

Contents

2021

1	Template	
1.1	template	
2	Combinatorics	
2.1	Burnside Lemma	
2.2	Catlan Numbers	
3	Algebra	
3.1	Gray Code	
3.2	Primitive Roots	
3.3	Discrete Logarithm minimum x for which $a^x = b \% m$	
3.4	Discrete Root finds all numbers x such that $x^k = a \% n$	
3.5	Factorial modulo in $p * \log(n)$ (Wilson Theroem)	
3.6	Iteration over submasks	
3.7	Totient function	
3.8	CRT and EGCD	
3.9	FFT	
3.10	FFT with mod	
3.11	convolutions of AND-XOR-OR	
3.12	NTT of KACTL	
3.13	Fibonacci	
3.14	Gauss Determinant	
3.15	GAUSS SLAE	
3.16	Matrix Inverse	
4	Data Structures	
4.1	UnionFindRollback	
4.2	2D BIT	
4.3	2D Sparse table	
4.4	Mo With Updates	
4.5	Ordered Set	
4.6	Persistent Seg Tree	
4.7	Treap	
4.8	Wavelet Tree	
4.9	SparseTable	
5	DP	
5.1	Dynamic Connectivity with SegTree	
5.2	CHT Line Container	
6	Geometry	
6.1	Convex Hull	
6.2	Geometry Template	
6.3	Half Plane Intersection	
6.4	Segments Intersection	
6.5	Rectangles Union	
7	Graphs	
7.1	2 SAD	
7.2	Ariculation Point	
7.3	Bridges Tree and Diameter	
7.4	Dinic With Scalling	
7.5	Gomory Hu	
7.6	HopcraftKarp BPM	
7.7	Hungarian	
7.8	Kosaraju	
7.9	Manhattan MST	
7.10	Maximum Clique	
7.11	MCMF	
7.12	Minimum Arbrosce in a Graph	
7.13	Minimium Vertex Cover (Bipartite)	

7.14	Prufer Code	16
7.15	Push Relabel Max Flow	16
7.16	Tarjan Algo	17
7.17	Bipartite Matching	17
8	Math	17
8.1	Sum Of floored division.	17
8.2	ModMulLL	17
8.3	MillerRabin Primality check	18
8.4	Pollard-rho randomized factorization algorithm $O(n^{1/4})$	18
8.5	ModSqrt Finds x s.t $x^2 = a \mod p$	18
8.6	Xor With Gauss	18
8.7	Josephus	18
9	Strings	18
9.1	Aho-Corasick Mostafa	18
9.2	KMP Anany	19
9.3	Manacher Kactl	19
9.4	Suffix Array Kactl	19
9.5	Suffix Automaton Mostafa	19
9.6	Zalgo Anany	20
9.7	lexicographically smallest rotation of a string	20
10	Trees	20
10.1	Centroid Decomposition	20
10.2	Dsu On Trees	20
10.3	Heavy Light Decomposition (Along with Euler Tour)	21
10.4	Mo on Trees	21
11	Numerical	21
11.1	Lagrange Polynomial	21
11.2	Polynomials	21
12	Guide	22
12.1	Notes	22
12.2	Assignment Problems	22
12.3	XOR problems	23
12.4	Decompositions	23
12.5	Strings	23
12.6	Trees	23
12.7	Flows	23
12.8	Geometry	23
12.9	Area	23
12.10	Volume	24
12.11	Combinatorics	24
12.12	Graph Theory	24
12.13	Max flow with lower bound	24
12.14	Joseph problem	24

1 Template

1.1 template

```
1 #include <bits/stdc++.h>
2 #define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
3 using namespace std;
4 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5
6 // Kactl defines
7 #define rep(i, a, b) for(int i = a; i < (b); ++i)
8 #define all(x) begin(x), end(x)
9 #define sz(x) (int)(x).size()
10 typedef long long ll;
11 typedef pair<int, int> pii;
12 typedef vector<int> vi;
13 typedef vector<double> vd;
```

2 Combinatorics

2.1 Burnside Lemma

```

1 // |Classes|=sum (k ^C(pi)) / |G|
2 // C(pi) the number of cycles in the permutation pi
3 // |G| the number of permutations

```

2.2 Catalan Numbers

```

1 void init() {
2     catalan[0] = catalan[1] = 1;
3     for (int i=2; i<=n; i++) {
4         catalan[i] = 0;
5         for (int j=0; j < i; j++) {
6             catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
7             if (catalan[i] >= MOD) {
8                 catalan[i] -= MOD;
9             }
10        }
11    }
12 }
13 // 1- Number of correct bracket sequence consisting of n opening and n
14 //    closing brackets.
15 // 2- The number of rooted full binary trees with n+1 leaves (vertices
16 //    are not numbered).
17 // 3- The number of ways to completely parenthesize n+1 factors.
18 // 4- The number of triangulations of a convex polygon with n+2 sides
19 // 5- The number of ways to connect the 2n points on a circle to form
20 //    n disjoint chords.
21 // 6- The number of non-isomorphic full binary trees with n internal
22 //    nodes (i.e. nodes having at least one son).
23 // 7- The number of monotonic lattice paths from point (0,0) to point
24 //    (n,n) in a square lattice of size nxn, which do not pass above the
25 //    main diagonal (i.e. connecting (0,0) to (n,n)).
26 // 8- Number of permutations of length n that can be stack sorted (it
27 //    can be shown that the rearrangement is stack sorted if and only if
28 //    there is no such index i<j<k, such that ak<ai<aj).
29 // 9- The number of non-crossing partitions of a set of n elements.
30 // 10- The number of ways to cover the ladder 1..n using n rectangles
31 //     (The ladder consists of n columns, where ith column has a height i
32 //     ).

```

3 Algebra

3.1 Gray Code

```

1 int g (int n) {
2     return n ^ (n >> 1);
3 }
4 int rev_g (int g) {
5     int n = 0;
6     for (; g; g >>= 1)
7         n ^= g;
8     return n;
9 }
10 int calc(int x, int y) { //2D Gray Code
11     int a = g(x), b = g(y);
12     int res = 0;
13     f(i, 0, LG) {
14         int k1 = (a & (1 << i));
15         int k2 = (b & (1 << i));
16         res |= k1 << (i + 1);
17         res |= k2 << i;
18     }
19     return res;
20 }

```

3.2 Primitive Roots

```

1 int primitive_root (int p) {
2     vector<int> fact;
3     int phi = p - 1, n = phi;
4     for (int i = 2; i * i <= n; ++i)
5         if (n % i == 0) {
6             fact.push_back (i);
7             while (n % i == 0)
8                 n /= i;
9         }
10    if (n > 1)
11        fact.push_back (n);
12    for (int res = 2; res <= p; ++res) {

```

```

14        bool ok = true;
15        for (size_t i = 0; i < fact.size() && ok; ++i)
16            ok &= powmod (res, phi / fact[i], p) != 1;
17        if (ok) return res;
18    }
19    return -1;
20 }

```

3.3 Discrete Logarithm minimum x for which $a^x = b \% m$

```

1 // Returns the smallest x > 0 : a^x = b mod m
2 ll modLog(ll a, ll b, ll m) {
3     ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
4     unordered_map<ll, ll> A;
5     while (j <= n && (e = f = e * a % m) != b % m)
6         A[e * b % m] = j++;
7     if (e == b % m) return j;
8     if ((__gcd(m, e) == (__gcd(m, b)))
9         rep(i, 2, n + 2) if (A.count(e = e * f % m))
10         return n * i - A[e];
11     return -1;
12 }

```

3.4 Discrete Root finds all numbers x such that $x^k = a \% n$

```

1 // This program finds all numbers x such that x^k = a (mod n)
2 vector<int> discrete_root(int n, int k, int a) {
3     if (a == 0)
4         return {0};
5
6     int g = primitive_root(n);
7     // Baby-step giant-step discrete logarithm algorithm
8     int sq = (int) sqrt(n + .0) + 1;
9     vector<pair<int, int>> dec(sq);
10    for (int i = 1; i <= sq; ++i)
11        dec[i - 1] = {powmod(g, i * sq * k % (n - 1), n), i};
12    sort(dec.begin(), dec.end());
13    int any_ans = -1;
14    for (int i = 0; i < sq; ++i) {
15        int my = powmod(g, i * k % (n - 1), n) * a % n;
16        auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0));
17        if (it != dec.end() && it->first == my) {
18            any_ans = it->second * sq - i;
19            break;
20        }
21    }
22    if (any_ans == -1) return {};
23
24    int delta = (n - 1) / __gcd(k, n - 1);
25    vector<int> ans;
26    for (int cur = any_ans % delta; cur < n - 1; cur += delta)
27        ans.push_back(powmod(g, cur, n));
28    sort(ans.begin(), ans.end());
29    return ans;
30 }

```

3.5 Factorial modulo in $p * \log(n)$ (Wilson Theroem)

```

1 int factmod(int n, int p) {
2     vector<int> f(p);
3     f[0] = 1;
4     for (int i = 1; i < p; i++)
5         f[i] = f[i-1] * i % p;
6
7     int res = 1;
8     while (n > 1) {
9         if ((n/p) % 2)
10             res = p - res;
11         res = res * f[n%p] % p;
12         n /= p;
13     }
14     return res;
15 }

```

3.6 Iteration over submasks

```

1 int s = m;
2 while (s > 0) {
3     s = (s-1) & m;
4 }

```

3.7 Totient function

```

1 void phi_1_to_n(int n) {
2     for (int i = 0; i <= n; i++)
3         phi[i] = i;
4     for (int i = 2; i <= n; i++) {
5         if (phi[i] == i) {
6             for (int j = i; j <= n; j += i)
7                 phi[j] -= phi[j] / i;
8         }
9     }
10 }

```

3.8 CRT and EGCD

```

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

```

3.9 FFT

```

1 typedef complex<double> C;
2 void fft(vector<C>& a) {
3     int n = sz(a), L = 31 - __builtin_clz(n);
4     static vector<complex<long double>> R(2, 1);
5     static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
6     for (static int k = 2; k < n; k *= 2) {
7         R.resize(n);
8         rt.resize(n);
9         auto x = polar(1.0L, acos(-1.0L) / k);
10        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i /
11        2];
12    }
13    vi rev(n);
14    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
15    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
16    for (int k = 1; k < n; k *= 2)
17        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
18            C z = rt[j + k] * a[i + j + k]; //
19            a[i + j + k] = a[i + j] - z;
20            a[i + j] += z;
21        }
22    vd conv(const vd& a, const vd& b) {
23        if (a.empty() || b.empty()) return {};
24        vd res(sz(a) + sz(b) - 1);
25        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;

```

```

26    vector<C> in(n), out(n);
27    copy(all(a), begin(in));
28    rep(i, 0, sz(b)) in[i].imag(b[i]);
29    fft(in);
30    for (C& x : in) x *= x;
31    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
32    fft(out);
33    /// rep(i,0,sz(res)) res[i] = (MOD+(ll)round(imag(out[i]) / (4 * n
34    )) % MOD; ///in case of mod
35    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
36    return res;
37 }
38 //Applications
39 //1-All possible sums
40 //2-All possible scalar products
41 // We are given two arrays a[] and b[] of length n.
42 //We have to compute the products of a with every cyclic shift of b.
43 //We generate two new arrays of size 2n: We reverse a and append n
44 //zeros to it.
45 //And we just append b to itself. When we multiply these two arrays as
46 //polynomials,
47 //and look at the coefficients c[n-1], c[n], ..., c[2n-2] of the
48 //product c, we get:
49 //c[k]=sum i+j=k a[i]b[j]
50 //3-Two stripes
51 //We are given two Boolean stripes (cyclic arrays of values 0 and 1) a
52 //and b.
53 //We want to find all ways to attach the first stripe to the second
54 //one,
55 //such that at no position we have a 1 of the first stripe next to a 1
56 //of the second stripe.

