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

Pillow, Isaac, Mostafa, Islam

Contents

2021

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

```cpp
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;
            }
        }
    }
}
```

```
17  // 1- Number of correct bracket sequence consisting of n opening and n closing
        brackets.
18  // 2- The number of rooted full binary trees with n+1 leaves (vertices are not
        numbered).
19  //    A rooted binary tree is full if every vertex has either two children or no
        children.
20  // 3- The number of ways to completely parenthesize n+1 factors.
21  // 4- The number of triangulations of a convex polygon with n+2 sides
22  //    (i.e. the number of partitions of polygon into disjoint triangles by using
        the diagonals).
23  // 5- The number of ways to connect the 2n points on a circle to form n disjoint
        chords.
24  // 6- The number of non-isomorphic full binary trees with n internal nodes (i.e.
        nodes having at least one son).
25  // 7- The number of monotonic lattice paths from point (0,0) to point (n,n) in a
        square lattice of size nxn,
26  //    which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n))
        .
27  // 8- Number of permutations of length n that can be stack sorted
28  //    (i.e. it can be shown that the rearrangement is stack sorted if and only
        if
29  //    there is no such index i<j<k, such that ak<ai<aj ).
30  // 9- The number of non-crossing partitions of a set of n elements.
31  // 10- The number of ways to cover the ladder 1..n using n rectangles
32  // (The ladder consists of n columns, where ith column has a height i).
```

2 Algebra

2.1 Primitive Roots

```
1   int powmod (int a, int b, int p) {
2       int res = 1;
3       while (b)
4           if (b & 1)
5               res = int (res * 1ll * a % p),  --b;
6           else
7               a = int (a * 1ll * a % p),  b >>= 1;
8       return res;
9   }
10
11  int generator (int p) {
12      vector<int> fact;
13      int phi = p - 1,  n = phi;
14      for (int i = 2; i * i <= n; ++i)
15          if (n % i == 0) {
16              fact.push_back (i);
17              while (n % i == 0)
18                  n /= i;
19          }
20      if (n > 1)
21          fact.push_back (n);
22
23      for (int res = 2; res <= p; ++res) {
24          bool ok = true;
25          for (size_t i = 0; i < fact.size() && ok; ++i)
26              ok &= powmod (res, phi / fact[i], p) != 1;
27          if (ok)  return res;
28      }
29      return -1;
30  }
```

2.2 Discrete Logarithm

```
1   // Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
2   int solve(int a, int b, int m) {
3       a %= m, b %= m;
4       int n = sqrt(m) + 1;
5       int an = 1;
6       for (int i = 0; i < n; ++i)
7           an = (an * 1ll * a) % m;
8
9       unordered_map<int, int> vals;
10      for (int q = 0, cur = b; q <= n; ++q) {
11          vals[cur] = q;
12          cur = (cur * 1ll * a) % m;
13      }
14
15      for (int p = 1, cur = 1; p <= n; ++p) {
16          cur = (cur * 1ll * an) % m;
17          if (vals.count(cur)) {
18              int ans = n * p - vals[cur];
19              return ans;
20          }
21      }
22      return -1;
23  }
24
25  //When a and m are not coprime
26  // Returns minimum x for which a ^ x % m = b % m.
27  int solve(int a, int b, int m) {
28      a %= m, b %= m;
29      int k = 1, add = 0, g;
30      while ((g = gcd(a, m)) > 1) {
31          if (b == k)
32              return add;
33          if (b % g)
34              return -1;
35          b /= g, m /= g, ++add;
36          k = (k * 1ll * a / g) % m;
37      }
38
39      int n = sqrt(m) + 1;
40      int an = 1;
41      for (int i = 0; i < n; ++i)
42          an = (an * 1ll * a) % m;
43
44      unordered_map<int, int> vals;
45      for (int q = 0, cur = b; q <= n; ++q) {
46          vals[cur] = q;
47          cur = (cur * 1ll * a) % m;
48      }
49
50      for (int p = 1, cur = k; p <= n; ++p) {
51          cur = (cur * 1ll * an) % m;
52          if (vals.count(cur)) {
53              int ans = n * p - vals[cur] + add;
54              return ans;
55          }
56      }
57      return -1;
58  }
```

2.3 Iteration over submasks

```
1   int s = m;
2   while (s > 0) {
3    ... you can use s ...
4    s = (s-1) & m;
5   }
```

2.4 Totient function

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

```
4        phi[1] = 1;
5        for (int i = 2; i <= n; i++)
6            phi[i] = i;
7
8        for (int i = 2; i <= n; i++) {
9            if (phi[i] == i) {
10               for (int j = i; j <= n; j += i)
11                   phi[j] -= phi[j] / i;
12               }
13           }
14   }
```

2.5   CRT and EEGCD

```
1    ll extended(ll a, ll b, ll &x, ll &y) {
2
3        if(b == 0) {
4            x = 1;
5            y = 0;
6            return a;
7        }
8        ll x0, y0;
9        ll g = extended(b, a % b, x0, y0);
10       x = y0;
11       y = x0 - a / b * y0;
12
13       return g ;
14   }
15   ll de(ll a, ll b, ll c, ll &x, ll &y) {
16
17       ll g = extended(abs(a), abs(b), x, y);
18       if(c % g) return -1;
19
20       x *= c / g;
21       y *= c / g;
22
23       if(a < 0)x = -x;
24       if(b < 0)y = -y;
25       return g;
26   }
27   pair<ll, ll> CRT(vector<ll> r, vector<ll> m) {
28
29       ll r1 = r[0], m1 = m[0];
30
31       for(int i = 1; i < r.size(); i++) {
32
33           ll r2 = r[i], m2 = m[i];
34           ll x0, y0;
35           ll g = de(m1, -m2, r2 - r1, x0, y0);
36
37           if(g == -1) return {-1, -1} ;
38
39           ll nr = x0 * m1 + r1;
40           ll nm = m1 / g * m2;
41
42           r1 = (nr % nm + nm) % nm;
43           m1 = nm;
44       }
45       return {r1, m1};
46   }
```

2.6   FFT

```
1    #include<iostream>
2    #include <bits/stdc++.h>
3    #define ll long long
4    #define ld long double
5    #define rep(i, a, b) for(int i = a; i < (b); ++i)
6    #define all(x) begin(x), end(x)
7    #define sz(x) (int)(x).size()
```

```
8    #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
9    using namespace std;
10   typedef complex<double> C;
11   typedef vector<double> vd;
12   typedef vector<int> vi;
13   typedef pair<int, int> pii;
14   void fft(vector<C>& a) {
15       int n = sz(a), L = 31 - __builtin_clz(n);
16       static vector<complex<long double>> R(2, 1);
17       static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
18       for (static int k = 2; k < n; k *= 2) {
19           R.resize(n);
20           rt.resize(n);
21           auto x = polar(1.0L, acos(-1.0L) / k);
22           rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
23       }
24       vi rev(n);
25       rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
26       rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
27       for (int k = 1; k < n; k *= 2)
28           for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
29               C z = rt[j + k] * a[i + j + k]; //
30               a[i + j + k] = a[i + j] - z;
31               a[i + j] += z;
32           }
33   }
34   vd conv(const vd& a, const vd& b) {
35       if (a.empty() || b.empty()) return {};
36       vd res(sz(a) + sz(b) - 1);
37       int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
38       vector<C> in(n), out(n);
39       copy(all(a), begin(in));
40       rep(i, 0, sz(b)) in[i].imag(b[i]);
41       fft(in);
42       for (C& x : in) x *= x;
43       rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
44       fft(out);
45       rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
46       return res;
47   }
48
49   int main() {
50       IO
51       //Applications
52       //1-All possible sums
53
54       //2-All possible scalar products
55       // We are given two arrays a[] and b[] of length n.
56       //We have to compute the products of a with every cyclic shift of b.
57       //We generate two new arrays of size 2n: We reverse a and append n zeros to
                it.
58       //And we just append b to itself. When we multiply these two arrays as
                polynomials,
59       //and look at the coefficients c[n-1], c[n], ..., c[2n-2] of the product c,
                we get:
60       //c[k]=sum i+j=k  a[i]b[j]
61
62       //3-Two stripes
63       //We are given two Boolean stripes (cyclic arrays of values 0 and 1) a and b
                .
64       //We want to find all ways to attach the first stripe to the second one,
65       //such that at no position we have a 1 of the first stripe next to a 1 of
                the second stripe.
66   }
```

2.7   Fibonacci

```
1
2
3    // F(n-1) * F(n+1) - F(n)^2 = (-1)^n
4
5    // F(n+k) = F(k) * F(n+1) + F(k-1) * F(n)
```

```
6
7    // F(2*n) = F(n) * (F(n+1) + F(n-1))
8
9    //GCD ( F(m) , F(n) ) = F(GCD(n,m))
```

2.8   Gauss Determinant

```
1    const double EPS = 1E-9;
2    int n;
3    vector < vector<double> > a (n, vector<double> (n));
4
5    double det = 1;
6    for (int i=0; i<n; ++i) {
7        int k = i;
8        for (int j=i+1; j<n; ++j)
9            if (abs (a[j][i]) > abs (a[k][i]))
10               k = j;
11       if (abs (a[k][i]) < EPS) {
12           det = 0;
13           break;
14       }
15       swap (a[i], a[k]);
16       if (i != k)
17           det = -det;
18       det *= a[i][i];
19       for (int j=i+1; j<n; ++j)
20           a[i][j] /= a[i][i];
21       for (int j=0; j<n; ++j)
22           if (j != i && abs (a[j][i]) > EPS)
23               for (int k=i+1; k<n; ++k)
24                   a[j][k] -= a[i][k] * a[j][i];
25   }
26
27   cout << det;
```

2.9   GAUSS SLAE

```
1    const double EPS = 1e-9;
2    const int INF = 2; // it doesn't actually have to be infinity or a big number
3
4    int gauss (vector < vector<double> > a, vector<double> & ans) {
5        int n = (int) a.size();
6        int m = (int) a[0].size() - 1;
7
8        vector<int> where (m, -1);
9        for (int col = 0, row = 0; col < m && row < n; ++col) {
10           int sel = row;
11           for (int i = row; i < n; ++i)
12               if (abs (a[i][col]) > abs (a[sel][col]))
13                   sel = i;
14           if (abs (a[sel][col]) < EPS)
15               continue;
16           for (int i = col; i <= m; ++i)
17               swap (a[sel][i], a[row][i]);
18           where[col] = row;
19
20           for (int i = 0; i < n; ++i)
21               if (i != row) {
22                   double c = a[i][col] / a[row][col];
23                   for (int j = col; j <= m; ++j)
24                       a[i][j] -= a[row][j] * c;
25               }
26           ++row;
27       }
28
29       ans.assign (m, 0);
30       for (int i = 0; i < m; ++i)
31           if (where[i] != -1)
32               ans[i] = a[where[i]][m] / a[where[i]][i];
33       for (int i = 0; i < n; ++i) {
34           double sum = 0;
35           for (int j = 0; j < m; ++j)
36               sum += ans[j] * a[i][j];
37           if (abs (sum - a[i][m]) > EPS)
38               return 0;
39       }
40
41       for (int i = 0; i < m; ++i)
42           if (where[i] == -1)
43               return INF;
44       return 1;
45   }
```

2.10   Matrix Inverse

```
1    // Sometimes, the questions are complicated - and the answers are simple. //
2    #pragma GCC optimize ("O3")
3    #pragma GCC optimize ("unroll-loops")
4    #include <bits/stdc++.h>
5    #define ll long long
6    #define ld  long double
7    #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
8    using namespace std;
9    vector < vector<double> > gauss (vector < vector<double> > a) {
10
11       int n = (int) a.size();
12       vector<vector<double> > ans(n, vector<double>(n, 0));
13
14       for(int i = 0; i < n; i++)
15           ans[i][i] = 1;
16       for(int i = 0; i < n; i++) {
17           for(int j = i + 1; j < n; j++)
18               if(a[j][i] > a[i][i]) {
19                   swap(a[j], a[i]);
20                   swap(ans[j], ans[i]);
21               }
22           double val = a[i][i];
23           for(int j = 0; j < n; j++) {
24               a[i][j] /= val;
25               ans[i][j] /= val;
26           }
27           for(int j = 0; j < n; j++) {
28               if(j == i)continue;
29               val = a[j][i];
30               for(int k = 0; k < n; k++) {
31                   a[j][k] -= val * a[i][k];
32                   ans[j][k] -= val * ans[i][k];
33               }
34           }
35       }
36       return ans;
37   }
38   int main() {
39
40       IO
41       vector<vector<double> > v(3, vector<double> (3) );
42       for(int i = 0; i < 3; i++)
43           for(int j = 0; j < 3; j++)
44               cin >> v[i][j];
45
46       for(auto i : gauss(v)) {
47           for(auto j : i)
48               cout << j << " ";
49           cout << "\n";
50       }
51   }
```

2.11   NTT

```cpp
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

```cpp
///(Note faster than the other NTT)
```

```cpp
///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

```cpp
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

```
1  /*
2      note this isn't the best cache-wise version
3      query O(1), Build O(NMlgNlgM)
4      be careful when using it and note the he build a dimension above another
5      i.e he builds a sparse table for each row
6      the build sparse table over each row's sparse table
7  */
8  const int N = 505, LG = 10;
9
10 int st[N][N][LG][LG];
11 int a[N][N], lg2[N];
12
13 int yo(int x1, int y1, int x2, int y2) {
14     x2++;
15     y2++;
16     int a = lg2[x2 - x1], b = lg2[y2 - y1];
17     return max(
18             max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
19             max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b
                    ])
20         );
21 }
22
23 void build(int n, int m) { // 0 indexed
24     for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
25     for (int i = 0; i < n; i++) {
26         for (int j = 0; j < m; j++) {
27             st[i][j][0][0] = a[i][j];
28         }
29     }
30     for (int a = 0; a < LG; a++) {
31         for (int b = 0; b < LG; b++) {
32             if (a + b == 0) continue;
33             for (int i = 0; i + (1 << a) <= n; i++) {
34                 for (int j = 0; j + (1 << b) <= m; j++) {
35                     if (!a) {
36                         st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a
                                ][b - 1]);
37                     } else {
38                         st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a
                                - 1][b]);
39                     }
40                 }
41             }
42         }
43     }
44 }
```

3.3   hillbert Order

```
1  ///Faster Sorting MO
2
3  const int infinity = (int)1e9 + 42;
4  const int64_t llInfinity = (int64_t)1e18 + 256;
5  const int module = (int)1e9 + 7;
6  const long double eps = 1e-8;
7
8  inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
9      if (pow == 0) {
10         return 0;
11     }
12     int hpow = 1 << (pow-1);
13     int seg = (x < hpow) ? (
14         (y < hpow) ? 0 : 3
15     ) : (
16         (y < hpow) ? 1 : 2
17     );
18     seg = (seg + rotate) & 3;
```

