Faculty of Computer and Information Sciences, Ain Shams University: Too Wrong to Pass Too Correct to Fail

Pillow, Isaac, Mostafa, Islam

2021

Contents

1 Combinatorics

1.1 Burnside Lemma

```

// |Classes|=sum (k ^C(pi))  / |G|

// C(pi)  the number of cycles in the permutation pi

// |G| the number of permutations
```

1.2 Catlan Numbers

```
const int MOD = ....
const int MAX = ....
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}

// 1- Number of correct bracket sequence consisting of n opening and n closing
     brackets.
// 2- The number of rooted full binary trees with n+1 leaves (vertices are not
     numbered).
```

```
//    A rooted binary tree is full if every vertex has either two children or no children.
// 3- The number of ways to completely parenthesize n+1 factors.
// 4- The number of triangulations of a convex polygon with n+2 sides
//    (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
// 5- The number of ways to connect the 2n points on a circle to form n disjoint chords.
// 6- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
// 7- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size nxn,
//    which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
// 8- Number of permutations of length n that can be stack sorted
//    (i.e. it can be shown that the rearrangement is stack sorted if and only if
//    there is no such index i<j<k, such that ak<ai<aj ).
// 9- The number of non-crossing partitions of a set of n elements.
// 10- The number of ways to cover the ladder 1..n using n rectangles
// (The ladder consists of n columns, where ith column has a height i).
```

2 Algebra

2.1 Primitive Roots

```
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
    int phi = p - 1,  n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

2.2 Discrete Logarithm

```
// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
        }
    }
    return -1;
}

//When a and m are not coprime
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

2.3 Iteration over submasks

```
int s = m;
while (s > 0) {
 ... you can use s ...
 s = (s-1) & m;
}
```

2.4 Totient function

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

2.5 CRT and EEGCD

```
ll extended(ll a, ll b, ll &x, ll &y) {

    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x0, y0;
    ll g = extended(b, a % b, x0, y0);
    x = y0;
    y = x0 - a / b * y0;

    return g ;
}
ll de(ll a, ll b, ll c, ll &x, ll &y) {

    ll g = extended(abs(a), abs(b), x, y);
    if(c % g) return -1;

    x *= c / g;
    y *= c / g;

    if(a < 0)x = -x;
    if(b < 0)y = -y;
    return g;
}
pair<ll, ll> CRT(vector<ll> r, vector<ll> m) {

    ll r1 = r[0], m1 = m[0];

    for(int i = 1; i < r.size(); i++) {

        ll r2 = r[i], m2 = m[i];
        ll x0, y0;
        ll g = de(m1, -m2, r2 - r1, x0, y0);

        if(g == -1) return {-1, -1} ;

        ll nr = x0 * m1 + r1;
        ll nm = m1 / g * m2;

        r1 = (nr % nm + nm) % nm;
        m1 = nm;
    }
    return {r1, m1};
}
```

2.6 FFT

```
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld long double
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
typedef complex<double> C;
typedef vector<double> vd;
typedef vector<int> vi;
typedef pair<int, int> pii;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            C z = rt[j + k] * a[i + j + k]; //
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i, 0, sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}

int main() {
    IO
    //Applications
    //1-All possible sums

    //2-All possible scalar products
    // We are given two arrays a[] and b[] of length n.
    //We have to compute the products of a with every cyclic shift of b.
    //We generate two new arrays of size 2n: We reverse a and append n zeros to it.
    //And we just append b to itself. When we multiply these two arrays as polynomials,
    //and look at the coefficients c[n-1], c[n], ..., c[2n-2] of the product c, we get:
    //c[k]=sum i+j=k  a[i]b[j]

    //3-Two stripes
    //We are given two Boolean stripes (cyclic arrays of values 0 and 1) a and b .
    //We want to find all ways to attach the first stripe to the second one,
    //such that at no position we have a 1 of the first stripe next to a 1 of the second stripe.
}
```

2.7 Fibonacci

```


// F(n-1) * F(n+1) - F(n)^2 = (-1)^n

// F(n+k) = F(k) * F(n+1) + F(k-1) * F(n)

// F(2*n) = F(n) * (F(n+1) + F(n-1))

//GCD ( F(m) , F(n) ) = F(GCD(n,m))
```

2.8 Gauss Determinant

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

2.9 GAUSS SLAE

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

2.10 Matrix Inverse

```
// Sometimes, the questions are complicated - and the answers are simple. //
#pragma GCC optimize ("O3")
#pragma GCC optimize ("unroll-loops")
#include <bits/stdc++.h>
#define ll long long
#define ld  long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
vector < vector<double> > gauss (vector < vector<double> > a) {

    int n = (int) a.size();
    vector<vector<double> > ans(n, vector<double>(n, 0));

    for(int i = 0; i < n; i++)
        ans[i][i] = 1;
    for(int i = 0; i < n; i++) {
        for(int j = i + 1; j < n; j++)
            if(a[j][i] > a[i][i]) {
                swap(a[j], a[i]);
                swap(ans[j], ans[i]);
            }
        double val = a[i][i];
        for(int j = 0; j < n; j++) {
            a[i][j] /= val;
            ans[i][j] /= val;
        }
        for(int j = 0; j < n; j++) {
            if(j == i)continue;
            val = a[j][i];
            for(int k = 0; k < n; k++) {
                a[j][k] -= val * a[i][k];
                ans[j][k] -= val * ans[i][k];
            }
        }
    }
    return ans;
}
int main() {

    IO
    vector<vector<double> > v(3, vector<double> (3) );
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            cin >> v[i][j];

    for(auto i : gauss(v)) {
        for(auto j : i)
            cout << j << " ";
        cout << "\n";
    }
}
```

2.11 NTT

```
struct NTT {
    int mod ;
    int root ;
    int root_1 ;
    int root_pw ;

    NTT(int _mod, int primtive_root, int NTT_Len) {

        mod = _mod;
        root_pw = NTT_Len;
        root = fastpower(primtive_root, (mod - 1) / root_pw);
        root_1 = fastpower(root, mod - 2);
    }
    void fft(vector<int> & 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) {
            int wlen = invert ? root_1 : root;
            for (int i = len; i < root_pw; i <<= 1)
                wlen = (int)(1LL * wlen * wlen % mod);


            for (int i = 0; i < n; i += len) {
                int w = 1;
                for (int j = 0; j < len / 2; j++) {
                    int u = a[i + j], v = (int)(1LL * a[i + j + len / 2] * w %
                         mod);
                    a[i + j] = u + v < mod ? u + v : u + v - mod;
                    a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
                    w = (int)(1LL * w * wlen % mod);
                }
            }
        }

        if (invert) {
            int n_1 = fastpower(n, mod - 2);
            for (int & x : a)
                x = (int)(1LL * x * n_1 % mod);
        }
    }
    vector<int> multiply(vector<int> &a, vector<int> &b) {
        vector<int> 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, 0);
        fft(fb, 0);

        for(int i = 0; i < n; i++)
            fa[i] = 1LL * fa[i] * fb[i] % mod;
        fft(fa, 1);
        return fa;
    }
};
```

2.12 NTT of KACTL

```
///(Note faster than the other NTT)
///If the mod changes don't forget to calculate the primitive root
using ll = long long;
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;

ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        f(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vector<int> rev(n);
    f(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    f(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) f(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    f(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
vector<int> v;
vector<ll> solve(int s, int e) {
    if(s==e) {
        vector<ll> res(2);
        res[0] = 1;
        res[1] = v[s];
        return res;
    }
    int md = (s + e) >> 1;
    return conv(solve(s,md),solve(md+1,e));
}
```

3 Data Structures

3.1 2D BIT

```
void upd(int x, int y, int val) {
    for(int i = x; i <= n; i += i & -i)
    for(int j = y; j <= m; j += j & -j)
    bit[i][j] += val;
}
int get(int x, int y) {
    int ans = 0;
    for(int i = x; i; i -= i & -i)
    for(int j = y; j; j -= j & -j)
    ans += bit[i][j];
}
```

3.2 2D Sparse table

```
/*
    note this isn't the best cache-wise version
    query O(1), Build O(NMlgNlgM)
    be careful when using it and note the he build a dimension above another
    i.e he builds a sparse table for each row
    the build sparse table over each row's sparse table
*/
const int N = 505, LG = 10;

int st[N][N][LG][LG];
int a[N][N], lg2[N];

int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
         max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
         max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b])
       );
}

void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
```

3.3 hillbert Order

```
///Faster Sorting MO

const int infinity = (int)1e9 + 42;
const int64_t llInfinity = (int64_t)1e18 + 256;
const int module = (int)1e9 + 7;
const long double eps = 1e-8;

inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
    if (pow == 0) {
        return 0;
    }
    int hpow = 1 << (pow-1);
    int seg = (x < hpow) ? (
        (y < hpow) ? 0 : 3
    ) : (
        (y < hpow) ? 1 : 2
    );
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2*pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
    return ans;
}

struct Query {
    int l, r, idx;
    int64_t ord;

    inline void calcOrder() {
        ord = gilbertOrder(l, r, 21, 0);
    }
};

inline bool operator<(const Query &a, const Query &b) {
    return a.ord < b.ord;
}

signed main() {
    #ifndef USE_FILE_IO
        ios_base::sync_with_stdio(false);
    #endif

    mt19937 rnd(42);

    int n, m, k; cin >> n >> m; k = rnd() % 1048576;
    vector<int> p(n+1);
    for (int i = 0; i < n; i++) {
        int val = rnd() % 1048576;
        p[i+1] = p[i] ^ val;
    }

    vector<Query> qry(m);
    for (int i = 0; i < m; i++) {
        int l = rnd() % n + 1, r = rnd() % n + 1;
        if (l > r) {
            swap(l, r);
        }
        qry[i].l = l; qry[i].r = r;
        qry[i].idx = i;
        qry[i].calcOrder();
    }

    int64_t ans = 0;
    vector<int64_t> res(m);
    vector<int64_t> cnt((int)2e6, 0);
    sort(qry.begin(), qry.end());
    int l = 0, r = 1;
    ans = (p[1] == k);
    cnt[p[0]]++; cnt[p[1]]++;

    for (Query q: qry) {
        q.l--;
        while (l > q.l) {
            l--;
            ans += cnt[p[l] ^ k];
            cnt[p[l]]++;
        }
        while (r < q.r) {
            r++;
            ans += cnt[p[r] ^ k];
            cnt[p[r]]++;
        }
        while (l < q.l) {
            cnt[p[l]]--;
            ans -= cnt[p[l] ^ k];
            l++;
        }
        while (r > q.r) {
            cnt[p[r]]--;
            ans -= cnt[p[r] ^ k];
            r--;
        }
        res[q.idx] = ans;
    }

    uint64_t rhsh = 0;
    for (int i = 0; i < m; i++) {
        rhsh *= (uint64_t)1e9 + 7;
        rhsh += (uint64_t)res[i];
    }
    cout << rhsh << "\n";

    return 0;
}
```

3.4 Merge Sort Bit with updates

```
//O(log ^ 2 N) updates and queries


#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/rope>

using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx;

template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;


Tree<int> t[N];

void add(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x){
        t[x].insert(v);
    }
}
void erase(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x)
        t[x].erase(v);
}
int get(int idx,  int limit){
    int ret = 0;
    for(int x = ++idx; x; x -= x & -x)
        ret += (t[x].order_of_key(limit+1));
    return ret;
}
```

