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

Pillow, Isaac, Mostafa, Islam

2021

Contents

1 Template 1
- 1.1 template 1

2 Combinatorics 1
- 2.1 Burnside Lemma 1
- 2.2 Catlan Numbers 1

3 Algebra 2
- 3.1 Gray Code 2
- 3.2 Factorial modulo in p*log(n) (Wilson Theroem) 2
- 3.3 Iteration over submasks 2
- 3.4 FFT 2
- 3.5 FFT with mod 2
- 3.6 convolutions of AND-XOR-OR 2
- 3.7 NTT of KACTL 3
- 3.8 Fibonacci 3
- 3.9 Gauss Determinant 3
- 3.10 GAUSS SLAE 3
- 3.11 Matrix Inverse 3

4 Data Structures 3
- 4.1 UnionFindRollback 3
- 4.2 2D BIT 4
- 4.3 2D Sparse table 4
- 4.4 Mo With Updates 4
- 4.5 Ordered Set 4
- 4.6 Persistent Seg Tree 4
- 4.7 Treap 4
- 4.8 Wavelet Tree 5
- 4.9 SparseTable 5

5 DP 5
- 5.1 CHT Line Container 5

6 Geometry 5
- 6.1 Convex Hull 5
- 6.2 Geometry Template 6
- 6.3 Half Plane Intersection 6
- 6.4 Segments Intersection 7
- 6.5 Rectangles Union 7

7 Graphs 8
- 7.1 Ariculation Point 8
- 7.2 Bridges Tree and Diameter 8
- 7.3 Dinic With Scalling 9
- 7.4 Gomory Hu 9
- 7.5 Kosaraju 9
- 7.6 Maximum Clique 10
- 7.7 HopcraftKarp matching (Bipartite) 10
- 7.8 Hungarian Weighted matching (Bipartite) 10
- 7.9 MinCostMaxFlow 10
- 7.10 Push Relabel Max Flow 11
- 7.11 Prufer Code 11
- 7.12 Tarjan Algo 12

8 NumberTheory 12
- 8.1 ModSum (Sum Of floored division) 12
- 8.2 ModMulLL 12
- 8.3 ModSqrt Finds x s.t $x^2 = a \mod p$ 12
- 8.4 MillerRabin Primality check 12
- 8.5 Pollard-rho randomized factorization algorithm $O(n^{1/4})$ 13
- 8.6 Primitive Roots 13
- 8.7 Discrete Logarithm minimum x for which $a^x = b\%m$ 13
- 8.8 Discrete Root finds all numbers x such that $x^k = a\%n$ 13
- 8.9 Totient function 13
- 8.10 CRT and EGCD 13
- 8.11 Xor With Gauss 13
- 8.12 Josephus 13

9 Strings 14
- 9.1 Aho-Corasick Mostafa 14
- 9.2 KMP Anany 14
- 9.3 Manacher Kactl 14
- 9.4 Suffix Array Kactl 14
- 9.5 Suffix Automaton Mostafa 15
- 9.6 Zalgo Anany 15
- 9.7 lexicographically smallest rotation of a string 15

10 Trees 15
- 10.1 Centroid Decomposition 15
- 10.2 Dsu On Trees 15
- 10.3 Heavy Light Decomposition (Along with Euler Tour) 16
- 10.4 Mo on Trees 16

11 Numerical 16
- 11.1 Lagrange Polynomial 16
- 11.2 Polynomials 17

12 Guide 17
- 12.1 Strings 17
- 12.2 Volume 17
- 12.3 Graph Theory 17
- 12.4 Joseph problem 17

1 Template

1.1 template

```cpp
#include <bits/stdc++.h>
#define IO ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
using namespace std;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

// Kactl defines
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<double> vd;
```

2 Combinatorics

2.1 Burnside Lemma

```cpp
// |Classes|=sum (k ^C(pi))  / |G|
// C(pi)  the number of cycles in the permutation pi
// |G| the number of permutations
```