```
19     const int rotateDelta[4] = {3, 0, 0, 1};
20     int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
21     int nrot = (rotate + rotateDelta[seg]) & 3;
22     int64_t subSquareSize = int64_t(1) << (2*pow - 2);
23     int64_t ans = seg * subSquareSize;
24     int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
25     ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
26     return ans;
27 }
28
29 struct Query {
30     int l, r, idx;
31     int64_t ord;
32
33     inline void calcOrder() {
34         ord = gilbertOrder(l, r, 21, 0);
35     }
36 };
37
38 inline bool operator<(const Query &a, const Query &b) {
39     return a.ord < b.ord;
40 }
41
42 signed main() {
43     #ifndef USE_FILE_IO
44         ios_base::sync_with_stdio(false);
45     #endif
46
47     mt19937 rnd(42);
48
49     int n, m, k; cin >> n >> m; k = rnd() % 1048576;
50     vector<int> p(n+1);
51     for (int i = 0; i < n; i++) {
52         int val = rnd() % 1048576;
53         p[i+1] = p[i] ^ val;
54     }
55
56     vector<Query> qry(m);
57     for (int i = 0; i < m; i++) {
58         int l = rnd() % n + 1, r = rnd() % n + 1;
59         if (l > r) {
60             swap(l, r);
61         }
62         qry[i].l = l; qry[i].r = r;
63         qry[i].idx = i;
64         qry[i].calcOrder();
65     }
66
67     int64_t ans = 0;
68     vector<int64_t> res(m);
69     vector<int64_t> cnt((int)2e6, 0);
70     sort(qry.begin(), qry.end());
71     int l = 0, r = 1;
72     ans = (p[1] == k);
73     cnt[p[0]]++; cnt[p[1]]++;
74
75     for (Query q: qry) {
76         q.l--;
77         while (l > q.l) {
78             l--;
79             ans += cnt[p[l] ^ k];
80             cnt[p[l]]++;
81         }
82         while (r < q.r) {
83             r++;
84             ans += cnt[p[r] ^ k];
85             cnt[p[r]]++;
86         }
87         while (l < q.l) {
88             cnt[p[l]]--;
89             ans -= cnt[p[l] ^ k];
90             l++;
91         }
92         while (r > q.r) {
93             cnt[p[r]]--;
```

```
 94              ans -= cnt[p[r] ^ k];
 95              r--;
 96          }
 97          res[q.idx] = ans;
 98      }
 99
100      uint64_t rhsh = 0;
101      for (int i = 0; i < m; i++) {
102          rhsh *= (uint64_t)1e9 + 7;
103          rhsh += (uint64_t)res[i];
104      }
105      cout << rhsh << "\n";
106
107      return 0;
108  }
```

3.4 Merge Sort Bit with updates

```
 1  //O(log ^ 2 N) updates and queries
 2
 3
 4  #include <ext/pb_ds/tree_policy.hpp>
 5  #include <ext/pb_ds/assoc_container.hpp>
 6  #include <ext/rope>
 7
 8  using namespace std;
 9  using namespace __gnu_pbds;
10  using namespace __gnu_cxx;
11
12  template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag,
         tree_order_statistics_node_update>;
13
14
15  Tree<int> t[N];
16
17  void add(int idx, int v){
18      for(int x = ++idx; x < N; x += x & -x){
19          t[x].insert(v);
20      }
21  }
22  void erase(int idx, int v){
23      for(int x = ++idx; x < N; x += x & -x)
24          t[x].erase(v);
25  }
26  int get(int idx,  int limit){
27      int ret = 0;
28      for(int x = ++idx; x; x -= x & -x)
29          ret += (t[x].order_of_key(limit+1));
30      return ret;
31  }
```

3.5 Mo's

```
 1  #include <bits/stdc++.h>
 2
 3  int n, qq, arr[N], sz = 1000; // sz  is the size of the bucket
 4  int co[N],ans = 0, ansq[N];
 5  int cul = 1, cur = 1;
 6
 7  void add(int x) {
 8      co[arr[x]]++;
 9      if (co[arr[x]] == 1)
10          ans++;
11      else if (co[arr[x]] == 2)
12          ans--;
13  }
14
15  void remove(int x) {
16      co[arr[x]]--;
```

```
17      if (co[arr[x]] == 1)
18          ans++;
19      else if (co[arr[x]] == 0)
20          ans--;
21  }
22
23  void solve(int l, int r,int ind) {
24      r+=1;
25      while (cul < l) remove(cul++);
26      while (cul > l) add(--cul);
27      while (cur < r) add(cur++);
28      while (cur > r) remove(--cur);
29      ansq[ind] = ans;
30  }
31
32
33  int main() {
34      FIO
35      cin >> qq;
36      //                      {l/sz,r},     { l , ind}
37      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
             ;
38      for (int i = 0; i < qq; i++) {
39          int l, r;
40          cin >> l >> r;
41          q.push({{l / sz, r},{l,i}});
42      }
43      while (q.size()) {
44          int ind=q.top().second.second,l=q.top().second.first,r=q.top().first.
             second;
45          solve(l, r,ind);
46          q.pop();
47      }
48      for (int i = 0; i < qq; i++)
49          cout << ansq[i] << endl;
50
51
52      return 0;
53  }
```

3.6 Mo With Updates

```
 1
 2  ///O(N^5/3) note that the block size is not a standard size
 3
 4  #pragma GCC optimize ("O3")
 5  #pragma GCC target ("sse4")
 6
 7  #include <bits/stdc++.h>
 8
 9  using namespace std;
10
11  using ll = long long;
12
13  const int N = 1e5 +5;
14  const int M = 2 * N;
15  const int blk = 2155;
16  const int mod = 1e9 + 7;
17  struct Query{
18    int l, r, t, idx;
19    Query(int a = 0,int b = 0,int c = 0,int d = 0){l=a,r=b,t=c,idx = d;}
20    bool operator < (Query o){
21      if(r / blk == o.r / blk && l / blk == o.l / blk)return t < o.t;
22      if(r / blk == o.r / blk)return l < o.l;
23      return r < o.r;
24    }
25  } Q[N];
26
27  int a[N], b[N];
28  int cnt1[M], cnt2[N];
29  int L = 0, R = -1, K = -1;
```

```
30    void add(int x){ ///add item to range
31    //   cout << x << '\n';
32      cnt2[cnt1[x]]--;
33      cnt1[x]++;
34      cnt2[cnt1[x]]++;
35    }
36    void del(int x){ ///delete item from range
37      cnt2[cnt1[x]]--;
38      cnt1[x]--;
39      cnt2[cnt1[x]]++;
40    }
41    map<int,int>id;
42    int cnt;
43    int ans[N];
44    int p[N], nxt[N];
45    int prv[N];
46    void upd(int idx){ ///update item value
47      if(p[idx] >= L && p[idx] <= R)
48        del(a[p[idx]]), add(nxt[idx]);
49      a[p[idx]] = nxt[idx];
50    }
51    void err(int idx){
52      if(p[idx] >= L && p[idx] <= R)
53        del(a[p[idx]]), add(prv[idx]);
54      a[p[idx]] = prv[idx];
55    }
56    int main(){
57
58      int n, q, l, r, tp;
59
60      scanf("%d%d", &n, &q);
61
62      for(int i = 0; i < n; i++){
63        scanf("%d", a + i);
64        if(id.count(a[i]) == 0)
65          id[a[i]] = cnt++;
66        a[i] = id[a[i]];
67        b[i] = a[i];
68      }
69      int qIdx = 0;
70      int ord = 0;
71      while(q--){
72
73        scanf("%d", &tp);
74        if(tp == 1){
75          /// ADD Query
76          scanf("%d%d", &l, &r);  --l, --r;
77          Q[qIdx] = Query(l,r,ord-1,qIdx); qIdx++;
78        } else{
79          /// ADD Update
80          scanf("%d%d",p + ord, nxt + ord); --p[ord];
81          if(id.count(nxt[ord]) == 0)
82            id[nxt[ord]] = cnt++;
83          nxt[ord] = id[nxt[ord]];
84          prv[ord] = b[p[ord]];
85          b[p[ord]] = nxt[ord];
86          ++ord;
87        }
88
89      }
90      sort(Q,Q+qIdx);
91      for(int i = 0; i < qIdx; i++){
92        while(L < Q[i].l)del(a[L++]);
93        while(L > Q[i].l)add(a[--L]);
94        while(R < Q[i].r)add(a[++R]);
95        while(R > Q[i].r)del(a[R--]);
96        while(K < Q[i].t)upd(++K);
97        while(K > Q[i].t)err(K--);
98        ///Solve Query I
99      }
100     for(int i = 0; i < qIdx; i++)
101       printf("%d\n", ans[i]);
102
103
104     return 0;
```

```
105   }
```

3.7 Ordered Set

```
1    #include <ext/pb_ds/assoc_container.hpp>
2    #include <ext/pb_ds/tree_policy.hpp>
3    using namespace __gnu_pbds;
4
5    #define ordered_set tree<int, null_type,less<int>, rb_tree_tag,
         tree_order_statistics_node_update>
6
7    // order_of_key(k): returns the number of elements in the set strictly less than
         k
8    // find_by_order(k): returns an iterator to the k-th element (zero-based) in the
         set
```

3.8 Persistent Seg Tree

```
1
2    int val[ N * 60 ], L[ N * 60 ], R[ N * 60 ], ptr, tree[N]; /// N * lgN
3    int upd(int root, int s, int e, int idx) {
4        int ret = ++ptr;
5        val[ret] = L[ret] = R[ret] = 0;
6        if (s == e) {
7            val[ret] = val[root] + 1;
8            return ret;
9        }
10       int md = (s + e) >> 1;
11       if (idx <= md) {
12           L[ret] = upd(L[root], s, md, idx), R[ret] = R[root];
13       } else {
14           R[ret] = upd(R[root], md + 1, e, idx), L[ret] = L[root];
15       }
16       val[ret] = max(val[L[ret]], val[R[ret]]);
17       return ret;
18   }
19   int qry(int node, int s, int e, int l, int r){
20     if(r < s || e < l || !node)return 0; //Punishment Value
21     if(l <= s && e <= r){
22       return val[node];
23     }
24     int md = (s+e)>>1;
25     return max(qry(L[node], s, md, l, r), qry(R[node],md+1,e,l,r));
26   }
27   int merge(int x, int y, int s, int e) {
28       if(!x||!y)return x | y;
29       if(s == e) {
30           val[x] += val[y];
31           return x;
32       }
33       int md = (s + e) >> 1;
34       L[x] = merge(L[x], L[y], s, md);
35       R[x] = merge(R[x], R[y], md+1,e);
36       val[x] = val[L[x]] + val[R[x]];
37       return x;
38   }
```

3.9 Sqrt Decomposition

```
1    // Source: https://cp-algorithms.com/data_structures/sqrt_decomposition.html
2
3    // input data
4    int n;
5    vector<int> a (n);
6
7    // preprocessing
```

```cpp
 8    int len = (int) sqrt (n + .0) + 1; // size of the block and the number of blocks
 9    vector<int> b (len);
10    for (int i=0; i<n; ++i)
11        b[i / len] += a[i];
12
13    // answering the queries
14    for (;;) {
15        int l, r;
16      // read input data for the next query
17        int sum = 0;
18        for (int i=l; i<=r; )
19            if (i % len == 0 && i + len - 1 <= r) {
20                // if the whole block starting at i belongs to [l, r]
21                sum += b[i / len];
22                i += len;
23            }
24            else {
25                sum += a[i];
26                ++i;
27            }
28    }
29
30    // If you're getting TLE and can't optimize more, you could reduce the number of
31    //       slow division operations using the following code:
32    int sum = 0;
33    int c_l = l / len,   c_r = r / len;
34    if (c_l == c_r)
35        for (int i=l; i<=r; ++i)
36            sum += a[i];
37    else {
38        for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
39            sum += a[i];
40        for (int i=c_l+1; i<=c_r-1; ++i)
41            sum += b[i];
42        for (int i=c_r*len; i<=r; ++i)
43            sum += a[i];
44    }
```

3.10   Treap

```cpp
 1    typedef struct item * pitem;
 2    struct item {
 3        int prior, value, cnt;
 4        bool rev;
 5        pitem l, r;
 6        item(int x, int y, int z){
 7            value = x;
 8            prior = y;
 9            cnt = z;
10            rev = 0;
11            l = r = NULL;
12        }
13    };
14
15    int cnt (pitem it) {
16        return it ? it->cnt : 0;
17    }
18
19    void upd_cnt (pitem it) {
20        if (it)
21            it->cnt = cnt(it->l) + cnt(it->r) + 1;
22    }
23
24    void push (pitem it) {
25        if (it && it->rev) {
26            it->rev = false;
27            swap (it->l, it->r);
28            if (it->l)  it->l->rev ^= true;
29            if (it->r)  it->r->rev ^= true;
30        }
31    }
```

```cpp
33    void merge (pitem & t, pitem l, pitem r) {
34        push (l);
35        push (r);
36        if (!l || !r)
37            t = l ? l : r;
38        else if (l->prior > r->prior)
39            merge (l->r, l->r, r),   t = l;
40        else
41            merge (r->l, l, r->l),   t = r;
42        upd_cnt (t);
43    }
44
45    void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
46        if (!t)
47            return void( l = r = 0 );
48        push (t);
49        int cur_key = add + cnt(t->l);
50        if (key <= cur_key)
51            split (t->l, l, t->l, key, add),   r = t;
52        else
53            split (t->r, t->r, r, key, add + 1 + cnt(t->l)),   l = t;
54        upd_cnt (t);
55    }
56
57    void reverse (pitem t, int l, int r) {
58        pitem t1, t2, t3;
59        split (t, t1, t2, l);
60        split (t2, t2, t3, r-l+1);
61        t2->rev ^= true;
62        merge (t, t1, t2);
63        merge (t, t, t3);
64    }
65
66    void output (pitem t) {
67        if (!t)  return;
68        push (t);
69        output (t->l);
70        printf ("%c", char(t->value));
71        output (t->r);
72    }
73
74    pitem gettreap(string s){
75            pitem ret=NULL;
76        int i;
77        for(i=0;i<s.size();i++)merge(ret,ret,new item(s[i],(rand()<<15)+rand(),
                1));
78        return ret;
79    }
```

3.11   Wavelet Tree