3.5 Mo's

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

int n, qq, arr[N], sz = 1000; // sz  is the size of the bucket
int co[N],ans = 0, ansq[N];
int cul = 1, cur = 1;

void add(int x) {
    co[arr[x]]++;
    if (co[arr[x]] == 1)
        ans++;
    else if (co[arr[x]] == 2)
        ans--;
}

void remove(int x) {
    co[arr[x]]--;
    if (co[arr[x]] == 1)
        ans++;
    else if (co[arr[x]] == 0)
        ans--;
}

void solve(int l, int r,int ind) {
    r+=1;
    while (cul < l) remove(cul++);
    while (cul > l) add(--cul);
    while (cur < r) add(cur++);
    while (cur > r) remove(--cur);
    ansq[ind] = ans;
}


int main() {
    FIO
    cin >> qq;
   //                       {l/sz,r},     { l , ind}
   priority_queue<pair<pair<int, int>, pair<int, int>>, vector<pair<pair<int,
        int>, pair<int, int>>>, greater<pair<pair<int, int>, pair<int, int>>>> q
        ;
    for (int i = 0; i < qq; i++) {
        int l, r;
        cin >> l >> r;
        q.push({{l / sz, r},{l,i}});
    }
    while (q.size()) {
        int ind=q.top().second.second,l=q.top().second.first,r=q.top().first.
            second;
        solve(l, r,ind);
        q.pop();
    }
    for (int i = 0; i < qq; i++)
        cout << ansq[i] << endl;


    return 0;
}
```

3.6 Mo With Updates

```

///O(N^5/3) note that the block size is not a standard size

#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int N = 1e5 +5;
const int M = 2 * N;
const int blk = 2155;
const int mod = 1e9 + 7;
struct Query{
  int l, r, t, idx;
  Query(int a = 0,int b = 0,int c = 0,int d = 0){l=a,r=b,t=c,idx = d;}
  bool operator < (Query o){
    if(r / blk == o.r / blk && l / blk == o.l / blk)return t < o.t;
    if(r / blk == o.r / blk)return l < o.l;
    return r < o.r;
  }
} Q[N];

int a[N], b[N];
int cnt1[M], cnt2[N];
int L = 0, R = -1, K = -1;
void add(int x){ ///add item to range
//  cout << x << '\n';
  cnt2[cnt1[x]]--;
  cnt1[x]++;
  cnt2[cnt1[x]]++;
}
void del(int x){ ///delete item from range
  cnt2[cnt1[x]]--;
  cnt1[x]--;
  cnt2[cnt1[x]]++;
}
map<int,int>id;
int cnt;
int ans[N];
int p[N], nxt[N];
int prv[N];
void upd(int idx){ ///update item value
  if(p[idx] >= L && p[idx] <= R)
    del(a[p[idx]]), add(nxt[idx]);
  a[p[idx]] = nxt[idx];
}
void err(int idx){
  if(p[idx] >= L && p[idx] <= R)
    del(a[p[idx]]), add(prv[idx]);
  a[p[idx]] = prv[idx];
}
int main(){

  int n, q, l, r, tp;

  scanf("%d%d", &n, &q);

  for(int i = 0; i < n; i++){
    scanf("%d", a + i);
    if(id.count(a[i]) == 0)
      id[a[i]] = cnt++;
    a[i] = id[a[i]];
    b[i] = a[i];
  }
  int qIdx = 0;
  int ord = 0;
  while(q--){

    scanf("%d", &tp);
    if(tp == 1){
      /// ADD Query
      scanf("%d%d", &l, &r);  --l, --r;
      Q[qIdx] = Query(l,r,ord-1,qIdx); qIdx++;
    } else{
      /// ADD Update
      scanf("%d%d",p + ord, nxt + ord); --p[ord];
      if(id.count(nxt[ord]) == 0)
        id[nxt[ord]] = cnt++;
      nxt[ord] = id[nxt[ord]];
      prv[ord] = b[p[ord]];
      b[p[ord]] = nxt[ord];
      ++ord;
    }

  }
  sort(Q,Q+qIdx);
  for(int i = 0; i < qIdx; i++){
    while(L < Q[i].l)del(a[L++]);
    while(L > Q[i].l)add(a[--L]);
    while(R < Q[i].r)add(a[++R]);
    while(R > Q[i].r)del(a[R--]);
    while(K < Q[i].t)upd(++K);
    while(K > Q[i].t)err(K--);
    ///Solve Query I
  }
  for(int i = 0; i < qIdx; i++)
    printf("%d\n", ans[i]);


  return 0;
}
```

3.7 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>, rb_tree_tag, tree_order_statistics_node_update>

// order_of_key(k): returns the number of elements in the set strictly less than k
// find_by_order(k): returns an iterator to the k-th element (zero-based) in the set
```

3.8 Persistent Seg Tree

```

int val[ N * 60 ], L[ N * 60 ], R[ N * 60 ], ptr, tree[N]; /// N * lgN
int upd(int root, int s, int e, int idx) {
    int ret = ++ptr;
    val[ret] = L[ret] = R[ret] = 0;
    if (s == e) {
        val[ret] = val[root] + 1;
        return ret;
    }
    int md = (s + e) >> 1;
    if (idx <= md) {
        L[ret] = upd(L[root], s, md, idx), R[ret] = R[root];
    } else {
        R[ret] = upd(R[root], md + 1, e, idx), L[ret] = L[root];
    }
    val[ret] = max(val[L[ret]], val[R[ret]]);
    return ret;
}
int qry(int node, int s, int e, int l, int r){
  if(r < s || e < l || !node)return 0; //Punishment Value
  if(l <= s && e <= r){
    return val[node];
  }
  int md = (s+e)>>1;
  return max(qry(L[node], s, md, l, r), qry(R[node],md+1,e,l,r));
}
int merge(int x, int y, int s, int e) {
    if(!x||!y)return x | y;
    if(s == e) {
        val[x] += val[y];
        return x;
    }
    int md = (s + e) >> 1;
    L[x] = merge(L[x], L[y], s, md);
    R[x] = merge(R[x], R[y], md+1,e);
    val[x] = val[L[x]] + val[R[x]];
    return x;
}
```

3.9 Sqrt Decomposition

```
// Source: https://cp-algorithms.com/data_structures/sqrt_decomposition.html

// input data
int n;
vector<int> a (n);

// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the block and the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];
```

```

// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}

// If you're getting TLE and can't optimize more, you could reduce the number of slow division operations using the following code:

int sum = 0;
int c_l = l / len,   c_r = r / len;
if (c_l == c_r)
    for (int i=l; i<=r; ++i)
        sum += a[i];
else {
    for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
        sum += a[i];
    for (int i=c_l+1; i<=c_r-1; ++i)
        sum += b[i];
    for (int i=c_r*len; i<=r; ++i)
        sum += a[i];
}
```

3.10 Treap

```
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
    item(int x, int y, int z){
        value = x;
        prior = y;
        cnt = z;
        rev = 0;
        l = r = NULL;
    }
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%c", char(t->value));
    output (t->r);
}

pitem gettreap(string s){
        pitem ret=NULL;
    int i;
       for(i=0;i<s.size();i++)merge(ret,ret,new item(s[i],(rand()<<15)+rand(),1));
    return ret;
}
```

3.11 Wavelet Tree

```
// remember your array and values must be 1-based
struct wavelet_tree {
    int lo, hi;
    wavelet_tree *l, *r;
    vector<int> b;

    //nos are in range [x,y]
    //array indices are [from, to)
    wavelet_tree(int *from, int *to, int x, int y) {
        lo = x, hi = y;
        if (lo == hi or from >= to)
            return;
        int mid = (lo + hi) / 2;
        auto f = [mid](int x) {
            return x <= mid;
        };
        b.reserve(to - from + 1);
        b.pb(0);
        for (auto it = from; it != to; it++)
            b.pb(b.back() + f(*it));
        //see how lambda function is used here
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid + 1, hi);
    }

    //kth smallest element in [l, r]
    int kth(int l, int r, int k) {
        if (l > r)
            return 0;
        if (lo == hi)
            return lo;
        int inLeft = b[r] - b[l - 1];
        int lb = b[l - 1]; //amt of nos in first (l-1) nos that go in left
        int rb = b[r]; //amt of nos in first (r) nos that go in left
        if (k <= inLeft)
            return this->l->kth(lb + 1, rb, k);
        return this->r->kth(l - lb, r - rb, k - inLeft);
    }

    //count of nos in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if (l > r or k < lo)
            return 0;
        if (hi <= k)
            return r - l + 1;
        int lb = b[l - 1], rb = b[r];
        return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
    }

    //count of nos in [l, r] equal to k
    int count(int l, int r, int k) {
        if (l > r or k < lo or k > hi)
            return 0;
        if (lo == hi)
            return r - l + 1;
        int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
        if (k <= mid)
            return this->l->count(lb + 1, rb, k);
        return this->r->count(l - lb, r - rb, k);
    }
};
```

4 DP

4.1 Dynamic Convex Hull Trick

```
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld  long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
struct Line
{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& other) const
    {
        return m < other.m;
    }
    bool operator<(const ll &x) const
    {
        const Line* s = succ();
        if (!s)
            return 0;
        return b - s->b < (s->m - m) * x;
    }
};
// will maintain upper hull for maximum
struct HullDynamic : public multiset<Line, less<>>
{
    bool bad(iterator y)
    {
        auto z = next(y);
        if (y == begin())
        {
            if (z == end())
                return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end())
            return y->m == x->m && y->b <= x->b;
        return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b)
    {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y))
        {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y)))
            erase(next(y));
        while (y != begin() && bad(prev(y)))
            erase(prev(y));
    }

    ll query(ll x)
    {

        auto l = *lower_bound(x);
        return l.m * x + l.b;
    }
};
int main()
{
    IO

}
```

4.2 Dynamic Connectivety with SegTree

```
/// MANGA
#pragma GCC optimize("O3")
#pragma GCC optimize ("unroll-loops")
#pragma GCC target("avx,avx2,fma")
using namespace std;

#include "bits/stdc++.h"

#define pb push_back
#define F first
#define S second
#define f(i, a, b)  for(int i = a; i < b; i++)
#define all(a)  a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define sz(x)  (int)(x).size()
//#define mp make_pair
#define popCnt(x)  (__builtin_popcountll(x))
typedef long long ll;
typedef pair<int, int> ii;
using ull = unsigned long long;
const int N = 1e5+5, LG = 17, MOD = 1e9 + 7;
const long double PI = acos(-1);
struct PT{
    ll x, y;
    PT() {}
    PT(ll a, ll b):x(a), y(b){}
    PT operator - (const PT & o) {return PT{x-o.x,y-o.y};}
    bool operator < (const PT & o) const {return make_pair(x,y) < make_pair(o.x, o.y);}
};
ll cross(PT x, PT y) {
    return x.x * y.y - x.y * y.x;
}
PT val[300005];
bool in[300005];
ll qr[300005];
bool ask[300005];
ll ans[N];
vector<PT> t[300005 * 4];   ///segment tree holding points to queries
void update(int node, int s, int e, int l, int r, PT x) {
    if(r < s || e < l)return;
    if(l <= s && e <= r) {  ///add this point to maximize it with queries in this range
        t[node].pb(x);
        return;
    }
    int md = (s + e) >> 1;
    update(node<<1,s,md,l,r,x);
    update(node<<1|1,md+1,e,l,r,x);
}
vector<PT> stk;
inline void addPts(vector<PT> v) {
    stk.clear();    ///reset the data structure you are using
    sort(all(v));
    ///build upper envelope
    for(int i = 0; i < v.size(); i++) {
        while(sz(stk) > 1 && cross(v[i] - stk.back(), stk.back() - stk[stk.size()-2]) <= 0)
            stk.pop_back();
        stk.push_back(v[i]);
    }
}
inline ll calc(PT x, ll val) {
    ///mb+y
    return x.x * val + x.y;
}

ll query(ll x) {
    if(stk.empty())
        return LLONG_MIN;
    int lo = 0, hi = stk.size() - 1;
    while(lo + 10 < hi) {
        int md = lo + (hi-lo) / 2;
        if(calc(stk[md+1],x) > calc(stk[md],x))
            lo = md + 1;
        else
            hi = md;
    }
    ll ans = LLONG_MIN;
    for(int i = lo; i <= hi; i++)
        ans = max(ans, calc(stk[i], x));
    return ans;
}
void solve(int node, int s, int e) {    ///Solve queries
    addPts(t[node]);    ///note that there is no need to add/delete just build for t[node]
    f(i,s,e+1){
        if(ask[i]) {
            ans[i] = max(ans[i], query(qr[i]));
        }
    }
    if(s==e)return;
    int md = (s + e) >> 1;
    solve(node<<1,s,md);
    solve(node<<1|1,md+1,e);
}
void doWork() {

    int n;
    cin >> n;
    stk.reserve(n);
    f(i,1,n+1) {
        int tp;
        cin >> tp;
        if(tp == 1) {   ///Add Query
            int x, y;
            cin >> x >> y;
            val[i] = PT(x, y);
            in[i] = 1;
        }   else if(tp == 2) {  ///Delete Query
            int x;
            cin >> x;
            if(in[x])update(1, 1, n, x, i - 1, val[x]);
            in[x] = 0;
        }   else {
            cin >> qr[i];
            ask[i] = true;
        }
    }
    f(i,1,n+1)  ///Finalize Query
        if(in[i])
            update(1, 1, n, i, n, val[i]);

    f(i,1,n+1)ans[i] = LLONG_MIN;
    solve(1, 1, n);
    f(i,1,n+1)
    if(ask[i]) {
        if(ans[i] == LLONG_MIN)
            cout << "EMPTY SET\n";
        else
            cout << ans[i] << '\n';
    }

}
int32_t main() {
#ifdef ONLINE_JUDGE
    ios_base::sync_with_stdio(0);
    cin.tie(0);
#endif // ONLINE_JUDGE
    int t = 1;
//    cin >> t;
    while (t--) {
        doWork();
    }
    return 0;
}
```