2.2 Catlan Numbers

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

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

3 Algebra

3.1 Gray Code

```
int g (int n) {
    return n ^ (n >> 1);
}
int rev_g (int g) {
  int n = 0;
  for (; g; g >>= 1)
    n ^= g;
  return n;
}
int calc(int x, int y) { ///2D Gray Code
    int a = g(x), b = g(y);
    int res = 0;
    f(i,0,LG) {
        int k1 = (a & (1 << i));
        int k2 = b & (1 << i);
        res |= k1 << (i + 1);
        res |= k2 << i;
    }
    return res;
}
```

3.2 Factorial modulo in p*log(n) (Wilson Theroem)

```
int factmod(int n, int p) {
    vector<int> f(p);
    f[0] = 1;
    for (int i = 1; i < p; i++)
        f[i] = f[i-1] * i % p;

    int res = 1;
    while (n > 1) {
        if ((n/p) % 2)
            res = p - res;
        res = res * f[n%p] % p;
        n /= p;
    }
    return res;
}
```

3.3 Iteration over submasks

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

3.4 FFT

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% fas te r i f double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            C z = rt[j + k] * a[i + j + k]; //
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i, 0, sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    /// rep(i,0,sz(res)) res[i] = (MOD+(ll)round(imag(out[i]) / (4 * n))) % MOD;
        ///in case of mod
    rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}

//Applications
//1-All possible sums

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

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

3.5 FFT with mod

```
"FastFourierTransform.cpp"
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

3.6 convolutions of AND-XOR-OR

```
// The size of a must be a power of two.
void FST(vi& a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                // inv ? pii(v, u - v) : pii(u + v, u); // OR /// include-line
                // pii(u + v, u - v);                   // XOR /// include-line
        }
    }
    // if (inv) for (int& x : a) x /= sz(a); // XOR only /// include-line
}
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
```

```
    rep(i,0,sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

3.7 NTT of KACTL

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
        n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n)
        out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

3.8 Fibonacci

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

3.9 Gauss Determinant

```
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
// for integers
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

3.10 GAUSS SLAE

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

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

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

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

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

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

3.11 Matrix Inverse

```
#define ld long double
vector < vector<ld> > gauss (vector < vector<ld> > a) {

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

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

4 Data Structures

4.1 UnionFindRollback

```
struct RollbackUF {
    vi e; vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
```

```
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return sz(st); }
    void rollback(int t) {
        for (int i = time(); i --> t;)
            e[st[i].first] = st[i].second;
        st.resize(t);
    }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]});
        st.push_back({b, e[b]});
        e[a] += e[b]; e[b] = a;
        return true;
    }
};
```

4.2 2D BIT

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

4.3 2D Sparse table

```
const int N = 505, LG = 10;
int st[N][N][LG][LG];
int a[N][N], lg2[N];
int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
         max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
         max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b])
       );
}
void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
```

4.4 Mo With Updates

```
///O(N^5/3) note that the block size is not a standard size
/// O(2SQ + N^2 / S + Q * N^2 / S^2) = O(Q * N^(2/3)) if S = n^(2/3)
/// fact: S = (2 * n * n)^(1/3) give the best complexity
const int block_size = 2000;
struct Query{
    int l, r, t, idx;
    Query(int l,int r,int t,int idx) : l(l),r(r),t(t),idx(idx) {}
    bool operator < (Query o) const{
        if(l / block_size != o.l / block_size) return l < o.l;
        if(r / block_size != o.r/block_size) return r < o.r;
        return t < o.t;
    }
};
int L = 0, R = -1, K = -1;
```

```
while(L < Q[i].l)del(a[L++]);
while(L > Q[i].l)add(a[--L]);
while(R < Q[i].r)add(a[++R]);
while(R > Q[i].r)del(a[R--]);
while(K < Q[i].t)upd(++K);
while(K > Q[i].t)err(K--);
```

4.5 Ordered Set

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

4.6 Persistent Seg Tree

```

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

4.7 Treap

```

mt19937_64 mrand(chrono::steady_clock::now().time_since_epoch().count());
struct Node {
    int key, pri = mrand(), sz = 1;
    int lz = 0;
    int idx;
    array<Node*, 2> c = {NULL,NULL};
    Node(int key, int idx) : key(key), idx(idx) {}
};
int getsz(Node* t){
    return t ? t->sz : 0;
}
Node* calc(Node* t) {
    t->sz = 1 + getsz(t->c[0]) + getsz(t->c[1]);
    return t;
}
void prop(Node* cur) {
    if(!cur || !cur->lz)
        return;
    cur->key += cur->lz;
    if(cur->c[0])
        cur->c[0]->lz += cur->lz;
    if(cur->c[1])
        cur->c[1]->lz += cur->lz;
    cur->lz = 0;
}
```