```cpp
 1    // remember your array and values must be 1-based
 2    struct wavelet_tree {
 3        int lo, hi;
 4        wavelet_tree *l, *r;
 5        vector<int> b;
 6
 7        //nos are in range [x,y]
 8        //array indices are [from, to)
 9        wavelet_tree(int *from, int *to, int x, int y) {
10            lo = x, hi = y;
11            if (lo == hi or from >= to)
12                return;
13            int mid = (lo + hi) / 2;
14            auto f = [mid](int x) {
15                return x <= mid;
16            };
17            b.reserve(to - from + 1);
18            b.pb(0);
19            for (auto it = from; it != to; it++)
20                b.pb(b.back() + f(*it));
```

```
21          //see how lambda function is used here
22          auto pivot = stable_partition(from, to, f);
23          l = new wavelet_tree(from, pivot, lo, mid);
24          r = new wavelet_tree(pivot, to, mid + 1, hi);
25      }
26
27      //kth smallest element in [l, r]
28      int kth(int l, int r, int k) {
29          if (l > r)
30              return 0;
31          if (lo == hi)
32              return lo;
33          int inLeft = b[r] - b[l - 1];
34          int lb = b[l - 1]; //amt of nos in first (l-1) nos that go in left
35          int rb = b[r]; //amt of nos in first (r) nos that go in left
36          if (k <= inLeft)
37              return this->l->kth(lb + 1, rb, k);
38          return this->r->kth(l - lb, r - rb, k - inLeft);
39      }
40
41      //count of nos in [l, r] Less than or equal to k
42      int LTE(int l, int r, int k) {
43          if (l > r or k < lo)
44              return 0;
45          if (hi <= k)
46              return r - l + 1;
47          int lb = b[l - 1], rb = b[r];
48          return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
49      }
50
51      //count of nos in [l, r] equal to k
52      int count(int l, int r, int k) {
53          if (l > r or k < lo or k > hi)
54              return 0;
55          if (lo == hi)
56              return r - l + 1;
57          int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
58          if (k <= mid)
59              return this->l->count(lb + 1, rb, k);
60          return this->r->count(l - lb, r - rb, k);
61      }
62  };
```

4 DP

4.1 Dynamic Convex Hull Trick

```
1   #include<iostream>
2   #include <bits/stdc++.h>
3   #define ll long long
4   #define ld  long double
5   #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
6   using namespace std;
7   struct Line
8   {
9       ll m, b;
10      mutable function<const Line*()> succ;
11      bool operator<(const Line& other) const
12      {
13          return m < other.m;
14      }
15      bool operator<(const ll &x) const
16      {
17          const Line* s = succ();
18          if (!s)
19              return 0;
20          return b - s->b < (s->m - m) * x;
21      }
22  };
23  // will maintain upper hull for maximum
```

```
24  struct HullDynamic : public multiset<Line, less<>>
25  {
26      bool bad(iterator y)
27      {
28          auto z = next(y);
29          if (y == begin())
30          {
31              if (z == end())
32                  return 0;
33              return y->m == z->m && y->b <= z->b;
34          }
35          auto x = prev(y);
36          if (z == end())
37              return y->m == x->m && y->b <= x->b;
38          return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)(y->b - z->b)*(y->m - x->m
                  );
39      }
40      void insert_line(ll m, ll b)
41      {
42          auto y = insert({ m, b });
43          y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
44          if (bad(y))
45          {
46              erase(y);
47              return;
48          }
49          while (next(y) != end() && bad(next(y)))
50              erase(next(y));
51          while (y != begin() && bad(prev(y)))
52              erase(prev(y));
53      }
54
55      ll query(ll x)
56      {
57
58          auto l = *lower_bound(x);
59          return l.m * x + l.b;
60      }
61  };
62  int main()
63  {
64      IO
65
66  }
```

4.2 Dynamic Connectivety with SegTree

```
1   /// MANGA
2   #pragma GCC optimize("O3")
3   #pragma GCC optimize ("unroll-loops")
4   #pragma GCC target("avx,avx2,fma")
5   using namespace std;
6
7   #include "bits/stdc++.h"
8
9   #define pb push_back
10  #define F first
11  #define S second
12  #define f(i, a, b)  for(int i = a; i < b; i++)
13  #define all(a)  a.begin(),a.end()
14  #define rall(a)  a.rbegin(),a.rend()
15  #define sz(x)  (int)(x).size()
16  //#define mp make_pair
17  #define popCnt(x) (__builtin_popcountll(x))
18  typedef long long ll;
19  typedef pair<int, int> ii;
20  using ull = unsigned long long;
21  const int N = 1e5+5, LG = 17, MOD = 1e9 + 7;
22  const long double PI = acos(-1);
23  struct PT{
24      ll x, y;
25      PT() {}
```

```
26        PT(ll a, ll b):x(a), y(b){}
27        PT operator - (const PT & o) {return PT{x-o.x,y-o.y};}
28        bool operator < (const PT & o) const {return make_pair(x,y) < make_pair(o.x,
              o.y);}
29   };
30   ll cross(PT x, PT y) {
31        return x.x * y.y - x.y * y.x;
32   }
33   PT val[300005];
34   bool in[300005];
35   ll qr[300005];
36   bool ask[300005];
37   ll ans[N];
38   vector<PT> t[300005 * 4];   ///segment tree holding points to queries
39   void update(int node, int s, int e, int l, int r, PT x) {
40        if(r < s || e < l)return;
41        if(l <= s && e <= r) {  ///add this point to maximize it with queries in
                this range
42            t[node].pb(x);
43            return;
44        }
45        int md = (s + e) >> 1;
46        update(node<<1,s,md,l,r,x);
47        update(node<<1|1,md+1,e,l,r,x);
48   }
49   vector<PT> stk;
50   inline void addPts(vector<PT> v) {
51        stk.clear();        ///reset the data structure you are using
52        sort(all(v));
53        ///build upper envelope
54        for(int i = 0; i < v.size(); i++) {
55            while(sz(stk) > 1 && cross(v[i] - stk.back(), stk.back() - stk[stk.size
                  ()-2]) <= 0)
56                stk.pop_back();
57            stk.push_back(v[i]);
58        }
59   }
60   inline ll calc(PT x, ll val) {
61        ///mb+y
62        return x.x * val + x.y;
63   }
64
65   ll query(ll x) {
66        if(stk.empty())
67            return LLONG_MIN;
68        int lo = 0, hi = stk.size() - 1;
69        while(lo + 10 < hi) {
70            int md = lo + (hi-lo) / 2;
71            if(calc(stk[md+1],x) > calc(stk[md],x))
72                lo = md + 1;
73            else
74                hi = md;
75        }
76        ll ans = LLONG_MIN;
77        for(int i = lo; i <= hi; i++)
78            ans = max(ans, calc(stk[i], x));
79        return ans;
80   }
81   void solve(int node, int s, int e) {     ///Solve queries
82        addPts(t[node]);      ///note that there is no need to add/delete just build
              for t[node]
83        f(i,s,e+1){
84            if(ask[i]) {
85                ans[i] = max(ans[i], query(qr[i]));
86            }
87        }
88        if(s==e)return;
89        int md = (s + e) >> 1;
90        solve(node<<1,s,md);
91        solve(node<<1|1,md+1,e);
92   }
93   void doWork() {
94
95        int n;
```

```
96        cin >> n;
97        stk.reserve(n);
98        f(i,1,n+1) {
99            int tp;
100           cin >> tp;
101           if(tp == 1) {    ///Add Query
102               int x, y;
103               cin >> x >> y;
104               val[i] = PT(x, y);
105               in[i] = 1;
106           } else if(tp == 2) {   ///Delete Query
107               int x;
108               cin >> x;
109               if(in[x])update(1, 1, n, x, i - 1, val[x]);
110               in[x] = 0;
111           } else {
112               cin >> qr[i];
113               ask[i] = true;
114           }
115       }
116       f(i,1,n+1)  ///Finalize Query
117           if(in[i])
118               update(1, 1, n, i, n, val[i]);
119
120       f(i,1,n+1)ans[i] = LLONG_MIN;
121       solve(1, 1, n);
122       f(i,1,n+1)
123       if(ask[i]) {
124           if(ans[i] == LLONG_MIN)
125               cout << "EMPTY SET\n";
126           else
127               cout << ans[i] << '\n';
128       }
129
130   }
131   int32_t main() {
132   #ifdef ONLINE_JUDGE
133       ios_base::sync_with_stdio(0);
134       cin.tie(0);
135   #endif // ONLINE_JUDGE
136       int t = 1;
137   //    cin >> t;
138       while (t--) {
139           doWork();
140       }
141       return 0;
142   }
```

4.3   Li Chao Tree

```
1    #include<iostream>
2    #include <bits/stdc++.h>
3    #define ll  long long
4    #define ld   long double
5    #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
6    using namespace std;
7    struct Line
8    {
9        ll m, b;
10       Line(ll m, ll b) : m(m), b(b) {}
11       ll operator()(ll x)
12       {
13           return m * x + b;
14       }
15   };
16   struct node
17   {
18       node * left,* right ;
19       Line line ;
20       node(node * left, node *right, Line line):left(left), right(right), line(
              line) {}
21       node * getLeft()
```

```
22          {
23              if(left==NULL)
24                  left= new node (NULL,NULL,Line(0,1e18)) ;
25              return left ;
26          }
27          node * getright()
28          {
29              if(right==NULL)
30                  right= new node (NULL,NULL,Line(0,1e18)) ;
31              return right ;
32          }
33          void insert(Line newline, int l, int r)
34          {
35              int m=(l+r)/2;
36              bool lef=newline(l)<line(l);
37              bool mid=newline(m)<line(m);
38
39              if(mid)
40                  swap(line,newline);
41              if(r-l==1)
42                  return ;
43              else if(lef!=mid)
44                  getLeft()->insert(newline,l,m);
45              else
46                  getright()->insert(newline,m,r);
47          }
48          ll query(int x, int l, int r)
49          {
50              int m = (l + r) / 2;
51              if(r - l == 1)
52                  return line(x);
53              else if(x < m)
54                  return min(line(x), getLeft()->query(x, l, m));
55              else
56                  return min(line(x), getright()->query(x, m, r));
57          }
58          void deletee()
59          {
60              if(left!=NULL)
61                  left->deletee();
62              if(right!=NULL)
63                  right->deletee();
64              free(this);
65          }
66      };
67      int main()
68      {
69          IO
70          node * root = new node(NULL,NULL,Line(0,5));
71          root->insert(Line(1,-3),1,100);
72
73          for(int i=1; i<=10; i++)
74              cout<<root->query(i,1,100)<<"\n";
75      }
```

4.4 CHT Line Container

```
1   struct Line
2   {
3       mutable ll m, b, p;
4       bool operator<(const Line& o) const
5       {
6           return m < o.m;
7       }
8       bool operator<(ll x) const
9       {
10          return p < x;
11      }
12  };
13
14  struct LineContainer : multiset<Line, less<>>
15  {
```

```
16      // (for doubles, use inf = 1/.0, div(a,b) = a/b)
17      static const ll inf = LLONG_MAX;
18      ll div(ll db, ll dm)    // floored division
19      {
20          return db / dm - ((db ^ dm) < 0 && db % dm);
21      }
22      bool isect(iterator x, iterator y)
23      {
24          if (y == end())
25          {
26              x->p = inf;
27              return false;
28          }
29          if (x->m == y->m)
30              x->p = x->b > y->b ? inf : -inf;
31          else
32              x->p = div(y->b - x->b, x->m - y->m);
33          return x->p >= y->p;
34      }
35      void add(ll m, ll b)
36      {
37          auto z = insert({m, b, 0}), y = z++, x = y;
38          while (isect(y, z))
39              z = erase(z);
40          if (x != begin() && isect(--x, y))
41              isect(x, y = erase(y));
42          while ((y = x) != begin() && (--x)->p >= y->p)
43              isect(x, erase(y));
44      }
45      ll query(ll x)
46      {
47          assert(!empty());
48          auto l = *lower_bound(x);
49          return l.m * x + l.b;
50      }
51  };
```

5   Geometry

5.1   Convex Hull

```
1   struct point {
2       ll x,  y;
3       point(ll x, ll y) : x(x), y(y) {}
4       point operator -(point other) {
5           return point(x - other.x, y - other.y);
6       }
7       bool operator <(const point &other) const {
8           return x != other.x ? x < other.x : y < other.y;
9       }
10  };
11  ll cross(point a, point b) {
12      return a.x * b.y - a.y * b.x;
13  }
14  ll dot(point a, point b) {
15      return a.x * b.x + a.y * b.y;
16  }
17  struct sortCCW {
18      point center;
19
20      sortCCW(point center) : center(center) {}
21
22      bool operator()(point a, point b) {
23          ll res = cross(a - center, b - center);
24          if(res)
25              return res > 0;
26          return dot(a - center, a - center) < dot(b - center, b - center);
27      }
28  };
29  vector<point> hull(vector<point> v) {
```

```
30        sort(v.begin(), v.end());
31        sort(v.begin() + 1, v.end(), sortCCW(v[0]));
32        v.push_back(v[0]);
33        vector<point> ans ;
34        for(auto i : v) {
35            int sz = ans.size();
36            while(sz > 1 && cross(i - ans[sz - 1], ans[sz - 2] - ans[sz - 1]) <= 0)
37                ans.pop_back(), sz--;
38            ans.push_back(i);
39        }
40        ans.pop_back();
41        return ans;
42    }
```

5.2   Geometry Template

```
1    using  ptype =  double edit this first ;
2    double EPS = 1e-9;
3    struct point {
4
5        ptype x,  y;
6        point(ptype x, ptype y) : x(x), y(y) {}
7
8        point operator -(const point & other)const {
9            return point(x - other.x, y - other.y);
10        }
11
12        point operator +(const point & other)const {
13            return point(x + other.x, y + other.y);
14        }
15
16        point operator *(ptype c) const {
17            return point(x * c, y * c);
18        }
19
20        point operator /(ptype c) const {
21            return point(x / c, y / c);
22        }
23        point prep() {
24            return point(-y, x);
25        }
26
27    };
28    ptype cross(point a, point b) {
29        return a.x * b.y - a.y * b.x;
30    }
31
32    ptype dot(point a, point b) {
33        return a.x * b.x + a.y * b.y;
34    }
35    double abs(point a) {
36        return sqrt(dot(a, a));
37    }
38    // angle between [0 , pi]
39    double angle (point a, point b) {
40        return acos(dot(a, b) / abs(a) / abs(b));
41    }
42    // a : point in Line
43    // d : Line direction
44    point LineLineIntersect(point a1, point d1, point a2, point d2) {
45        return a1 + d1 * cross(a2 - a1, d2) / cross(d1, d2);
46    }
47    // Line a---b
48    // point C
49    point ProjectPointLine(point a, point b, point c) {
50        return a + (b - a) * 1.0 * dot(c - a, b - a) / dot(b - a, b - a);
51    }
52    // segment a---b
53    // point C
54    point ProjectPointSegment(point a, point b, point c) {
55        double r = dot(c - a, b - a) / dot(b - a, b - a);
56        if(r < 0)
```