4.3 Li Chao Tree

```
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld  long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
struct Line
{
    ll m, b;
    Line(ll m, ll b) : m(m), b(b) {}
    ll operator()(ll x)
    {
        return m * x + b;
    }
};
struct node
{
    node * left,* right ;
    Line line ;
    node(node * left, node *right, Line line):left(left), right(right), line(line) {}
    node * getLeft()
    {
        if(left==NULL)
            left= new node (NULL,NULL,Line(0,1e18)) ;
        return left ;
    }
    node * getright()
    {
        if(right==NULL)
            right= new node (NULL,NULL,Line(0,1e18)) ;
        return right ;
    }
    void insert(Line newline, int l, int r)
    {
        int m=(l+r)/2;
        bool lef=newline(l)<line(l);
        bool mid=newline(m)<line(m);

        if(mid)
            swap(line,newline);
        if(r-l==1)
            return ;
        else if(lef!=mid)
            getLeft()->insert(newline,l,m);
        else
            getright()->insert(newline,m,r);
    }
    ll query(int x, int l, int r)
    {
        int m = (l + r) / 2;
        if(r - l == 1)
            return line(x);
        else if(x < m)
            return min(line(x), getLeft()->query(x, l, m));
        else
            return min(line(x), getright()->query(x, m, r));
    }
    void deletee()
    {
        if(left!=NULL)
            left->deletee();
        if(right!=NULL)
            right->deletee();
        free(this);
    }
};
int main()
{
    IO
    node * root = new node(NULL,NULL,Line(0,5));
    root->insert(Line(1,-3),1,100);

    for(int i=1; i<=10; i++)
        cout<<root->query(i,1,100)<<"\n";
}
```

4.4 CHT Line Container

```
struct Line
{
    mutable ll m, b, p;
    bool operator<(const Line& o) const
    {
        return m < o.m;
    }
    bool operator<(ll x) const
    {
        return p < x;
    }
};

struct LineContainer : multiset<Line, less<>>
{
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll db, ll dm)   // floored division
    {
        return db / dm - ((db ^ dm) < 0 && db % dm);
    }
    bool isect(iterator x, iterator y)
    {
        if (y == end())
        {
            x->p = inf;
            return false;
        }
        if (x->m == y->m)
            x->p = x->b > y->b ? inf : -inf;
        else
            x->p = div(y->b - x->b, x->m - y->m);
        return x->p >= y->p;
    }
    void add(ll m, ll b)
    {
        auto z = insert({m, b, 0}), y = z++, x = y;
        while (isect(y, z))
            z = erase(z);
        if (x != begin() && isect(--x, y))
            isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x)
    {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.m * x + l.b;
    }
};
```

5 Geometry

5.1 Convex Hull

```
struct point {
    ll x,  y;
    point(ll x, ll y) : x(x), y(y) {}
    point operator -(point other) {
        return point(x - other.x, y - other.y);
    }
    bool operator <(const point  &other) const {
        return x != other.x ? x < other.x : y < other.y;
    }
};
ll cross(point a, point b) {
    return a.x * b.y - a.y * b.x;
}
ll dot(point a, point b) {
    return a.x * b.x + a.y * b.y;
}
struct sortCCW {
    point center;

    sortCCW(point center) : center(center) {}

    bool operator()(point a, point b) {
        ll res = cross(a - center, b - center);
        if(res)
            return res > 0;
        return dot(a - center, a - center) < dot(b - center, b - center);
    }
};
vector<point> hull(vector<point> v) {
    sort(v.begin(), v.end());
    sort(v.begin() + 1, v.end(), sortCCW(v[0]));
    v.push_back(v[0]);
    vector<point> ans ;
    for(auto i : v) {
        int sz = ans.size();
        while(sz > 1 && cross(i - ans[sz - 1], ans[sz - 2] - ans[sz - 1]) <= 0)
            ans.pop_back(), sz--;
        ans.push_back(i);
    }
    ans.pop_back();
    return ans;
}
```

5.2 Geometry Template

```
using  ptype =  double edit this first ;
double EPS = 1e-9;
struct point {

    ptype x,  y;
    point(ptype x, ptype y) : x(x), y(y) {}

    point operator -(const point & other)const {
        return point(x - other.x, y - other.y);
    }

    point operator +(const point & other)const {
        return point(x + other.x, y + other.y);
    }

    point operator *(ptype c) const {
        return point(x * c, y * c);
    }

    point operator /(ptype c) const {
        return point(x / c, y / c);
    }
    point prep() {
        return point(-y, x);
    }

};
ptype cross(point a, point b) {
    return a.x * b.y - a.y * b.x;
}

ptype dot(point a, point b) {
    return a.x * b.x + a.y * b.y;
}
double abs(point a) {
    return sqrt(dot(a, a));
}
// angle between [0 , pi]
double angle (point a, point b) {
    return acos(dot(a, b) / abs(a) / abs(b));
}
// a : point in Line
// d : Line direction
point LineLineIntersect(point a1, point d1, point a2, point d2) {
    return a1 + d1 * cross(a2 - a1, d2) / cross(d1, d2);
}
// Line a---b
// point C
point ProjectPointLine(point a, point b, point c) {
    return a + (b - a) * 1.0 * dot(c - a, b - a) / dot(b - a, b - a);
}
// segment a---b
// point C
point ProjectPointSegment(point a, point b, point c) {
    double r = dot(c - a, b - a) / dot(b - a, b - a);
    if(r < 0)
        return a;
    if(r > 1)
        return b;
    return a + (b - a) * r ;
}
// Line a---b
// point p
point reflectAroundLine(point a, point b, point p) {
    //(proj-p) *2 + p
    return ProjectPointLine(a, b, p) * 2 - p;
}
// Around origin
point RotateCCW(point p, double t) {
    return point(p.x * cos(t) - p.y * sin(t),
                 p.x * sin(t) + p.y * cos(t));
}
// Line a---b
vector<point> CircleLineIntersect(point a, point b, point center, double r) {
    a = a - center;
    b = b - center;
    point p = ProjectPointLine(a, b, point(0, 0)); // project point from center to the Line
    if(dot(p, p) > r * r)
        return {};
    double len = sqrt(r * r - dot(p, p));
    if(len < EPS)
        return {center + p};

    point d = (a - b) / abs(a - b);
    return {center + p + d * len, center + p - d * len};
}
vector<point> CircleCircleIntersect(point c1, double r1, point c2, double r2) {

    if(r1 < r2) {
        swap(r1, r2);
        swap(c1, c2);
    }
    double d = abs(c1 - c2); // distance between c1,c2
    if(d > r1 + r2 || d < r1 - r2)
        return {};

    double angle = acos(min((d * d + r1 * r1 - r2 * r2) / (2 * r1 * d), 1.0));
    point p = (c2 - c1) / d * r1;

    if(angle < EPS)
        return {p};

    return {RotateCCW(p, angle), RotateCCW(p, -angle)};

}
point circumcircle(point p1, point p2, point p3) {

    return LineLineIntersect((p1 + p2) / 2, (p1 - p2).prep(),
                             (p1 + p3) / 2, (p1 - p3).prep() );
}
//S : Area.
//I : number points with integer coordinates lying strictly inside the polygon.
//B : number of points lying on polygon sides by B.
//S = I + B/2 - 1
```

5.3 Half Plane Intersection

```
// Redefine epsilon and infinity as necessary. Be mindful of precision errors.
const long double eps = 1e-9, inf = 1e9;

// Basic point/vector struct.
struct Point {

    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}

    // Addition, substraction, multiply by constant, cross product.

    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }
```

```cpp

    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }

    friend Point operator * (const Point& p, const long double& k) {
        return Point(p.x * k, p.y * k);
    }

    friend long double cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};

// Basic half-plane struct.
struct Halfplane {

    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;

    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }

    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }

    // Comparator for sorting.
    // If the angle of both half-planes is equal, the leftmost one should go first.
    bool operator < (const Halfplane& e) const {
        if (fabsl(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
        return angle < e.angle;
    }

    // We use equal comparator for std::unique to easily remove parallel half-planes.
    bool operator == (const Halfplane& e) const {
        return fabsl(angle - e.angle) < eps;
    }

    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};



// Actual algorithm
vector<Point> hp_intersect(vector<Halfplane>& H) {

    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };

    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }

    // Sort and remove duplicates
    sort(H.begin(), H.end());
    H.erase(unique(H.begin(), H.end()), H.end());

    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {

        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }

        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }

        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }

    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }

    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();

    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

5.4 Segments Intersection

```cpp
const double EPS = 1E-9;

struct pt {
    double x, y;
};

struct seg {
    pt p, q;
    int id;

    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

bool intersect1d(double l1, double r1, double l2, double r2) {
    if (l1 > r1)
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}

int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}

bool intersect(const seg& a, const seg& b)
{
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
           intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
           vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
           vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}

bool operator<(const seg& a, const seg& b)
{
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}

struct event {
    double x;
    int tp, id;

    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};

set<seg> s;
vector<set<seg>::iterator> where;

set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}

set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}

pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());

