blossom algorithm O(N^3)
// https://www.codechef.com/problems/HAMILG
*/
int mQueue[MAXN], qHead, qTail, match[MAXN], cur_blossom[MAXN], parent[MAXN], N;
bool inPath[MAXN], inQueue[MAXN], inBlossom[MAXN];

vector<vector<int> > G;

void push(int u) {
        mQueue[qTail++] = u;
        inQueue[u] = true;
}

int lca(int u, int v) {
        memset(inPath, 0, sizeof(inPath));
```

```
        u = cur_blossom[u];
        while(true) {
                inPath[u] = true;
                if(match[u] == -1 || parent[match[u]] == -1) break;
                u = cur_blossom[parent[match[u]]];
        }

        v = cur_blossom[v];
        while(true) {
                if(inPath[v]) return v;
                v = cur_blossom[parent[match[v]]];
        }

        return v;
}

int find_aug_path(int R) {
        int u, viz;
        memset(inQueue, 0, sizeof(inQueue));
        memset(parent, -1, sizeof(parent));
        for(int i = 0; i < N; i++) cur_blossom[i] = i;
        qTail = qHead = 0;
        push(R);

        while(qHead < qTail) {
                u = mQueue[qHead++];
                for (int i = 0; i < G[u].size(); i++) {
                        viz = G[u][i];
                        if(viz != match[u] && cur_blossom[viz] != cur_blossom[u]) {
                                if(parent[viz] == -1 && !(match[viz] != -1 && parent[match[viz]] !=
                                        -1)) {
                                        if(viz == R) continue;
                                        parent[viz] = u;
                                        if(match[viz] == -1) return viz; // augmenting path to viz
                                        push(match[viz]);
                                        continue;
                                }

                                if(viz != R && !(match[viz] != -1 && parent[match[viz]] != -1))
                                        continue;

                                // blossom found
                                int new_blossom = lca(u,viz);
                                memset(inBlossom, 0, sizeof(inBlossom));

                                int v = u, parent_v = viz;
                                while(cur_blossom[v] != new_blossom) {
                                        inBlossom[cur_blossom[v]] = inBlossom[cur_blossom[match[v]]] =
                                                true;
                                        parent[v] = parent_v;
                                        parent_v = match[v];
                                        v = parent[match[v]];
                                }

                                v = viz, parent_v = u;
                                while(cur_blossom[v] != new_blossom) {
                                        inBlossom[cur_blossom[v]] = inBlossom[cur_blossom[match[v]]] =
                                                true;
                                        parent[v] = parent_v;
                                        parent_v = match[v];
                                        v = parent[match[v]];
                                }

                                for(int i =0; i < N; i++) if(inBlossom[cur_blossom[i]]) {
                                        cur_blossom[i] = new_blossom;
                                        if(!inQueue[i]) {
                                                push(i);
                                        }
                                }
                        }
                }
        }

        return -1;
}

void MaxMatch() {
        memset(match, -1, sizeof(match));

        for (int i = 0; i < N; i++) {
                if (match[i] < 0) {
                        int v = find_aug_path(i);

                        while(v != -1) {
                                int parent_v = match[parent[v]];
                                match[v] = parent[v];
                                match[parent[v]] = v;
                                v = parent_v;
                        }
                }
        }
}
```

```
bool marked[MAXN];

void dfs(int R) {
        int u, viz;
        memset(inQueue, 0, sizeof(inQueue));
        memset(parent, -1, sizeof(parent));

        for(int i =0; i < N; i++) cur_blossom[i] = i;

        qTail = qHead = 0;
        push(R);

        while(qHead < qTail) {
                u = mQueue[qHead++];
                marked[u] = true;
                for (int i = 0; i < G[u].size(); i++) {
                        viz = G[u][i];
                        if(viz != match[u] && cur_blossom[viz] != cur_blossom[u]) {
                                if(parent[viz] == -1 && !(match[viz] != -1 && parent[match[viz]] !=
                                        -1)) {
                                        if(viz == R) continue;
                                        parent[viz] = u;
                                        if(match[viz] == -1) return; // augmenting path to viz
                                        push(match[viz]);
                                        continue;
                                }

                                if(viz != R && !(match[viz] != -1 && parent[match[viz]] != -1))
                                        continue;

                                // blossom found
                                int new_blossom = lca(u,viz);
                                memset(inBlossom, 0, sizeof(inBlossom));

                                int v = u, parent_v = viz;
                                while(cur_blossom[v] != new_blossom) {
                                        inBlossom[cur_blossom[v]] = inBlossom[cur_blossom[match[v]]] =
                                                true;
                                        parent[v] = parent_v;
                                        parent_v = match[v];
                                        v = parent[match[v]];}

                                v =viz, parent_v = u;
                                while(cur_blossom[v] != new_blossom) {
                                        inBlossom[cur_blossom[v]] = inBlossom[cur_blossom[match[v]]] =
                                                true;
                                        parent[v] =parent_v;
                                        parent_v =match[v];
                                        v =parent[match[v]];}

                                for(int i =0; i < N; i++) if(inBlossom[cur_blossom[i]]) {
                                        cur_blossom[i] = new_blossom;
                                        if(!inQueue[i]) {
                                                push(i);
                                        }
                                }
                        }
                }
        }

        return;
}

int main() {
        int t, a, b, M;
        scanf("%d", &t);
        while(t--) {
                scanf("%d %d", &N, &M);

                G.clear();
                G.resize(N+5);

                for (int i = 0; i < M; i++) {
                        scanf("%d %d", &a, &b);a--;b--;
                        G[a].push_back(b);
                        G[b].push_back(a);
                }
                MaxMatch();

                int ans = 0;
                memset(marked, 0, sizeof(marked));

                for (int u = 0; u < N; u++) {
                        if (!marked[u] && match[u] == -1) {
                                dfs(u);
                        }
                }
                for (int u = 0; u < N; u++) {
                        if (marked[u]) ans++;
                }
                printf("%d\n", ans);
```