```
57            return a;
58        if(r > 1)
59            return b;
60        return a + (b - a) * r ;
61    }
62    // Line a---b
63    // point p
64    point reflectAroundLine(point a, point b, point p) {
65        //(proj-p) *2 + p
66        return ProjectPointLine(a, b, p) * 2 - p;
67    }
68    // Around origin
69    point RotateCCW(point p, double t) {
70        return point(p.x * cos(t) - p.y * sin(t),
71                     p.x * sin(t) + p.y * cos(t));
72    }
73    // Line a---b
74    vector<point> CircleLineIntersect(point a, point b, point center, double r) {
75        a = a - center;
76        b = b - center;
77        point p = ProjectPointLine(a, b, point(0, 0)); // project point from center
                to the Line
78        if(dot(p, p) > r * r)
79            return {};
80        double len = sqrt(r * r - dot(p, p));
81        if(len < EPS)
82            return {center + p};
83
84        point d = (a - b) / abs(a - b);
85        return {center + p + d * len, center + p - d * len};
86    }
87    vector<point> CircleCircleIntersect(point c1, double r1, point c2, double r2) {
88
89        if(r1 < r2) {
90            swap(r1, r2);
91            swap(c1, c2);
92        }
93        double d = abs(c1 - c2); // distance between c1,c2
94        if(d > r1 + r2 || d < r1 - r2)
95            return {};
96
97        double angle = acos(min((d * d + r1 * r1 - r2 * r2) / (2 * r1 * d), 1.0));
98        point p = (c2 - c1) / d * r1;
99
100        if(angle < EPS)
101            return {p};
102
103        return {RotateCCW(p, angle), RotateCCW(p, -angle)};
104
105    }
106    point circumcircle(point p1, point p2, point p3) {
107
108        return LineLineIntersect((p1 + p2) / 2, (p1 - p2).prep(),
109                                 (p1 + p3) / 2, (p1 - p3).prep() );
110    }
111    //S : Area.
112    //I : number points with integer coordinates lying strictly inside the polygon.
113    //B : number of points lying on polygon sides by B.
114    //S = I + B/2 - 1
```

5.3   Half Plane Intersection

```
1    // Redefine epsilon and infinity as necessary. Be mindful of precision errors.
2    const long double eps = 1e-9, inf = 1e9;
3
4    // Basic point/vector struct.
5    struct Point {
6
7        long double x, y;
8        explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}
9
10        // Addition, substraction, multiply by constant, cross product.
```

```
11
12        friend Point operator + (const Point& p, const Point& q) {
13            return Point(p.x + q.x, p.y + q.y);
14        }
15
16        friend Point operator - (const Point& p, const Point& q) {
17            return Point(p.x - q.x, p.y - q.y);
18        }
19
20        friend Point operator * (const Point& p, const long double& k) {
21            return Point(p.x * k, p.y * k);
22        }
23
24        friend long double cross(const Point& p, const Point& q) {
25            return p.x * q.y - p.y * q.x;
26        }
27    };
28
29    // Basic half-plane struct.
30    struct Halfplane {
31
32        // 'p' is a passing point of the line and 'pq' is the direction vector of
            //     the line.
33        Point p, pq;
34        long double angle;
35
36        Halfplane() {}
37        Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
38            angle = atan2l(pq.y, pq.x);
39        }
40
41        // Check if point 'r' is outside this half-plane.
42        // Every half-plane allows the region to the LEFT of its line.
43        bool out(const Point& r) {
44            return cross(pq, r - p) < -eps;
45        }
46
47        // Comparator for sorting.
48        // If the angle of both half-planes is equal, the leftmost one should go
            //     first.
49        bool operator < (const Halfplane& e) const {
50            if (fabsl(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
51            return angle < e.angle;
52        }
53
54        // We use equal comparator for std::unique to easily remove parallel half-
            //     planes.
55        bool operator == (const Halfplane& e) const {
56            return fabsl(angle - e.angle) < eps;
57        }
58
59        // Intersection point of the lines of two half-planes. It is assumed they're
            //     never parallel.
60        friend Point inter(const Halfplane& s, const Halfplane& t) {
61            long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
62            return s.p + (s.pq * alpha);
63        }
64    };
65
66
67
68    // Actual algorithm
69    vector<Point> hp_intersect(vector<Halfplane>& H) {
70
71        Point box[4] = {  // Bounding box in CCW order
72            Point(inf, inf),
73            Point(-inf, inf),
74            Point(-inf, -inf),
75            Point(inf, -inf)
76        };
77
78        for(int i = 0; i<4; i++) { // Add bounding box half-planes.
79            Halfplane aux(box[i], box[(i+1) % 4]);
80            H.push_back(aux);
81        }
```

```
82
83        // Sort and remove duplicates
84        sort(H.begin(), H.end());
85        H.erase(unique(H.begin(), H.end()), H.end());
86
87        deque<Halfplane> dq;
88        int len = 0;
89        for(int i = 0; i < int(H.size()); i++) {
90
91            // Remove from the back of the deque while last half-plane is redundant
92            while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
93                dq.pop_back();
94                --len;
95            }
96
97            // Remove from the front of the deque while first half-plane is
                //     redundant
98            while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
99                dq.pop_front();
100               --len;
101           }
102
103           // Add new half-plane
104           dq.push_back(H[i]);
105           ++len;
106       }
107
108       // Final cleanup: Check half-planes at the front against the back and vice-
            //     versa
109       while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
110           dq.pop_back();
111           --len;
112       }
113
114       while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
115           dq.pop_front();
116           --len;
117       }
118
119       // Report empty intersection if necessary
120       if (len < 3) return vector<Point>();
121
122       // Reconstruct the convex polygon from the remaining half-planes.
123       vector<Point> ret(len);
124       for(int i = 0; i+1 < len; i++) {
125           ret[i] = inter(dq[i], dq[i+1]);
126       }
127       ret.back() = inter(dq[len-1], dq[0]);
128       return ret;
129   }
```

5.4   Segments Intersection

```
1    const double EPS = 1E-9;
2
3    struct pt {
4        double x, y;
5    };
6
7    struct seg {
8        pt p, q;
9        int id;
10
11       double get_y(double x) const {
12           if (abs(p.x - q.x) < EPS)
13               return p.y;
14           return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
15       }
16   };
17
18   bool intersect1d(double l1, double r1, double l2, double r2) {
19       if (l1 > r1)
```

```
20              swap(l1, r1);
21          if (l2 > r2)
22              swap(l2, r2);
23          return max(l1, l2) <= min(r1, r2) + EPS;
24  }
25
26  int vec(const pt& a, const pt& b, const pt& c) {
27      double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
28      return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
29  }
30
31  bool intersect(const seg& a, const seg& b)
32  {
33      return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
34             intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
35             vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
36             vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
37  }
38
39  bool operator<(const seg& a, const seg& b)
40  {
41      double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
42      return a.get_y(x) < b.get_y(x) - EPS;
43  }
44
45  struct event {
46      double x;
47      int tp, id;
48
49      event() {}
50      event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
51
52      bool operator<(const event& e) const {
53          if (abs(x - e.x) > EPS)
54              return x < e.x;
55          return tp > e.tp;
56      }
57  };
58
59  set<seg> s;
60  vector<set<seg>::iterator> where;
61
62  set<seg>::iterator prev(set<seg>::iterator it) {
63      return it == s.begin() ? s.end() : --it;
64  }
65
66  set<seg>::iterator next(set<seg>::iterator it) {
67      return ++it;
68  }
69
70  pair<int, int> solve(const vector<seg>& a) {
71      int n = (int)a.size();
72      vector<event> e;
73      for (int i = 0; i < n; ++i) {
74          e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
75          e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
76      }
77      sort(e.begin(), e.end());
78
79      s.clear();
80      where.resize(a.size());
81      for (size_t i = 0; i < e.size(); ++i) {
82          int id = e[i].id;
83          if (e[i].tp == +1) {
84              set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
85              if (nxt != s.end() && intersect(*nxt, a[id]))
86                  return make_pair(nxt->id, id);
87              if (prv != s.end() && intersect(*prv, a[id]))
88                  return make_pair(prv->id, id);
89              where[id] = s.insert(nxt, a[id]);
90          } else {
91              set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
92              if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
93                  return make_pair(prv->id, nxt->id);
94              s.erase(where[id]);
```

```
95          }
96      }
97
98      return make_pair(-1, -1);
99  }
```

5.5   Rectangles Union

```
1   #include<bits/stdc++.h>
2   #define P(x,y) make_pair(x,y)
3   using namespace std;
4   class Rectangle {
5   public:
6       int x1, y1, x2, y2;
7       static Rectangle empt;
8       Rectangle() {
9           x1 = y1 = x2 = y2 = 0;
10      }
11      Rectangle(int X1, int Y1, int X2, int Y2) {
12          x1 = X1;
13          y1 = Y1;
14          x2 = X2;
15          y2 = Y2;
16      }
17  };
18  struct Event {
19      int x, y1, y2, type;
20      Event() {}
21      Event(int x, int y1, int y2, int type): x(x), y1(y1), y2(y2), type(type) {}
22  };
23  bool operator < (const Event&A, const Event&B) {
24  //if(A.x != B.x)
25      return A.x < B.x;
26  //if(A.y1 != B.y1) return A.y1 < B.y1;
27  //if(A.y2 != B.y2()) A.y2 < B.y2;
28  }
29  const int MX = (1 << 17);
30  struct Node {
31      int prob, sum, ans;
32      Node() {}
33      Node(int prob, int sum, int ans): prob(prob), sum(sum), ans(ans) {}
34  };
35  Node tree[MX * 4];
36  int interval[MX];
37  void build(int x, int a, int b) {
38      tree[x] = Node(0, 0, 0);
39      if(a == b) {
40          tree[x].sum += interval[a];
41          return;
42      }
43      build(x * 2, a, (a + b) / 2);
44      build(x * 2 + 1, (a + b) / 2 + 1, b);
45      tree[x].sum = tree[x * 2].sum + tree[x * 2 + 1].sum;
46  }
47  int ask(int x) {
48      if(tree[x].prob)
49          return tree[x].sum;
50      return tree[x].ans;
51  }
52  int st, en, V;
53  void update(int x, int a, int b) {
54      if(st > b || en < a)
55          return;
56      if(a >= st && b <= en) {
57          tree[x].prob += V;
58          return;
59      }
60      update(x * 2, a, (a + b) / 2);
61      update(x * 2 + 1, (a + b) / 2 + 1, b);
62      tree[x].ans = ask(x * 2) + ask(x * 2 + 1);
63  }
64  Rectangle Rectangle::empt = Rectangle();
```

```cpp
65      vector < Rectangle > Rect;
66      vector < int > sorted;
67      vector < Event > sweep;
68      void compressncalc() {
69          sweep.clear();
70          sorted.clear();
71          for(auto R : Rect) {
72              sorted.push_back(R.y1);
73              sorted.push_back(R.y2);
74          }
75          sort(sorted.begin(), sorted.end());
76          sorted.erase(unique(sorted.begin(), sorted.end()), sorted.end());
77          int sz = sorted.size();
78          for(int j = 0; j < sorted.size() - 1; j++)
79              interval[j + 1] = sorted[j + 1] - sorted[j];
80          for(auto R : Rect) {
81              sweep.push_back(Event(R.x1, R.y1, R.y2, 1));
82              sweep.push_back(Event(R.x2, R.y1, R.y2, -1));
83          }
84          sort(sweep.begin(), sweep.end());
85          build(1, 1, sz - 1);
86      }
87      long long ans;
88      void Sweep() {
89          ans = 0;
90          if(sorted.empty() || sweep.empty())
91              return;
92          int last = 0, sz_ = sorted.size();
93          for(int j = 0; j < sweep.size(); j++) {
94              ans += 1ll * (sweep[j].x - last) * ask(1);
95              last = sweep[j].x;
96              V = sweep[j].type;
97              st = lower_bound(sorted.begin(), sorted.end(), sweep[j].y1 - sorted.
                      begin() + 1;
98              en = lower_bound(sorted.begin(), sorted.end(), sweep[j].y2 - sorted.
                      begin();
99              update(1, 1, sz_ - 1);
100         }
101     }
102     int main() {
103     //     freopen("in.in","r",stdin);
104         int n;
105         scanf("%d", &n);
106         for(int j = 1; j <= n; j++) {
107             int a, b, c, d;
108             scanf("%d %d %d %d", &a, &b, &c, &d);
109             Rect.push_back(Rectangle(a, b, c, d));
110         }
111         compressncalc();
112         Sweep();
113         cout << ans << endl;
114     }
```

6 Graphs

6.1 2 SAD

```cpp
1  /**
2   * Author: Emil Lenngren, Simon Lindholm
3   * Date: 2011-11-29
4   * License: CC0
5   * Source: folklore
6   * Description: Calculates a valid assignment to boolean variables a, b, c,...
7       to a 2-SAT problem, so that an expression of the type $(a\/\/b)\&\&(!a\/\/
8       c)\&\&(d\/\/!b)\&\&...$ becomes true, or reports that it is unsatisfiable.
9   * Negated variables are represented by bit-inversions (\texttt{\tilde{}x}).
10  * Usage:
11  *  TwoSat ts(number of boolean variables);
12  *  ts.either(0, \tilde3); // Var 0 is true or var 3 is false
13  *  ts.setValue(2); // Var 2 is true
```