    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv, a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }

    return make_pair(-1, -1);
}
```

5.5 Rectangles Union

```
#include<bits/stdc++.h>
#define P(x,y) make_pair(x,y)
using namespace std;
class Rectangle {
public:
    int x1, y1, x2, y2;
    static Rectangle empt;
    Rectangle() {
        x1 = y1 = x2 = y2 = 0;
    }
    Rectangle(int X1, int Y1, int X2, int Y2) {
        x1 = X1;
        y1 = Y1;
        x2 = X2;
        y2 = Y2;
    }
};
struct Event {
    int x, y1, y2, type;
    Event() {}
    Event(int x, int y1, int y2, int type): x(x), y1(y1), y2(y2), type(type) {}
};
bool operator < (const Event&A, const Event&B) {
//if(A.x != B.x)
    return A.x < B.x;
//if(A.y1 != B.y1) return A.y1 < B.y1;
//if(A.y2 != B.y2()) A.y2 < B.y2;
}
const int MX = (1 << 17);
struct Node {
    int prob, sum, ans;
    Node() {}
    Node(int prob, int sum, int ans): prob(prob), sum(sum), ans(ans) {}
};
Node tree[MX * 4];
int interval[MX];
void build(int x, int a, int b) {
    tree[x] = Node(0, 0, 0);
    if(a == b) {
        tree[x].sum += interval[a];
        return;
    }
    build(x * 2, a, (a + b) / 2);
    build(x * 2 + 1, (a + b) / 2 + 1, b);
    tree[x].sum = tree[x * 2].sum + tree[x * 2 + 1].sum;
}
int ask(int x) {
    if(tree[x].prob)
        return tree[x].sum;
    return tree[x].ans;
}
int st, en, V;
void update(int x, int a, int b) {
    if(st > b || en < a)
        return;
    if(a >= st && b <= en) {
        tree[x].prob += V;
        return;
    }
    update(x * 2, a, (a + b) / 2);
    update(x * 2 + 1, (a + b) / 2 + 1, b);
    tree[x].ans = ask(x * 2) + ask(x * 2 + 1);
}
Rectangle Rectangle::empt = Rectangle();
vector < Rectangle > Rect;
vector < int > sorted;
vector < Event > sweep;
void compressncalc() {
```

```cpp
    sweep.clear();
    sorted.clear();
    for(auto R : Rect) {
        sorted.push_back(R.y1);
        sorted.push_back(R.y2);
    }
    sort(sorted.begin(), sorted.end());
    sorted.erase(unique(sorted.begin(), sorted.end()), sorted.end());
    int sz = sorted.size();
    for(int j = 0; j < sorted.size() - 1; j++)
        interval[j + 1] = sorted[j + 1] - sorted[j];
    for(auto R : Rect) {
        sweep.push_back(Event(R.x1, R.y1, R.y2, 1));
        sweep.push_back(Event(R.x2, R.y1, R.y2, -1));
    }
    sort(sweep.begin(), sweep.end());
    build(1, 1, sz - 1);
}
long long ans;
void Sweep() {
    ans = 0;
    if(sorted.empty() || sweep.empty())
        return;
    int last = 0, sz_ = sorted.size();
    for(int j = 0; j < sweep.size(); j++) {
        ans += 1ll * (sweep[j].x - last) * ask(1);
        last = sweep[j].x;
        V = sweep[j].type;
        st = lower_bound(sorted.begin(), sorted.end(), sweep[j].y1) - sorted.begin() + 1;
        en = lower_bound(sorted.begin(), sorted.end(), sweep[j].y2) - sorted.begin();
        update(1, 1, sz_ - 1);
    }
}
int main() {
//    freopen("in.in","r",stdin);
    int n;
    scanf("%d", &n);
    for(int j = 1; j <= n; j++) {
        int a, b, c, d;
        scanf("%d %d %d %d", &a, &b, &c, &d);
        Rect.push_back(Rectangle(a, b, c, d));
    }
    compressncalc();
    Sweep();
    cout << ans << endl;
}
```

6 Graphs

6.1 2 SAD

```cpp
/**
 * Author: Emil Lenngren, Simon Lindholm
 * Date: 2011-11-29
 * License: CC0
 * Source: folklore
 * Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ becomes true, or reports that it is unsatisfiable.
 * Negated variables are represented by bit-inversions (\texttt{\tilde{}x}).
 * Usage:
 *  TwoSat ts(number of boolean variables);
 *  ts.either(0, \tilde3); // Var 0 is true or var 3 is false
 *  ts.setValue(2); // Var 2 is true
 *  ts.atMostOne({0,\tilde1,2}); // <= 1 of vars 0, \tilde1 and 2 are true
 *  ts.solve(); // Returns true iff it is solvable
 *  ts.values[0..N-1] holds the assigned values to the vars
 * Time: O(N+E), where N is the number of boolean variables, and E is the number of clauses.
 * Status: stress-tested
 */
#pragma once

struct TwoSat {
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

    TwoSat(int n = 0) : N(n), gr(2*n) {}

    int addVar() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++;
    }

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }
    void setValue(int x) { either(x, x); }

    void atMostOne(const vi& li) { // (optional)
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        rep(i,2,sz(li)) {
            int next = addVar();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }

    vi val, comp, z; int time = 0;
    int dfs(int i) {
        int low = val[i] = ++time, x; z.push_back(i);
        for(int e : gr[i]) if (!comp[e])
            low = min(low, val[e] ?: dfs(e));
        if (low == val[i]) do {
            x = z.back(); z.pop_back();
            comp[x] = low;
            if (values[x>>1] == -1)
                values[x>>1] = x&1;
        } while (x != i);
        return val[i] = low;
    }

    bool solve() {
        values.assign(N, -1);
        val.assign(2*N, 0); comp = val;
        rep(i,0,2*N) if (!comp[i]) dfs(i);
        rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
        return 1;
    }
};
```

6.2 Ariculation Point

```cpp
vector<int> adj[N];
int dfsn[N], low[N], instack[N], ar_point[N], timer;
stack<int> st;

void dfs(int node, int par){
    dfsn[node] = low[node] = ++timer;
    int kam = 0;
    for(auto i: adj[node]){
        if(i == par) continue;
        if(dfsn[i] == 0){
            kam++;
            dfs(i, node);
            low[node] = min(low[node], low[i]);
            if(dfsn[node] <= low[i] && par != 0) ar_point[node] = 1;
        }
        else low[node] = min(low[node], dfsn[i]);
    }
    if(par == 0 && kam > 1) ar_point[node] = 1;
}

void init(int n){
    for(int i = 1; i <= n; i++){
        adj[i].clear();
        low[i] = dfsn[i] = 0;
        instack[i] = 0;
        ar_point[i] = 0;
    }
    timer = 0;
}

int main(){
    int tt;
    cin >> tt;
    while(tt--){
        // Input
        init(n);
        for(int i = 1; i <= n; i++){
            if(dfsn[i] == 0) dfs(i, 0);
        }
        int c = 0;
        for(int i = 1; i <= n; i++){
            if(ar_point[i]) c++;
        }
        cout << c << '\n';
    }
    return 0;
}
```

6.3 Bridges Tree and Diameter

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 3e5 + 5, mod = 1e9 + 7;

vector<int> adj[N], bridge_tree[N];
int dfsn[N], low[N], cost[N], timer, cnt, comp_id[N], kam[N], ans;
stack<int> st;


void dfs(int node, int par){
    dfsn[node] = low[node] = ++timer;
    st.push(node);
    for(auto i: adj[node]){
        if(i == par) continue;
        if(dfsn[i] == 0){
            dfs(i, node);
            low[node] = min(low[node], low[i]);
        }
        else low[node] = min(low[node], dfsn[i]);
    }
    if(dfsn[node] == low[node]){
        cnt++;
        while(1){
            int cur = st.top();
            st.pop();
            comp_id[cur] = cnt;
            if(cur == node) break;
        }
    }
}

void dfs2(int node, int par){
    kam[node] = 0;
    int mx = 0, second_mx = 0;
    for(auto i: bridge_tree[node]){
        if(i == par) continue;
        dfs2(i, node);
        kam[node] = max(kam[node], 1 + kam[i]);
        if(kam[i] > mx){
            second_mx = mx;
            mx = kam[i];
        }
        else second_mx = max(second_mx, kam[i]);
    }
    ans = max(ans, kam[node]);
    if(second_mx) ans = max(ans, 2 + mx + second_mx);
}

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n, m;
    cin >> n >> m;
    while(m--){
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1, 0);
    for(int i = 1; i <= n; i++){
        for(auto j: adj[i]){
            if(comp_id[i] != comp_id[j]){
                bridge_tree[comp_id[i]].push_back(comp_id[j]);
            }
        }
    }
    dfs2(1, 0);
    cout << ans;

    return 0;
}
```

6.4 Dinic With Scalling

```
///O(ElgFlow) on Bipratite Graphs and O(EVlgFlow) on other graphs (I think)
struct Dinic {
    #define vi vector<int>
    #define rep(i,a,b)  f(i,a,b)
    struct Edge {
        int to, rev;
        ll c, oc;
        int id;
        ll flow() { return max(oc - c, 0LL); } // if you need flows
    };
    vi lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, int id, ll rcap = 0) {
        adj[a].push_back({b, sz(adj[b]), c, c, id});
        adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap,id});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < sz(adj[v]); i++) {
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0; q[0] = s;
        rep(L,0,31) do { // 'int L=30' maybe faster for random data
            lvl = ptr = vi(sz(q));
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) {
                int v = q[qi++];
                for (Edge e : adj[v])
                    if (!lvl[e.to] && e.c >> (30 - L))
                        q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
            }
            while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
        } while (lvl[t]);
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

6.5 Gomory Hu

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2020-04-03
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/graph/gomory_hu_tree.cc#L102
 * Description: Given a list of edges representing an undirected flow graph,
 * returns edges of the Gomory-Hu tree. The max flow between any pair of
 * vertices is given by minimum edge weight along the Gomory-Hu tree path.
 * Time: $O(V)$ Flow Computations
 * Status: Tested on CERC 2015 J, stress-tested
 *
 * Details: The implementation used here is not actually the original
 * Gomory-Hu, but Gusfield's simplified version: "Very simple methods for all
 * pairs network flow analysis". PushRelabel is used here, but any flow
 * implementation that supports 'leftOfMinCut' also works.
 */
#pragma once

#include "PushRelabel.h"

typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    }
    return tree;
}
```

6.6 HopcraftKarp BPM

```
/**
 * Author: Chen Xing
 * Date: 2009-10-13
 * License: CC0
 * Source: N/A
 * Description: Fast bipartite matching algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and $btoa$ should be a vector full of
 * -1's of the same size as the right partition. Returns the size of
 * the matching. $btoa[i]$ will be the match for vertex $i$ on the right side,
 * or $-1$ if it's not matched.
 * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
 * Time: O(\sqrt{V}E)
 * Status: stress-tested by MinimumVertexCover, and tested on oldkattis.adkbipmatch and SPOJ:MATCHING
 */
#pragma once

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        /// Find the starting nodes for BFS (i.e. layer 0).
        cur.clear();
        for (int a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
        /// Find all layers using bfs.
        for (int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]);
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        /// Use DFS to scan for augmenting paths.
        rep(a,0,sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    }
}
```

6.7 Hungarian

```
/*
    Notes:
        note that n must be <= m
        so in case in your problem n >= m, just swap
    also note this
    void set(int x, int y, ll v){a[x+1][y+1]=v;}
    the algorithim assumes you're using 0-index
    but it's using 1-based
*/
struct Hungarian {
    const ll INF = 1000000000000000000; ///10^18
    int n,m;
    vector<vector<ll> > a;
    vector<ll> u,v;vector<int> p,way;
    Hungarian(int n, int m):
    n(n),m(m),a(n+1,vector<ll>(m+1,INF-1)),u(n+1),v(m+1),p(m+1),way(m+1){}
    void set(int x, int y, ll v){a[x+1][y+1]=v;}
    ll assign(){
        for(int i = 1; i <= n; i++){
            int j0=0;p[0]=i;
            vector<ll> minv(m+1,INF);
            vector<char> used(m+1,false);
            do {
                used[j0]=true;
                int i0=p[j0],j1;ll delta=INF;
                for(int j = 1; j <= m; j++)if(!used[j]){
                    ll cur=a[i0][j]-u[i0]-v[j];
                    if(cur<minv[j])minv[j]=cur,way[j]=j0;
                    if(minv[j]<delta)delta=minv[j],j1=j;
                }
                for(int j = 0; j <= m; j++)
                    if(used[j])u[p[j]]+=delta,v[j]-=delta;
                    else minv[j]-=delta;
                j0=j1;
            } while(p[j0]);
            do {
                int j1=way[j0];p[j0]=p[j1];j0=j1;
            } while(j0);
        }
        return -v[0];
    }
    vector<int> restoreAnswer() {   ///run it after assign
        vector<int> ans (n+1);
        for (int j=1; j<=m; ++j)
            ans[p[j]] = j;
        return ans;
    }
};
```