```
        }
}
```

5 Geometry

5.1 Main structures

5.1.1 Point/Vector structure

```
#define PI (4*atan(1LL))

bool eq(double d1, double d2) { return fabs(d1 - d2) < EPS;}
bool dif(double d1, double d2) { return !eq(d1, d2);}
bool men(double d1, double d2) { return d1 + EPS < d2;}
bool mai(double d1, double d2) { return d1 > d2 + EPS;}

typedef struct pt;
struct pt {double x, y;
    bool operator < (pt o) {
        return eq(x, o.x) ? y < o.y : x < o.x;}
    bool operator == (pt o) {
        return eq(x, o.x) && eq(y, o.y);
    }
    pt operator + (pt o) { //Soma de pontos
        return  {x + o.x, y + o.y};
    }
    pt operator - (pt o) { //Diferença de pontos
        return {x - o.x, y - o.y};
    }
    pt operator * (double o) { // Escalar ponto
        return {x * o, y * o};
    }
    pt operator / (double o) { // Escalar ponto
        return {x / o, y / o};
    }
    double operator * (pt o) { // Produto escalar |this|*|o|*cos(this->o)
        return x * o.x + y * o.y;
    }
    double operator ^ (pt o) { // Produto cruzado |this|*|o|*sin(this->o)
        return x * o.y - y * o.x;
    }
    double operator ~ () { // Comprimento do vetor
        return hypot(x, y);
    }
    double operator | (pt o) { //Distância entre 2 pontos Pre: (pt: -, ~)
        return ~((*this)-o);
    }
    double operator - () { // Comprimento de vetor ao quadrado
        return x * x + y * y;
    }
    pt rotate(double an) { //Rotaciona ponto ao redor da origem em ângulo an (em radianos)
        return {x*cos(an)-y*sin(an), x*sin(an)+y*cos(an)};
    }
    pt rotateAround(pt p, double an) { //Rotaciona ao redor do ponto p em ângulo an(radianos)
    return (((*this)-p).rotate(an))+p;
    }
    double an() { // Ângulo do vetor em relação ao eixo x
        return atan2(y, x);
    }
    pt rot90() { return {-y, x}; }
    pt rot90cw() { return {y, -x}; }
    pt unit() { return (*this)/(~(*this)); }
};

//Para printar ponto com cout/trace
ostream &operator<<(ostream &out, const pt &p) { return out << "{" << p.x << ", " << p.y << "}"; }

// Distância entre ponto p e linha com pointos a e b. Pre: (pt: -, ^, |)
double distToLn(pt p, pt a, pt b) {
    return fabs(((a-p)^(b-p))/(a|b));
}

// Distância entre ponto p e segmento entre pointos a e b. Pre: (pt: -, +, *, *(double), -(), |)
double distToSeg(pt p, pt a, pt b) {
    double u = ((p-a)*(b-a)) / (-(b-a));
    return p|(a + ((b-a) * max(0.0, min(1.0, u))));
}

// Retorna ângulo aob em radianos Pre: (pt: -, an())
// an(a, b, c)*180.0/pi para obter o resultado em graus
double an(pt a, pt o, pt b) {
    return (b-o).an() - (a-o).an();
}
```

```
}

//Retorna se o ponto p está a esquerda da linha que passa de q para r Pre: (pt: -, ^)
bool isLeft( pt p, pt q, pt r ) {
    return mai((r-q)^(p-q), 0.0);
}

//Retorna se os pontos p, q e r são colineares Pre: (pt: -, ^)
bool col(pt p, pt q, pt r) {
    return eq((r-q)^(p-q), 0.0);
}

// Retorna intersecção de linhas ln(a, b) e ln(c, d)
pt linesIntersection(pt a, pt b, pt c, pt d) {
    double s = (b-a)^(d-c);
    if (fabs(s) < EPS) {//parallel or equal lines
        return {1e18, 1e18};
    }
    double s1 = (c-a)^(d-a);
    return a + ((b-a)*(s1/s));
}
```

5.1.2 Line structure

```
struct ln {pt p, v;
    //Cria linha com ponto p e vetor direcional v a partir de pontos _p e q Pre: (pt: -, /(double), ~)
    ln(pt _p, pt q) {
        p = _p; v = (q-p).unit();
    }
    //Cria linha com ponto p e ângulo do vetor direcional an (em radianos)
    ln(pt _p, double an) {
        p = _p; v = {cos(an), sin(an)};
    }
    //Distância entre esta linha e ponto o Pre: (pt: -, +, ^)
    double operator | (pt o) {
        return fabs((p-o)^(p+v-o));
    }
    //Projeção do ponto o nesta linha Pre: (pt: -, +, ^) OBS: Falta testar
    pt proj(pt o) {
        double dist = (*this)|o;
        if (((o-p)^(v)) > 0.0) return o + ((v.rot90())*dist);
        return o + ((v.rot90cw())*dist);
    }
    //Retorna true se esta linha e linha o são paralelas
    bool operator || (ln o) {
        return eq(v^o.v, 0.0);
    }
    //Retorna true se esta linha e linha o são a mesma linha Pre: (pt: +), (ln: ||), col
    bool operator == (ln o) {
        return ((*this)||o) && col(p, p+v, o.p);
    }
};

pt linesIntersection(ln l1, ln l2) {
    if (l1||l2) {
        if (l1 == l2) {
            //stub: equal lines
        } else {
            //stub: empty intersection
        }
    }
        return linesIntersection(l1.p, l1.p+l1.v, l2.p, l2.p+l2.v);
}
```

5.1.3 Segment structure