```cpp
12   *  ts.atMostOne({0,\tilde1,2}); // <= 1 of vars 0, \tilde1 and 2 are true
13   *  ts.solve(); // Returns true iff it is solvable
14   *  ts.values[0..N-1] holds the assigned values to the vars
15   * Time: O(N+E), where N is the number of boolean variables, and E is the number
          of clauses.
16   * Status: stress-tested
17   */
18  #pragma once
19
20  struct TwoSat {
21      int N;
22      vector<vi> gr;
23      vi values; // 0 = false, 1 = true
24
25      TwoSat(int n = 0) : N(n), gr(2*n) {}
26
27      int addVar() { // (optional)
28          gr.emplace_back();
29          gr.emplace_back();
30          return N++;
31      }
32
33      void either(int f, int j) {
34          f = max(2*f, -1-2*f);
35          j = max(2*j, -1-2*j);
36          gr[f].push_back(j^1);
37          gr[j].push_back(f^1);
38      }
39      void setValue(int x) { either(x, x); }
40
41      void atMostOne(const vi& li) { // (optional)
42          if (sz(li) <= 1) return;
43          int cur = ~li[0];
44          rep(i,2,sz(li)) {
45              int next = addVar();
46              either(cur, ~li[i]);
47              either(cur, next);
48              either(~li[i], next);
49              cur = ~next;
50          }
51          either(cur, ~li[1]);
52      }
53
54      vi val, comp, z; int time = 0;
55      int dfs(int i) {
56          int low = val[i] = ++time, x; z.push_back(i);
57          for(int e : gr[i]) if (!comp[e])
58              low = min(low, val[e] ?: dfs(e));
59          if (low == val[i]) do {
60              x = z.back(); z.pop_back();
61              comp[x] = low;
62              if (values[x>>1] == -1)
63                  values[x>>1] = x&1;
64          } while (x != i);
65          return val[i] = low;
66      }
67
68      bool solve() {
69          values.assign(N, -1);
70          val.assign(2*N, 0); comp = val;
71          rep(i,0,2*N) if (!comp[i]) dfs(i);
72          rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
73          return 1;
74      }
75  };
```

6.2 Ariculation Point

```cpp
1  vector<int> adj[N];
2  int dfsn[N], low[N], instack[N], ar_point[N], timer;
3  stack<int> st;
4
```

```
5   void dfs(int node, int par){
6       dfsn[node] = low[node] = ++timer;
7       int kam = 0;
8       for(auto i: adj[node]){
9           if(i == par) continue;
10          if(dfsn[i] == 0){
11              kam++;
12              dfs(i, node);
13              low[node] = min(low[node], low[i]);
14              if(dfsn[node] <= low[i] && par != 0) ar_point[node] = 1;
15          }
16          else low[node] = min(low[node], dfsn[i]);
17      }
18      if(par == 0 && kam > 1) ar_point[node] = 1;
19  }
20
21  void init(int n){
22      for(int i = 1; i <= n; i++){
23          adj[i].clear();
24          low[i] = dfsn[i] = 0;
25          instack[i] = 0;
26          ar_point[i] = 0;
27      }
28      timer = 0;
29  }
30
31  int main(){
32      int tt;
33      cin >> tt;
34      while(tt--){
35          // Input
36          init(n);
37          for(int i = 1; i <= n; i++){
38              if(dfsn[i] == 0) dfs(i, 0);
39          }
40          int c = 0;
41          for(int i = 1; i <= n; i++){
42              if(ar_point[i]) c++;
43          }
44          cout << c << '\n';
45      }
46      return 0;
47  }
```

```
27              comp_id[cur] = cnt;
28              if(cur == node) break;
29          }
30      }
31  }
32
33  void dfs2(int node, int par){
34      kam[node] = 0;
35      int mx = 0, second_mx = 0;
36      for(auto i: bridge_tree[node]){
37          if(i == par) continue;
38          dfs2(i, node);
39          kam[node] = max(kam[node], 1 + kam[i]);
40          if(kam[i] > mx){
41              second_mx = mx;
42              mx = kam[i];
43          }
44          else second_mx = max(second_mx, kam[i]);
45      }
46      ans = max(ans, kam[node]);
47      if(second_mx) ans = max(ans, 2 + mx + second_mx);
48  }
49
50  int main(){
51      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
52      int n, m;
53      cin >> n >> m;
54      while(m--){
55          int u, v;
56          cin >> u >> v;
57          adj[u].push_back(v);
58          adj[v].push_back(u);
59      }
60      dfs(1, 0);
61      for(int i = 1; i <= n; i++){
62          for(auto j: adj[i]){
63              if(comp_id[i] != comp_id[j]){
64                  bridge_tree[comp_id[i]].push_back(comp_id[j]);
65              }
66          }
67      }
68      dfs2(1, 0);
69      cout << ans;
70
71      return 0;
72  }
```

6.3   Bridges Tree and Diameter

```
1   #include <bits/stdc++.h>
2   #define ll long long
3   using namespace std;
4   const int N = 3e5 + 5, mod = 1e9 + 7;
5
6   vector<int> adj[N], bridge_tree[N];
7   int dfsn[N], low[N], cost[N], timer, cnt, comp_id[N], kam[N], ans;
8   stack<int> st;
9
10
11  void dfs(int node, int par){
12      dfsn[node] = low[node] = ++timer;
13      st.push(node);
14      for(auto i: adj[node]){
15          if(i == par) continue;
16          if(dfsn[i] == 0){
17              dfs(i, node);
18              low[node] = min(low[node], low[i]);
19          }
20          else low[node] = min(low[node], dfsn[i]);
21      }
22      if(dfsn[node] == low[node]){
23          cnt++;
24          while(1){
25              int cur = st.top();
26              st.pop();
```

6.4   Dinic With Scalling

```
1   ///O(ElgFlow) on Bipratite Graphs and O(EVlgFlow) on other graphs (I think)
2   struct Dinic {
3       #define vi vector<int>
4       #define rep(i,a,b)   f(i,a,b)
5       struct Edge {
6           int to, rev;
7           ll c, oc;
8           int id;
9           ll flow() { return max(oc - c, 0LL); } // if you need flows
10      };
11      vi lvl, ptr, q;
12      vector<vector<Edge>> adj;
13      Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
14      void addEdge(int a, int b, ll c, int id, ll rcap = 0) {
15          adj[a].push_back({b, sz(adj[b]), c, c, id});
16          adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap,id});
17      }
18      ll dfs(int v, int t, ll f) {
19          if (v == t || !f) return f;
20          for (int& i = ptr[v]; i < sz(adj[v]); i++) {
21              Edge& e = adj[v][i];
22              if (lvl[e.to] == lvl[v] + 1)
23                  if (ll p = dfs(e.to, t, min(f, e.c))) {
```

```
24                    e.c -= p, adj[e.to][e.rev].c += p;
25                    return p;
26                }
27            }
28            return 0;
29        }
30        ll calc(int s, int t) {
31            ll flow = 0; q[0] = s;
32            rep(L,0,31) do { // 'int L=30' maybe faster for random data
33                lvl = ptr = vi(sz(q));
34                int qi = 0, qe = lvl[s] = 1;
35                while (qi < qe && !lvl[t]) {
36                    int v = q[qi++];
37                    for (Edge e : adj[v])
38                        if (!lvl[e.to] && e.c >> (30 - L))
39                            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
40                }
41                while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
42            } while (lvl[t]);
43            return flow;
44        }
45        bool leftOfMinCut(int a) { return lvl[a] != 0; }
46    };
```

6.5   Gomory Hu

```
1  /**
2   * Author: chilli, Takanori MAEHARA
3   * Date: 2020-04-03
4   * License: CC0
5   * Source: https://github.com/spaghetti-source/algorithm/blob/master/graph/
        gomory_hu_tree.cc#L102
6   * Description: Given a list of edges representing an undirected flow graph,
7   * returns edges of the Gomory-Hu tree. The max flow between any pair of
8   * vertices is given by minimum edge weight along the Gomory-Hu tree path.
9   * Time: $O(V)$ Flow Computations
10  * Status: Tested on CERC 2015 J, stress-tested
11  *
12  * Details: The implementation used here is not actually the original
13  * Gomory-Hu, but Gusfield's simplified version: "Very simple methods for all
14  * pairs network flow analysis". PushRelabel is used here, but any flow
15  * implementation that supports 'leftOfMinCut' also works.
16  */
17 #pragma once
18
19 #include "PushRelabel.h"
20
21 typedef array<ll, 3> Edge;
22 vector<Edge> gomoryHu(int N, vector<Edge> ed) {
23     vector<Edge> tree;
24     vi par(N);
25     rep(i,1,N) {
26         PushRelabel D(N); // Dinic also works
27         for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
28         tree.push_back({i, par[i], D.calc(i, par[i])});
29         rep(j,i+1,N)
30             if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
31     }
32     return tree;
33 }
```

6.6   HopcraftKarp BPM

```
1  /**
2   * Author: Chen Xing
3   * Date: 2009-10-13
4   * License: CC0
5   * Source: N/A
6   * Description: Fast bipartite matching algorithm. Graph $g$ should be a list
```

```
7   * of neighbors of the left partition, and $btoa$ should be a vector full of
8   * -1's of the same size as the right partition. Returns the size of
9   * the matching. $btoa[i]$ will be the match for vertex $i$ on the right side,
10  * or $-1$ if it's not matched.
11  * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
12  * Time: O(\sqrt{V}E)
13  * Status: stress-tested by MinimumVertexCover, and tested on oldkattis.
        adkbipmatch and SPOJ:MATCHING
14  */
15 #pragma once
16
17 bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
18     if (A[a] != L) return 0;
19     A[a] = -1;
20     for (int b : g[a]) if (B[b] == L + 1) {
21         B[b] = 0;
22         if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
23             return btoa[b] = a, 1;
24     }
25     return 0;
26 }
27
28 int hopcroftKarp(vector<vi>& g, vi& btoa) {
29     int res = 0;
30     vi A(g.size()), B(btoa.size()), cur, next;
31     for (;;) {
32         fill(all(A), 0);
33         fill(all(B), 0);
34         /// Find the starting nodes for BFS (i.e. layer 0).
35         cur.clear();
36         for (int a : btoa) if(a != -1) A[a] = -1;
37         rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
38         /// Find all layers using bfs.
39         for (int lay = 1;; lay++) {
40             bool islast = 0;
41             next.clear();
42             for (int a : cur) for (int b : g[a]) {
43                 if (btoa[b] == -1) {
44                     B[b] = lay;
45                     islast = 1;
46                 }
47                 else if (btoa[b] != a && !B[b]) {
48                     B[b] = lay;
49                     next.push_back(btoa[b]);
50                 }
51             }
52             if (islast) break;
53             if (next.empty()) return res;
54             for (int a : next) A[a] = lay;
55             cur.swap(next);
56         }
57         /// Use DFS to scan for augmenting paths.
58         rep(a,0,sz(g))
59             res += dfs(a, 0, g, btoa, A, B);
60     }
61 }
```

6.7   Hungarian

```
1  /*
2      Notes:
3          note that n must be <= m
4          so in case your problem n >= m, just swap
5      also note this
6      void set(int x, int y, ll v){a[x+1][y+1]=v;}
7      the algorithim assumes you're using 0-index
8      but it's using 1-based
9  */
10 struct Hungarian {
11     const ll INF = 1000000000000000000; ///10^18
12     int n,m;
13     vector<vector<ll> > a;
```

```
14        vector<ll> u,v;vector<int> p,way;
15        Hungarian(int n, int m):
16        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){}
17        void set(int x, int y, ll v){a[x+1][y+1]=v;}
18        ll assign(){
19            for(int i = 1; i <= n; i++){
20                int j0=0;p[0]=i;
21                vector<ll> minv(m+1,INF);
22                vector<char> used(m+1,false);
23                do {
24                    used[j0]=true;
25                    int i0=p[j0],j1;ll delta=INF;
26                    for(int j = 1; j <= m; j++)if(!used[j]){
27                        ll cur=a[i0][j]-u[i0]-v[j];
28                        if(cur<minv[j])minv[j]=cur,way[j]=j0;
29                        if(minv[j]<delta)delta=minv[j],j1=j;
30                    }
31                    for(int j = 0; j <= m; j++)
32                        if(used[j])u[p[j]]+=delta,v[j]-=delta;
33                        else minv[j]-=delta;
34                    j0=j1;
35                } while(p[j0]);
36                do {
37                    int j1=way[j0];p[j0]=p[j1];j0=j1;
38                } while(j0);
39            }
40            return -v[0];
41        }
42        vector<int> restoreAnswer() {   ///run it after assign
43            vector<int> ans (n+1);
44            for (int j=1; j<=m; ++j)
45                ans[p[j]] = j;
46            return ans;
47        }
48    };
```

6.8   Kosaraju

```
1  /*
2   g :  Adjacency List of the original graph
3   rg : Reversed Adjacency List
4   vis : A bitset to mark visited nodes
5   adj : Adjacency List of the super graph
6   stk : holds dfs ordered elements
7   cmp[i] : holds the component of node i
8   go[i] : holds the nodes inside the strongly connected component i
9  */
10
11 #define FOR(i,a,b) for(int i = a; i < b; i++)
12 #define pb push_back
13
14 const int N = 1e5+5;
15
16 vector<vector<int>>g, rg;
17 vector<vector<int>>go;
18 bitset<N>vis;
19 vector<vector<int>>adj;
20 stack<int>stk;
21 int n, m, cmp[N];
22 void add_edge(int u, int v){
23    g[u].push_back(v);
24    rg[v].push_back(u);
25 }
26 void dfs(int u){
27    vis[u]=1;
28    for(auto v : g[u])if(!vis[v])dfs(v);
29    stk.push(u);
30 }
31 void rdfs(int u,int c){
32    vis[u] = 1;
33    cmp[u] = c;
34    go[c].push_back(u);
```

```
35        for(auto v : rg[u])if(!vis[v])rdfs(v,c);
36    }
37    int scc(){
38        vis.reset();
39        for(int i = 0; i < n; i++)if(!vis[i])
40            dfs(i);
41        vis.reset();
42        int c = 0;
43        while(stk.size()){
44            auto cur = stk.top();
45            stk.pop();
46            if(!vis[cur])
47                rdfs(cur,c++);
48
49        }
50        return c;
51    }
```

6.9   Krichoff