```

3.10 FFT with mod

```

1 "FastFourierTransform.cpp"
2 typedef vector<ll> vl;
3 template<int M> vl convMod(const vl &a, const vl &b) {
4     if (a.empty() || b.empty()) return {};
5     vl res(sz(a) + sz(b) - 1);
6     int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
7     vector<C> L(n), R(n), outs(n), outl(n);
8     rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
9     rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
10    fft(L), fft(R);
11    rep(i,0,n) {
12        int j = -i & (n - 1);
13        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
14        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
15    }
16    fft(outl), fft(outs);
17    rep(i,0,sz(res)) {
18        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
19        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
20        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
21    }
22    return res;
23 }

```

3.11 convolutions of AND-XOR-OR

```

1 // The size of a must be a power of two.
2 void FST(vi& a, bool inv) {
3     for (int n = sz(a), step = 1; step < n; step *= 2) {
4         for (int i = 0; i < n; i += 2 * step) rep(j, i, i+step) {
5             int &u = a[j], &v = a[j + step]; tie(u, v) =
6             inv ? pii(v - u, u) : pii(v, u + v); // AND
7             inv ? pii(v, u - v) : pii(u + v, u); // OR
8             pii(u + v, u - v); // XOR
9         }
10    }
11    if (inv) for (int& x : a) x /= sz(a); // XOR only
12 }
13 vi conv(vi a, vi b) {
14     FST(a, 0); FST(b, 0);
15     rep(i,0,sz(a)) a[i] *= b[i];
16     FST(a, 1); return a;
17 }

```

3.12 NTT of KACTL

```

1  const ll mod = (119 << 23) + 1, root = 62; // = 998244353
2  // For p < 2^30 there is a lso e . g . 5 << 25, 7 << 26, 479 << 21
3  // and 483 << 21 (same root) . The l as t two are > 10^9.
4  typedef vector<ll> vl;
5  void ntt(vl &a) {
6      int n = sz(a), L = 31 - __builtin_clz(n);
7      static vl rt(2, 1);
8      for (static int k = 2, s = 2; k < n; k *= 2, s++) {
9          rt.resize(n);
10         ll z[] = {1, modpow(root, mod >> s)};
11         rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
12     }
13     vi rev(n);
14     rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
15     rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
16     for (int k = 1; k < n; k *= 2)
17         for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
18             ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
19             a[i + j + k] = ai - z + (z > ai ? mod : 0);
20             ai += (ai + z >= mod ? z - mod : z);
21         }
22 }
23 vl conv(const vl &a, const vl &b) {
24     if (a.empty() || b.empty()) return {};
25     int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
26     int inv = modpow(n, mod - 2);
27     vl L(a), R(b), out(n);
28     L.resize(n), R.resize(n);
29     ntt(L), ntt(R);
30     rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
31     ntt(out);
32     return {out.begin(), out.begin() + s};
33 }

```

3.13 Fibonacci

```

1  // F(n-1) * F(n+1) - F(n)^2 = (-1)^n
2  // F(n+k) = F(k) * F(n+1) + F(k-1) * F(n)
3  // F(2*n) = F(n) * (F(n+1) + F(n-1))
4  // GCD ( F(m) , F(n) ) = F(GCD(n,m))

```

3.14 Gauss Determinant

```

1  double det(vector<vector<double>>& a) {
2      int n = sz(a); double res = 1;
3      rep(i,0,n) {
4          int b = i;
5          rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
6          if (i != b) swap(a[i], a[b]), res *= -1;
7          res *= a[i][i];
8          if (res == 0) return 0;
9          rep(j,i+1,n) {
10             double v = a[j][i] / a[i][i];
11             if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
12         }
13     }
14     return res;
15 }
16 // for integers
17 const ll mod = 12345;
18 ll det(vector<vector<ll>>& a) {
19     int n = sz(a); ll ans = 1;
20     rep(i,0,n) {
21         rep(j,i+1,n) {
22             while (a[j][i] != 0) { // gcd step
23                 ll t = a[i][i] / a[j][i];
24                 if (t) rep(k,i,n)
25                     a[i][k] = (a[i][k] - a[j][k] * t) % mod;
26                 swap(a[i], a[j]);
27                 ans *= -1;
28             }
29         }
30         ans = ans * a[i][i] % mod;
31         if (!ans) return 0;
32     }
33     return (ans + mod) % mod;
34 }

```

3.15 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     ans.assign (m, 0);
29     for (int i = 0; i < m; ++i)
30         if (where[i] != -1)
31             ans[i] = a[where[i]][m] / a[where[i]][i];
32     for (int i = 0; i < n; ++i) {
33         double sum = 0;
34         for (int j = 0; j < m; ++j)
35             sum += ans[j] * a[i][j];
36         if (abs (sum - a[i][m]) > EPS)
37             return 0;
38     }
39
40     for (int i = 0; i < m; ++i)
41         if (where[i] == -1)
42             return INF;
43     return 1;
44 }
45

```

3.16 Matrix Inverse

```

1  #define ld long double
2  vector < vector<ld> > gauss (vector < vector<ld> > a) {
3
4      int n = (int) a.size();
5      vector<vector<ld> > ans (n, vector<ld> (n, 0));
6
7      for (int i = 0; i < n; i++)
8          ans[i][i] = 1;
9      for (int i = 0; i < n; i++) {
10         for (int j = i + 1; j < n; j++)
11             if (a[j][i] > a[i][i]) {
12                 a[j].swap(a[i]);
13                 ans[j].swap(ans[i]);
14             }
15         ld val = a[i][i];
16         for (int j = 0; j < n; j++) {
17             a[i][j] /= val;
18             ans[i][j] /= val;
19         }
20         for (int j = 0; j < n; j++) {
21             if (j == i) continue;
22             val = a[j][i];
23             for (int k = 0; k < n; k++) {
24                 a[j][k] -= val * a[i][k];
25                 ans[j][k] -= val * ans[i][k];
26             }
27         }
28     }
29     return ans;
30 }

```

4 Data Structures

4.1 UnionFindRollback

```

1 struct RollbackUF {
2     vi e; vector<pli> st;
3     RollbackUF(int n) : e(n, -1) {}
4     int size(int x) { return -e[find(x)]; }
5     int find(int x) { return e[x] < 0 ? x : find(e[x]); }
6     int time() { return sz(st); }
7     void rollback(int t) {
8         for (int i = time(); i --> t;)
9             e[st[i].first] = st[i].second;
10        st.resize(t);
11    }
12    bool join(int a, int b) {
13        a = find(a), b = find(b);
14        if (a == b) return false;
15        if (e[a] > e[b]) swap(a, b);
16        st.push_back({a, e[a]});
17        st.push_back({b, e[b]});
18        e[a] += e[b]; e[b] = a;
19        return true;
20    }
21 };

```

4.2 2D BIT

```

1 void upd(int x, int y, int val) {
2     for(int i = x; i <= n; i += i & -i)
3         for(int j = y; j <= m; j += j & -j)
4             bit[i][j] += val;
5 }
6 int get(int x, int y) {
7     int ans = 0;
8     for(int i = x; i; i -= i & -i)
9         for(int j = y; j; j -= j & -j)
10            ans += bit[i][j];
11 }

```

4.3 2D Sparse table

```

1 const int N = 505, LG = 10;
2 int st[N][N][LG][LG];
3 int a[N][N], lg2[N];
4 int yo(int x1, int y1, int x2, int y2) {
5     x2++;
6     y2++;
7     int a = lg2[x2 - x1], b = lg2[y2 - y1];
8     return max(
9         max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
10        max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b])
11    );
12 }
13 void build(int n, int m) { // 0 indexed
14     for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
15     for (int i = 0; i < n; i++) {
16         for (int j = 0; j < m; j++) {
17             st[i][j][0][0] = a[i][j];
18         }
19     }
20     for (int a = 0; a < LG; a++) {
21         for (int b = 0; b < LG; b++) {
22             if (a + b == 0) continue;
23             for (int i = 0; i + (1 << a) <= n; i++) {
24                 for (int j = 0; j + (1 << b) <= m; j++) {
25                     if (!a) {
26                         st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << b) - 1][a][b - 1]);
27                     } else {
28                         st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << a) - 1][j][a - 1][b]);
29                     }
30                 }
31             }
32         }
33     }
34 }

```

4.4 Mo With Updates

```

1 //O(N^5/3) note that the block size is not a standard size
2 //O(2SQ + N^2 / S + Q * N^2 / S^2) = O(Q * N^(2/3)) if S = n^(2/3)
3 //fact: S = (2 * n * n)^(1/3) give the best complexity
4 const int block_size = 2000;
5 struct Query{
6     int l, r, t, idx;
7     Query(int l, int r, int t, int idx) : l(l), r(r), t(t), idx(idx) {}
8     bool operator < (Query o) const{
9         if(l / block_size != o.l / block_size) return l < o.l;
10        if(r / block_size != o.r / block_size) return r < o.r;
11        return t < o.t;
12    }
13 };
14 int L = 0, R = -1, K = -1;
15 while(L < Q[i].l) del(a[L++]);
16 while(L > Q[i].l) add(a[--L]);
17 while(R < Q[i].r) add(a[++R]);
18 while(R > Q[i].r) del(a[R--]);
19 while(K < Q[i].t) upd(++K);
20 while(K > Q[i].t) err(K--);

```

4.5 Ordered Set

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 #define ordered_set tree<int, null_type, less<int>, rb_tree_tag,
5     tree_order_statistics_node_update>
6 //order_of_key(k): returns the number of elements strictly less than k
7 //find_by_order(k): returns an iterator to the k-th element (0-based)