```
bool intersect(seg s1, seg s2, pt &inter)
{
    pt v1, v2, v3;
    v1 = s1.q-s1.p;
    v2 = s2.q-s2.p;
    v3 = s1.p-s2.p;
    if(eq(v1^v2, 0)) {
                if (eq(v2^v3, 0)) {
                        //Collinear, you should decide what to do
                        if (s2.q < s2.p) swap(s2.p, s2.q);
                        if (s1.q < s1.p) swap(s1.p, s1.q);
                        if (s1.p > s2.p) swap(s1, s2);
                        if (s1.q.y > s2.p.y) {
                                inter = {s2.p}; // Menor ponto da intersecção
                                //inter = {s1.q}; // Maior ponto
                                return true;
                        }
                        return false;
                }
                //line(s1) != line(s2)
                return false;
        }
```

```
    double s = (v1^v3)/(v1^v2);
    double t = (v2^v3)/(v1^v2);

    if (s >= 0 && s <= 1 && t >= 0 && t <= 1) {
        // Collision detected
        inter = s1.p + (v1*t);
        return true;
    }

    return false; // No collision
}
```

5.1.4 Circle structure

```
#define sq(x) ((x)*(x))

struct circle {
    pt c;
    double r;
    circle (pt _c, double _r) { r = _r; c = _c; }
};

// Retorna intersecção de linha l e circulo com centro a e raio r
vector<pt> lineCircleIntersection(ln l, circle ci) {
    pt c = ci.c; double r = ci.r;
    pt p = l.proj(c);
    double d = p|c;
    if (mai(d, r)) return {};
    if (eq(d, r)) return {p};
    double tmp = sqrt(sq(r)-sq(d));
    return {p + (l.v*tmp), p - (l.v*tmp)};
}

vector<pt> circlesIntersction(circle c1, circle c2) {
    if (c1.r < c2.r) swap(c1, c2);
    pt a = c1.c, b = c2.c; double r1 = c1.r, r2 = c2.r;
    double d = a|b;
    pt v = (b-a).unit();
    if (mai(d, r1 + r2) || mai(r1, d+r2)) return {};
    if (eq(r1 + r2, d) || eq(r1, d + r2)) return {a + (v*r1)};
    double tmp1 = (sq(r1) - sq(r2) + sq(d))/ (2*d);
    double tmp2 = sqrt(sq(r1)-sq(tmp1));
    return {a+(v*tmp1)+((v.rot90())*tmp2), a+(v*tmp1)-((v.rot90())*tmp2)};
}

vector<pt> circleTangents(pt p, circle c) {
    double d = p|(c.c);
    if (eq(c.r, d)) return {p};
    if (c.r > d) return {};
    double len = sqrt(sq(d) - sq(c.r));
    vector<pt> res;
    pt v = (c.c - p).unit() * len;
    for (int sgn : {-1, 1}) res.pb(p + v.rotate(atan2(r, len)*sgn));
    return res;
}
```

5.1.5 Geometric structures on SVG

```
struct SVG {

    FILE *out;
    ld sc = 10;

    void open() {
        out = fopen("image.svg", "w");
        fprintf(out, "<svg xmlns='http://www.w3.org/2000/svg' viewBox='-1000 -1000 2000 2000'>\n");
        grid();
    }

    void segment(pt a, pt b, string cor = "black") {
        a = a * sc, b = b * sc;
        fprintf(out, "<line x1='%lf' y1='%lf' x2='%lf' y2='%lf' stroke='%s'/>\n", a.x, -a.y, b.x, -b.y
              , cor.c_str());
    }

    void circle(pt a, double r, string dentro = "red", string borda = "blue") {
        r = r * sc;
        a = a * sc;
        fprintf(out, "<circle cx='%lf' cy='%lf' r='%lf' fill='%s' stroke='%s'/>\n", a.x, -a.y, r,
             dentro.c_str(), borda.c_str());
    }

    void point(pt a, string cor = "blue") {
        circle(a, 0.25, cor, cor);
    }

    void polygon(vector<pt> poly, string dentro = "blue", string borda = "black") {
        string polyString = "";
        for(pt p: poly) polyString +=  to_string(p.x * sc)+","+to_string(-(p.y) * sc)+" ";
        fprintf(out, "<polygon stroke='%s' fill='%s' points='%s' />", borda.c_str(), dentro.c_str(),
              polyString.c_str());
    }

    void text(pt a, string s) {
        a = a * sc;
        fprintf(out, "<text x='%lf' y='%lf' font-size='10px'>%s</text>\n", a.x, -a.y, s.c_str());
    }

    void grid() {
        fprintf(out, "<rect fill='white' height='2000' width='2000' x='-1000' y='-1000'/>");
        fprintf(out, "<pattern id='grid' width='10' height='10' patternUnits='userSpaceOnUse'>");
        fprintf(out, "<path d='M 10 0 L 0 0 0 10' fill='none' stroke='gray' stroke-width='0.5'/>");
        fprintf(out, "</pattern>");
        fprintf(out, "<rect fill='url(#grid)' height='2000' width='2000' x='-1000' y='-1000'/>");
        segment({-1000, 0}, {1000, 0}); segment({0, -1000}, {0, 1000});
    }

    void close() {
        fprintf(out, "</svg>\n");
        fclose(out);
        out = 0;
    }

    ~SVG() {
        if (out)
            close();
    }
} svg;

//Usage of SVG
int main() {
    pt a  = {5, 10}, b = {45, -1}, c = {38, -23};
    svg.open();
    svg.circle(a, 10);
    svg.polygon({a, b, c});
    svg.close();
        return 0;
}
```

5.2 Sweep Line Utils

5.2.1 Radial Ordering

```
int quadrante(pt p) {
        if (p.y > 0) {
                if (p.x > 0) return 1;
                return 2;
        }
        if (p.x > 0) return 4;
        return 3;
}