```
1  /*
2      Count number of spanning trees in a graph
3  */
4  int power(long long n, long long k) {
5    int ans = 1;
6    while (k) {
7      if (k & 1) ans = (long long) ans * n % mod;
8      n = (long long) n * n % mod;
9      k >>= 1;
10   }
11   return ans;
12 }
13 int det(vector<vector<int>> a) {
14   int n = a.size(), m = (int)a[0].size();
15   int free_var = 0;
16   const long long MODSQ = (long long)mod * mod;
17   int det = 1, rank = 0;
18   for (int col = 0, row = 0; col < m && row < n; col++) {
19     int mx = row;
20     for (int k = row; k < n; k++) if (a[k][col] > a[mx][col]) mx = k;
21     if (a[mx][col] == 0) {
22       det = 0;
23       continue;
24     }
25     for (int j = col; j < m; j++) swap(a[mx][j], a[row][j]);
26     if (row != mx) det = det == 0 ? 0 : mod - det;
27     det = 1LL * det * a[row][col] % mod;
28     int inv = power(a[row][col], mod - 2);
29     for (int i = 0; i < n && inv; i++) {
30       if (i != row && a[i][col]) {
31         int x = ((long long)a[i][col] * inv) % mod;
32         for (int j = col; j < m && x; j++) {
33           if (a[row][j]) a[i][j] = (MODSQ + a[i][j] - ((long long)a[row][j] * x)
34                  ) % mod;
35         }
36       }
37     }
38     row++;
39     ++rank;
40   }
41   return det;
42 }
```

6.10   Manhattan MST

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N = 2e5 + 9;
```

```cpp
5   int n;
6   vector<pair<int, int>> g[N];
7   struct PT {
8       int x, y, id;
9       bool operator < (const PT &p) const {
10          return x == p.x ? y < p.y : x < p.x;
11      }
12  } p[N];
13  struct node {
14      int val, id;
15  } t[N];
16  struct DSU {
17      int p[N];
18      void init(int n) { for (int i = 1; i <= n; i++)  p[i] = i; }
19      int find(int u) { return p[u] == u ? u : p[u] = find(p[u]); }
20      void merge(int u, int v) { p[find(u)] = find(v); }
21  } dsu;
22  struct edge {
23      int u, v, w;
24      bool operator < (const edge &p) const { return w < p.w; }
25  };
26  vector<edge> edges;
27  int query(int x) {
28      int r = 2e9 + 10, id = -1;
29      for (; x <= n; x += (x & -x)) if (t[x].val < r) r = t[x].val, id = t[x].id;
30      return id;
31  }
32  void modify(int x, int w, int id) {
33      for (; x > 0; x -= (x & -x)) if (t[x].val > w) t[x].val = w, t[x].id = id;
34  }
35  int dist(PT &a, PT &b) {
36      return abs(a.x - b.x) + abs(a.y - b.y);
37  }
38  void add(int u, int v, int w) {
39      edges.push_back({u, v, w});
40  }
41  long long Kruskal() {
42      dsu.init(n);
43      sort(edges.begin(), edges.end());
44      long long ans = 0;
45      for (edge e : edges) {
46          int u = e.u, v = e.v, w = e.w;
47          if (dsu.find(u) != dsu.find(v)) {
48              ans += w;
49              g[u].push_back({v, w});
50              //g[v].push_back({u, w});
51              dsu.merge(u, v);
52          }
53      }
54      return ans;
55  }
56  void Manhattan() {
57      for (int i = 1; i <= n; ++i) p[i].id = i;
58      for (int dir = 1; dir <= 4; ++dir) {
59          if (dir == 2 || dir == 4) {
60              for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
61          }
62          else if (dir == 3) {
63              for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
64          }
65          sort(p + 1, p + 1 + n);
66          vector<int> v;
67          static int a[N];
68          for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x, v.push_back(a[i]);
69          sort(v.begin(), v.end());
70          v.erase(unique(v.begin(), v.end()), v.end());
71          for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;
72          for (int i = 1; i <= n; ++i) t[i].val = 2e9 + 10, t[i].id = -1;
73          for (int i = n; i >= 1; --i) {
74              int pos = query(a[i]);
75              if (pos != -1) add(p[i].id, p[pos].id, dist(p[i], p[pos]));
76              modify(a[i], p[i].x + p[i].y, i);
```

```cpp
78          }
79      }
80  }
81  int32_t main() {
82      ios_base::sync_with_stdio(0);
83      cin.tie(0);
84      cin >> n;
85      for (int i = 1; i <= n; i++) cin >> p[i].x >> p[i].y;
86      Manhattan();
87      cout << Kruskal() << '\n';
88      for (int u = 1; u <= n; u++) {
89          for (auto x: g[u]) cout << u - 1 << ' ' << x.first - 1 << '\n';
90      }
91      return 0;
92  }
```

6.11 Maximum Clique

```cpp
1   ///Complexity O(3 ^ (N/3)) i.e works for 50
2   ///you can change it to maximum independent set by flipping the edges 0->1, 1->0
3   ///if you want to extract the nodes they are 1-bits in R
4   int g[60][60];
5   int res;
6   long long edges[60];
7   void BronKerbosch(int n, long long R, long long P, long long X) {
8       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
9           int t = __builtin_popcountll(R);
10          res = max(res, t);
11          return;
12      }
13      int u = 0;
14      while (!((1LL << u) & (P | X))) u ++;
15      for (int v = 0; v < n; v++) {
16          if (((1LL << v) & P) && !((1LL << v) & edges[u])) {
17              BronKerbosch(n, R | (1LL << v), P & edges[v], X & edges[v]);
18              P -= (1LL << v);
19              X |= (1LL << v);
20          }
21      }
22  }
23  int max_clique (int n) {
24      res = 0;
25      for (int i = 1; i <= n; i++) {
26          edges[i - 1] = 0;
27          for (int j = 1; j <= n; j++)  if (g[i][j]) edges[i - 1] |= ( 1LL << (j - 1)
                );
28      }
29      BronKerbosch(n, 0, (1LL << n) - 1, 0);
30      return res;
31  }
```

6.12 MCMF

```cpp
1   /*
2       Notes:
3           make sure you notice the #define int ll
4           focus on the data types of the max flow everythign inside is integer
5           addEdge(u,v,cap,cost)
6           note that for min cost max flow the cost is sum of cost * flow over all
                edges
7   */
8
9   struct Edge {
10      int to;
11      int cost;
12      int cap, flow, backEdge;
13  };
14
```

```
15  struct MCMF {
16
17      const int inf = 1000000010;
18      int n;
19      vector<vector<Edge>> g;
20
21      MCMF(int _n) {
22          n = _n + 1;
23          g.resize(n);
24      }
25
26      void addEdge(int u, int v, int cap, int cost) {
27          Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
28          Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
29          g[u].push_back(e1);
30          g[v].push_back(e2);
31      }
32
33      pair<int, int> minCostMaxFlow(int s, int t) {
34          int flow = 0;
35          int cost = 0;
36          vector<int> state(n), from(n), from_edge(n);
37          vector<int> d(n);
38          deque<int> q;
39          while (true) {
40              for (int i = 0; i < n; i++)
41                  state[i] = 2, d[i] = inf, from[i] = -1;
42              state[s] = 1;
43              q.clear();
44              q.push_back(s);
45              d[s] = 0;
46              while (!q.empty()) {
47                  int v = q.front();
48                  q.pop_front();
49                  state[v] = 0;
50                  for (int i = 0; i < (int) g[v].size(); i++) {
51                      Edge e = g[v][i];
52                      if (e.flow >= e.cap || (d[e.to] <= d[v] + e.cost))
53                          continue;
54                      int to = e.to;
55                      d[to] = d[v] + e.cost;
56                      from[to] = v;
57                      from_edge[to] = i;
58                      if (state[to] == 1) continue;
59                      if (!state[to] || (!q.empty() && d[q.front()] > d[to]))
60                          q.push_front(to);
61                      else q.push_back(to);
62                      state[to] = 1;
63                  }
64              }
65              if (d[t] == inf) break;
66              int it = t, addflow = inf;
67              while (it != s) {
68                  addflow = min(addflow,
69                                g[from[it]][from_edge[it]].cap
70                                - g[from[it]][from_edge[it]].flow);
71                  it = from[it];
72              }
73              it = t;
74              while (it != s) {
75                  g[from[it]][from_edge[it]].flow += addflow;
76                  g[it][g[from[it]][from_edge[it]].backEdge].flow -= addflow;
77                  cost += g[from[it]][from_edge[it]].cost * addflow;
78                  it = from[it];
79              }
80              flow += addflow;
81          }
82          return {cost, flow};
83      }
84  };
```

6.13   Minimum Arbroscene in a Graph

```
1   const int maxn = 2510, maxm = 7000000;
2   const ll maxint = 0x3f3f3f3f3f3f3f3fLL;
3
4   int n, ec, ID[maxn], pre[maxn], vis[maxn];
5   ll in[maxn];
6
7   struct edge_t {
8       int u, v;
9       ll w;
10  } edge[maxm];
11  void add(int u, int v, ll w) {
12      edge[++ec].u = u, edge[ec].v = v, edge[ec].w = w;
13  }
14
15  ll arborescence(int n, int root) {
16      ll res = 0, index;
17      while (true) {
18          for (int i = 1; i <= n; ++i) {
19              in[i] = maxint, vis[i] = -1, ID[i] = -1;
20          }
21          for (int i = 1; i <= ec; ++i) {
22              int u = edge[i].u, v = edge[i].v;
23              if (u == v || in[v] <= edge[i].w) continue;
24              in[v] = edge[i].w, pre[v] = u;
25          }
26          pre[root] = root, in[root] = 0;
27          for (int i = 1; i <= n; ++i) {
28              res += in[i];
29              if (in[i] == maxint) return -1;
30          }
31          index = 0;
32          for (int i = 1; i <= n; ++i) {
33              if (vis[i] != -1) continue;
34              int u = i, v;
35              while (vis[u] == -1) {
36                  vis[u] = i;
37                  u = pre[u];
38              }
39              if (vis[u] != i || u == root) continue;
40              for (v = u, u = pre[u], ++index; u != v; u = pre[u]) ID[u] = index;
41              ID[v] = index;
42          }
43          if (index == 0) return res;
44          for (int i = 1; i <= n; ++i) if (ID[i] == -1) ID[i] = ++index;
45          for (int i = 1; i <= ec; ++i) {
46              int u = edge[i].u, v = edge[i].v;
47              edge[i].u = ID[u], edge[i].v = ID[v];
48              edge[i].w -= in[v];
49          }
50          n = index, root = ID[root];
51      }
52      return res;
53  }
```

6.14   Minmimum Vertex Cover (Bipartite)

```
1   int myrandom (int i) { return std::rand()%i;}
2
3   struct MinimumVertexCover {
4       int n, id;
5       vector<vector<int> > g;
6       vector<int> color, m, seen;
7       vector<int> comp[2];
8       MinimumVertexCover() {}
9       MinimumVertexCover(int n, vector<vector<int> > g) {
10
11          this->n = n;
12          this->g = g;
13          color = m = vector<int>(n, -1);
```

```
14          seen = vector<int>(n, 0);
15          makeBipartite();
16      }
17
18      void dfsBipartite(int node, int col) {
19          if (color[node] != -1) {
20              assert(color[node] == col); /* MSH BIPARTITE YA BASHMOHANDES */
21              return;
22          }
23          color[node] = col;
24          comp[col].push_back(node);
25          for (int i = 0; i < int(g[node].size()); i++)
26              dfsBipartite(g[node][i], 1 - col);
27      }
28
29      void makeBipartite() {
30          for (int i = 0; i < n; i++)
31              if (color[i] == -1)
32                  dfsBipartite(i, 0);
33      }
34
35      // match a node
36      bool dfs(int node) {
37        random_shuffle(g[node].begin(),g[node].end());
38          for (int i = 0; i < g[node].size(); i++) {
39              int child = g[node][i];
40              if (m[child] == -1) {
41                  m[node] = child;
42                  m[child] = node;
43                  return true;
44              }
45              if (seen[child] == id)
46                  continue;
47              seen[child] = id;
48              int enemy = m[child];
49              m[node] = child;
50              m[child] = node;
51              m[enemy] = -1;
52              if (dfs(enemy))
53                  return true;
54              m[node] = -1;
55              m[child] = enemy;
56              m[enemy] = child;
57          }
58          return false;
59      }
60
61      void makeMatching() {
62      for(int j = 0; j < 5; j++)
63        random_shuffle(comp[0].begin(),comp[0].end(),myrandom );
64          for (int i = 0; i < int(comp[0].size()); i++) {
65              id++;
66              if(m[comp[0][i]] == -1)
67                  dfs(comp[0][i]);
68          }
69      }
70
71
72      void recurse(int node, int x, vector<int> &minCover, vector<int> &done) {
73          if (m[node] != -1)
74              return;
75          if (done[node])return;
76          done[node] = 1;
77          for (int i = 0; i < int(g[node].size()); i++) {
78              int child = g[node][i];
79              int newnode = m[child];
80              if (done[child]) continue;
81              if(newnode == -1) {
82                  continue;
83              }
84              done[child] = 2;
85              minCover.push_back(child);
86              m[newnode] = -1;
87              recurse(newnode, x, minCover, done);
```

```
88          }
89      }
90
91      vector<int> getAnswer() {
92          vector<int> minCover, maxIndep;
93          vector<int> done(n, 0);
94          makeMatching();
95          for (int x = 0; x < 2; x++)
96              for (int i = 0; i < int(comp[x].size()); i++) {
97                  int node = comp[x][i];
98                  if (m[node] == -1)
99                      recurse(node, x, minCover, done);
100             }
101
102         for (int i = 0; i < int(comp[0].size()); i++)
103             if (!done[comp[0][i]])
104                 minCover.push_back(comp[0][i]);
105         }
106         return minCover;
107     }
108 };
```

6.15   Prufer Code

```
1   #include<bits/stdc++.h>
2   using namespace std;
3
4   const int N = 3e5 + 9;
5
6   /*
7   prufer code is a sequence of length n-2 to uniquely determine a labeled tree
        with n vertices
8   Each time take the leaf with the lowest number and add the node number the leaf
        is connected to
9   the sequence and remove the leaf. Then break the algo after n-2 iterations
10  */
11  //0-indexed
12  int n;
13  vector<int> g[N];
14  int parent[N], degree[N];
15
16  void dfs (int v) {
17    for (size_t i = 0; i < g[v].size(); ++i) {
18      int to = g[v][i];
19      if (to != parent[v]) {
20        parent[to] = v;
21        dfs (to);
22      }
23    }
24  }
25
26  vector<int> prufer_code() {
27    parent[n - 1] = -1;
28    dfs (n - 1);
29    int ptr = -1;
30    for (int i = 0; i < n; ++i) {
31      degree[i] = (int) g[i].size();
32      if (degree[i] == 1 && ptr == -1) ptr = i;
33    }
34    vector<int> result;
35    int leaf = ptr;
36    for (int iter = 0; iter < n - 2; ++iter) {
37      int next = parent[leaf];
38      result.push_back (next);
39      --degree[next];
40      if (degree[next] == 1 && next < ptr) leaf = next;
41      else {
42        ++ptr;
43        while (ptr < n && degree[ptr] != 1) ++ptr;
44        leaf = ptr;
45      }
46    }
```

```
47        return result;
48    }
49    vector < pair<int, int> > prufer_to_tree(const vector<int> & prufer_code) {
50        int n = (int) prufer_code.size() + 2;
51        vector<int> degree (n, 1);
52        for (int i = 0; i < n - 2; ++i) ++degree[prufer_code[i]];
53
54        int ptr = 0;
55        while (ptr < n && degree[ptr] != 1) ++ptr;
56        int leaf = ptr;
57        vector < pair<int, int> > result;
58        for (int i = 0; i < n - 2; ++i) {
59            int v = prufer_code[i];
60            result.push_back (make_pair (leaf, v));
61            --degree[leaf];
62            if (--degree[v] == 1 && v < ptr) leaf = v;
63            else {
64                ++ptr;
65                while (ptr < n && degree[ptr] != 1) ++ptr;
66                leaf = ptr;
67            }
68        }
69        for (int v = 0; v < n - 1; ++v) if (degree[v] == 1) result.push_back (
                make_pair (v, n - 1));
70        return result;
71    }
72
73    int32_t main() {
74
75        return 0;
76    }
```