```

4.6 Persistent Seg Tree

```

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

```

4.7 Treap

```

1 mt19937_64 mrand(chrono::steady_clock::now().time_since_epoch().count
2     ());
3 struct Node {

```

```

4   int key, pri = mrand(), sz = 1;
5   int lz = 0;
6   int idx;
7   array<Node*, 2> c = {NULL, NULL};
8   Node(int key, int idx) : key(key), idx(idx) {}
9 };
10 int getsz(Node* t) {
11     return t ? t->sz : 0;
12 }
13 Node* calc(Node* t) {
14     t->sz = 1 + getsz(t->c[0]) + getsz(t->c[1]);
15     return t;
16 }
17 void prop(Node* cur) {
18     if(!cur || !cur->lz)
19         return;
20     cur->key += cur->lz;
21     if(cur->c[0])
22         cur->c[0]->lz += cur->lz;
23     if(cur->c[1])
24         cur->c[1]->lz += cur->lz;
25     cur->lz = 0;
26 }
27 array<Node*, 2> split(Node* t, int k) {
28     prop(t);
29     if(!t)
30         return {t, t};
31     if(getsz(t->c[0]) >= k) { ///answer is in left node
32         auto ret = split(t->c[0], k);
33         t->c[0] = ret[1];
34         return {ret[0], calc(t)};
35     } else { ///k > t->c[0]
36         auto ret = split(t->c[1], k - 1 - getsz(t->c[0]));
37         t->c[1] = ret[0];
38         return {calc(t), ret[1]};
39     }
40 }
41 Node* merge(Node* u, Node* v) {
42     prop(u);
43     prop(v);
44     if(!u || !v)
45         return u ? u : v;
46     if(u->pri > v->pri) {
47         u->c[1] = merge(u->c[1], v);
48         return calc(u);
49     } else {
50         v->c[0] = merge(u, v->c[0]);
51         return calc(v);
52     }
53 }
54 int cnt(Node* cur, int x) {
55     prop(cur);
56     if(!cur)
57         return 0;
58     if(cur->key <= x)
59         return getsz(cur->c[0]) + 1 + cnt(cur->c[1], x);
60     return cnt(cur->c[0], x);
61 }
62 Node* ins(Node* root, int val, int idx, int pos) {
63     auto splitted = split(root, pos);
64     root = merge(splitted[0], new Node(val, idx));
65     return merge(root, splitted[1]);
66 }

```

4.8 Wavelet Tree

```

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
35             left
36         int rb = b[r]; ///amt of nos in first (r) nos that go in left
37         if (k <= inleft)
38             return this->l->kth(lb + 1, rb, k);
39         return this->r->kth(l - lb, r - rb, k - inleft);
40     }
41
42     ///count of nos in [l, r] Less than or equal to k
43     int LTE(int l, int r, int k) {
44         if (l > r or k < lo)
45             return 0;
46         if (hi <= k)
47             return r - l + 1;
48         int lb = b[l - 1], rb = b[r];
49         return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r -
50             rb, k);
51     }
52
53     ///count of nos in [l, r] equal to k
54     int count(int l, int r, int k) {
55         if (l > r or k < lo or k > hi)
56             return 0;
57         if (lo == hi)
58             return r - l + 1;
59         int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;
60         if (k <= mid)
61             return this->l->count(lb + 1, rb, k);
62         return this->r->count(l - lb, r - rb, k);
63     }
64 };

```

4.9 SparseTable

```

1 int S[N];
2 for(int i = 2; i < N; i++) S[i] = S[i >> 1] + 1;
3 for (int i = 1; i <= K; i++)
4     for (int j = 0; j + (1 << i) <= N; j++)
5         st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
6
7 int query(int l, int r) {
8     int k = S[r - l + 1];
9     return mrg(st[k][l], st[k][r - (1 << k) + 1]);
10 }

```

5 DP

5.1 Dynamic Connectivity with SegTree

```

1 #define f(i, a, b) for(int i = a; i < b; i++)
2 #define all(a) a.begin(), a.end()
3 #define sz(x) (int)(x).size()
4 typedef long long ll;
5 const int N = 1e5 + 5;
6
7 struct PT {
8     ll x, y;
9     PT() {}
10     PT(ll a, ll b) : x(a), y(b) {}
11     PT operator-(const PT &o) { return PT(x - o.x, y - o.y); }
12     bool operator<(const PT &o) const { return make_pair(x, y) <
13         make_pair(o.x, o.y); }
14 };

```



```

14 ll cross(PT x, PT y) {
15     return x.x * y.y - x.y * y.x;
16 }
17 PT val[300005];
18 bool in[300005];
19 ll qr[300005];
20 bool ask[300005];
21 ll ans[N];
22 vector<PT> t[300005 * 4]; //segment tree holding points to queries
23 void update(int node, int s, int e, int l, int r, PT x) {
24     if (r < s || e < l) return;
25     if (l <= s && e <= r) { //add this point to maximize it with
        queries in this range
        t[node].push_back(x);
        return;
26     }
27     int md = (s + e) >> 1;
28     update(node << 1, s, md, l, r, x);
29     update(node << 1 | 1, md + 1, e, l, r, x);
30 }
31 vector<PT> stk;
32 inline void addPts(vector<PT> v) {
33     stk.clear(); //reset the data structure you are using
34     sort(all(v));
35     //build upper envelope
36     for (int i = 0; i < v.size(); i++) {
37         while (sz(stk) > 1 && cross(v[i] - stk.back(), stk.back() -
38             stk[stk.size() - 2]) <= 0)
39             stk.pop_back();
40         stk.push_back(v[i]);
41     }
42 }
43 inline ll calc(PT x, ll val) {
44     return x.x * val + x.y;
45 }
46 ll query(ll x) {
47     if (stk.empty())
48         return LLONG_MIN;
49     int lo = 0, hi = stk.size() - 1;
50     while (lo + 10 < hi) {
51         int md = lo + (hi - lo) / 2;
52         if (calc(stk[md + 1], x) > calc(stk[md], x))
53             lo = md + 1;
54         else
55             hi = md;
56     }
57     ll ans = LLONG_MIN;
58     for (int i = lo; i <= hi; i++)
59         ans = max(ans, calc(stk[i], x));
60     return ans;
61 }
62 void solve(int node, int s, int e) { //Solve queries
63     addPts(t[node]); //note that there is no need to add/delete
64     just build for t[node]
65     f(i, s, e + 1) {
66         if (ask[i]) {
67             ans[i] = max(ans[i], query(qr[i]));
68         }
69     }
70     if (s == e) return;
71     int md = (s + e) >> 1;
72     solve(node << 1, s, md);
73     solve(node << 1 | 1, md + 1, e);
74 }
75 void doWork() {
76     int n;
77     cin >> n;
78     stk.reserve(n);
79     f(i, 1, n + 1) {
80         int tp;
81         cin >> tp;
82         if (tp == 1) { //Add Query
83             int x, y;
84             cin >> x >> y;
85             val[i] = PT(x, y);
86             in[i] = 1;
87         } else if (tp == 2) { //Delete Query
88             int x;
89             cin >> x;
90             if (in[x]) update(1, 1, n, x, i - 1, val[x]);
91         }

```

```

92         in[x] = 0;
93     } else {
94         cin >> qr[i];
95         ask[i] = true;
96     }
97 }
98 f(i, 1, n + 1) //Finalize Query
99 if (in[i])
100     update(1, 1, n, i, n, val[i]);
101
102 f(i, 1, n + 1) ans[i] = LLONG_MIN;
103 solve(1, 1, n);
104 f(i, 1, n + 1) if (ask[i]) {
105     if (ans[i] == LLONG_MIN)
106         cout << "EMPTY SET\n";
107     else
108         cout << ans[i] << '\n';
109 }
110 }
111 }

```

5.2 CHT Line Container

```

1 struct Line {
2     mutable ll m, b, p;
3     bool operator<(const Line &o) const { return m < o.m; }
4     bool operator<(ll x) const { return p < x; }
5 };
6 struct LineContainer : multiset<Line, less<>> {
7     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
8     static const ll inf = LLONG_MAX;
9     ll div(ll db, ll dm) { // floored division
10         return db / dm - ((db ^ dm) < 0 && db % dm);
11     }
12     bool isect(iterator x, iterator y) {
13         if (y == end()) {
14             x->p = inf;
15             return false;
16         }
17         if (x->m == y->m)
18             x->p = x->b > y->b ? inf : -inf;
19         else
20             x->p = div(y->b - x->b, x->m - y->m);
21         return x->p >= y->p;
22     }
23     void add(ll m, ll b) {
24         auto z = insert({m, b, 0}), y = z++, x = y;
25         while (isect(y, z))
26             z = erase(z);
27         if (x != begin() && isect(--x, y))
28             isect(x, y = erase(y));
29         while ((y = x) != begin() && (--x)->p >= y->p)
30             isect(x, erase(y));
31     }
32     ll query(ll x) {
33         assert(!empty());
34         auto l = *lower_bound(x);
35         return l.m * x + l.b;
36     }
37 };

```

6 Geometry

6.1 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 -
27             center);
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 -
37             1]) <= 0)
38             ans.pop_back(), sz--;
39         ans.push_back(i);
40     }
41     ans.pop_back();
42     return ans;

```

6.2 Geometry Template

```

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

```

```

49     point p = ProjectPointLine(a, b, point(0, 0)); // project point
50         from center to the Line
51     if(dot(p, p) > r * r)
52         return {};
53     double len = sqrt(r * r - dot(p, p));
54     if(len < EPS)
55         return {center + p};
56     point d = (a - b) / abs(a - b);
57     return {center + p + d * len, center + p - d * len};
58 }
59
60 vector<point> CircleCircleIntersect(point c1, ld r1, point c2, ld r2)
61 {
62     if (r1 < r2) {
63         swap(r1, r2);
64         swap(c1, c2);
65     }
66     ld d = abs(c2 - c1); // distance between c1, c2
67     if (d > r1 + r2 || d < r1 - r2 || d < EPS) // zero or infinite
68         solutions
69         return {};
70     ld angle = acos(min((d * d + r1 * r1 - r2 * r2) / (2 * r1 * d), (
71         ld) 1.0));
72     point p = (c2 - c1) / d * r1;
73     if (angle < EPS)
74         return {c1 + p};
75     return {c1 + RotateCCW(p, angle), c1 + RotateCCW(p, -angle)};
76 }
77 point circumcircle(point p1, point p2, point p3) {
78     return LineLineIntersect((p1 + p2) / 2, (p1 - p2).prep(),
79         (p1 + p3) / 2, (p1 - p3).prep());
80 }
81 //I : number points with integer coordinates lying strictly inside the
82     polygon.
83 //B : number of points lying on polygon sides by B.
84 //Area = I + B/2 - 1

```

6.3 Half Plane Intersection

```

1 // Redefine epsilon and infinity as necessary. Be mindful of precision
2     errors.
3 #define ld long double
4 const ld eps = 1e-9, inf = 1e9;
5
6 // Basic point/vector struct.
7 struct Point {
8     ld x, y;
9     explicit Point(ld x = 0, ld y = 0) : x(x), y(y) {}
10
11     // Addition, subtraction, multiply by constant, cross product.
12     friend Point operator + (const Point& p, const Point& q) {
13         return Point(p.x + q.x, p.y + q.y);
14     }
15     friend Point operator - (const Point& p, const Point& q) {
16         return Point(p.x - q.x, p.y - q.y);
17     }
18     friend Point operator * (const Point& p, const ld& k) {
19         return Point(p.x * k, p.y * k);
20     }
21     friend ld cross(const Point& p, const Point& q) {
22         return p.x * q.y - p.y * q.x;
23     }
24 };
25
26 // Basic half-plane struct.
27 struct Halfplane {
28     // 'p' is a passing point of the line and 'pq' is the direction
29     // vector of the line.
30     Point p, pq;
31     ld angle;
32
33     Halfplane() {}
34     Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
35         angle = atan2l(pq.y, pq.x);
36     }
37     // Check if point 'r' is outside this half-plane.
38     // Every half-plane allows the region to the LEFT of its line.

```