```cpp
array<Node*, 2> split(Node* t, int k) {
    prop(t);
    if(!t)
        return {t, t};
    if(getsz(t->c[0]) >= k) {   ///answer is in left node
        auto ret = split(t->c[0], k);
        t->c[0] = ret[1];
        return {ret[0], calc(t)};
    } else {  ///k > t->c[0]
        auto ret = split(t->c[1], k - 1 - getsz(t->c[0]));
        t->c[1] = ret[0];
        return {calc(t), ret[1]};
    }
}
Node* merge(Node* u, Node* v) {
    prop(u);
    prop(v);
    if(!u || !v)
        return u ? u : v;
    if(u->pri>v->pri) {
        u->c[1] = merge(u->c[1], v);
        return calc(u);
    } else {
        v->c[0] = merge(u, v->c[0]);
        return calc(v);
    }
}
int cnt(Node* cur, int x) {
    prop(cur);
    if(!cur)
        return 0;
    if(cur->key <= x)
        return getsz(cur->c[0]) + 1 + cnt(cur->c[1], x);
    return cnt(cur->c[0], x);
}
Node* ins(Node* root, int val, int idx, int pos) {
    auto splitted = split(root, pos);
    root = merge(splitted[0], new Node(val, idx));
    return merge(root, splitted[1]);
}
```

4.8 Wavelet Tree

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

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

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

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

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

4.9 SparseTable

```cpp
int S[N];
for(int i = 2; i < N; i++) S[i] = S[i >> 1] + 1;
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);

int query(int l, int r) {
    int k = S[r - l + 1];
    return mrg(st[k][l], st[k][r-(1<<k)+1]);
}
```

5 DP

5.1 CHT Line Container

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

6 Geometry

6.1 Convex Hull

```cpp
struct point {
    ll x,  y;
    point(ll x, ll y) : x(x), y(y) {}
    point operator -(point other) {
        return point(x - other.x, y - other.y);
    }
```

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

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

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

6.2 Geometry Template

```cpp
using  ptype =  double edit this first ;
double EPS = 1e-9;
struct point {
    ptype x,  y;
    point(ptype x, ptype y) : x(x), y(y) {}
    point operator -(const point & other)const { return point(x - other.x, y - other.y);}
    point operator +(const point & other)const { return point(x + other.x, y + other.y);}
    point operator *(ptype c) const { return point(x * c, y * c); }
    point operator /(ptype c) const { return point(x / c, y / c); }
    point prep() { return point(-y, x); }
};
ptype cross(point a, point b) { return a.x * b.y - a.y * b.x;}
ptype dot(point a, point b) {return a.x * b.x + a.y * b.y;}
double abs(point a) {return sqrt(dot(a, a));}

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

    point d = (a - b) / abs(a - b);
    return {center + p + d * len, center + p - d * len};
}

vector<point> CircleCircleIntersect(point c1, ld r1, point c2, ld r2) {
    if (r1 < r2) {
        swap(r1, r2);
        swap(c1, c2);
    }
    ld d = abs(c2 - c1); // distance between c1,c2
    if (d > r1 + r2 || d < r1 - r2 || d < EPS)  // zero or infinite solutions
        return {};
    ld angle = acos(min((d * d + r1 * r1 - r2 * r2) / (2 * r1 * d), (ld) 1.0));
    point p = (c2 - c1) / d * r1;

    if (angle < EPS)
        return {c1 + p};

    return {c1 + RotateCCW(p, angle), c1 + RotateCCW(p, -angle)};

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

6.3 Half Plane Intersection

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

// Basic point/vector struct.
struct Point {

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

    // Addition, substraction, multiply by constant, cross product.
    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }
    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }
    friend Point operator * (const Point& p, const ld& k) {
        return Point(p.x * k, p.y * k);
    }
    friend ld cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};

// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the direction vector of the line
    Point p, pq;
    ld angle;