6.8 Kosaraju

```
/*
  g : Adjacency List of the original graph
  rg : Reversed Adjacency List
  vis : A bitset to mark visited nodes
  adj : Adjacency List of the super graph
  stk : holds dfs ordered elements
  cmp[i] : holds the component of node i
  go[i] : holds the nodes inside the strongly connected component i
*/

#define FOR(i,a,b) for(int i = a; i < b; i++)
#define pb push_back

const int N = 1e5+5;

vector<vector<int>>g, rg;
vector<vector<int>>go;
bitset<N>vis;
vector<vector<int>>adj;
stack<int>stk;
int n, m, cmp[N];
void add_edge(int u, int v){
  g[u].push_back(v);
  rg[v].push_back(u);
}
void dfs(int u){
  vis[u]=1;
  for(auto v : g[u])if(!vis[v])dfs(v);
  stk.push(u);
}
void rdfs(int u,int c){
  vis[u] = 1;
  cmp[u] = c;
  go[c].push_back(u);
  for(auto v : rg[u])if(!vis[v])rdfs(v,c);
}
int scc(){
  vis.reset();
  for(int i = 0; i < n; i++)if(!vis[i])
    dfs(i);
  vis.reset();
  int c = 0;
  while(stk.size()){
    auto cur = stk.top();
    stk.pop();
    if(!vis[cur])
      rdfs(cur,c++);

  }
  return c;
}
```

6.9 Krichoff

```
/*
    Count number of spanning trees in a graph
*/
int power(long long n, long long k) {
  int ans = 1;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
int det(vector<vector<int>> a) {
  int n = a.size(), m = (int)a[0].size();
  int free_var = 0;
  const long long MODSQ = (long long)mod * mod;
  int det = 1, rank = 0;
  for (int col = 0, row = 0; col < m && row < n; col++) {
    int mx = row;
    for (int k = row; k < n; k++) if (a[k][col] > a[mx][col]) mx = k;
    if (a[mx][col] == 0) {
      det = 0;
      continue;
    }
    for (int j = col; j < m; j++) swap(a[mx][j], a[row][j]);
    if (row != mx) det = det == 0 ? 0 : mod - det;
    det = 1LL * det * a[row][col] % mod;
    int inv = power(a[row][col], mod - 2);
    for (int i = 0; i < n && inv; i++) {
      if (i != row && a[i][col]) {
        int x = ((long long)a[i][col] * inv) % mod;
        for (int j = col; j < m && x; j++) {
          if (a[row][j]) a[i][j] = (MODSQ + a[i][j] - ((long long)a[row][j] * x)
               ) % mod;
        }
      }
    }
    row++;
    ++rank;
  }
  return det;
}
```

6.10 Manhattan MST

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

const int N = 2e5 + 9;

int n;
vector<pair<int, int>> g[N];
```

```
struct PT {
  int x, y, id;
  bool operator < (const PT &p) const {
    return x == p.x ? y < p.y : x < p.x;
  }
} p[N];
struct node {
  int val, id;
} t[N];
struct DSU {
  int p[N];
  void init(int n) { for (int i = 1; i <= n; i++)  p[i] = i; }
  int find(int u) { return p[u] == u ? u : p[u] = find(p[u]); }
  void merge(int u, int v) { p[find(u)] = find(v); }
} dsu;
struct edge {
  int u, v, w;
  bool operator < (const edge &p) const { return w < p.w; }
};
vector<edge> edges;
int query(int x) {
  int r = 2e9 + 10, id = -1;
  for (; x <= n; x += (x & -x)) if (t[x].val < r) r = t[x].val, id = t[x].id;
  return id;
}
void modify(int x, int w, int id) {
  for (; x > 0; x -= (x & -x)) if (t[x].val > w) t[x].val = w, t[x].id = id;
}
int dist(PT &a, PT &b) {
  return abs(a.x - b.x) + abs(a.y - b.y);
}
void add(int u, int v, int w) {
  edges.push_back({u, v, w});
}
long long Kruskal() {
  dsu.init(n);
  sort(edges.begin(), edges.end());
  long long ans = 0;
  for (edge e : edges) {
    int u = e.u, v = e.v, w = e.w;
    if (dsu.find(u) != dsu.find(v)) {
      ans += w;
      g[u].push_back({v, w});
      //g[v].push_back({u, w});
      dsu.merge(u, v);
    }
  }
  return ans;
}
void Manhattan() {
  for (int i = 1; i <= n; ++i) p[i].id = i;
  for (int dir = 1; dir <= 4; ++dir) {
    if (dir == 2 || dir == 4) {
      for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
    }
    else if (dir == 3) {
      for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
    }
    sort(p + 1, p + 1 + n);
    vector<int> v;
    static int a[N];
    for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x, v.push_back(a[i]);
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;
    for (int i = 1; i <= n; ++i) t[i].val = 2e9 + 10, t[i].id = -1;
    for (int i = n; i >= 1; --i) {
      int pos = query(a[i]);
      if (pos != -1) add(p[i].id, p[pos].id, dist(p[i], p[pos]));
      modify(a[i], p[i].x + p[i].y, i);
    }
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cin >> n;
  for (int i = 1; i <= n; i++) cin >> p[i].x >> p[i].y;
  Manhattan();
  cout << Kruskal() << '\n';
  for (int u = 1; u <= n; u++) {
    for (auto x: g[u]) cout << u - 1 << ' ' << x.first - 1 << '\n';
  }
  return 0;
}
```

6.11 Maximum Clique

```
///Complexity O(3 ^ (N/3)) i.e works for 50
///you can change it to maximum independent set by flipping the edges 0->1, 1->0
///if you want to extract the nodes they are 1-bits in R
int g[60][60];
int res;
long long edges[60];
void BronKerbosch(int n, long long R, long long P, long long X) {
  if (P == 0LL && X == 0LL) { //here we will find all possible maximal cliques (not maximum) i.e. there is no node which can be included in this set
    int t = __builtin_popcountll(R);
    res = max(res, t);
    return;
  }
  int u = 0;
  while (!((1LL << u) & (P | X))) u ++;
  for (int v = 0; v < n; v++) {
    if (((1LL << v) & P) && !((1LL << v) & edges[u])) {
      BronKerbosch(n, R | (1LL << v), P & edges[v], X & edges[v]);
      P -= (1LL << v);
      X |= (1LL << v);
    }
  }
}
int max_clique (int n) {
  res = 0;
  for (int i = 1; i <= n; i++) {
    edges[i - 1] = 0;
    for (int j = 1; j <= n; j++)  if (g[i][j]) edges[i - 1] |= ( 1LL << (j - 1));
  }
  BronKerbosch(n, 0, (1LL << n) - 1, 0);
  return res;
}
```

6.12 MCMF

```
/*
    Notes:
        make sure you notice the #define int ll
        focus on the data types of the max flow everythign inside is integer
        addEdge(u,v,cap,cost)
        note that for min cost max flow the cost is sum of cost * flow over all edges
*/

struct Edge {
    int to;
    int cost;
    int cap, flow, backEdge;
};

struct MCMF {

    const int inf = 1000000010;
    int n;
```

```cpp
    vector<vector<Edge>> g;

    MCMF(int _n) {
        n = _n + 1;
        g.resize(n);
    }

    void addEdge(int u, int v, int cap, int cost) {
        Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
        Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
        g[u].push_back(e1);
        g[v].push_back(e2);
    }

    pair<int, int> minCostMaxFlow(int s, int t) {
        int flow = 0;
        int cost = 0;
        vector<int> state(n), from(n), from_edge(n);
        vector<int> d(n);
        deque<int> q;
        while (true) {
            for (int i = 0; i < n; i++)
                state[i] = 2, d[i] = inf, from[i] = -1;
            state[s] = 1;
            q.clear();
            q.push_back(s);
            d[s] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop_front();
                state[v] = 0;
                for (int i = 0; i < (int) g[v].size(); i++) {
                    Edge e = g[v][i];
                    if (e.flow >= e.cap || (d[e.to] <= d[v] + e.cost))
                        continue;
                    int to = e.to;
                    d[to] = d[v] + e.cost;
                    from[to] = v;
                    from_edge[to] = i;
                    if (state[to] == 1) continue;
                    if (!state[to] || (!q.empty() && d[q.front()] > d[to]))
                        q.push_front(to);
                    else q.push_back(to);
                    state[to] = 1;
                }
            }
            if (d[t] == inf) break;
            int it = t, addflow = inf;
            while (it != s) {
                addflow = min(addflow,
                              g[from[it]][from_edge[it]].cap
                              - g[from[it]][from_edge[it]].flow);
                it = from[it];
            }
            it = t;
            while (it != s) {
                g[from[it]][from_edge[it]].flow += addflow;
                g[it][g[from[it]][from_edge[it]].backEdge].flow -= addflow;
                cost += g[from[it]][from_edge[it]].cost * addflow;
                it = from[it];
            }
            flow += addflow;
        }
        return {cost, flow};
    }
};
```

6.13 Minimum Arbroscene in a Graph

```cpp
const int maxn = 2510, maxm = 7000000;
const ll maxint = 0x3f3f3f3f3f3f3f3fLL;

int n, ec, ID[maxn], pre[maxn], vis[maxn];
ll in[maxn];

struct edge_t {
    int u, v;
    ll w;
} edge[maxm];
void add(int u, int v, ll w) {
    edge[++ec].u = u, edge[ec].v = v, edge[ec].w = w;
}

ll arborescence(int n, int root) {
    ll res = 0, index;
    while (true) {
        for (int i = 1; i <= n; ++i) {
            in[i] = maxint, vis[i] = -1, ID[i] = -1;
        }
        for (int i = 1; i <= ec; ++i) {
            int u = edge[i].u, v = edge[i].v;
            if (u == v || in[v] <= edge[i].w) continue;
            in[v] = edge[i].w, pre[v] = u;
        }
        pre[root] = root, in[root] = 0;
        for (int i = 1; i <= n; ++i) {
            res += in[i];
            if (in[i] == maxint) return -1;
        }
        index = 0;
        for (int i = 1; i <= n; ++i) {
            if (vis[i] != -1) continue;
            int u = i, v;
            while (vis[u] == -1) {
                vis[u] = i;
                u = pre[u];
            }
            if (vis[u] != i || u == root) continue;
            for (v = u, u = pre[u], ++index; u != v; u = pre[u]) ID[u] = index;
            ID[v] = index;
        }
        if (index == 0) return res;
        for (int i = 1; i <= n; ++i) if (ID[i] == -1) ID[i] = ++index;
        for (int i = 1; i <= ec; ++i) {
            int u = edge[i].u, v = edge[i].v;
            edge[i].u = ID[u], edge[i].v = ID[v];
            edge[i].w -= in[v];
        }
        n = index, root = ID[root];
    }
    return res;
}
```

6.14 Minmimum Vertex Cover (Bipartite)