```

38 bool out(const Point& r) {
39     return cross(pq, r - p) < -eps;
40 }
41 // Comparator for sorting.
42 // If the angle of both half-planes is equal, the leftmost one
43 // should go first.
44 bool operator < (const Halfplane& e) const {
45     if (fabs(angle - e.angle) < eps) return cross(pq, e.p - p) <
46         0;
47     return angle < e.angle;
48 }
49 // We use equal comparator for std::unique to easily remove
50 // parallel half-planes.
51 bool operator == (const Halfplane& e) const {
52     return fabs(angle - e.angle) < eps;
53 }
54 // Intersection point of the lines of two half-planes. It is
55 // assumed they're never parallel.
56 friend Point inter(const Halfplane& s, const Halfplane& t) {
57     ld alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
58     return s.p + (s.pq * alpha);
59 }
60 };
61 // Actual algorithm
62 vector<Point> hp_intersect(vector<Halfplane>& H) {
63     Point box[4] = { // Bounding box in CCW order
64         Point(-inf, -inf),
65         Point(inf, -inf),
66         Point(inf, inf),
67         Point(-inf, inf),
68     };
69     for(int i = 0; i < 4; i++) { // Add bounding box half-planes.
70         Halfplane aux(box[i], box[(i+1) % 4]);
71         H.push_back(aux);
72     }
73     // Sort and remove duplicates
74     sort(H.begin(), H.end());
75     H.erase(unique(H.begin(), H.end()), H.end());
76     deque<Halfplane> dq;
77     int len = 0;
78     for(int i = 0; i < int(H.size()); i++) {
79         // Remove from the back of the deque while last half-plane is
80         // redundant
81         while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
82             dq.pop_back();
83             --len;
84         }
85         // Remove from the front of the deque while first half-plane
86         // is redundant
87         while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
88             dq.pop_front();
89             --len;
90         }
91         // Add new half-plane
92         dq.push_back(H[i]);
93         ++len;
94     }
95     // Final cleanup: Check half-planes at the front against the back
96     // and vice-versa
97     while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
98         dq.pop_back();
99         --len;
100     }
101     while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
102         dq.pop_front();
103         --len;
104     }
105     // Report empty intersection if necessary
106     if (len < 3) return vector<Point>();
107     // Reconstruct the convex polygon from the remaining half-planes.
108     vector<Point> ret(len);
109     for(int i = 0; i+1 < len; i++) {
110         ret[i] = inter(dq[i], dq[i+1]);
111     }
112     ret.back() = inter(dq[len-1], dq[0]);
113     return ret;
114 }

```

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

```

```

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(
85             nxt);
86         if (nxt != s.end() && intersect(*nxt, a[id]))
87             return make_pair(nxt->id, id);
88         if (prv != s.end() && intersect(*prv, a[id]))
89             return make_pair(prv->id, id);
90         where[id] = s.insert(nxt, a[id]);
91     } else {
92         set<seg>::iterator nxt = next(where[id]), prv = prev(where
93             [id]);
94         if (nxt != s.end() && prv != s.end() && intersect(*nxt, *
95             prv))
96             return make_pair(prv->id, nxt->id);
97         s.erase(where[id]);
98     }
99     return make_pair(-1, -1);

```

6.5 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 empty;
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
22         (type) {}
23 };
24 bool operator < (const Event&A, const Event&B) {
25     //if(A.x != B.x)
26     return A.x < B.x;
27     //if(A.y1 != B.y1) return A.y1 < B.y1;
28     //if(A.y2 != B.y2()) A.y2 < B.y2;
29 }
30 const int MX = (1 << 17);
31 struct Node {
32     int prob, sum, ans;
33     Node() {}
34     Node(int prob, int sum, int ans): prob(prob), sum(sum), ans(ans)
35     {}
36 };
37 Node tree[MX * 4];
38 int interval[MX];
39 void build(int x, int a, int b) {
40     tree[x] = Node(0, 0, 0);
41     if(a == b) {
42         tree[x].sum += interval[a];
43         return;
44     }
45     build(x * 2, a, (a + b) / 2);
46     build(x * 2 + 1, (a + b) / 2 + 1, b);
47     tree[x].sum = tree[x * 2].sum + tree[x * 2 + 1].sum;
48 }
49 int ask(int x) {
50     if(tree[x].prob)
51         return tree[x].sum;
52     return tree[x].ans;
53 }
54 int st, en, V;
55 void update(int x, int a, int b) {
56     if(st > b || en < a)
57         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::empty = Rectangle();
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) -
98             sorted.begin() + 1;
99         en = lower_bound(sorted.begin(), sorted.end(), sweep[j].y2) -
100             sorted.begin();
101         update(1, 1, sz_ - 1);
102     }
103 }
104 int main() {
105     // freopen("in.in", "r", stdin);
106     int n;
107     scanf("%d", &n);
108     for(int j = 1; j <= n; j++) {
109         int a, b, c, d;
110         scanf("%d %d %d %d", &a, &b, &c, &d);
111         Rect.push_back(Rectangle(a, b, c, d));
112     }
113     compressncalc();
114     Sweep();
115     cout << ans << endl;

```

7 Graphs

7.1 2 SAD

```

1  /**
2   * Description: Calculates a valid assignment to boolean variables a,
3   *               b, c, ... to a 2-SAT problem, so that an expression of the type $(
4   *               a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ becomes true, or
5   *               reports that it is unsatisfiable.
6   * Negated variables are represented by bit-inversions (\texttt{\tilde
7   *               {x}}).
8   * Usage:
9   * TwoSat ts(number of boolean variables);
10  * ts.either(0, \tilde{3}); // Var 0 is true or var 3 is false
11  * ts.setValue(2); // Var 2 is true
12  * ts.atMostOne({0, \tilde{1}, 2}); // <= 1 of vars 0, \tilde{1} and 2 are
13  *               true

```

```

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

```

7.2 Articulation Point

```

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 int main(){
21     // Input
22     for(int i = 1; i <= n; i++){
23         if(dfsn[i] == 0) dfs(i, 0);
24     }
25     int c = 0;
26     for(int i = 1; i <= n; i++){
27         if(ar_point[i]) c++;
28     }
29     cout << c << '\n';
30 }

```

7.3 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 void dfs(int node, int par){
11     dfsn[node] = low[node] = ++timer;
12     st.push(node);
13     for(auto i: adj[node]){
14         if(i == par) continue;
15         if(dfsn[i] == 0){
16             dfs(i, node);
17             low[node] = min(low[node], low[i]);
18         }
19         else low[node] = min(low[node], dfsn[i]);
20     }
21     if(dfsn[node] == low[node]){
22         cnt++;
23         while(1){
24             int cur = st.top();
25             st.pop();
26             comp_id[cur] = cnt;
27             if(cur == node) break;
28         }
29     }
30 }
31
32 void dfs2(int node, int par){
33     kam[node] = 0;
34     int mx = 0, second_mx = 0;
35     for(auto i: bridge_tree[node]){
36         if(i == par) continue;
37         dfs2(i, node);
38         kam[node] = max(kam[node], 1 + kam[i]);
39         if(kam[i] > mx){
40             second_mx = mx;
41             mx = kam[i];
42         }
43         else second_mx = max(second_mx, kam[i]);
44     }
45     ans = max(ans, kam[node]);
46     if(second_mx) ans = max(ans, 2 + mx + second_mx);
47 }
48
49 int main(){
50     ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
51     int n, m;
52     cin >> n >> m;
53     while(m--){
54         int u, v;
55         cin >> u >> v;
56         adj[u].push_back(v);
57         adj[v].push_back(u);
58     }
59     dfs(1, 0);
60     for(int i = 1; i <= n; i++){
61         for(auto j: adj[i]){
62             if(comp_id[i] != comp_id[j]){
63                 bridge_tree[comp_id[i]].push_back(comp_id[j]);
64             }
65         }
66     }
67     dfs2(1, 0);
68 }

```

```

69     cout << ans;
70
71     return 0;
72 }

```

7.4 Dinic With Scalling

```

1  ///O(ElgFlow) on Bipratite Graphs and O(EVlgFlow) on other graphs (I
2  think)
3  struct Dinic {
4      #define vi vector<int>
5      #define rep(i,a,b) f(i,a,b)
6      struct Edge {
7          int to, rev;
8          ll c, oc;
9          int id;
10         ll flow() { return max(oc - c, 0LL); } // if you need flows
11     };
12     vi lvl, ptr, q;
13     vector<vector<Edge>> adj;
14     Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
15     void addEdge(int a, int b, ll c, int id, ll rcap = 0) {
16         adj[a].push_back({b, sz(adj[b]), c, id});
17         adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap, id});
18     }
19     ll dfs(int v, int t, ll f) {
20         if (v == t || !f) return f;
21         for (int& i = ptr[v]; i < sz(adj[v]); i++) {
22             Edge& e = adj[v][i];
23             if (lvl[e.to] == lvl[v] + 1)
24                 if (ll p = dfs(e.to, t, min(f, e.c))) {
25                     e.c -= p, adj[e.to][e.rev].c += p;
26                     return p;
27                 }
28         }
29         return 0;
30     }
31     ll calc(int s, int t) {
32         ll flow = 0; q[0] = s;
33         rep(L,0,31) do { // 'int L=30' maybe faster for random data
34             lvl = ptr = vi(sz(q));
35             int qi = 0, qe = lvl[s] = 1;
36             while (qi < qe && !lvl[t]) {
37                 int v = q[qi++];
38                 for (Edge e : adj[v])
39                     if (!lvl[e.to] && e.c >> (30 - L))
40                         q[qi++] = e.to, lvl[e.to] = lvl[v] + 1;
41             }
42             while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
43             while (lvl[t]);
44             return flow;
45         }
46         bool leftOfMinCut(int a) { return lvl[a] != 0; }

```

7.5 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/
6   * Description: Given a list of edges representing an undirected flow
7   * graph,
8   * returns edges of the Gomory-Hu tree. The max flow between any pair
9   * of
10  * vertices is given by minimum edge weight along the Gomory-Hu tree
11  * path.
12  * Time:  $\mathcal{O}(V)$  Flow Computations
13  * Status: Tested on CERC 2015 J, stress-tested
14  * Details: The implementation used here is not actually the original
15  * Gomory-Hu, but Gusfield's simplified version: "Very simple methods
16  * for all
17  * pairs network flow analysis". PushRelabel is used here, but any
18  * flow
19  * implementation that supports 'leftOfMinCut' also works.
20  */
21 #pragma once
22 #include "PushRelabel.h"

```