// Muuuuuito mais rápido que apenas comparar os ângulos obtidos com atan2
ll comparaAngulo(pt p, pt q) {
        int qp = quadrante(p), qq  = quadrante(q);
        if (qp == qq) {
                return p ^ q;
        }
        return (qp < qq) ? 1LL : -1LL;
}

bool comp(pt p, pt q) {
        ll x = comparaAngulo(p-ptcomp, q-ptcomp);
        if (x != 0) {
                return x > 0LL;
        }
        // No caso de uma ordenação de eventos, pode ser útil ordenar eventos de entrada
        // antes de eventos de saída quando estes possuem a mesma angulação
        return p.tipo < q.tipo;
}
```

5.3 Convex hull (Monotone chain) - considering including collinear points

Given n points, compute how many recursive convex hulls exists in this set of points

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

struct pt {
        int x,y;
        int operator ^ (pt o) { // Produto cruzado |this|*|o|*sin(this->o)
        return x * o.y - y * o.x;
        }
        pt operator - (pt o) { //Diferença de pontos
        return {x - o.x, y - o.y};
        }
        bool operator == (pt o) {
        return x == o.x && y == o.y;
        }
};

inline bool operator<(const pt& a, const pt& b){
        if(a.x != b.x)  return a.x < b.x;
        else return a.y < b.y;
}

bool ccw(pt p, pt q, pt r) {
        pt p1 = q-p;
        pt p2 = r-p;
        return (p1 ^ p2) > 0;
}

vector<pt> convexhull(vector<pt> points){
        int n = (int) points.size();
        if(n <= 3) return points;

        sort(points.begin(), points.end());

        int k = 0;
        vector<pt> hull(2*n);

        for (int i = 0; i < n; ++i) {
                while (k >= 2 && ccw(hull[k-2], hull[k-1], points[i])) k--;
                hull[k++] = points[i];
        }
        for (int i = n-1, t = k+1; i > 0; --i) {
                while (k >= t && ccw(hull[k-2], hull[k-1], points[i-1])) k--;
                hull[k++] = points[i-1];
        }

        hull.resize(k-1);
        return hull;
}

int main(){
        int n,x,y;
        while (1){
                scanf("%d", &n);
                if (!n)break;
                vector<pt> points;
                for (int i = 0; i < n; i++){
                        scanf("%d %d", &x, &y);
                        points.pb({x,y});
                }

                int qtd = 0;
                while ((int) points.size() > 0){
                        qtd++;
                        vector<pt> hull = convexhull(points);
                        set<pt> dots(hull.begin(), hull.end());
                        vector<pt> npts;
                        for (auto dot: points)
                                if (!dots.count(dot))
                                        npts.pb(dot);

                        points = npts;
                }

                if(qtd&1)
                        printf("Take this onion to the lab!\n");
                else
                        printf("Do not take this onion to the lab!\n");
        }

        return 0;
}
```

6 DP

6.1 Optimizations

6.1.1 Convex Hull Trick for Maximal points (addLine and get: $O(log(Xmax - Xmin))$)

```cpp
const ll inf = 2e18;

struct ln {
    ll a, b;
    ll f(ll x) {
        return a*x + b;
    }
};

struct Node {
        Node *l, *r;
    ln Ln;
    Node() : l(NULL), r(NULL), Ln({0,-inf}) {}
};

typedef Node *pNode;

void add_line(ln nw, pNode &node, ll xl, ll xr) {
    if (!node) node = new Node();
    ll xm = (xl+xr)>>1;

    ln llow = node->Ln, lhigh = nw;
    if (llow.f(xl) > lhigh.f(xl)) swap(llow, lhigh);

    if (llow.f(xr) <= lhigh.f(xr)) {
        node->Ln = lhigh;
        return;
    } else if (llow.f(xm) < lhigh.f(xm)) {
        node->Ln = lhigh;
        add_line(llow, node->r, xm+1, xr);
    } else {
        node->Ln = llow;
        add_line(lhigh, node->l, xl, xm);
    }
}

ll get(ll x, pNode &node, ll xl, ll xr) {
    if (!node) return -inf;
    ll xm = (xl+xr)>>1;
    if (x <= xm) return max(node->Ln.f(x), get(x, node->l, xl, xm));
    else return max(node->Ln.f(x), get(x, node->r, xm+1, xr));
}
pNode root;
ll xmin = -1e9, xmax = 1e9;
void add_line(ln nw) {
    add_line(nw, root, xmin, xmax);
}
ll get(ll x) {
    return get(x, root, xmin, xmax);
}

int main(){
        add_line({1, 1});
    add_line({2, -5});
    add_line({0, 2});
    add_line({-2, -8});
    add_line({-1, 0});

    trace(get(-10LL)); //resp: 12
    trace(get(-4LL)); //resp: 4
    trace(get(-2LL)); //resp: 2
    trace(get(0LL)); //resp: 2
    trace(get(2LL)); //resp: 3
    trace(get(7LL)); //resp: 9

        return 0;
}
```

6.1.2 Divide and Conquer. From $O(n^2 * k)$ to $O(n * k * log(n))$

```cpp
// DP de divisão e conquista para particionar array de tamanho n em
// k partições com custo mínimo.
int n, k, tempK, atual = 0;
int dpt[2][MAXN];

void dp(int l, int r, int OPmin, int OPmax) {
    if (l > r) return;
```

```
    int m = (l+r)/2;
    int mini = 1e9;
    int op = m;

    for(int i = max(OPmin, tempK-1); i <= min(OPmax, m-1); i++) {
        if (dpt[!atual][i] + getCusto(i+1, m) < mini) {
            mini = min(mini, dpt[!atual][i] + getCusto(i+1, m));
            op = i;
        }
    }
    dpt[atual][m] = mini;
    dp(l, m-1, OPmin, op);
    dp(m+1, r, op, OPmax);
}


int main(){
    sc("%d %d", &n, &k);

    funcao_que_pega_entrada_e_inicializa_custos_de_intervalos();

    for(int i = 1; i <= n; i++) dpt[!atual][i] = getCusto(1, i); // Caso base
    for(int i = 2; i <= k; i++) {
        tempK = i;
        dp(1, n, 1, n);
        atual = !atual;
    }

    pr("%d\n", dpt[!atual][n]);

    return 0;
}
```