```cpp
int myrandom (int i) { return std::rand()%i;}

struct MinimumVertexCover {
    int n, id;
    vector<vector<int> > g;
    vector<int> color, m, seen;
    vector<int> comp[2];
    MinimumVertexCover() {}
    MinimumVertexCover(int n, vector<vector<int> > g) {

        this->n = n;
        this->g = g;
        color = m = vector<int>(n, -1);
        seen = vector<int>(n, 0);
        makeBipartite();
    }

    void dfsBipartite(int node, int col) {
        if (color[node] != -1) {
            assert(color[node] == col); /* MSH BIPARTITE YA BASHMOHANDES */
            return;
        }
        color[node] = col;
        comp[col].push_back(node);
        for (int i = 0; i < int(g[node].size()); i++)
            dfsBipartite(g[node][i], 1 - col);
    }

    void makeBipartite() {
        for (int i = 0; i < n; i++)
            if (color[i] == -1)
                dfsBipartite(i, 0);
    }

    // match a node
    bool dfs(int node) {
      random_shuffle(g[node].begin(),g[node].end());
        for (int i = 0; i < g[node].size(); i++) {
            int child = g[node][i];
            if (m[child] == -1) {
                m[node] = child;
                m[child] = node;
                return true;
            }
            if (seen[child] == id)
                continue;
            seen[child] = id;
            int enemy = m[child];
            m[node] = child;
            m[child] = node;
            m[enemy] = -1;
            if (dfs(enemy))
                return true;
            m[node] = -1;
            m[child] = enemy;
            m[enemy] = child;
        }
        return false;
    }

    void makeMatching() {
    for(int j = 0; j < 5; j++)
      random_shuffle(comp[0].begin(),comp[0].end(),myrandom );
        for (int i = 0; i < int(comp[0].size()); i++) {
            id++;
            if(m[comp[0][i]] == -1)
                dfs(comp[0][i]);
        }
    }


    void recurse(int node, int x, vector<int> &minCover, vector<int> &done) {
        if (m[node] != -1)
            return;
        if (done[node])return;
        done[node] = 1;
        for (int i = 0; i < int(g[node].size()); i++) {
            int child = g[node][i];
            int newnode = m[child];
            if (done[child]) continue;
            if(newnode == -1) {
                continue;
            }
            done[child] = 2;
            minCover.push_back(child);
            m[newnode] = -1;
            recurse(newnode, x, minCover, done);
        }
    }

    vector<int> getAnswer() {
        vector<int> minCover, maxIndep;
        vector<int> done(n, 0);
        makeMatching();
        for (int x = 0; x < 2; x++)
            for (int i = 0; i < int(comp[x].size()); i++) {
                int node = comp[x][i];
                if (m[node] == -1)
                    recurse(node, x, minCover, done);
            }

        for (int i = 0; i < int(comp[0].size()); i++)
            if (!done[comp[0][i]]) {
                minCover.push_back(comp[0][i]);
            }
        return minCover;
    }
};
```

6.15 Prufer Code

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

const int N = 3e5 + 9;

/*
prufer code is a sequence of length n-2 to uniquely determine a labeled tree with n vertices
Each time take the leaf with the lowest number and add the node number the leaf is connected to
the sequence and remove the leaf. Then break the algo after n-2 iterations
*/
//0-indexed
int n;
vector<int> g[N];
int parent[N], degree[N];

void dfs (int v) {
  for (size_t i = 0; i < g[v].size(); ++i) {
    int to = g[v][i];
    if (to != parent[v]) {
      parent[to] = v;
      dfs (to);
    }
  }
}

vector<int> prufer_code() {
  parent[n - 1] = -1;
  dfs (n - 1);
  int ptr = -1;
  for (int i = 0; i < n; ++i) {
    degree[i] = (int) g[i].size();
    if (degree[i] == 1 && ptr == -1) ptr = i;
  }
  vector<int> result;
  int leaf = ptr;
  for (int iter = 0; iter < n - 2; ++iter) {
    int next = parent[leaf];
    result.push_back (next);
    --degree[next];
    if (degree[next] == 1 && next < ptr) leaf = next;
    else {
      ++ptr;
      while (ptr < n && degree[ptr] != 1) ++ptr;
      leaf = ptr;
    }
  }
  return result;
}
vector < pair<int, int> > prufer_to_tree(const vector<int> & prufer_code) {
  int n = (int) prufer_code.size() + 2;
  vector<int> degree (n, 1);
  for (int i = 0; i < n - 2; ++i) ++degree[prufer_code[i]];

  int ptr = 0;
  while (ptr < n && degree[ptr] != 1) ++ptr;
  int leaf = ptr;
  vector < pair<int, int> > result;
  for (int i = 0; i < n - 2; ++i) {
    int v = prufer_code[i];
    result.push_back (make_pair (leaf, v));
    --degree[leaf];
    if (--degree[v] == 1 && v < ptr) leaf = v;
    else {
      ++ptr;
      while (ptr < n && degree[ptr] != 1) ++ptr;
      leaf = ptr;
    }
  }
  for (int v = 0; v < n - 1; ++v) if (degree[v] == 1) result.push_back (
      make_pair (v, n - 1));
  return result;
}

int32_t main() {

  return 0;
}
```

6.16 Push Relabel Max Flow

```
struct edge
{
    int from, to, cap, flow, index;
    edge(int from, int to, int cap, int flow, int index):
        from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel
{
    int n;
    vector<vector<edge> > g;
    vector<long long> excess;
    vector<int> height, active, count;
    queue<int> Q;

    PushRelabel(int n):
        n(n), g(n), excess(n), height(n), active(n), count(2*n) {}

    void addEdge(int from, int to, int cap)
    {
        g[from].push_back(edge(from, to, cap, 0, g[to].size()));
        if(from==to)
            g[from].back().index++;
        g[to].push_back(edge(to, from, 0, 0, g[from].size()-1));
    }

    void enqueue(int v)
    {
        if(!active[v] && excess[v] > 0)
        {
            active[v]=true;
            Q.push(v);
        }
    }

    void push(edge &e)
    {
        int amt=(int)min(excess[e.from], (long long)e.cap - e.flow);
        if(height[e.from]<=height[e.to] || amt==0)
            return;
        e.flow += amt;
        g[e.to][e.index].flow -= amt;
        excess[e.to] += amt;
        excess[e.from] -= amt;
        enqueue(e.to);
    }

    void relabel(int v)
    {
        count[height[v]]--;
        int d=2*n;
        for(auto &it:g[v])
        {
            if(it.cap-it.flow>0)
                d=min(d, height[it.to]+1);
        }
        height[v]=d;
        count[height[v]]++;
        enqueue(v);
    }

    void gap(int k)
    {
        for(int v=0;v<n;v++)
        {
            if(height[v]<k)
                continue;
            count[height[v]]--;
            height[v]=max(height[v], n+1);
            count[height[v]]++;
            enqueue(v);
        }
    }

    void discharge(int v)
    {
        for(int i=0; excess[v]>0 && i<g[v].size(); i++)
            push(g[v][i]);
        if(excess[v]>0)
        {
            if(count[height[v]]==1)
                gap(height[v]);
            else
                relabel(v);
        }
    }

    long long max_flow(int source, int dest)
    {
        count[0] = n-1;
        count[n] = 1;
        height[source] = n;
        active[source] = active[dest] = 1;
        for(auto &it:g[source])
        {
            excess[source]+=it.cap;
            push(it);
        }

        while(!Q.empty())
        {
            int v=Q.front();
            Q.pop();
            active[v]=false;
            discharge(v);
        }

        long long max_flow=0;
        for(auto &e:g[source])
            max_flow+=e.flow;

        return max_flow;
    }
};
```

6.17 Tarjan Algo

```
vector< vector<int> > scc;
vector<int> adj[N];
int dfsn[N], low[N], cost[N], timer, in_stack[N];
stack<int> st;

// to detect all the components (cycles) in a directed graph
void tarjan(int node){
    dfsn[node] = low[node] = ++timer;
    in_stack[node] = 1;
    st.push(node);
    for(auto i: adj[node]){
        if(dfsn[i] == 0){
            tarjan(i);
            low[node] = min(low[node], low[i]);
        }
        else if(in_stack[i]) low[node] = min(low[node], dfsn[i]);
    }
    if(dfsn[node] == low[node]){
        scc.push_back(vector<int>());
        while(1){
            int cur = st.top();
            st.pop();
            in_stack[cur] = 0;
            scc.back().push_back(cur);
            if(cur == node) break;
        }
    }
}
int main(){
    int m;
    cin >> m;
    while(m--){
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }
    for(int i = 1; i <= n; i++){
        if(dfsn[i] == 0){
            tarjan(i);
        }
    }

    return 0;
}
```

6.18 Bipartite Matching

```
#include<iostream>
#include <bits/stdc++.h>
#define ll long long
#define ld long double
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
struct graph
{
    int L, R;
    vector<vector<int> > adj;
    graph(int l, int r) : L(l), R(r), adj(l+1) {}
    void add_edge(int u, int v)
    {
        adj[u].push_back(v+L);
    }
    int maximum_matching()
    {
        vector<int> mate(L+R+1,-1), level(L+1);
        function<bool (void)> levelize  = [&]()
        {
            queue<int> q;
            for(int i=1; i<=L; i++)
            {
                level[i]=-1;
                if(mate[i]<0)
                    q.push(i), level[i]=0;
            }
            while(!q.empty())
            {
                int node=q.front();
                q.pop();
                for(auto i : adj[node])
                {
                    int v=mate[i];
                    if(v<0)
                        return true;
                    if(level[v]<0)
                    {
                        level[v]=level[node]+1;
                        q.push(v);
                    }
                }
            }
            return false;
        };
        function<bool (int)> augment =[&](int node)
        {
            for(auto i : adj[node])
            {
                int v=mate[i];
                if(v<0 || (level[v]>level[node] && augment(v)))
                {
                    mate[node]=i;
                    mate[i]=node;
                    return true;
                }
            }
            return false;
        };
        int match=0;
        while(levelize())
            for(int i=1; i<=L; i++)
                if(mate[i] < 0 && augment(i))
                    match++;
        return match;
    }
};

int main()
{
    IO
    int L, R, m;
    cin>>L>>R>>m;
    graph g(L, R);
    for (int i = 0; i < m; ++i)
    {
        int u, v;
        cin>>u>>v;
        g.add_edge(u, v);
    }
    cout<<g.maximum_matching();
}
```

7 Math

7.1 Xor With Gauss

```
/*
    Some applications
    If you want to find the maximum in xor subset
    just ans = max(ans, ans ^ p[i]) for all i
    if you want to count the number of subsets with a certain value
    check all different subsets of p
*/
ll p[66];
bool add(ll x) {
    for(int i = 60; (~i) && x; --i) {
        if(x >> i & 1) {
            if(!p[i]) {
                p[i] = x;
                return true;
            }   else {
                x ^= p[i];
            }
        }
    }
    return false;
}
```

7.2 Josephus

```
// n = total person
// will kill every kth person, if k = 2, 2,4,6,...
// returns the mth killed person
ll josephus(ll n, ll k, ll m) {
  m = n - m;
  if (k <= 1)return n - m;
  ll i = m;
  while (i < n) {
    ll r = (i - m + k - 2) / (k - 1);
    if ((i + r) > n) r = n - i;
    else if (!r) r = 1;
    i += r;
    m = (m + (r * k)) % i;
  } return m + 1;
}
```