```

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 }

```

7.6 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
7   * a list
8   * of neighbors of the left partition, and $btoa$ should be a vector
9   * full of
10  * -1's of the same size as the right partition. Returns the size of
11  * the matching. $btoa[i]$ will be the match for vertex $i$ on the
12  * right side,
13  * or $-1$ if it's not matched.
14  * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
15  * Time:  $\mathcal{O}(\sqrt{V}E)$ 
16  * Status: stress-tested by MinimumVertexCover, and tested on
17  * oldkattis.adkbipmatch and SPOJ:MATCHING
18  */
19 #pragma once
20 bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
21     if (A[a] != L) return 0;
22     A[a] = -1;
23     for (int b : g[a]) if (B[b] == L + 1) {
24         B[b] = 0;
25         if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
26             return btoa[b] = a, 1;
27     }
28     return 0;
29 }
30 int hopcroftKarp(vector<vi>& g, vi& btoa) {
31     int res = 0;
32     vi A(g.size(), -1), B(btoa.size(), -1), cur, next;
33     for (;;) {
34         fill(all(A), 0);
35         fill(all(B), 0);
36         /// Find the starting nodes for BFS (i.e. layer 0).
37         cur.clear();
38         for (int a : btoa) if (a != -1) A[a] = -1;
39         rep(a,0,sz(g)) if (A[a] == 0) cur.push_back(a);
40         /// Find all layers using bfs.
41         for (int lay = 1; lay++) {
42             bool islast = 0;
43             next.clear();
44             for (int a : cur) for (int b : g[a]) {
45                 if (btoa[b] == -1) {
46                     B[b] = lay;
47                     islast = 1;
48                 }
49                 else if (btoa[b] != a && !B[b]) {
50                     B[b] = lay;
51                     next.push_back(btoa[b]);
52                 }
53             }
54             if (islast) break;
55             if (next.empty()) return res;
56             for (int a : next) A[a] = lay;
57             cur.swap(next);
58         }
59         /// Use DFS to scan for augmenting paths.
60         rep(a,0,sz(g))
61             res += dfs(a, 0, g, btoa, A, B);

```

7.7 Hungarian

```

1  /*
2   Notes:
3   note that n must be <= m
4   so in case in 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 algorithm 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
17     +1){}
18     void set(int x, int y, ll v){a[x+1][y+1]=v;}
19     ll assign(){
20         for(int i = 1; i <= n; i++){
21             int j0=0;p[0]=i;
22             vector<ll> minv(m+1,INF);
23             vector<char> used(m+1,false);
24             do {
25                 used[j0]=true;
26                 int i0=p[j0],j1,ll delta=INF;
27                 for(int j = 1; j <= m; j++){if(!used[j]){
28                     ll cur=a[i0][j]-u[i0]-v[j];
29                     if(cur<minv[j])minv[j]=cur,way[j]=j0;
30                     if(minv[j]<delta)delta=minv[j],j1=j;
31                 }
32                 for(int j = 0; j <= m; j++){
33                     if(used[j])u[p[j]]+=delta,v[j]-=delta;
34                     else minv[j]-=delta;
35                 }
36                 j0=j1;
37             } while(p[j0]);
38             do {
39                 int j1=way[j0];p[j0]=p[j1];j0=j1;
40             } while(j0);
41         }
42         return -v[0];
43     }
44     vector<int> restoreAnswer() { ///run it after assign
45         vector<int> ans (n+1);
46         for (int j=1; j<=m; ++j)
47             ans[p[j]] = j;
48         return ans;
49     }
50 };

```

7.8 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 #define FOR(i,a,b) for(int i = a; i < b; i++)
11 #define pb push_back
12
13 const int N = 1e5+5;
14
15 vector<vector<int>>g, rg;
16 vector<vector<int>>go;
17 bitset<N>vis;
18 vector<vector<int>>adj;
19 stack<int>stk;
20 int n, m, cmp[N];
21 void add_edge(int u, int v){
22     g[u].push_back(v);
23     rg[v].push_back(u);
24 }
25 void dfs(int u){
26     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     return c;
50 }
51 }

```

7.9 Manhattan MST

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N = 2e5 + 9;
5
6  int n;
7  vector<pair<int, int>> g[N];
8  struct PT {
9      int x, y, id;
10     bool operator < (const PT &p) const {
11         return x == p.x ? y < p.y : x < p.x;
12     }
13 } p[N];
14 struct node {
15     int val, id;
16 } t[N];
17 struct DSU {
18     int p[N];
19     void init(int n) { for (int i = 1; i <= n; i++) p[i] = i; }
20     int find(int u) { return p[u] == u ? u : p[u] = find(p[u]); }
21     void merge(int u, int v) { p[find(u)] = find(v); }
22 } dsu;
23 struct edge {
24     int u, v, w;
25     bool operator < (const edge &p) const { return w < p.w; }
26 };
27 vector<edge> edges;
28 int query(int x) {
29     int r = 2e9 + 10, id = -1;
30     for (; x <= n; x += (x & -x)) if (t[x].val < r) r = t[x].val, id = t[x].id;
31     return id;
32 }
33 void modify(int x, int w, int id) {
34     for (; x > 0; x -= (x & -x)) if (t[x].val > w) t[x].val = w, t[x].id = id;
35 }
36 int dist(PT &a, PT &b) {
37     return abs(a.x - b.x) + abs(a.y - b.y);
38 }
39 void add(int u, int v, int w) {
40     edges.push_back({u, v, w});
41 }
42 long long Kruskal() {
43     dsu.init(n);
44     sort(edges.begin(), edges.end());
45     long long ans = 0;
46     for (edge e : edges) {
47         int u = e.u, v = e.v, w = e.w;
48         if (dsu.find(u) != dsu.find(v)) {
49             ans += w;
50             g[u].push_back({v, w});
51             //g[v].push_back({u, w});
52             dsu.merge(u, v);
53         }
54     }

```

```

55     return ans;
56 }
57 void Manhattan() {
58     for (int i = 1; i <= n; ++i) p[i].id = i;
59     for (int dir = 1; dir <= 4; ++dir) {
60         if (dir == 2 || dir == 4) {
61             for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
62         }
63         else if (dir == 3) {
64             for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
65         }
66         sort(p + 1, p + 1 + n);
67         vector<int> v;
68         static int a[N];
69         for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x, v.push_back(a[i]);
70         sort(v.begin(), v.end());
71         v.erase(unique(v.begin(), v.end()), v.end());
72         for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;
73         for (int i = 1; i <= n; ++i) t[i].val = 2e9 + 10, t[i].id = -1;
74         for (int i = n; i >= 1; --i) {
75             int pos = query(a[i]);
76             if (pos != -1) add(p[i].id, p[pos].id, dist(p[i], p[pos]));
77             modify(a[i], p[i].x + p[i].y, i);
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 }

```

7.10 Maximum Clique

```

1  ///Complexity  $O(3^{N/3})$  i.e works for 50
2  ///you can change it to maximum independent set by flipping the edges
3  0->1, 1->0
4  ///if you want to extract the nodes they are 1-bits in R
5  int g[60][60];
6  int res;
7  long long edges[60];
8  void BronKerbosch(int n, long long R, long long P, long long X) {
9      if (P == 0LL && X == 0LL) { //here we will find all possible maximal
10         cliques (not maximum) i.e. there is no node which can be
11         included in this set
12         int t = __builtin_popcountll(R);
13         res = max(res, t);
14         return;
15     }
16     int u = 0;
17     while (!(1LL << u) & (P | X)) u++;
18     for (int v = 0; v < n; ++v) {
19         if (((1LL << v) & P) && !(1LL << v) & edges[u])) {
20             BronKerbosch(n, R | (1LL << v), P & edges[v], X & edges[v]);
21             P |= (1LL << v);
22             X |= (1LL << v);
23         }
24     }
25 }
26 int max_clique (int n) {
27     res = 0;
28     for (int i = 1; i <= n; ++i) {
29         edges[i - 1] = 0;
30         for (int j = 1; j <= n; ++j) if (g[i][j]) edges[i - 1] |= (1LL << (j - 1));
31     }
32     BronKerbosch(n, 0, (1LL << n) - 1, 0);
33     return res;
34 }

```

7.11 MCMF

```

1  /*
2  Notes:
3  make sure you notice the #define int ll
4  focus on the data types of the max flow everythingn inside is
5  integer
6  addEdge(u,v,cap,cost)
7  note that for min cost max flow the cost is sum of cost * flow
8  over all edges
9  */
10 struct Edge {
11     int to;
12     int cost;
13     int cap, flow, backEdge;
14 };
15 struct MCMF {
16     const int inf = 1000000010;
17     int n;
18     vector<vector<Edge>> g;
19
20     MCMF(int _n) {
21         n = _n + 1;
22         g.resize(n);
23     }
24
25     void addEdge(int u, int v, int cap, int cost) {
26         Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
27         Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
28         g[u].push_back(e1);
29         g[v].push_back(e2);
30     }
31
32     pair<int, int> minCostMaxFlow(int s, int t) {
33         int flow = 0;
34         int cost = 0;
35         vector<int> state(n), from(n), from_edge(n);
36         vector<int> d(n);
37         deque<int> q;
38         while (true) {
39             for (int i = 0; i < n; ++i)
40                 state[i] = 2, d[i] = inf, from[i] = -1;
41             state[s] = 1;
42             q.clear();
43             q.push_back(s);
44             d[s] = 0;
45             while (!q.empty()) {
46                 int v = q.front();
47                 q.pop_front();
48                 state[v] = 0;
49                 for (int i = 0; i < (int) g[v].size(); ++i) {
50                     Edge e = g[v][i];
51                     if (e.flow >= e.cap || (d[e.to] <= d[v] + e.cost))
52                         continue;
53                     int to = e.to;
54                     d[to] = d[v] + e.cost;
55                     from[to] = v;
56                     from_edge[to] = i;
57                     if (state[to] == 1) continue;
58                     if (!state[to] || (!q.empty() && d[q.front()] > d[to]))
59                         q.push_front(to);
60                     else q.push_back(to);
61                     state[to] = 1;
62                 }
63             }
64             if (d[t] == inf) break;
65             int it = t, addflow = inf;
66             while (it != s) {
67                 addflow = min(addflow,
68                     g[from[it]][from_edge[it]].cap
69                     - g[from[it]][from_edge[it]].flow);
70                 it = from[it];
71             }
72             it = t;
73             while (it != s) {
74                 g[from[it]][from_edge[it]].flow += addflow;
75                 g[it][g[from[it]][from_edge[it]].backEdge].flow -= addflow;
76                 cost += g[from[it]][from_edge[it]].cost * addflow;
77                 it = from[it];
78             }
79         }
80     }
81 }

```



```

79         }
80         flow += addflow;
81     }
82     return {cost, flow};
83 }
84 };

```

7.12 Minimum Arboscene in a Graph

```

1  const int maxn = 2510, maxm = 7000000;
2  const ll maxint = 0x3f3f3f3f3f3f3fLL;
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 arboscene(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 }

```

7.13 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         this->n = n;
11         this->g = g;
12         color = m = vector<int>(n, -1);
13         seen = vector<int>(n, 0);
14         makeBipartite();
15     }
16 }

```