6.16   Push Relabel Max Flow

```
1     struct edge
2     {
3         int from, to, cap, flow, index;
4         edge(int from, int to, int cap, int flow, int index):
5             from(from), to(to), cap(cap), flow(flow), index(index) {}
6     };
7
8     struct PushRelabel
9     {
10        int n;
11        vector<vector<edge> > g;
12        vector<long long> excess;
13        vector<int> height, active, count;
14        queue<int> Q;
15
16        PushRelabel(int n):
17            n(n), g(n), excess(n), height(n), active(n), count(2*n) {}
18
19        void addEdge(int from, int to, int cap)
20        {
21            g[from].push_back(edge(from, to, cap, 0, g[to].size()));
22            if(from==to)
23                g[from].back().index++;
24            g[to].push_back(edge(to, from, 0, 0, g[from].size()-1));
25        }
26
27        void enqueue(int v)
28        {
29            if(!active[v] && excess[v] > 0)
30            {
31                active[v]=true;
32                Q.push(v);
33            }
34        }
35
36        void push(edge &e)
37        {
38            int amt=(int)min(excess[e.from], (long long)e.cap - e.flow);
39            if(height[e.from]<=height[e.to] || amt==0)
40                return;
41            e.flow += amt;
42            g[e.to][e.index].flow -= amt;
43            excess[e.to] += amt;
44            excess[e.from] -= amt;
45            enqueue(e.to);
46        }
47
48        void relabel(int v)
49        {
50            count[height[v]]--;
51            int d=2*n;
52            for(auto &it:g[v])
53            {
54                if(it.cap-it.flow>0)
55                    d=min(d, height[it.to]+1);
56            }
57            height[v]=d;
58            count[height[v]]++;
59            enqueue(v);
60        }
61
62        void gap(int k)
63        {
64            for(int v=0;v<n;v++)
65            {
66                if(height[v]<k)
67                    continue;
68                count[height[v]]--;
69                height[v]=max(height[v], n+1);
70                count[height[v]]++;
71                enqueue(v);
72            }
73        }
74
75        void discharge(int v)
76        {
77            for(int i=0; excess[v]>0 && i<g[v].size(); i++)
78                push(g[v][i]);
79            if(excess[v]>0)
80            {
81                if(count[height[v]]==1)
82                    gap(height[v]);
83                else
84                    relabel(v);
85            }
86        }
87
88        long long max_flow(int source, int dest)
89        {
90            count[0] = n-1;
91            count[n] = 1;
92            height[source] = n;
93            active[source] = active[dest] = 1;
94            for(auto &it:g[source])
95            {
96                excess[source]+=it.cap;
97                push(it);
98            }
99
100           while(!Q.empty())
101           {
102               int v=Q.front();
103               Q.pop();
104               active[v]=false;
105               discharge(v);
106           }
107
108           long long max_flow=0;
109           for(auto &e:g[source])
110               max_flow+=e.flow;
111
112           return max_flow;
113       }
```

```
114   };
```

6.17   Tarjan Algo

```cpp
1   vector< vector<int> > scc;
2   vector<int> adj[N];
3   int dfsn[N], low[N], cost[N], timer, in_stack[N];
4   stack<int> st;
5
6   // to detect all the components (cycles) in a directed graph
7   void tarjan(int node){
8       dfsn[node] = low[node] = ++timer;
9       in_stack[node] = 1;
10      st.push(node);
11      for(auto i: adj[node]){
12          if(dfsn[i] == 0){
13              tarjan(i);
14              low[node] = min(low[node], low[i]);
15          }
16          else if(in_stack[i]) low[node] = min(low[node], dfsn[i]);
17      }
18      if(dfsn[node] == low[node]){
19          scc.push_back(vector<int>());
20          while(1){
21              int cur = st.top();
22              st.pop();
23              in_stack[cur] = 0;
24              scc.back().push_back(cur);
25              if(cur == node) break;
26          }
27      }
28  }
29  int main(){
30      int m;
31      cin >> m;
32      while(m--){
33          int u, v;
34          cin >> u >> v;
35          adj[u].push_back(v);
36      }
37      for(int i = 1; i <= n; i++){
38          if(dfsn[i] == 0){
39              tarjan(i);
40          }
41      }
42
43      return 0;
44  }
```

6.18   Bipartite Matching

```cpp
1   #include<iostream>
2   #include <bits/stdc++.h>
3   #define ll long long
4   #define ld long double
5   #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
6   using namespace std;
7   struct graph
8   {
9       int L, R;
10      vector<vector<int> > adj;
11      graph(int l, int r) : L(l), R(r), adj(l+1) {}
12      void add_edge(int u, int v)
13      {
14          adj[u].push_back(v+L);
15      }
16      int maximum_matching()
17      {
18          vector<int> mate(L+R+1,-1), level(L+1);
```

```cpp
19          function<bool (void)> levelize = [&]()
20          {
21              queue<int> q;
22              for(int i=1; i<=L; i++)
23              {
24                  level[i]=-1;
25                  if(mate[i]<0)
26                      q.push(i), level[i]=0;
27              }
28              while(!q.empty())
29              {
30                  int node=q.front();
31                  q.pop();
32                  for(auto i : adj[node])
33                  {
34                      int v=mate[i];
35                      if(v<0)
36                          return true;
37                      if(level[v]<0)
38                      {
39                          level[v]=level[node]+1;
40                          q.push(v);
41                      }
42                  }
43              }
44              return false;
45          };
46          function<bool (int)> augment =[&](int node)
47          {
48              for(auto i : adj[node])
49              {
50                  int v=mate[i];
51                  if(v<0 || (level[v]>level[node] && augment(v)))
52                  {
53                      mate[node]=i;
54                      mate[i]=node;
55                      return true;
56                  }
57              }
58              return false;
59          };
60          int match=0;
61          while(levelize())
62              for(int i=1; i<=L; i++)
63                  if(mate[i] < 0 && augment(i))
64                      match++;
65          return match;
66      }
67  };
68
69  int main()
70  {
71      IO
72      int L, R, m;
73      cin>>L>>R>>m;
74      graph g(L, R);
75      for (int i = 0; i < m; ++i)
76      {
77          int u, v;
78          cin>>u>>v;
79          g.add_edge(u, v);
80      }
81      cout<<g.maximum_matching();
82  }
```

7   Math

7.1   Xor With Gauss

```cpp
1   /*
```

```
2        Some applications
3        If you want to find the maximum in xor subset
4        just ans = max(ans, ans ^ p[i]) for all i
5        if you want to count the number of subsets with a certain value
6        check all different subsets of p
7    */
8    ll p[66];
9    bool add(ll x) {
10       for(int i = 60; (~i) && x; --i) {
11           if(x >> i & 1) {
12               if(!p[i]) {
13                   p[i] = x;
14                   return true;
15               } else {
16                   x ^= p[i];
17               }
18           }
19       }
20       return false;
21   }
```

7.2 Josephus

```
1    // n = total person
2    // will kill every kth person, if k = 2, 2,4,6,...
3    // returns the mth killed person
4    ll josephus(ll n, ll k, ll m) {
5      m = n - m;
6      if (k <= 1) return n - m;
7      ll i = m;
8      while (i < n) {
9          ll r = (i - m + k - 2) / (k - 1);
10         if ((i + r) > n) r = n - i;
11         else if (!r) r = 1;
12         i += r;
13         m = (m + (r * k)) % i;
14     } return m + 1;
15   }
```

7.3 Matrix Power/Multiplication

```
1    struct Matrix {
2
3        const static int D = 100;
4        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

```cpp
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

```cpp
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

```cpp
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);
```

```
15      vector<int> ret;
16      for(int i = 0, g = 0; i < s.size(); i++) {
17          while (g && s[i] != t[g])
18              g = pi[g-1];
19          g += s[i] == t[g];
20          if(g == t.size()) { ///occurrence found
21              ret.push_back(i-t.size()+1);
22              g = pi[g-1];
23          }
24      }
25      return ret;
26  }
```

8.4   Manacher Kactl

```
1   // If the size of palindrome centered at i is x, then d1[i] stores (x+1)/2.
2
3   vector<int> d1(n);
4   for (int i = 0, l = 0, r = -1; i < n; i++) {
5       int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
6       while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
7           k++;
8       }
9       d1[i] = k--;
10      if (i + k > r) {
11          l = i - k;
12          r = i + k;
13      }
14  }
15
16
17  // If the size of palindrome centered at i is x, then d2[i] stores x/2
18
19  vector<int> d2(n);
20  for (int i = 0, l = 0, r = -1; i < n; i++) {
21      int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
22      while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
23          k++;
24      }
25      d2[i] = k--;
26      if (i + k > r) {
27          l = i - k - 1;
28          r = i + k ;
29      }
30  }
```

8.5   Suffix Array Kactl

```
1   struct SuffixArray {
2       using vi = vector<int>;
3       #define rep(i,a,b)  for(int i = a; i < b; i++)
4       /*
5           Note this code is considers also the empty suffix
6           so hear sa[0] = n and sa[1] is the smallest non empty suffix
7           and sa[n] is the largest non empty suffix
8           also LCP[i] = LCP(sa[i-1], sa[i]), meaning LCP[0] = LCP[1] = 0
9           if you want to get LCP(i..j) you need to build a mapping between
10          sa[i] and i, and build a min sparse table to calculate the minimum
11          note that this minimum should consider sa[i+1...j] since you don't want
12          to consider LCP(sa[i], sa[i-1])
13
14          you should also print the suffix array and lcp at the beginning of the
              contest
15          to clarify this stuff
16      */
17      vi sa, lcp;
18      SuffixArray(string& s, int lim=256) { // or basic_string<int>
19          int n = sz(s) + 1, k = 0, a, b;
20          vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
```

```
21          sa = lcp = y, iota(all(sa), 0);
22          for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
23              p = j, iota(all(y), n - j);
24              rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
25              fill(all(ws), 0);
26              rep(i,0,n) ws[x[i]]++;
27              rep(i,1,lim) ws[i] += ws[i - 1];
28              for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
29              swap(x, y), p = 1, x[sa[0]] = 0;
30              rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
31                  (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
32          }
33          rep(i,1,n) rank[sa[i]] = i;
34          for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
35              for (k && k--, j = sa[rank[i] - 1];
36                  s[i + k] == s[j + k]; k++);
37      }
38  };
```

8.6   Suffix Automaton Anany

```
1   ///Note it's better to use addNode to clear a node before using it
2   ///at the start of each test case use initAutomaton
3
4   int last = 0, cntState = 1;
5   int nxt[N * 2][26];
6   int len[N * 2], link[N * 2], firstPos[N * 2], cnt[N * 2];
7
8   void addNode(int i) {
9       memset(nxt[i], 0, sizeof nxt[i]);
10      link[i] = -1;
11      cnt[i] = 0;
12  }
13
14  void initAutomaton() {
15      cntState = 1;
16      last = 0;
17      addNode(last);
18  }
19
20  int addChar(char c) {
21
22      c -= 'a';    ///note this offset
23      int p = last;
24      int cur = cntState++;
25      addNode(cur);
26      cnt[cur] = 1; ///extra
27      len[cur] = len[last] + 1;
28      firstPos[cur] = len[cur] - 1;   ///extra
29      while(p != -1 && nxt[p][c] == 0) {
30          nxt[p][c] = cur;
31          p = link[p];
32      }
33
34      if(p == -1) {
35          link[cur] = 0;
36      } else {
37          int q = nxt[p][c];
38          if(len[q] == len[p] + 1) {
39              link[cur] = q;
40          } else {
41              int clone = cntState++;
42              link[clone] = link[q];
43              firstPos[clone] = firstPos[q];   ///extra
44              len[clone] = len[p] + 1;
45              link[q] = link[cur] = clone;
46              memcpy(nxt[clone], nxt[q], sizeof nxt[q]);
47              cnt[clone] = 0; ///extra
48              f(i,0,26)nxt[clone][i] = nxt[q][i];
49              while(p != -1 && nxt[p][c] == q) {
50                  nxt[p][c] = clone;
51                  p = link[p];
```

```
52                  }
53              }
54          }
55          last = cur;
56          return cur;
57      }
```

8.7   Suffix Automaton Mostafa

```
1   #include <bits/stdc++.h>
2
3   #define FIO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
4   using namespace std;
5   typedef long long ll;
6   typedef long double ld;
7   const int N = 2e6 + 9, M = 5e5 + 9;
8
9   struct SA {
10      struct node {
11          int to[26];
12          int link, len, co = 0;
13
14          node() {
15              memset(to, 0, sizeof to);
16              co = 0, link = 0, len = 0;
17          }
18      };
19
20      int last, sz;
21      vector<node> v;
22
23      SA() {
24          v = vector<node>(1);
25          last = 0, sz = 1;
26      }
27
28      void add_letter(int c) {
29          int p = last;
30          last = sz++;
31          v.push_back({});
32          v[last].len = v[p].len + 1;
33          v[last].co = 1;
34          for (; v[p].to[c] == 0; p = v[p].link)
35              v[p].to[c] = last;
36          if (v[p].to[c] == last) {
37              v[last].link = 0;
38              return;
39          }
40          int q = v[p].to[c];
41          if (v[q].len == v[p].len + 1) {
42              v[last].link = q;
43              return;
44          }
45          int cl = sz++;
46          v.push_back(v[q]);
47          v.back().co = 0;
48          v.back().len = v[p].len + 1;
49          v[last].link = v[q].link = cl;
50
51          for (; v[p].to[c] == q; p = v[p].link)
52              v[p].to[c] = cl;
53      }
54
55      void build_co() {
56          priority_queue<pair<int, int>> q;
57          for (int i = sz - 1; i > 0; i--)
58              q.push({v[i].len, i});
59          while (q.size()) {
60              int i = q.top().second;
61              q.pop();
62              v[v[i].link].co += v[i].co;
63          }
```