6.1.3 Knuth optimization. From $O(n^3)$ to $O(n^2)$

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 1010

int n, m, a[maxn];
int dp[maxn][maxn];
int op[maxn][maxn];

int main(){
    while(cin >> n >> m){
        a[0] = 0;
        a[m+1] = n;
        for(int i = 1; i <= m; i++)
            cin >> a[i];

        memset(dp, 0, sizeof dp);
        for(int i = 1; i <= m+1; i++)
            for(int j = 0; i+j <= m+1; j++){
                if(i == 1){
                    dp[j][j+i] = 0;
                    continue;
                }

                int ans = 1e9;

                int bg,nd;
                if(i == 2)
                    bg = j+1, nd = i+j;
                else
                    bg = op[j][i+j-1], nd = op[j+1][i+j];

                for(int k = bg; k <= nd; k++){
                    int v = dp[j][k]+dp[k][j+i];
                    if(ans > v){
                        ans = v;
                        op[j][j+i] = k;
                    }
                }
                dp[j][j+i] = ans + a[j+i] - a[j];
            }
        cout << dp[0][m+1] << '\n';
    }
    return 0;
}
```

6.1.4 Aliens Trick.

```
#define maxn 5000
#define maxValue 1000000000LL
#define maxP maxValue * maxn + 1
int n, b, c, k, x;
ll a;
ll li[5001];
ll dp[4*5001];
int ind[4*5001];
ll diminui[5001];
int lx[5001];
ll l, r, m;

int zeraDp() {memset(lx, -1, sizeof lx);
    for(int i = 0; i < 4*5001; i++){
        dp[i] = LLONG_MIN;
        ind[i] = INT_MIN;
    }}
pair<ll, int> query(int id, int l, int r, int i, int j){
    //trace(id, l, r, i, j);
    if (l > r || l > j || r < i) return {LLONG_MIN, -1};
    if (l >= i && r <= j) return {dp[id], ind[id]};
    int m = (l+r)/2;
    pair<ll, int> q1 = query(id*2+1, l, m, i, j), q2 = query(id*2+2, m+1, r, i, j);
    //trace(q1.fi, q2.fi, l, r, i, j);
    if (q1.fi > q2.fi) return q1;
    else return q2;
}
void update(int id, int l, int r, int i, pair<ll, int> v){
    if (l == r) {
        //trace("l = r = ", l);
        dp[id] = v.fi;
        ind[id] = v.se;
        return;
    }
    int m = (l+r)/2;
    if (i <= m) update(id*2+1, l, m, i, v);
    else update(id*2+2, m+1, r, i, v);

    dp[id] = dp[2*id+1]; ind[id] = ind[2*id+1];
    if (dp[2*id+2] > dp[id]) {
        dp[id] = dp[2*id+2]; ind[id] = ind[2*id+2];
    }
}
pair<ll, int> getMaxDp(int i) {
    return query(0, 0, 5000, max(0, i-k), i-1);
}
void updMaxDp(ll newVal, int newIndex, int i) {
    update(0, 0, 5000, i, {newVal, newIndex});
}
void populaDp() {
    zeraDp();
    for(int i = 0; i < n; i++) {
        //trace("i:", i);
        ll cost = li[i]+m;
        // /trace("cost[", i, "]:", cost);
        ll maxi = LLONG_MIN;
        int maxiX = 1;
        diminui[i] = 0;
        if (i < k) {
            maxi = max(maxi, cost);
            diminui[i] = m+i;
        }
        if (i > 0) {
            pair<ll, int> pa = getMaxDp(i);
            //trace("que",0, 0, 5000, max(0, i-k), i-1);
            //trace(pa.fi, pa.se);


            if (pa.fi+cost > maxi){
                maxi = pa.fi+cost;
                maxiX = lx[pa.se]+1;
                diminui[i] = diminui[pa.se]+m+i;
            }
        }

        //trace("lx[", i, "]:", maxiX);
        //trace("dp[", i, "]:", maxi);
        //trace("upd",0, 0, 5000, i, maxi, i);
        updMaxDp(maxi, i, i);
        lx[i] = maxiX;
    }
}

int main(){
    sc("%d %d %d", &n, &k, &x);
    ll maxi = 0;

    for(int i = 0; i < n; i++){
        sc("%lld", &a);
        li[i] = a*10000+i;
        //trace(li[i]);
        maxi = max(maxi, li[i]);
    }

    int mini = ((n-k+1)/k)+(((n-k+1)%k) ? 1 : 0);
    if (x < mini) {
        pr("-1\n");return 0;
```

```
    }

    li[n] = n;diminui[n] = n;
    n++;


    //trace(maxi);
    l = -5000*maxi-1; r = 0;
    while (1) {
        m = (l+r)/2;
        //trace("l, m, r:", l, m, r);
        cin >> a;
        populaDp();
        //trace("lx:",lx[n-1]-1, "query:", query(0, 0, 5000, n-1, n-1).fi);
        if (lx[n-1]-1 == x) {
            break;
        } else if (lx[n-1]-1 < x){
            l = m+1;
        } else {
            r = m-1;
        }
    }
    pr("%lld\n", (query(0, 0, 5000, n-1, n-1).fi-diminui[n-1])/10000);
        return 0;
}
```

7 String

7.1 ZFunction

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

7.2 KMP

```
#include <bits/stdc++.h>
#define MAX 1000007

using namespace std;

int k[MAX];
string txt;

void kmp(){
        int i = 0, j = -1;
        k[0] = -1;
        while(i < txt.size()){
                while(j >= 0 && txt[i] != txt[j]) j = k[j];
                i++; j++;
                k[i] = j;
        }
}
int main(){
        cin >> txt;
        kmp();
        return 0;
}
```

7.3 Hash

7.3.1 Silple Hash