```

17 void dfsBipartite(int node, int col) {
18     if (color[node] != -1) {
19         assert(color[node] == col); /* MSH BIPARTITE YA BASHMOHANDES */
20         return;
21     }
22     color[node] = col;
23     comp[col].push_back(node);
24     for (int i = 0; i < int(g[node].size()); i++)
25         dfsBipartite(g[node][i], 1 - col);
26 }
27
28 void makeBipartite() {
29     for (int i = 0; i < n; i++)
30         if (color[i] == -1)
31             dfsBipartite(i, 0);
32 }
33
34 // match a node
35 bool dfs(int node) {
36     random_shuffle(g[node].begin(), g[node].end());
37     for (int i = 0; i < g[node].size(); i++) {
38         int child = g[node][i];
39         if (m[child] == -1) {
40             m[node] = child;
41             m[child] = node;
42             return true;
43         }
44         if (seen[child] == id)
45             continue;
46         seen[child] = id;
47         int enemy = m[child];
48         m[node] = child;
49         m[child] = node;
50         m[enemy] = -1;
51         if (dfs(enemy))
52             return true;
53         m[node] = -1;
54         m[child] = enemy;
55         m[enemy] = child;
56     }
57     return false;
58 }
59
60 void makeMatching() {
61     for (int j = 0; j < 5; j++)
62         random_shuffle(comp[0].begin(), comp[0].end(), myrandom);
63     for (int i = 0; i < int(comp[0].size()); i++) {
64         id++;
65         if (m[comp[0][i]] == -1)
66             dfs(comp[0][i]);
67     }
68 }
69
70 void recurse(int node, int x, vector<int> &minCover, vector<int> &done) {
71     if (m[node] != -1)
72         return;
73     if (done[node]) return;
74     done[node] = 1;
75     for (int i = 0; i < int(g[node].size()); i++) {
76         int child = g[node][i];
77         int newnode = m[child];
78         if (done[child]) continue;
79         if (newnode == -1) {
80             continue;
81         }
82         done[child] = 2;
83         minCover.push_back(child);
84         m[newnode] = -1;
85         recurse(newnode, x, minCover, done);
86     }
87 }
88
89 vector<int> getAnswer() {
90     vector<int> minCover, maxIndep;
91     vector<int> done(n, 0);
92     makeMatching();
93     for (int x = 0; x < 2; x++)
94         for (int i = 0; i < int(comp[x].size()); i++) {
95             int node = comp[x][i];

```

```

97         if (m[node] == -1)
98             recurse(node, x, minCover, done);
99     }
100
101     for (int i = 0; i < int(comp[0].size()); i++)
102         if (!done[comp[0][i]]) {
103             minCover.push_back(comp[0][i]);
104         }
105     return minCover;
106 }
107 };

```

7.14 Prufer Code

```

1  const int N = 3e5 + 9;
2  /*
3  prufer code is a sequence of length n-2 to uniquely determine a
4  labeled tree with n vertices
5  Each time take the leaf with the lowest number and add the node number
6  the leaf is connected to
7  the sequence and remove the leaf. Then break the algo after n-2
8  iterations
9  */
10 //0-indexed
11 int n;
12 vector<int> g[N];
13 int parent[N], degree[N];
14
15 void dfs (int v) {
16     for (size_t i = 0; i < g[v].size(); ++i) {
17         int to = g[v][i];
18         if (to != parent[v]) {
19             parent[to] = v;
20             dfs (to);
21         }
22     }
23 }
24
25 vector<int> prufer_code() {
26     parent[n - 1] = -1;
27     dfs (n - 1);
28     int ptr = -1;
29     for (int i = 0; i < n; ++i) {
30         degree[i] = (int) g[i].size();
31         if (degree[i] == 1 && ptr == -1) ptr = i;
32     }
33     vector<int> result;
34     int leaf = ptr;
35     for (int iter = 0; iter < n - 2; ++iter) {
36         int next = parent[leaf];
37         result.push_back (next);
38         --degree[next];
39         if (degree[next] == 1 && next < ptr) leaf = next;
40         else {
41             ++ptr;
42             while (ptr < n && degree[ptr] != 1) ++ptr;
43             leaf = ptr;
44         }
45     }
46     return result;
47 }
48
49 vector < pair<int, int> > prufer_to_tree(const vector<int> &
50     prufer_code) {
51     int n = (int) prufer_code.size() + 2;
52     vector<int> degree (n, 1);
53     for (int i = 0; i < n - 2; ++i) ++degree[prufer_code[i]];
54
55     int ptr = 0;
56     while (ptr < n && degree[ptr] != 1) ++ptr;
57     int leaf = ptr;
58     vector < pair<int, int> > result;
59     for (int i = 0; i < n - 2; ++i) {
60         int v = prufer_code[i];
61         result.push_back (make_pair (leaf, v));
62         --degree[leaf];
63         if (--degree[v] == 1 && v < ptr) leaf = v;
64         else {
65             ++ptr;
66             while (ptr < n && degree[ptr] != 1) ++ptr;
67             leaf = ptr;
68         }
69     }
70 }

```

```

65     for (int v = 0; v < n - 1; ++v) if (degree[v] == 1) result.push_back
66         (make_pair (v, n - 1));
67     return result;

```

7.15 Push Relabel Max Flow

```

1  struct edge {
2      int from, to, cap, flow, index;
3      edge(int from, int to, int cap, int flow, int index) :
4          from(from), to(to), cap(cap), flow(flow), index(index) {}
5  };
6
7  struct PushRelabel {
8      int n;
9      vector <vector<edge>> g;
10     vector<long long> excess;
11     vector<int> height, active, count;
12     queue<int> Q;
13
14     PushRelabel(int n) :
15         n(n), g(n), excess(n), height(n), active(n), count(2 * n)
16         {}
17
18     void addEdge(int from, int to, int cap) {
19         g[from].push_back(edge(from, to, cap, 0, g[to].size()));
20         if (from == to)
21             g[from].back().index++;
22         g[to].push_back(edge(to, from, 0, 0, g[from].size() - 1));
23     }
24
25     void enqueue(int v) {
26         if (!active[v] && excess[v] > 0) {
27             active[v] = true;
28             Q.push(v);
29         }
30     }
31
32     void push(edge &e) {
33         int amt = (int) min(excess[e.from], (long long) e.cap - e.flow);
34         if (height[e.from] <= height[e.to] || amt == 0)
35             return;
36         e.flow += amt;
37         g[e.to][e.index].flow -= amt;
38         excess[e.to] += amt;
39         excess[e.from] -= amt;
40         enqueue(e.to);
41     }
42
43     void relabel(int v) {
44         count[height[v]]--;
45         int d = 2 * n;
46         for (auto &it: g[v]) {
47             if (it.cap - it.flow > 0)
48                 d = min(d, height[it.to] + 1);
49         }
50         height[v] = d;
51         count[height[v]]++;
52         enqueue(v);
53     }
54
55     void gap(int k) {
56         for (int v = 0; v < n; v++) {
57             if (height[v] < k)
58                 continue;
59             count[height[v]]--;
60             height[v] = max(height[v], n + 1);
61             count[height[v]]++;
62             enqueue(v);
63         }
64     }
65
66     void discharge(int v) {
67         for (int i = 0; excess[v] > 0 && i < g[v].size(); i++)
68             push(g[v][i]);
69         if (excess[v] > 0) {
70             if (count[height[v]] == 1)
71                 gap(height[v]);
72             else
73                 relabel(v);
74         }
75     }

```

```

75 long long max_flow(int source, int dest) {
76     count[0] = n - 1;
77     count[n] = 1;
78     height[source] = n;
79     active[source] = active[dest] = 1;
80     for (auto &it: g[source]) {
81         excess[source] += it.cap;
82         push(it);
83     }
84
85     while (!Q.empty()) {
86         int v = Q.front();
87         Q.pop();
88         active[v] = false;
89         discharge(v);
90     }
91
92     long long max_flow = 0;
93     for (auto &e: g[source])
94         max_flow += e.flow;
95
96     return max_flow;
97 }
98 };
99

```

7.16 Tarjan Algo

```

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     return 0;
43 }
44

```

7.17 Bipartite Matching

```

1  // vertex are one based
2  struct graph
3  {
4      int L, R;
5      vector<vector<int> > adj;
6      graph(int l, int r) : L(l), R(r), adj(l+1) {}
7      void add_edge(int u, int v)
8      {

```

```

9          adj[u].push_back(v+L);
10     }
11     int maximum_matching()
12     {
13         vector<int> mate(L+R+1, -1), level(L+1);
14         function<bool (void)> levelize = [&]() {
15             {
16                 queue<int> q;
17                 for(int i=1; i<=L; i++)
18                 {
19                     level[i] = -1;
20                     if(mate[i] < 0)
21                         q.push(i), level[i] = 0;
22                 }
23                 while(!q.empty())
24                 {
25                     int node = q.front();
26                     q.pop();
27                     for(auto i : adj[node])
28                     {
29                         int v = mate[i];
30                         if(v < 0)
31                             return true;
32                         if(level[v] < 0)
33                         {
34                             level[v] = level[node] + 1;
35                             q.push(v);
36                         }
37                     }
38                 }
39                 return false;
40             };
41             function<bool (int)> augment = [&](int node)
42             {
43                 for(auto i : adj[node])
44                 {
45                     int v = mate[i];
46                     if(v < 0 || (level[v] > level[node] && augment(v)))
47                     {
48                         mate[node] = i;
49                         mate[i] = node;
50                         return true;
51                     }
52                 }
53                 return false;
54             };
55             int match = 0;
56             while(levelize())
57                 for(int i=1; i<=L; i++)
58                     if(mate[i] < 0 && augment(i))
59                         match++;
60             return match;
61         };
62     };

```

8 Math

8.1 Sum Of floored division.