7.3 Matrix Power/Multiplication

```
struct Matrix {

    const static int D = 100;
    int a[D][D];

    Matrix(int val) {
        for(int i = 0; i < D; i++)
            for(int j = 0; j < D; j++)
                a[i][j] = val;
    }
    void clear() {
        memset(a, 0, sizeof a);
    }
    void initIdentity() {
        clear();
        for(int i = 0; i < D; i++)
            a[i][i] = 1;
    }
    int * operator [](int r) {
        return a[r];
    }
    const int * operator [](int r) const{
        return a[r];
    }

    friend Matrix operator * (const Matrix & a, const Matrix & b) {
        Matrix ret(0);
        for(int k = 0; k < D; k++)
            for(int i = 0; i < D; i++)if(a[i][k])
                for(int j = 0; j < D; j++)
                    ret[i][j] = (ret[i][j] + 1ll * a[i][k] * b[k][j]) % MOD;
        return ret;
    }

};
Matrix raiseMatrix(Matrix trans, ll k) {
    Matrix res(0);
    res.initIdentity();
    for(;k;k>>=1,trans = trans * trans)
        if(k & 1)
            res = res * trans;
    return res;
}
```

7.4 Rabin Miller Primality check

```

// n < 4,759,123,141                3 :  2, 7, 61
// n < 1,122,004,669,633            4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383            6 :  pirmes <= 13
// n < 3,825,123,056,546,413,051    9 :  primes <= 23

int testPrimes[] = {2,3,5,7,11,13,17,19,23};

struct MillerRabin{
  ///change K according to n
  const int K = 9;
  ll mult(ll s, ll m, ll mod){
    if(!m) return 0;
    ll ret = mult(s, m/2, mod);
    ret = (ret + ret) % mod;
    if(m & 1) ret = (ret + s) % mod;
    return ret;
  }

  ll power(ll x, ll p, ll mod){
    ll s = 1, m = x;
    while(p){
      if(p&1) s = mult(s, m, mod);
      p >>= 1;
      m = mult(m, m, mod);
    }
    return s;
  }

  bool witness(ll a,ll n,ll u,int t){
    ll x = power(a, u, n), nx;
    for(int i = 0; i < t; i++){
      nx = mult(x, x, n);
      if(nx == 1 and x != 1 and x != n-1) return 1;
      x = nx;
    }
    return x != 1;
  }

  bool isPrime(ll n){   // return 1 if prime, 0 otherwise
    if(n < 2) return 0;
    if(!(n&1)) return n == 2;
    for(int i = 0; i < K; i++)if(n == testPrimes[i])return 1;
    ll u = n-1; int t = 0;

    while(u&1) u >>= 1, t++;    // n-1 = u*2^t

    for(int i = 0; i < K; i++) if(witness(testPrimes[i], n, u, t)) return 0;
    return 1;
  }
}tester;
```

8 Strings

8.1 Aho-Corasick Mostafa

```
struct AC_FSM {
#define ALPHABET_SIZE 26

    struct Node {
        int child[ALPHABET_SIZE], failure = 0, match_parent = -1;
        vector<int> match;

        Node() {
            for (int i = 0; i < ALPHABET_SIZE; ++i)child[i] = -1;
        }
    };

    vector<Node> a;

    AC_FSM() {
        a.push_back(Node());
    }

    void construct_automaton(vector<string> &words) {
        for (int w = 0, n = 0; w < words.size(); ++w, n = 0) {
            for (int i = 0; i < words[w].size(); ++i) {
                if (a[n].child[words[w][i] - ’a’] == -1) {
                    a[n].child[words[w][i] - ’a’] = a.size();
                    a.push_back(Node());
                }
                n = a[n].child[words[w][i] - ’a’];
            }
            a[n].match.push_back(w);
        }
        queue<int> q;
        for (int k = 0; k < ALPHABET_SIZE; ++k) {
            if (a[0].child[k] == -1) a[0].child[k] = 0;
            else if (a[0].child[k] > 0) {
                a[a[0].child[k]].failure = 0;
                q.push(a[0].child[k]);
            }
        }
        while (!q.empty()) {
            int r = q.front();
            q.pop();
            for (int k = 0, arck; k < ALPHABET_SIZE; ++k) {
                if ((arck = a[r].child[k]) != -1) {
                    q.push(arck);
                    int v = a[r].failure;
                    while (a[v].child[k] == -1) v = a[v].failure;
                    a[arck].failure = a[v].child[k];
                    a[arck].match_parent = a[v].child[k];
                    while (a[arck].match_parent != -1 &&
                           a[a[arck].match_parent].match.empty())
                        a[arck].match_parent =
                                a[a[arck].match_parent].match_parent;
                }
            }
        }
    }

    void aho_corasick(string &sentence, vector<string> &words,
                      vector<vector<int> > &matches) {
        matches.assign(words.size(), vector<int>());
        int state = 0, ss = 0;
        for (int i = 0; i < sentence.length(); ++i, ss = state) {
            while (a[ss].child[sentence[i] - ’a’] == -1)
                ss = a[ss].failure;
            state = a[state].child[sentence[i] - ’a’] = a[ss].child[sentence[i]
                - ’a’];
            for (ss = state; ss != -1; ss = a[ss].match_parent)
                for (int w: a[ss].match)
                    matches[w].push_back(i + 1 - words[w].length());
        }
    }
};
```

8.2 Aho-Corasick Anany

```
int trie[N][A];
int go[N][A]; ///holds the node that you will go to after failure and stuff
int ptr;
ll ans[N]; ///this node is a string terminator;
int fail[N];    ///the failure function for each
void BFS() {
    queue<int> q;
    f(i,0,A) {
        if(trie[0][i]) {
            q.push(trie[0][i]);
            fail[trie[0][i]] = 0;
        }
        go[0][i] = trie[0][i];
    }

    while(q.size()) {
        auto node = q.front();
        q.pop();
        ans[node] += ans[fail[node]];   ///propagate fail[i] to ans[i]
        for(int i = 0; i < A; i++) {
            if(trie[node][i]) { ///calculate failure for you child
                int to = trie[node][i];
                int cur = fail[node]; ///int g = pi[i-1]
                while(cur && !trie[cur][i]) ///while(g && s[g] != s[i])
                    cur = fail[cur];    ///g = pi[g-1]
                if(trie[cur][i])cur = trie[cur][i]; ///g += s[i] == s[g]
                fail[to] = cur; ///pi[i] = g
                q.push(to);
                go[node][i] = trie[node][i];
            }   else {
                go[node][i] = go[fail[node]][i];
            }
        }
    }
}
void ins(string s, ll val) {
    int cur = 0;
    string sx = "";
    for(char c : s) {
        sx.push_back(c);
        if(!trie[cur][c - ’a’]) {
            trie[cur][c - ’a’] = ++ptr;
        }
        cur = trie[cur][c - ’a’];
    }
    ans[cur] += val;
}
```

8.3 KMP Anany

```
vector<int> fail(string s) {
    int n = s.size();
    vector<int> pi(n);
    for(int i = 1; i < n; i++) {
        int g = pi[i-1];
        while(g && s[i] != s[g])
            g = pi[g-1];
        g += s[i] == s[g];
        pi[i] = g;
    }
    return pi;
}
vector<int> KMP(string s, string t) {
    vector<int> pi = fail(t);
```

```cpp
    vector<int> ret;
    for(int i = 0, g = 0; i < s.size(); i++) {
        while (g && s[i] != t[g])
            g = pi[g-1];
        g += s[i] == t[g];
        if(g == t.size()) { ///occurrence found
            ret.push_back(i-t.size()+1);
            g = pi[g-1];
        }
    }
    return ret;
}
```

8.4 Manacher Kactl

```cpp
// If the size of palindrome centered at i is x, then d1[i] stores (x+1)/2.

vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
        k++;
    }
    d1[i] = k--;
    if (i + k > r) {
        l = i - k;
        r = i + k;
    }
}


// If the size of palindrome centered at i is x, then d2[i] stores x/2

vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
        k++;
    }
    d2[i] = k--;
    if (i + k > r) {
        l = i - k - 1;
        r = i + k ;
    }
}
```

8.5 Suffix Array Kactl

```cpp
struct SuffixArray {
    using vi = vector<int>;
    #define rep(i,a,b)  for(int i = a; i < b; i++)
    /*
        Note this code is considers also the empty suffix
        so hear sa[0] = n and sa[1] is the smallest non empty suffix
        and sa[n] is the largest non empty suffix
        also LCP[i] = LCP(sa[i-1], sa[i]), meanining LCP[0] = LCP[1] = 0
        if you want to get LCP(i..j) you need to build a mapping between
        sa[i] and i, and build a min sparse table to calculate the minimum
        note that this minimum should consider sa[i+1...j] since you don't want
        to consider LCP(sa[i], sa[i-1])

        you should also print the suffix array and lcp at the beginning of the contest
        to clarify this stuff
    */
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                    s[i + k] == s[j + k]; k++);
    }
};
```

8.6 Suffix Automaton Anany

```cpp
///Note it's better to use addNode to clear a node before using it
///at the start of each test case use initAutomaton

int last = 0, cntState = 1;
int nxt[N * 2][26];
int len[N * 2], link[N * 2], firstPos[N * 2], cnt[N * 2];

void addNode(int i) {
    memset(nxt[i], 0, sizeof nxt[i]);
    link[i] = -1;
    cnt[i] = 0;
}

void initAutomaton() {
    cntState = 1;
    last = 0;
    addNode(last);
}

int addChar(char c) {

    c -= 'a';   ///note this offset
    int p = last;
    int cur = cntState++;
    addNode(cur);
    cnt[cur] = 1; ///extra
    len[cur] = len[last] + 1;
    firstPos[cur] = len[cur] - 1;   ///extra
    while(p != -1 && nxt[p][c] == 0) {
        nxt[p][c] = cur;
        p = link[p];
    }

    if(p == -1) {
        link[cur] = 0;
    }   else {
        int q = nxt[p][c];
        if(len[q] == len[p] + 1) {
            link[cur] = q;
        }   else {
            int clone = cntState++;
            link[clone] = link[q];
            firstPos[clone] = firstPos[q];  ///extra
            len[clone] = len[p] + 1;
            link[q] = link[cur] = clone;
            memcpy(nxt[clone], nxt[q], sizeof nxt[q]);
            cnt[clone] = 0; ///extra
            f(i,0,26)nxt[clone][i] = nxt[q][i];
            while(p != -1 && nxt[p][c] == q) {
                nxt[p][c] = clone;
                p = link[p];
            }
        }
    }
    last = cur;
    return cur;
}
```

8.7 Suffix Automaton Mostafa

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

#define FIO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 2e6 + 9, M = 5e5 + 9;

struct SA {
    struct node {
        int to[26];
        int link, len, co = 0;

        node() {
            memset(to, 0, sizeof to);
            co = 0, link = 0, len = 0;
        }
    };

    int last, sz;
    vector<node> v;