```
#include <bits/stdc++.h>
#define MAX 1000007
#define HMAX 2

using namespace std;
typedef long long ll;
typedef pair<ll, int> pii;


ll inverse(int a, int b, int s0 = 1, int s1 = 0){
    return b == 0? s0: inverse(b, a%b, s1, s0-s1*(a/b));
}

struct H{
    ll base[HMAX] = {317, 307};
    ll mod[HMAX] = {104000717, 104000711};
    ll power[HMAX][MAX], inv[HMAX][MAX], h[HMAX][MAX];

    void init(){
        for(int k = 0; k < HMAX; k++){
            power[k][0] = inv[k][0] = 1;
            power[k][1] = base[k];
            inv[k][1] = inverse(base[k], mod[k]);
            for(int i = 2; i < MAX; i++){
                power[k][i] = (power[k][i-1]*power[k][1])%mod[k];
                inv[k][i] = (inv[k][i-1]*inv[k][1])%mod[k];
            }
        }
    }
    H() {
        init();
    }

    H(string &s){
        init();
        build(s);
    }
    vector<ll> build(string &s){
        vector<ll> ret;

        for(int k = 0; k < HMAX; k++){
            h[k][0] = s[0];
            for(int i = 1; i < s.size(); i++){
                h[k][i] = (h[k][i-1] + (s[i]*power[k][i])%mod[k])%mod[k];
            }
            ret.push_back(h[k][s.size()-1]);
        }

        return ret;
    }

    vector<ll> sub(int l, int r){
        vector<ll> ret;
        for(int k = 0; k < HMAX; k++){
            ll hr = h[k][r];
            ll hl = (l > 0)? h[k][l-1]:0;
            ll ans = ((hr+mod[k] - hl) * inv[k][l])%mod[k];

            ret.push_back(ans);
        }

        return ret;
    }
};

bool cmp(vector<ll> a, vector<ll> b){
    for(int i = 0; i < a.size(); i++){
        if(a[i] != b[i]) return false;
    }

    return true;
}
int main(){
    return 0;
}
```

7.3.2 BIT de Hash

```
#include <bits/stdc++.h>
#define MAX 100007

using namespace std;

typedef long long ll;

ll h[4*MAX], power[MAX], inv[MAX];
ll base[2] = {317, 307};
ll mod[2] = {104000717, 104000711};
int n;
```

```
ll gcd(ll a, ll m){
    ll x = 1, y = 0, ini = m;
    if(m == 1) return 0;

    while(a > 1){
        ll q = a/m;

        ll t = m;
        m = a%m;
        a = t;
        t = y;
        y = x-q*y;
        x = t;
    }

    if(x < 0) x += ini;
    return x;
}

void init(){
    power[0] = base[0];
    inv[0] = gcd(base[0], mod[0]);
    cout << inv[0] << endl;
    for(int i = 1; i < MAX; i++){
        power[i] = (power[i-1]*base[0])%mod[0];
        inv[i] = (inv[i-1]*inv[0])%mod[0];
    }
}

void update(int idx, char now, char old = '-'){
    cout << now << " " << old << endl;
    int pos = idx;
    if(old == '-') for(; idx <= n; idx += idx&(-idx)) h[idx] = (h[idx] + (now-'a'+1)*power[pos])%mod
          [0];
    else{
        for(; idx <= n; idx += idx&(-idx)) {
            h[idx] = (h[idx] - (old-'a'+1)*power[pos])%mod[0];
            while(h[idx] < 0) h[idx] += mod[0];
            h[idx] = (h[idx] + (now-'a'+1)*power[pos])%mod[0];
            while(h[idx] < 0) h[idx] += mod[0];
        }
    }
}
void getHash(string s){
    for(int i = 1; i <= s.size(); i++) update(i, s[i-1]);
}

ll query(int idx){
    ll hash = 0;
    for(; idx > 0; idx -= idx&(-idx)) hash = (hash + h[idx])%mod[0];
    return hash;
}

ll query(int l, int r){
    return (((query(r)-query(l-1)+ mod[0])%mod[0])*inv[l])%mod[0];
}
int main(){
    string txt; cin >> txt;
    n = txt.size();
    init();
    //cout << "ola\n";
    getHash(txt);
    //cout << "oi\n";
    int k; cin >> k;
    while(k--){
        int tipo; cin >> tipo;
        if(!tipo){
            int l, r; cin >> l >> r;
            cout << query(l, r) << endl;
        }else{
            int idx;
            char letra; cin >> idx >> letra;
            update(idx, letra, txt[idx-1]);
            txt[idx-1] = letra;
        }
    }
    return 0;
}
```

7.4 Suffix Array

```
// Suffix Array N log N

#include <bits/stdc++.h>
#define MAX 100007

using namespace std;

vector<int> suffix_array(string s){
    int n = s.size(), N = n + 256;
    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;
        for(int i = 0; i < n; i++) cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i - 1];
        for(int i = n - 1; i >= 0; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        int r = 0;
        for(int i = 1; i < n; i++){
            if(ra[sa[i]] != ra[sa[i - 1]]) r++;
            else if(ra[(sa[i] + k) % n] != ra[(sa[i - 1] + k) % n]) r++;
            nra[sa[i]] = r;
        }
        ra = nra;
    }
    return sa;
}

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++){
        if(k) k--;
        if(ra[i] == n - 1) {k = 0; continue;}
        int j = sa[ra[i] + 1];
        while(k < n && s[(i + k) % n] == s[(j + k) % n]) k++;
        lcp[ra[i]] = k;
        if(ra[(sa[ra[i]] + 1) % n] > ra[(sa[ra[j]] + 1) % n]) k = 0;
    }
    return lcp;
}

char s[MAX];
int main(){
    //cout << "oe\n";
    scanf(" %s", s);
    vector<int> arr = suffix_array(s + '$');
    int n = strlen(s) + 1;
    for(int i = 0; i < arr.size(); i++) printf("%d\n", arr[i]);
    return 0;
}
```

7.5 Suffix Automata