```

1  typedef unsigned long long ull;
2  ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
3
4  // return sum_{i=0}^{to-1} floor((ki + c) / m) (mod 2^64)
5  ull divsum(ull to, ull c, ull k, ull m) {
6      ull res = k / m * sumsq(to) + c / m * to;
7      k %= m; c %= m;
8      if (!k) return res;
9      ull to2 = (to * k + c) / m;
10     return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
11 }
12 // return sum_{i=0}^{to-1} (ki+c) % m
13 ll modsum(ull to, ll c, ll k, ll m) {
14     c = ((c % m) + m) % m;
15     k = ((k % m) + m) % m;
16     return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
17 }

```

8.2 ModMulLL

```

1 // Calculate a^b % c and a*b % c
2 ull modmul(ull a, ull b, ull M) {
3     ll ret = a * b - M * ull(1.L / M * a * b);
4     return ret + M * (ret < 0) - M * (ret >= (ll)M);
5 }
6 ull modpow(ull b, ull e, ull mod) {
7     ull ans = 1;
8     for (; e; b = modmul(b, b, mod), e /= 2)
9         if (e & 1) ans = modmul(ans, b, mod);
10    return ans;
11 }

```

8.3 MillerRabin Primality check

```

1 "ModMulLL.cpp"
2 typedef unsigned long long ull;
3 ull modmul(ull a, ull b, ull M) {
4     ll ret = a * b - M * ull(1.L / M * a * b);
5     return ret + M * (ret < 0) - M * (ret >= (ll)M);
6 }
7 ull modpow(ull b, ull e, ull mod) {
8     ull ans = 1;
9     for (; e; b = modmul(b, b, mod), e /= 2)
10        if (e & 1) ans = modmul(ans, b, mod);
11    return ans;
12 }
13
14 bool isPrime(ull n) {
15     if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
16     ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
17             s = __builtin_ctzll(n - 1), d = n >> s;
18     for (ull a: A) { // ~count trailing zeroes
19         ull p = modpow(a % n, d, n), i = s;
20         while (p != 1 && p != n - 1 && a % n && i--)
21             p = modmul(p, p, n);
22         if (p != n - 1 && i != s) return 0;
23     }
24     return 1;
25 }

```

8.4 Pollard-rho randomized factorization algorithm $O(n^{1/4})$

```

1 "ModMulLL.cpp", "MillerRabin.cpp"
2 ull pollard(ull n) {
3     auto f = [n](ull x) { return modmul(x, x, n) + 1; };
4     ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
5     while (t++ % 40 || __gcd(prd, n) == 1) {
6         if (x == y) x = ++i, y = f(x);
7         if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
8         x = f(x), y = f(f(y));
9     }
10    return __gcd(prd, n);
11 }
12 vector<ull> factor(ull n) {
13     if (n == 1) return {};
14     if (isPrime(n)) return {n};
15     ull x = pollard(n);
16     auto l = factor(x), r = factor(n / x);
17     l.insert(l.end(), all(r));
18     return l;
19 }
20 }

```

8.5 ModSqrt Finds x s.t $x^2 = a \pmod p$

```

1 ll sqrt(ll a, ll p) {
2     a %= p; if (a < 0) a += p;
3     if (a == 0) return 0;
4     assert(modpow(a, (p-1)/2, p) == 1); // else no solution
5     if (p % 4 == 3) return modpow(a, (p+1)/4, p);
6     // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
7     ll s = p - 1, n = 2;
8     int r = 0, m;
9     while (s % 2 == 0)
10        ++r, s /= 2;
11     while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
12     ll x = modpow(a, (s + 1) / 2, p);
13     ll b = modpow(a, s, p), g = modpow(n, s, p);
14     for (; r = m) {
15         ll t = b;
16         for (m = 0; m < r && t != 1; ++m)
17             t = t * t % p;

```

```

18         if (m == 0) return x;
19         ll gs = modpow(g, 1LL << (r - m - 1), p);
20         g = gs * gs % p;
21         x = x * gs % p;
22         b = b * g % p;
23     }
24 }

```

8.6 Xor With Gauss

```

1 void insertVector(int mask) {
2     for (int i = d - 1; i >= 0; i--) {
3         if ((mask & 1 << i) == 0) continue;
4         if (!basis[i]) {
5             basis[i] = mask;
6             return;
7         }
8         mask ^= basis[i];
9     }
10 }

```

8.7 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 }

```

9 Strings

9.1 Aho-Corasick Mostafa

```

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

```

```

38 while (!q.empty()) {
39     int r = q.front();
40     q.pop();
41     for (int k = 0, arck; k < ALPHABET_SIZE; ++k) {
42         if ((arck = a[r].child[k]) != -1) {
43             q.push(arck);
44             int v = a[r].failure;
45             while (a[v].child[k] == -1) v = a[v].failure;
46             a[arck].failure = a[v].child[k];
47             a[arck].match_parent = a[v].child[k];
48             while (a[arck].match_parent != -1 &&
49                 a[a[arck].match_parent].match.empty())
50                 a[arck].match_parent =
51                     a[a[arck].match_parent].match_parent;
52         }
53     }
54 }
55
56 void aho_corasick(string &sentence, vector<string> &words,
57                 vector<vector<int>> &matches) {
58     matches.assign(words.size(), vector<int>());
59     int state = 0, ss = 0;
60     for (int i = 0; i < sentence.length(); ++i, ss = state) {
61         while (a[ss].child[sentence[i] - 'a'] == -1)
62             ss = a[ss].failure;
63         state = a[state].child[sentence[i] - 'a'] = a[ss].child[
64             sentence[i] - 'a'];
65         for (ss = state; ss != -1; ss = a[ss].match_parent)
66             for (int w: a[ss].match)
67                 matches[w].push_back(i + 1 - words[w].length());
68     }
69 }
70 };

```

9.2 KMP Anany

```

1 vector<int> fail(string s) {
2     int n = s.size();
3     vector<int> pi(n);
4     for (int i = 1; i < n; i++) {
5         int g = pi[i-1];
6         while (g && s[i] != s[g])
7             g = pi[g-1];
8         g += s[i] == s[g];
9         pi[i] = g;
10    }
11    return pi;
12 }
13 vector<int> KMP(string s, string t) {
14     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 }

```

9.3 Manacher Kactl

```

1 // If the size of palindrome centered at i is x, then d1[i] stores (x
2 // +1)/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 // If the size of palindrome centered at i is x, then d2[i] stores x/2
17 vector<int> d2(n);
18 for (int i = 0, l = 0, r = -1; i < n; i++) {
19     int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
20     while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
21         k++;
22     }
23     d2[i] = k--;
24     if (i + k > r) {
25         l = i - k - 1;
26         r = i + k;
27     }
28 }

```

9.4 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     #define all(x) begin(x), end(x)
5     /*
6      * Note this code is considers also the empty suffix
7      * so hear sa[0] = n and sa[1] is the smallest non empty suffix
8      * and sa[n] is the largest non empty suffix
9      * also LCP[i] = LCP(sa[i-1], sa[i]), meaning LCP[0] = LCP[1] =
10      * 0
11      * if you want to get LCP(i..j) you need to build a mapping
12      * between
13      * sa[i] and i, and build a min sparse table to calculate the
14      * minimum
15      * note that this minimum should consider sa[i+1...j] since you
16      * don't want
17      * to consider LCP(sa[i], sa[i-1])
18      *
19      * you should also print the suffix array and lcp at the
20      * beginning of the contest
21      * to clarify this stuff
22      */
23     vi sa, lcp;
24     SuffixArray(string& s, int lim=256) { // or basic_string<int>
25         int n = sz(s) + 1, k = 0, a, b;
26         vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
27         sa = lcp = y, iota(all(sa), 0);
28         for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
29             p = j, iota(all(y), n - j);
30             rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
31             fill(all(ws), 0);
32             rep(i,0,n) ws[x[i]]++;
33             rep(i,1,lim) ws[i] += ws[i - 1];
34             for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
35             swap(x, y), p = 1, x[sa[0]] = 0;
36             rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
37                 (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
38         }
39         rep(i,1,n) rank[sa[i]] = i;
40         for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
41             for (k && k--, j = sa[rank[i] - 1];
42                 s[i + k] == s[j + k]; k++);
43     }
44 };

```

9.5 Suffix Automaton Mostafa

```

1 struct SA {
2     struct node {
3         int to[26];
4         int link, len, co = 0;
5     };
6     node() {
7         memset(to, 0, sizeof to);
8         co = 0, link = 0, len = 0;
9     }
10 };
11
12 int last, sz;
13 vector<node> v;
14
15 SA() {
16     v = vector<node>(1);
17     last = 0, sz = 1;
18 }
19

```

```

20 void add_letter(int c) {
21     int p = last;
22     last = sz++;
23     v.push_back({});
24     v[last].len = v[p].len + 1;
25     v[last].co = 1;
26     for (; v[p].to[c] == 0; p = v[p].link)
27         v[p].to[c] = last;
28     if (v[p].to[c] == last) {
29         v[last].link = 0;
30         return;
31     }
32     int q = v[p].to[c];
33     if (v[q].len == v[p].len + 1) {
34         v[last].link = q;
35         return;
36     }
37     int cl = sz++;
38     v.push_back(v[q]);
39     v.back().co = 0;
40     v.back().len = v[p].len + 1;
41     v[last].link = v[q].link = cl;
42
43     for (; v[p].to[c] == q; p = v[p].link)
44         v[p].to[c] = cl;
45 }
46
47 void build_co() {
48     priority_queue<pair<int, int>> q;
49     for (int i = sz - 1; i > 0; i--)
50         q.push({v[i].len, i});
51     while (q.size()) {
52         int i = q.top().second;
53         q.pop();
54         v[v[i].link].co += v[i].co;
55     }
56 }
57 };

```

9.6 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 }

```

9.7 lexicographically smallest rotation of a string

```

1  int minRotation(string s) {
2      int a=0, N=sz(s); s += s;
3      rep(b,0,N) rep(k,0,N) {
4          if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
5          if (s[a+k] > s[b+k]) {a = b; break;}
6      }
7      return a;
8  }

```

10 Trees

10.1 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
         a descendant
8
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
34         others and decompose
35         centPar[decompose(p.F)] = c;
36     }
37     return c;
38 }
39 void update(int node, int distance, int col) {
40     int centroid = node;
41     while(centroid) {
42         ///solve
43         centroid = centPar[centroid];
44     }
45 }
46 int query(int node) {
47     int ans = 0;
48
49     int centroid = node;
50     while(centroid) {
51         ///solve
52         centroid = centPar[centroid];
53     }
54
55     return ans;
56 }

```

10.2 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     ///modify node to data structure
15
16     for(auto v : adj[node])
17         if(v != par && v != bigChild)
18             add(v, node, bigChild, delta);
19 }
20
21 void dfs2(int node, int par, bool keep) {
22     for(auto v : adj[node]) if(v != par && v != bigChild[node]) {
23         dfs2(v, node, 0);
24     }
25     if(bigChild[node]) {
26         dfs2(bigChild[node], node, true);
27     }

```



```

28     }
29     add(node, par, bigChild[node], 1);
30     ///process queries
31     if(!keep) {
32         add(node, par, -1, -1);
33     }
34 }

```

10.3 Heavy Light Decomposition (Along with Euler Tour)

```

1  /*
2  Notes:
3  1. 0-based
4  2. solve function iterates over segments and handles them
   seperately
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
47 int solve(int u,int v) {
48     vector<pair<int,int>> > segu;
49     vector<pair<int,int>> > segv;
50     if(isChild(u,v)){
51         while(nxt[u] != nxt[v]){
52             segv.push_back(make_pair(in[nxt[v]], in[v]));
53             v = par[nxt[v]];
54         }
55         segv.push_back({in[u], in[v]});
56     } else if(isChild(v,u)){
57         while(nxt[u] != nxt[v]){
58             segu.push_back(make_pair(in[nxt[u]], in[u]));
59             u = par[nxt[u]];
60         }
61         segu.push_back({in[v], in[u]});
62     } else {
63         while(u != v) {
64             if(nxt[u] == nxt[v]) {
65                 if(in[u] < in[v]) segv.push_back({in[u],in[v]}), R.push_back
66                     ({u+1,v+1});
67                 else segu.push_back({in[v],in[u]}), L.push_back({v+1,u+1});
68                 u = v;
69                 break;
70             } else if(in[u] > in[v]) {