    SA() {
        v = vector<node>(1);
        last = 0, sz = 1;
    }

    void add_letter(int c) {
        int p = last;
        last = sz++;
        v.push_back({});
        v[last].len = v[p].len + 1;
        v[last].co = 1;
        for (; v[p].to[c] == 0; p = v[p].link)
            v[p].to[c] = last;
        if (v[p].to[c] == last) {
            v[last].link = 0;
            return;
        }
        int q = v[p].to[c];
        if (v[q].len == v[p].len + 1) {
            v[last].link = q;
            return;
        }
        int cl = sz++;
        v.push_back(v[q]);
        v.back().co = 0;
        v.back().len = v[p].len + 1;
        v[last].link = v[q].link = cl;

        for (; v[p].to[c] == q; p = v[p].link)
            v[p].to[c] = cl;
    }

    void build_co() {
        priority_queue<pair<int, int>> q;
        for (int i = sz - 1; i > 0; i--)
            q.push({v[i].len, i});
        while (q.size()) {
            int i = q.top().second;
            q.pop();
            v[v[i].link].co += v[i].co;
        }
    }
};

int main() {
    FIO

    return 0;
}
```

8.8 Suffix Automaton With Rollback Mostafa

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

#define FIO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 2e6 + 9, M = 5e5 + 9;

struct SA {
    struct node {
        int to[26];
        int link, len, co = 0;

        node() {
            memset(to, 0, sizeof to);
            co = 0, link = 0, len = 0;
        }
    };

    struct LogNode {
        int last, sz;
        vector<pair<pair<int, int>, int>> edges;
        pair<int, int> LinksUpdate = {0, 0};
    };

    int last, sz;
    vector<node> v;
    vector<LogNode> logs;

    SA() {
        v = vector<node>(1);
        last = 0, sz = 1;
    }

    void add_letter(int c) {
        logs.push_back({});
        logs.back().last = last;
        logs.back().sz = sz;

        int p = last;
        last = sz++;
        v.push_back({});
        v[last].len = v[p].len + 1;
        v[last].co = 1;
        for (; v[p].to[c] == 0; p = v[p].link) {
            logs.back().edges.push_back({{p, c}, 0});
            v[p].to[c] = last;
        }
        if (v[p].to[c] == last) {
            v[last].link = 0;
            return;
        }
        int q = v[p].to[c];
        if (v[q].len == v[p].len + 1) {
            v[last].link = q;
            return;
        }
        int cl = sz++;
        v.push_back(v[q]);
        v.back().co = 0;
        v.back().len = v[p].len + 1;
        logs.back().LinksUpdate = {q, v[q].link};
        v[last].link = v[q].link = cl;
        for (; v[p].to[c] == q; p = v[p].link) {
            logs.back().edges.push_back({{p, c}, q});
            v[p].to[c] = cl;
        }
    }
    void rollback() {
        assert(logs.size());
        auto log = logs.back();
        while (v.size() > log.sz)
            v.pop_back();
        for (auto edge: log.edges)
            v[edge.first.first].to[edge.first.second] = edge.second;
        if (log.LinksUpdate.first != 0)
            v[log.LinksUpdate.first].link = log.LinksUpdate.second;
        last = log.last;
        sz = log.sz;
        logs.pop_back();
    }
};

int main() {
    FIO

    return 0;
}
```

8.9 Zalgo Anany

```
int z[N], n;
void Zalgo(string s) {
    int L = 0, R = 0;
    for(int i = 1; i < n; i++) {
        if(i<=R&&z[i-L] < R - i + 1)z[i] = z[i-L];
        else {
            L = i;
            R = max(R,i);
            while(R < n && s[R-L] == s[R])R++;
            z[i] = R-L; --R;
        }
    }
}
```

9 Trees

9.1 Centroid Decomposition

```
/*
    Properties:
        1. consider path(a,b) can be decomposed to path(a,lca(a,b)) and path(b,lca(a,b))
        where lca(a,b) is the lca on the centroid tree
        2. Each one of the n^2 paths is the concatenation of two paths in a set of O(n lg(n))
        paths from a node to all its ancestors in the centroid decomposition.
        3. Ancestor of a node in the original tree is either an ancestor in the CD tree or
        a descendadnt
*/
vector<int> adj[N]; ///adjacency list of original graph
int n;
int sz[N];
bool used[N];
int centPar[N]; ///parent in centroid
void init(int node, int par) {  ///initialize size
    sz[node] = 1;
    for(auto p : adj[node])
        if(p != par && !used[p]) {
            init(p, node);
            sz[node] += sz[p];
        }
}
int centroid(int node, int par, int limit) {    ///get centroid
    for(int p : adj[node])
        if(!used[p] && p != par && sz[p] * 2 > limit)
        return centroid(p, node, limit);
    return node;
}
int decompose(int node) {
    init(node,node);    ///calculate size
    int c = centroid(node, node, sz[node]); ///get centroid
    used[c] = true;
    for(auto p : adj[c])if(!used[p.F]) {    ///initialize parent for others and decompose
        centPar[decompose(p.F)] = c;
    }
    return c;
}
void update(int node, int distance, int col) {
    int centroid = node;
    while(centroid){
        ///solve
        centroid = centPar[centroid];
    }
}
int query(int node) {

    int ans = 0;

    int centroid = node;
    while(centroid) {
        ///solve
        centroid = centPar[centroid];
    }

    return ans;
}
```

9.2 Dsu On Trees

```
const int N = 1e5 + 9;
vector<int> adj[N];
int bigChild[N], sz[N];
void dfs(int node, int par) {
    for(auto v : adj[node]) if(v != par){
        dfs(v, node);
        sz[node] += sz[v];
        if(!bigChild[node] || sz[v] > sz[bigChild[node]]) {
            bigChild[node] = v;
        }
    }
}
void add(int node, int par, int bigChild, int delta) {

    ///modify node to data structure

    for(auto v : adj[node])
    if(v != par && v != bigChild)
        add(v, node, bigChild, delta);

}
void dfs2(int node, int par, bool keep) {
    for(auto v : adj[node])if(v != par && v != bigChild[node]) {
        dfs2(v, node, 0);
    }
    if(bigChild[node]) {
        dfs2(bigChild[node], node, true);
    }
    add(node, par, bigChild[node], 1);
    ///process queries
    if(!keep) {
        add(node, par, -1, -1);
    }
}
```

9.3 Heavy Light Decomposition (Along with Euler Tour)

```
/*
    Notes:
        1. 0-based
        2. solve function iterates over segments and handles them seperatly
        if you’re gonna use it make sure you know what you’re doing
        3. to update/query segment in[node], out[node]
        4. to update/query chain in[nxt[node]], in[node]
        nxt[node]: is the head of the chain so to go to the next chain node = par[nxt[node]]
*/
int sz[mxN], nxt[mxN];
int in[N], out[N], rin[N];
vector<int> g[mxN];
int par[mxN];

void dfs_sz(int v = 0, int p = -1) {
    sz[v] = 1;
    par[v] = p;
    for (auto &u : g[v]) {
        if (u == p) {
            swap(u, g[v].back());
        }
        if(u == p) continue;
        dfs_sz(u,v);
        sz[v] += sz[u];
        if (sz[u] > sz[g[v][0]])
            swap(u, g[v][0]);
    }
    if(v != 0)
        g[v].pop_back();
}

void dfs_hld(int v = 0) {
    in[v] = t++;
    rin[in[v]] = v;
    for (auto u : g[v]) {
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u);
    }
    out[v] = t;
}

int n;
bool isChild(int p, int u){
  return in[p] <= in[u] && out[u] <= out[p];
}
int solve(int u,int v) {
    vector<pair<int,int> > segu;
    vector<pair<int,int> > segv;
    if(isChild(u,v)){
      while(nxt[u] != nxt[v]){
        segv.push_back(make_pair(in[nxt[v]], in[v]));
        v = par[nxt[v]];
      }
      segv.push_back({in[u], in[v]});
    } else if(isChild(v,u)){
      while(nxt[u] != nxt[v]){
      segu.push_back(make_pair(in[nxt[u]], in[u]));
      u = par[nxt[u]];
      }
      segu.push_back({in[v], in[u]});
  } else {
      while(u  != v) {
        if(nxt[u] == nxt[v]) {
          if(in[u] < in[v]) segv.push_back({in[u],in[v]}), R.push_back({u+1,v+1});
          else segu.push_back({in[v],in[u]}), L.push_back({v+1,u+1});
          u = v;
          break;
        } else if(in[u] > in[v]) {
          segu.push_back({in[nxt[u]],in[u]}), L.push_back({nxt[u]+1, u+1});
          u = par[nxt[u]];
        } else {
          segv.push_back({in[nxt[v]],in[v]}), R.push_back({nxt[v]+1, v+1});
          v = par[nxt[v]];
        }
      }
    }
    reverse(segv.begin(),segv.end());
    int res = 0,state = 0;
    for(auto p : segu) {
        qry(1,1,0,n-1,p.first,p.second,state,res);
    }
    for(auto p : segv) {
        qry(0,1,0,n-1,p.first,p.second,state,res);
    }
    return res;
}
```

9.4 LCA

```
const int N = 1e5 + 5;
const int LG = 18;

vector<int> adj[N];
int pa[N][LG], lvl[N];
int in[N], out[N], timer;
void dfs(int u, int p){
  in[u] = ++timer;
  for(int k = 1; k < LG; k++)
    pa[u][k] = pa[pa[u][k-1]][k-1];
  for(auto v : adj[u])
    if(v != p){
       lvl[v] = lvl[u] + 1;
       pa[v][0] = u;
       dfs(v, u);
     }
  out[u] = timer;
}
int LCA(int u, int v){
  if(lvl[u] > lvl[v])
    swap(u,v);
   int d = lvl[v] - lvl[u];
   for(int k = 0; k < LG; k++)
    if(d >> k & 1)
      v = pa[v][k];
   if(u == v)return u;
   for(int i = LG - 1; i >= 0; --i)
    if(pa[u][i] != pa[v][i]){
      u = pa[u][i];
      v = pa[v][i];
    }
  return pa[u][0];
}
```

9.5 Mo on Trees

```
int BL[N << 1], ID[N << 1];
int lvl[N], par[17][N];
int ans[N];
vector<ii> adj[N];
struct query{
  int id, l, r, lc;
  bool operator < (const query & rhs){
    return (BL[l] == BL[rhs.l]) ? (r < rhs.r) : (BL[l] < BL[rhs.l]);
  }
}Q[N];
int in[N], out[N], val[N], timer;
void dfs(int node, int p){
  in[node] = ++timer;  ID[timer] = node;
  for(int i = 1; i < 17; i++)par[i][node] = par[i-1][par[i-1][node]];
  for(auto child : adj[node])if(child.F != p){
    lvl[child.F] = lvl[node] + 1;
    par[0][child.F] = node;
    val[child.F] = child.S;
    dfs(child.F, node);
  }
  out[node] = ++timer;  ID[timer] = node;
}
int LCA(int u, int v){
  if(lvl[u] > lvl[v])swap(u,v);
  for(int k = 0; k < 17; k++)
    if((lvl[v] - lvl[u]) >> k & 1)
      v = par[k][v];
  if(u == v)
    return u;
  for(int i = 16; i >= 0; --i)
    if(par[i][u] != par[i][v])
      u = par[i][u], v = par[i][v];
  return par[0][u];
}
bool vis[N];
int inSet[N];
void add(int node, int & res){
  if(val[node] > N)return;
  if(!vis[node]){
    inSet[val[node]]++;
    while(inSet[res])res++;
  } else {
    inSet[val[node]]--;
    if(!inSet[val[node]] && val[node] < res)
      res = val[node];
  }
  vis[node] ^= 1;
}
//-------------------Adding Queries-----------------/
f(i,0,q){
    int u, v;
    cin >> u >> v;  if(lvl[u] > lvl[v])swap(u, v);
    int lca = LCA(u, v);
    Q[i].id = i;
    Q[i].lc = lca;
    if(lca == u)Q[i].l = in[u], Q[i].r = in[v];
    else {
      Q[i].l = out[u];
      Q[i].r = in[v];
    }
  }
//-------------------Processing Queries-----------------/
f(i,0,q){
        while (curL < Q[i].l) add(ID[curL++], res);
        while (curL > Q[i].l) add(ID[--curL], res);
        while (curR < Q[i].r) add(ID[++curR], res);
        while (curR > Q[i].r) add(ID[curR--], res);
        int u = ID[Q[i].l];
        int v = ID[Q[i].r];
        if(Q[i].lc == u)add(Q[i].lc, res);
        ans[Q[i].id] = res;
        if(Q[i].lc == u)add(Q[i].lc, res);
  }
```