```
#include <bits/stdc++.h>
#define MAX 1000007

using namespace std;

struct node{
        int link, len;
        map<char, int> next;
} st[MAX];

int sz, last;

void init(){
        sz = 1;
        last = 0;
        st[0].len = 0;
        st[0].link = -1;
        st[0].next.clear();
}

void add(char c){
        int cur = sz++;
        st[cur].len = st[last].len + 1;

        for(; last != -1 && !st[last].next[c]; last = st[last].link) st[last].next[c] = cur;

        if(last == -1) st[cur].link = 0;
        else{
                int q = st[last].next[c];
                if(st[q].len == st[last].len + 1) st[cur].link = q;
                else{
                        int clone = sz++;
                        st[clone].len = st[last].len + 1;
                        st[clone].link = st[q].link;
                        st[clone].next = st[q].next;
```

```
                for(; last != -1 && st[last].next[c] == q; last = st[last].link) st[last].next
                        [c] = clone;
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
}
int main(){
        string palavra; cin >> palavra;
        init();
        for(int i = 0; i < palavra.size(); i++) add(palavra[i]);
        return 0;
}
```

7.6 Trie

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

using namespace std;

struct node{
        bool end;
        struct node* word[26];
} *head;

void init(){
        head = new node();
        head->end = false;
}

void add(string palavra){
        node* current = head;
        for(int i = 0; i < palavra.size(); i++){
                int letra = palavra[i]-'a';

                if(current->word[letra] == NULL) current->word[letra] = new node();

                current = current->word[letra];
        }
        current->end = true;
}
int main(){
        string palavra; cin >> palavra;
        init();
        add(palavra);
        return 0;
}
```

7.7 String Tricks

```
// ------- ----------- ------------- SUFFIX AUTOMATA
      -----------------------------------------------------

//Number of distinct substring is also equal to  (i = 0 to sz ) sum( length[i]-length[link[i]] ),
//because each node corresponds to different substrings with lengths in range [length[link[i]] + 1,
      length[i]].

// LCS

void extends(int c){
    int cur = sz++;
    sa[cur].len = sa[last].len + 1;
    mn[cur] = sa[cur].len;
    for(; last != -1 && !sa[last].next[c]; last = sa[last].link) sa[last].next[c] = cur;

    if(last == -1) sa[cur].link = 0;
    else{
        int q = sa[last].next[c];
        if(sa[q].len == sa[last].len + 1) sa[cur].link = q;
        else{
            int clone = sz++;
            sa[clone].len = sa[last].len + 1;
            sa[clone].link = sa[q].link;
            mn[clone] = sa[clone].len;
            memcpy(sa[clone].next, sa[q].next, sizeof sa[q].next);
            for(; last != -1 && sa[last].next[c] == q; last = sa[last].link) sa[last].next[c] = clone;
            sa[q].link = sa[cur].link = clone;
        }
    }
    last = cur;
}

void lcs(){
    int v = 0, l = 0;
    for(int i = 0; i <= sz; i++) mx[i] = 0;
    for(int i = 0; i < palavra.size(); i++){
        int id = palavra[i]-'a';
        while(v && !sa[v].next[id]){
            v = sa[v].link;
            l = sa[v].len;
        }

        if(sa[v].next[id]){
            v = sa[v].next[id];
            l++;
        }
        mx[v] = max(mx[v], l);
    }
    for(int i = sz; i > 0; i--) mx[sa[i].link] = max(mx[sa[i].link], mx[i]);
    for(int i = 0; i <= sz; i++) mn[i] = min(mn[i], mx[i]);
}

void solve(){
    int ans = 0;
    for(int i = 0; i <= sz; i++) ans = max(mn[i], ans);
    cout << ans << endl;
}


// ----------------------------  PALINDROMIC ARRAY (MANACHER'S) ----------------------------

// NOT USED
// It calculates half-length of maximum odd palindrome with center in i
// ex: ABACABA -> {1, 2, 1, 4, 1, 2, 1}
//
// In case of even palindromes I suggest to use the same code with the string s1#s2#... #sn
//
// q: What is the use of characters "_" and "-"?
//      r: Using them we can get rid of cases when  - len or i + len are out of bound.

vector<int> pal_array(string s)
{
    int n = s.size();
    s = "_" + s + "-";
    vector<int> len(n + 1);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++)
    {
        len[i] = min(r - i, len[l + (r - i)]);
        while(s[i - len[i]] == s[i + len[i]])
            len[i]++;
        if(i + len[i] > r)
        {
            l = i - len[i];
            r = i + len[i];
        }
    }
    len.erase(begin(len));
    return len;
}
```

7.8 Aho-Corasick

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

using namespace std;

#define MN 1000100
#define ALP 26

int C(char c) { return c - 'a'; }

struct aho {
    int parent[MN], suffix[MN], transition[MN][ALP], super[MN];
    char from[MN];
    int id[MN], cnt, built;
    long long ending[MN], lazy[MN];

    int new_node(int _parent = 0, char _from = ' ') {
        parent[cnt] = _parent; from[cnt] = _from; id[cnt] = -1;
        suffix[cnt] = super[cnt] = ending[cnt] = lazy[cnt] = 0;
        for(int i = 0; i < ALP; i++) transition[cnt][i] = 0;
        return cnt++;
    }

    aho() {
        cnt = built = 0;
        new_node();
    }

    int add_word(string &word, int _id = 0) {
```

```
        int node = 0;
        for(int i = 0; i < word.size(); i++) {
            int nxt = C(word[i]);
            if(!transition[node][nxt]) transition[node][nxt] = new_node(node, word[i]);
            node = transition[node][nxt];
        }
        ending[node]++;
        if(id[node] == -1) id[node] = _id;
        return id[node];
    }

    void build() {
        built = 1;
        queue<int> q;
        for(int i = 0; i < ALP; i++)
            if(transition[0][i]) q.push(transition[0][i]);

        while(!q.empty()) {
            int v = q.front(); q.pop();
            if(parent[v]) suffix[v] = transition[suffix[parent[v]]][C(from[v])];

            if(id[v] != -1) super[v] = v;
            else super[v] = super[suffix[v]];

            ending[v] += ending[suffix[v]];
            for(int i = 0; i < ALP; i++) {
                if(!transition[v][i]) transition[v][i] = transition[suffix[v]][i];
                else q.push(transition[v][i]);
            }
        }
    }

    long long search_text(string &text) {
        if(!built) build();
        int prefix = 0;
        long long count = ending[0];
        for(char c : text) {
            prefix = transition[prefix][C(c)];
            count += ending[prefix];
        }
        return count;
    }