```

```

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

```

10.4 Mo on Trees

```

1  // Calculate the DFS order, {1, 2, 3, 3, 4, 4, 2, 5, 6, 6, 5, 1}.
2  // Let a query be (u, v), ST(u) <= ST(v), P = LCA(u,v)
3  // Case 1: P = u : the query range would be [ST(u), ST(v)]
4  // Case 2: P != u : range would be [EN(u), ST(v)] + [ST(P), ST(P)].
5  // the path will be the nodes that appears exactly once in that range

```

11 Numerical

11.1 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 };

```

11.2 Polynomials

```

1  struct Poly {
2      vector<double> a;
3      double operator()(double x) const {
4          double val = 0;
5          for (int i = sz(a); i--;) (val += x) += a[i];
6          return val;
7      }
8      void diff() {
9          rep(i, 1, sz(a)) a[i-1] = i*a[i];
10         a.pop_back();
11     }
12     void divroot(double x0) {
13         double b = a.back(), c; a.back() = 0;
14         for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
15         a.pop_back();
16     }
17 }
18 // Finds the real roots to a polynomial
19 // O(n^2 log(1/e))
20 vector<double> polyRoots(Poly p, double xmin, double xmax) {
21     if (sz(p.a) == 2) { return {-p.a[0] / p.a[1]}; }
22     vector<double> ret;
23     Poly der = p;
24     der.diff();
25     auto dr = polyRoots(der, xmin, xmax);
26     dr.push_back(xmin - 1);
27     dr.push_back(xmax + 1);
28     sort(all(dr));
29     rep(i, 0, sz(dr) - 1) {
30         double l = dr[i], h = dr[i + 1];
31         bool sign = p(l) > 0;
32         if (sign ^ (p(h) > 0)) {
33             rep(it, 0, 60) { // while (h - l > 1e-8)
34                 double m = (l + h) / 2, f = p(m);
35                 if ((f <= 0) ^ sign) l = m;
36                 else h = m;
37             }
38             ret.push_back((l + h) / 2);
39         }
40     }
41     return ret;
42 }
43 // Given n points (x[i], y[i]), computes an n-1-degree polynomial that
44 // passes through them.
45 // For numerical precision pick x[k] = c * cos(k / (n - 1) * pi).
46 // O(n^2)
47 typedef vector<double> vd;
48 vd interpolate(vd x, vd y, int n) {
49     vd res(n), temp(n);
50     rep(k, 0, n - 1) rep(i, k + 1, n)
51     y[i] = (y[i] - y[k]) / (x[i] - x[k]);
52     double last = 0;
53     temp[0] = 1;
54     rep(k, 0, n) rep(i, 0, n) {
55         res[i] += y[k] * temp[i];
56         swap(last, temp[i]);
57         temp[i] -= last * x[k];
58     }
59     return res;
60 }
61 // Recovers any n-order linear recurrence relation from the first 2n
62 // terms of the recurrence.
63 // Useful for guessing linear recurrences after bruteforcing the first
64 // terms.
65 // Should work on any field, but numerical stability for floats is not
66 // guaranteed.
67 // O(n^2)
68 vector<ll> berlekampMassey(vector<ll> s) {
69     int n = sz(s), L = 0, m = 0;
70     vector<ll> C(n), B(n), T;
71     C[0] = B[0] = 1;
72     ll b = 1;
73     rep(i, 0, n) { ++m;
74         ll d = s[i] % mod;
75         rep(j, 1, L + 1) d = (d + C[j] * s[i - j]) % mod;
76         if (!d) continue;
77         T = C; ll coef = d * modpow(b, mod - 2) % mod;
78         rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) % mod;
79         if (2 * L > i) continue;

```

```

79         L = i + 1 - L; B = T; b = d; m = 0;
80     }
81     C.resize(L + 1); C.erase(C.begin());
82     for (ll &x: C) x = (mod - x) % mod;
83     return C;
84 }
85 // Generates the kth term of an n-order linear recurrence
86 // S[i] = S[i - j - 1]tr[j], given S[0..>= n - 1] and tr[0..n - 1]
87 // Useful together with Berlekamp-Massey.
88 // O(n^2 * log(k))
89 typedef vector<ll> Poly;
90 ll linearRec(Poly S, Poly tr, ll k) {
91     int n = sz(tr);
92     auto combine = [&](Poly a, Poly b) {
93         Poly res(n * 2 + 1);
94         rep(i, 0, n + 1) rep(j, 0, n + 1)
95             res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
96         for (int i = 2 * n; i > n; --i) rep(j, 0, n)
97             res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
98         res.resize(n + 1);
99         return res;
100     };
101     Poly pol(n + 1, e(pol));
102     pol[0] = e[1] = 1;
103     for (++k; k; k /= 2) {
104         if (k % 2) pol = combine(pol, e);
105         e = combine(e, e);
106     }
107     ll res = 0;
108     rep(i, 0, n) res = (res + pol[i + 1] * S[i]) % mod;
109     return res;
110 }
111 }
112 }
113 }

```

12 Guide

12.1 Notes

- Don't forget to solve the problem in reverse (i.e deleting->adding or adding->deleting, ...etc)
- Max flow is just choosing the maximum number of paths between source and sink
- If you have a problem that tells you choose a[i] or b[i] (or a range) choose one of them initially and play a take or leave on the other
- If the problem tells you to do something cyclic solving it for $x + x$
- Problems that are close to NP problems sometimes have greedy solutions for large input i.e $n \leq 20-30$
- in case of merging between sets try bitsets (i.e $i + j$ or sth)
- If you have a TLE soln using bitset might help
- If everything else fails think Brute force or randomization

12.2 Assignment Problems

- If you see a problem that tells you out of N choose K that has some property (think flows or aliens trick)

- If you see a problem that tells for some X choose a Y (think flows)
- If the problem tells you to choose a Y from L-ıR (think range flow i.e putting edges between the same layer)

12.3 XOR problems

- If the problem tells your something about choosing an XOR of a subset (think FWHT or XOR-basis)
- If the problem tells you about getting XOR of a tree path let $a[i] = \text{XOR tree from root to } i$ and solve this as an array
- If the problem tells you range XOR sth it's better to have prefix XOR and make it pairs XOR.

12.4 Decompositions

- If a problem is asking you to calculate the answer after K steps you can calculate the answer for K
- If the number of queries is significantly larger than updates or vice versa you can use square root Decompositions to give advantage to one over the other

12.5 Strings

- Longest Common Substring is easier with suffix automaton
- Problems that tell you count stuff that appears X times or count appearances (Use suffix links)
- Problems that tell you find the largest substring with some property (Use Suffix links)
- Remember suffix links are the same as aho corasic failure links (you can memoize them with dp)
- Problems that ask you to get the k-th string (can be either suffix automaton or array)
- Longest Common Prefix is mostly a (suffix automaton-array) thing
- try thinking bitsets

12.6 Trees

- For problems that ask you to count stuff in a subtree think (Euler Tour with RQ - Small to Large - DSU on Trees - PersistentSegTree)
- Note that the farthest node to any node in the tree is one of the two diameter heads
- In case of asking $F(\text{node}, x)$ for each node it's probably DP on Trees

12.7 Flows

- If you want to make a K-covering instead of considering lit edges consider non-lit edges
- To get mincost while maintaining a flow network (note that flows are batched together according to cost)
- If the problem asks you to choose some stuff that minimizes use Min Cut (If maximizes sum up stuff and subtract min cut)

12.8 Geometry

- Manhattan to King distance $(x,y) \rightarrow (x+y, x-y)$
- Lattice points on line: $\gcd(dx, dy) + 1$
- Pick's theorem: $A = I + \frac{B}{2} - 1$
- cosine rule: $C^2 = A^2 + B^2 - 2AB \times \cos(c)$
- Rotation around axis: $R = (\cos(a) \times Id + \sin(a) \times \text{cross}U + (1 - \cos(a)) \times \text{outer}U)$
- Triangulation of n-gon = Catalan (n-2)

12.9 Area

- triangle = $\sqrt{(S \times (S - A) \times (S - B) \times (S - C))}$, $S = \text{PERIMETER}/2$
- triangle = $r \times S$, r = radius of inscribed circle
- ellipse = $\pi \times r_1 \times r_2$
- sector = $\frac{(r^2 \times a)}{2}$
- circular cap = $\frac{R^2 \times (a - \sin(a))}{2}$
- prism = $\text{perimeter}(B)L + 2\text{area}(B)$
- sphere = $4\pi r^2$

12.10 Volume

- Right circular cylinder = $\pi r^2 h$
- Pyramid = $\frac{Bh}{3}$
- Right circular cone = $\frac{\pi r^2 h}{3}$
- Sphere = $\frac{4}{3}\pi r^2 h$
- Sphere sector = $\frac{2}{3}\pi r^2 h = \frac{2}{3}\pi r^3(1 - \cos(a))$
- Sphere cap = $\frac{\pi h^2(3r-h)}{3}$

12.11 Combinatorics

- Cayley formula: number of forest with k trees where first k nodes belongs to different trees = kn^{n-k-1} . Multinomial theorem for trees of given degree sequence $\binom{n}{d_i}$
- Prufer sequence (M5da calls it parent array)
- K-Cyclic permutation = $\binom{n}{k} \times (k-1)!$
- Stirling numbers $S(n, k) = k \times S(n-1, k) + S(n, k-1)$ number of way to partition n in k sets.
- Bell number $B_n = \sum_1^n (n-1, k) B_k$
- # ways to make a graph with k components connected $n^{k-2} \times \prod_{i=1}^k s_i$
- Arithmetic-geometric-progression $S_n = \frac{A_1 \times G_1 - A_{n+1} \times G_{n+1}}{1-r} + \frac{dr}{(1-r)^2} \times (G_1 - G_{n+1})$

12.12 Graph Theory

- Graph realization problem: sorted decreasing degrees: $\sum_1^k d_i = k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ (first k form clique and all other nodes are connected to them).
- Euler formula: $v + f = e + c + 1$
- # perfect matching in bipartite graph, $DP[S][j] = DP[S][j-1] + DP[S/v][j-1]$ for all v connected to the j node.

12.13 Max flow with lower bound

- feasible flow in a network with both upper and lower capacity constraints, no source or sink: capacities are changed to upper bound - lower bound. Add a new source and a sink. let $M[v] = (\text{sum of lower bounds of ingoing edges to } v) - (\text{sum of lower bounds of outgoing edges from } v)$. For all v, if $M[v] < 0$ then add edge (S,v) with capacity $M[v]$, otherwise add (v,T) with capacity $-M[v]$. If all outgoing edges from S are full, then a feasible flow exists, it is the flow plus the original lower bounds.
- maximum flow in a network with both upper and lower capacity constraints, with source s and sink t: add edge (t,s) with capacity infinity. Binary search for the lower bound, check whether a feasible exists for a network WITHOUT source or sink (B).

12.14 Joseph problem

$$g(n, k) = \begin{cases} 0 & \text{if } n = 1 \\ (g(n-1, k) + k) \bmod n & \text{if } 1 < n < k \\ \left\lfloor \frac{k((g(n', k) - n \bmod k) \bmod n')}{k-1} \right\rfloor \text{ where } n' = n - \left\lfloor \frac{n}{k} \right\rfloor & \text{if } k \leq n \end{cases}$$