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

```cpp
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

```cpp
// 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

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

2.4 Totient function

```cpp
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

```cpp
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;
```

```cpp
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

```cpp
#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

```cpp

///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

6 Graphs

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];
```

```
5
6       Matrix(int val) {
7           for(int i = 0; i < D; i++)
8               for(int j = 0; j < D; j++)
9                   a[i][j] = val;
10      }
11      void clear() {
12          memset(a, 0, sizeof a);
13      }
14      void initIdentity() {
15          clear();
16          for(int i = 0; i < D; i++)
17              a[i][i] = 1;
18      }
19      int * operator [](int r) {
20          return a[r];
21      }
22      const int * operator [](int r) const{
23          return a[r];
24      }
25
26      friend Matrix operator * (const Matrix & a, const Matrix & b) {
27          Matrix ret(0);
28          for(int k = 0; k < D; k++)
29              for(int i = 0; i < D; i++)if(a[i][k])
30                  for(int j = 0; j < D; j++)
31                      ret[i][j] = (ret[i][j] + 1ll * a[i][k] * b[k][j]) % MOD;
32          return ret;
33      }
34
35  };
36  Matrix raiseMatrix(Matrix trans, ll k) {
37      Matrix res(0);
38      res.initIdentity();
39      for(;k;k>>=1,trans = trans * trans)
40          if(k & 1)
41              res = res * trans;
42      return res;
43  }
```

7.4 Rabin Miller Primality check

```
1
2   // n < 4,759,123,141                3 :  2, 7, 61
3   // n < 1,122,004,669,633            4 :  2, 13, 23, 1662803
4   // n < 3,474,749,660,383            6 :  pirmes <= 13
5   // n < 3,825,123,056,546,413,051    9 :  primes <= 23
6
7   int testPrimes[] = {2,3,5,7,11,13,17,19,23};
8
9   struct MillerRabin{
10    ///change K according to n
11    const int K = 9;
12    ll mult(ll s, ll m, ll mod){
13      if(!m) return 0;
14      ll ret = mult(s, m/2, mod);
15      ret = (ret + ret) % mod;
16      if(m & 1) ret = (ret + s) % mod;
17      return ret;
18    }
19
20    ll power(ll x, ll p, ll mod){
21      ll s = 1, m = x;
22      while(p){
23        if(p&1) s = mult(s, m, mod);
24        p >>= 1;
25        m = mult(m, m, mod);
26      }
27      return s;
28    }
29
30    bool witness(ll a,ll n,ll u,int t){
31      ll x = power(a, u, n), nx;
32      for(int i = 0; i < t; i++){
33        nx = mult(x, x, n);
34        if(nx == 1 and x != 1 and x != n-1) return 1;
35        x = nx;
36      }
37      return x != 1;
38    }
39
40    bool isPrime(ll n){   // return 1 if prime, 0 otherwise
41      if(n < 2) return 0;
42      if(!(n&1)) return n == 2;
43      for(int i = 0; i < K; i++)if(n == testPrimes[i])return 1;
44      ll u = n-1; int t = 0;
45
46      while(u&1) u >>= 1, t++;     // n-1 = u*2^t
47
48      for(int i = 0; i < K; i++) if(witness(testPrimes[i], n, u, t)) return 0;
49      return 1;
50    }
51  }tester;
```

8 Strings

8.1 Aho-Corasick Mostafa

```
1   struct AC_FSM {
2   #define ALPHABET_SIZE 26
3
4       struct Node {
5           int child[ALPHABET_SIZE], failure = 0, match_parent = -1;
6           vector<int> match;
7
8           Node() {
9               for (int i = 0; i < ALPHABET_SIZE; ++i)child[i] = -1;
10          }
11      };
12
13      vector<Node> a;
14
15      AC_FSM() {
16          a.push_back(Node());
17      }
18
19      void construct_automaton(vector<string> &words) {
20          for (int w = 0, n = 0; w < words.size(); ++w, n = 0) {
21              for (int i = 0; i < words[w].size(); ++i) {
22                  if (a[n].child[words[w][i] - 'a'] == -1) {
23                      a[n].child[words[w][i] - 'a'] = a.size();
24                      a.push_back(Node());
25                  }
26                  n = a[n].child[words[w][i] - 'a'];
27              }
28              a[n].match.push_back(w);
29          }
30          queue<int> q;
31          for (int k = 0; k < ALPHABET_SIZE; ++k) {
32              if (a[0].child[k] == -1) a[0].child[k] = 0;
33              else if (a[0].child[k] > 0) {
34                  a[a[0].child[k]].failure = 0;
35                  q.push(a[0].child[k]);
36              }
37          }
38          while (!q.empty()) {
39              int r = q.front();
40              q.pop();
41              for (int k = 0, arck; k < ALPHABET_SIZE; ++k) {
42                  if ((arck = a[r].child[k]) != -1) {
43                      q.push(arck);
44                      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);
    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

```
// 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

```cpp
#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);
  }
```