```
64          }
65      };
66
67      int main() {
68          FIO
69
70          return 0;
71      }
```

8.8   Suffix Automaton With Rollback Mostafa

```
1   #include <bits/stdc++.h>
2
3   #define FIO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
4   using namespace std;
5   typedef long long ll;
6   typedef long double ld;
7   const int N = 2e6 + 9, M = 5e5 + 9;
8
9   struct SA {
10      struct node {
11          int to[26];
12          int link, len, co = 0;
13
14          node() {
15              memset(to, 0, sizeof to);
16              co = 0, link = 0, len = 0;
17          }
18      };
19
20      struct LogNode {
21          int last, sz;
22          vector<pair<pair<int, int>, int>> edges;
23          pair<int, int> LinksUpdate = {0, 0};
24      };
25
26      int last, sz;
27      vector<node> v;
28      vector<LogNode> logs;
29
30      SA() {
31          v = vector<node>(1);
32          last = 0, sz = 1;
33      }
34
35      void add_letter(int c) {
36          logs.push_back({});
37          logs.back().last = last;
38          logs.back().sz = sz;
39
40          int p = last;
41          last = sz++;
42          v.push_back({});
43          v[last].len = v[p].len + 1;
44          v[last].co = 1;
45          for (; v[p].to[c] == 0; p = v[p].link) {
46              logs.back().edges.push_back({{p, c}, 0});
47              v[p].to[c] = last;
48          }
49          if (v[p].to[c] == last) {
50              v[last].link = 0;
51              return;
52          }
53          int q = v[p].to[c];
54          if (v[q].len == v[p].len + 1) {
55              v[last].link = q;
56              return;
57          }
58          int cl = sz++;
59          v.push_back(v[q]);
60          v.back().co = 0;
61          v.back().len = v[p].len + 1;
```

```
62              logs.back().LinksUpdate = {q, v[q].link};
63              v[last].link = v[q].link = cl;
64              for (; v[p].to[c] == q; p = v[p].link) {
65                  logs.back().edges.push_back({{p, c}, q});
66                  v[p].to[c] = cl;
67              }
68          }
69          void rollback() {
70              assert(logs.size());
71              auto log = logs.back();
72              while (v.size() > log.sz)
73                  v.pop_back();
74              for (auto edge: log.edges)
75                  v[edge.first.first].to[edge.first.second] = edge.second;
76              if (log.LinksUpdate.first != 0)
77                  v[log.LinksUpdate.first].link = log.LinksUpdate.second;
78              last = log.last;
79              sz = log.sz;
80              logs.pop_back();
81          }
82      };
83
84      int main() {
85          FIO
86
87          return 0;
88      }
```

8.9   Zalgo Anany

```
1   int z[N], n;
2   void Zalgo(string s) {
3       int L = 0, R = 0;
4       for(int i = 1; i < n; i++) {
5           if(i<=R&&z[i-L] < R - i + 1)z[i] = z[i-L];
6           else {
7               L = i;
8               R = max(R,i);
9               while(R < n && s[R-L] == s[R])R++;
10              z[i] = R-L; --R;
11          }
12      }
13  }
```

9   Trees

9.1   Centroid Decomposition

```
1   /*
2       Properties:
3           1. consider path(a,b) can be decomposed to path(a,lca(a,b)) and path(b,
                lca(a,b))
4           where lca(a,b) is the lca on the centroid tree
5           2. Each one of the n^2 paths is the concatenation of two paths in a set
                of O(n lg(n))
6           paths from a node to all its ancestors in the centroid decomposition.
7           3. Ancestor of a node in the original tree is either an ancestor in the
                CD tree or
8           a descendadnt
9   */
10  vector<int> adj[N]; ///adjacency list of original graph
11  int n;
12  int sz[N];
13  bool used[N];
14  int centPar[N]; ///parent in centroid
15  void init(int node, int par) {  ///initialize size
16      sz[node] = 1;
17      for(auto p : adj[node])
18          if(p != par && !used[p]) {
19              init(p, node);
20              sz[node] += sz[p];
21          }
22  }
23  int centroid(int node, int par, int limit) {    ///get centroid
24      for(int p : adj[node])
25          if(!used[p] && p != par && sz[p] * 2 > limit)
26              return centroid(p, node, limit);
27      return node;
28  }
29  int decompose(int node) {
30      init(node,node);     ///calculate size
31      int c = centroid(node, node, sz[node]); ///get centroid
32      used[c] = true;
33      for(auto p : adj[c])if(!used[p.F]) {     ///initialize parent for others and
            decompose
34          centPar[decompose(p.F)] = c;
35      }
36      return c;
37  }
38  void update(int node, int distance, int col) {
39      int centroid = node;
40      while(centroid){
41          ///solve
42          centroid = centPar[centroid];
43      }
44  }
45  int query(int node) {
46
47      int ans = 0;
48
49      int centroid = node;
50      while(centroid) {
51          ///solve
52          centroid = centPar[centroid];
53      }
54
55      return ans;
56  }
```

9.2   Dsu On Trees

```
1   const int N = 1e5 + 9;
2   vector<int> adj[N];
3   int bigChild[N], sz[N];
4   void dfs(int node, int par) {
5       for(auto v : adj[node]) if(v != par){
6           dfs(v, node);
7           sz[node] += sz[v];
8           if(!bigChild[node] || sz[v] > sz[bigChild[node]]) {
9               bigChild[node] = v;
10          }
11      }
12  }
13  void add(int node, int par, int bigChild, int delta) {
14
15      ///modify node to data structure
16
17      for(auto v : adj[node])
18      if(v != par && v != bigChild)
19          add(v, node, bigChild, delta);
20
21  }
22  void dfs2(int node, int par, bool keep) {
23      for(auto v : adj[node])if(v != par && v != bigChild[node]) {
24          dfs2(v, node, 0);
25      }
26      if(bigChild[node]) {
27          dfs2(bigChild[node], node, true);
28      }
```

```
29        add(node, par, bigChild[node], 1);
30        ///process queries
31        if(!keep) {
32            add(node, par, -1, -1);
33        }
34    }
```

9.3 Heavy Light Decomposition (Along with Euler Tour)

```
1    /*
2        Notes:
3            1. 0-based
4            2. solve function iterates over segments and handles them seperatly
5            if you're gonna use it make sure you know what you're doing
6            3. to update/query segment in[node], out[node]
7            4. to update/query chain in[nxt[node]], in[node]
8            nxt[node]: is the head of the chain so to go to the next chain node =
                    par[nxt[node]]
9    */
10   int sz[mxN], nxt[mxN];
11   int in[N], out[N], rin[N];
12   vector<int> g[mxN];
13   int par[mxN];
14
15   void dfs_sz(int v = 0, int p = -1) {
16       sz[v] = 1;
17       par[v] = p;
18       for (auto &u : g[v]) {
19           if (u == p) {
20               swap(u, g[v].back());
21           }
22           if(u == p) continue;
23           dfs_sz(u,v);
24           sz[v] += sz[u];
25           if (sz[u] > sz[g[v][0]])
26               swap(u, g[v][0]);
27       }
28       if(v != 0)
29           g[v].pop_back();
30   }
31
32   void dfs_hld(int v = 0) {
33       in[v] = t++;
34       rin[in[v]] = v;
35       for (auto u : g[v]) {
36           nxt[u] = (u == g[v][0] ? nxt[v] : u);
37           dfs_hld(u);
38       }
39       out[v] = t;
40   }
41
42   int n;
43   bool isChild(int p, int u){
44     return in[p] <= in[u] && out[u] <= out[p];
45   }
46   int solve(int u,int v) {
47       vector<pair<int,int> > segu;
48       vector<pair<int,int> > segv;
49       if(isChild(u,v)){
50           while(nxt[u] != nxt[v]){
51               segv.push_back(make_pair(in[nxt[v]], in[v]));
52               v = par[nxt[v]];
53           }
54           segv.push_back({in[u], in[v]});
55       } else if(isChild(v,u)){
56           while(nxt[u] != nxt[v]){
57               segu.push_back(make_pair(in[nxt[u]], in[u]));
58               u = par[nxt[u]];
59           }
60           segu.push_back({in[v], in[u]});
61       } else {
62           while(u  != v) {
```

```
63           if(nxt[u] == nxt[v]) {
64               if(in[u] < in[v]) segv.push_back({in[u],in[v]}), R.push_back({u+1,v
                    +1});
65               else segu.push_back({in[v],in[u]}), L.push_back({v+1,u+1});
66               u = v;
67               break;
68           } else if(in[u] > in[v]) {
69               segu.push_back({in[nxt[u]],in[u]}), L.push_back({nxt[u]+1, u+1});
70               u = par[nxt[u]];
71           } else {
72               segv.push_back({in[nxt[v]],in[v]}), R.push_back({nxt[v]+1, v+1});
73               v = par[nxt[v]];
74           }
75       }
76   }
77   reverse(segv.begin(),segv.end());
78   int res = 0,state = 0;
79   for(auto p : segu) {
80       qry(1,1,0,n-1,p.first,p.second,state,res);
81   }
82   for(auto p : segv) {
83       qry(0,1,0,n-1,p.first,p.second,state,res);
84   }
85   return res;
86 }
```

9.4 LCA

```
1    const int N = 1e5 + 5;
2    const int LG = 18;
3
4    vector<int> adj[N];
5    int pa[N][LG], lvl[N];
6    int in[N], out[N], timer;
7    void dfs(int u, int p){
8      in[u] = ++timer;
9      for(int k = 1; k < LG; k++)
10         pa[u][k] = pa[pa[u][k-1]][k-1];
11     for(auto v : adj[u])
12         if(v != p){
13             lvl[v] = lvl[u] + 1;
14             pa[v][0] = u;
15             dfs(v, u);
16         }
17     out[u] = timer;
18   }
19   int LCA(int u, int v){
20     if(lvl[u] > lvl[v])
21         swap(u,v);
22     int d = lvl[v] - lvl[u];
23     for(int k = 0; k < LG; k++)
24         if(d >> k & 1)
25             v = pa[v][k];
26     if(u == v)return u;
27     for(int i = LG - 1; i >= 0; --i)
28         if(pa[u][i] != pa[v][i]){
29             u = pa[u][i];
30             v = pa[v][i];
31         }
32     return pa[u][0];
33   }
```

9.5 Mo on Trees

```
1    int BL[N << 1], ID[N << 1];
2    int lvl[N], par[17][N];
3    int ans[N];
4    vector<ii> adj[N];
5    struct query{
```

```
6        int id, l, r, lc;
7        bool operator < (const query & rhs){
8            return (BL[l] == BL[rhs.l]) ? (r < rhs.r) : (BL[l] < BL[rhs.l]);
9        }
10   }Q[N];
11   int in[N], out[N], val[N], timer;
12   void dfs(int node, int p){
13       in[node] = ++timer;  ID[timer] = node;
14       for(int i = 1; i < 17; i++)par[i][node] = par[i-1][par[i-1][node]];
15       for(auto child : adj[node])if(child.F != p){
16           lvl[child.F] = lvl[node] + 1;
17           par[0][child.F] = node;
18           val[child.F] = child.S;
19           dfs(child.F, node);
20       }
21       out[node] = ++timer;  ID[timer] = node;
22   }
23   int LCA(int u, int v){
24       if(lvl[u] > lvl[v])swap(u,v);
25       for(int k = 0; k < 17; k++)
26           if((lvl[v] - lvl[u]) >> k & 1)
27               v = par[k][v];
28       if(u == v)
29           return u;
30       for(int i = 16; i >= 0; --i)
31           if(par[i][u] != par[i][v])
32               u = par[i][u], v = par[i][v];
33       return par[0][u];
34   }
35   bool vis[N];
36   int inSet[N];
37   void add(int node, int & res){
38       if(val[node] > N)return;
39       if(!vis[node]){
40           inSet[val[node]]++;
41           while(inSet[res])res++;
42       } else {
43           inSet[val[node]]--;
44           if(!inSet[val[node]] && val[node] < res)
45               res = val[node];
46       }
47       vis[node] ^= 1;
48   }
49   //------------------Adding Queries-----------------/
50   f(i,0,q){
51       int u, v;
52       cin >> u >> v;  if(lvl[u] > lvl[v])swap(u, v);
53       int lca = LCA(u, v);
54       Q[i].id = i;
55       Q[i].lc = lca;
56       if(lca == u)Q[i].l = in[u], Q[i].r = in[v];
57       else {
58           Q[i].l = out[u];
59           Q[i].r = in[v];
60       }
61   }
62   //------------------Processing Queries-----------------/
63   f(i,0,q){
64           while (curL < Q[i].l) add(ID[curL++], res);
65           while (curL > Q[i].l) add(ID[--curL], res);
66           while (curR < Q[i].r) add(ID[++curR], res);
67           while (curR > Q[i].r) add(ID[curR--], res);
68           int u = ID[Q[i].l];
```

```
69           int v = ID[Q[i].r];
70           if(Q[i].lc == u)add(Q[i].lc, res);
71           ans[Q[i].id] = res;
72           if(Q[i].lc == u)add(Q[i].lc, res);
73   }
```

10   Numerical

10.1   Lagrange Polynomial

```
1    class LagrangePoly {
2    public:
3        LagrangePoly(std::vector<long long> _a) {
4            //f(i) = _a[i]
5            //interpola o vetor em um polinomio de grau y.size() - 1
6            y = _a;
7            den.resize(y.size());
8            int n = (int) y.size();
9            for(int i = 0; i < n; i++) {
10               y[i] = (y[i] % MOD + MOD) % MOD;
11               den[i] = ifat[n - i - 1] * ifat[i] % MOD;
12               if((n - i - 1) % 2 == 1) {
13                   den[i] = (MOD - den[i]) % MOD;
14               }
15           }
16       }
17
18       long long getVal(long long x) {
19           int n = (int) y.size();
20           x = (x % MOD + MOD) % MOD;
21           if(x < n) {
22               //return y[(int) x];
23           }
24           std::vector<long long> l, r;
25           l.resize(n);
26           l[0] = 1;
27           for(int i = 1; i < n; i++) {
28               l[i] = l[i - 1] * (x - (i - 1) + MOD) % MOD;
29           }
30           r.resize(n);
31           r[n - 1] = 1;
32           for(int i = n - 2; i >= 0; i--) {
33               r[i] = r[i + 1] * (x - (i + 1) + MOD) % MOD;
34           }
35           long long ans = 0;
36           for(int i = 0; i < n; i++) {
37               long long coef = l[i] * r[i] % MOD;
38               ans = (ans + coef * y[i] % MOD * den[i]) % MOD;
39           }
40           return ans;
41       }
42
43   private:
44       std::vector<long long> y, den;
45   };
```