    /// array of number of occurrences of pattern with id = i
    void search_text(string &text, int *ans) {
        if(!built) build();
        int prefix = 0;
        for(char c : text) {
            prefix = transition[prefix][C(c)];
            for(int u = super[prefix]; u; u = super[suffix[u]])
                ans[id[u]]++;
        }
    }

    void search_text_linear(string &text, int *ans) { /// O(N+M)
        if(!built) build();
        int prefix = 0;
        for(char c : text) {
            prefix = transition[prefix][C(c)];
            lazy[prefix]++;
        }
        push_lazy(ans);
    }

    void push_lazy(int *ans) {
        vector<int> deg(cnt, 0);
        for(int i = 1; i < cnt; i++)
            deg[suffix[i]]++;

        queue<int> fila;
        for(int i = 1; i < cnt; i++)
            if(deg[i] == 0) fila.push(i);

        while(!fila.empty()) {
            int u = fila.front(); fila.pop();

            if(id[u] != -1) ans[id[u]] += lazy[u];
            lazy[suffix[u]] += lazy[u];

            deg[suffix[u]]--;
            if(suffix[u] && deg[suffix[u]] == 0) fila.push(suffix[u]);
        }
    }
} dict;

int main() {
    string str = "aa";
    dict.add_word(str);

    str = "a";
    dict.add_word(str);

    dict.build();

    string cmon = "aaaaa";
    printf("%d\n", dict.search_text(cmon));
    return 0;
}
```

8 Game Theory

8.1 Normal Nim vs. Misére Nim

8.1.1 Normal Nim

```
void printaVencedor(vector<int> piles) {
    int x = 0;
    for (int pile : piles) {
        x ^= pile;
    }
    pr(x ? "First\n" : "Second\n");
}
```

8.1.2 Misére Nim

```
void printaVencedor(vector<int> piles) {
    int x = 0, maxi = 1;
    for (int pile : piles) {
        x ^= pile;
        maxi = max(maxi, pile);
    }
    if(maxi == 1) pr((n & 1) ? "Second\n" : "First\n");
    else pr(x ? "First\n" : "Second\n");
}
```

9 Specific

9.1 Ad-Hoc

9.1.1 Huffman Coding

```
/*
 * Huffman Cost - O(nlogn)
 */

// Exemplo 1:
// Dado a frequência de cada letra presente em uma string, calcula a menor quantidade de
// bits necessária para acodificar essa string dando uma sequência de bits como
// representante de cada letra e sem causar ambiguidade na decodificação (decodificando
// sempre da esquerda para a direita)
// Exemplo 2:
// Calcular o menor custo para particionar um conjunto em grupos de tamanhos correspondente
// aos valores presentes no array (dado que o custo para particionar um grupo é a quantidade
// de membros nesse grupo)

ll huffman(vector<ll> a) {
        ll ans = 0, u, v;
        priority_queue<ll, vector<ll>, greater<ll>> pq;
        for(ll frq : a) pq.push(frq);
        while(pq.size() > 1) {
                u = pq.top(); pq.pop();
                v = pq.top(); pq.pop();
                pq.push(u+v);
                ans += u + v;
        }
        return ans;
}
```

9.2 Graph

9.2.1 Erdos-Gallai

```cpp
/*
 * Erdos-Gallai's theorem
 * O(nlogn), O(n) if already sorted
 * checks if a given array of degrees can represent a graph
 */

bool erdosgallai(vector<int> d) {
        sort(d.begin(), d.end(), greater<int>());
        vector<long long> pd(d.size());
        int n = d.size(), p = n-1;
        for(int i = 0; i < n; i++)
                pd[i] = d[i] + (i > 0 ? pd[i-1] : 0);
        for(int k = 1; k <= n; k++) {
                while(p >= 0 && d[p] < k) p--;
                long long sum;
                if (p >= k-1) sum = (p-k+1)*1ll*k + pd[n-1] - pd[p];
                else sum = pd[n-1] - pd[k-1];
                if (pd[k-1] > k*(k-1LL) + sum) return false;
        }
        return pd[n-1] % 2 == 0;
}
```

10 Pra ficar atento

10.1 Possiveis erros

10.1.1 WA

- Utilizar int em vez de long long.
- Em caso de busca por intersecções, se esquecer de procurar os 3 tipos:
  1. $A \cup B = A$.
  2. $A \cup B = B$.
  3. $A \cap B \neq \emptyset$ e $A, B \subset A \cup B$
- Tirar módulo de algo que está sendo dividido ou shiftado para a direita.
- Se esquecer de limpar o array para um novo caso de teste.
- Se esquecer de limpar a lazy propagation.
- Usar a mesma variável para loops dentro de outros loops.

10.2 Possiveis ideias

10.2.1 Sem ideia:

- Se não estiver pensando em como acumular uma lazy, talvez o ideal seja usar outra estrutura como sqrt decomposition.
- Caso não haja atualização, mesmo que haja uma busca binária no resultado uma Sparse Table ainda diminuirá de $O(\log^2(N))$ para $O(\log(N))$ ou de $O(\log(N))$ para $O(1)$.
- Tirar módulo de algo que está sendo dividido ou shiftado para a direita.
- Trocar uma BIT de maps por uma seg tree implícita também melhora a complexidade.