    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }
    // Comparator for sorting.
    // If the angle of both half-planes is equal, the leftmost one should go first.
    bool operator < (const Halfplane& e) const {
        if (fabsl(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
        return angle < e.angle;
    }
    // We use equal comparator for std::unique to easily remove parallel half-planes.
```

```cpp
    bool operator == (const Halfplane& e) const {
        return fabsl(angle - e.angle) < eps;
    }
    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        ld alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
// Actual algorithm
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };

    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }
    // Sort and remove duplicates
    sort(H.begin(), H.end());
    H.erase(unique(H.begin(), H.end()), H.end());

    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }
        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();

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

6.4 Segments Intersection

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

struct pt {
    double x, y;
};

struct seg {
    pt p, q;
    int id;

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

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

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

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

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

struct event {
    double x;
    int tp, id;

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

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

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

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

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

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

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

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

6.5 Rectangles Union

```cpp
#include<bits/stdc++.h>
#define P(x,y) make_pair(x,y)
using namespace std;
class Rectangle {
public:
    int x1, y1, x2, y2;
    static Rectangle empt;
```

```cpp
    Rectangle() {
        x1 = y1 = x2 = y2 = 0;
    }
    Rectangle(int X1, int Y1, int X2, int Y2) {
        x1 = X1;
        y1 = Y1;
        x2 = X2;
        y2 = Y2;
    }
};
struct Event {
    int x, y1, y2, type;
    Event() {}
    Event(int x, int y1, int y2, int type): x(x), y1(y1), y2(y2), type(type) {}
};
bool operator < (const Event&A, const Event&B) {
//if(A.x != B.x)
    return A.x < B.x;
//if(A.y1 != B.y1) return A.y1 < B.y1;
//if(A.y2 != B.y2()) A.y2 < B.y2;
}
const int MX = (1 << 17);
struct Node {
    int prob, sum, ans;
    Node() {}
    Node(int prob, int sum, int ans): prob(prob), sum(sum), ans(ans) {}
};
Node tree[MX * 4];
int interval[MX];
void build(int x, int a, int b) {
    tree[x] = Node(0, 0, 0);
    if(a == b) {
        tree[x].sum += interval[a];
        return;
    }
    build(x * 2, a, (a + b) / 2);
    build(x * 2 + 1, (a + b) / 2 + 1, b);
    tree[x].sum = tree[x * 2].sum + tree[x * 2 + 1].sum;
}
int ask(int x) {
    if(tree[x].prob)
        return tree[x].sum;
    return tree[x].ans;
}
int st, en, V;
void update(int x, int a, int b) {
    if(st > b || en < a)
        return;
    if(a >= st && b <= en) {
        tree[x].prob += V;
        return;
    }
    update(x * 2, a, (a + b) / 2);
    update(x * 2 + 1, (a + b) / 2 + 1, b);
    tree[x].ans = ask(x * 2) + ask(x * 2 + 1);
}
Rectangle Rectangle::empt = Rectangle();
vector < Rectangle > Rect;
vector < int > sorted;
vector < Event > sweep;
void compressncalc() {
    sweep.clear();
    sorted.clear();
    for(auto R : Rect) {
        sorted.push_back(R.y1);
        sorted.push_back(R.y2);
    }
    sort(sorted.begin(), sorted.end());
    sorted.erase(unique(sorted.begin(), sorted.end()), sorted.end());
    int sz = sorted.size();
    for(int j = 0; j < sorted.size() - 1; j++)
        interval[j + 1] = sorted[j + 1] - sorted[j];
    for(auto R : Rect) {
        sweep.push_back(Event(R.x1, R.y1, R.y2, 1));
        sweep.push_back(Event(R.x2, R.y1, R.y2, -1));
    }
    sort(sweep.begin(), sweep.end());
    build(1, 1, sz - 1);
}
long long ans;
void Sweep() {
    ans = 0;
    if(sorted.empty() || sweep.empty())
        return;
    int last = 0, sz_ = sorted.size();
    for(int j = 0; j < sweep.size(); j++) {
        ans += 1ll * (sweep[j].x - last) * ask(1);
        last = sweep[j].x;
        V = sweep[j].type;
        st = lower_bound(sorted.begin(), sorted.end(), sweep[j].y1) - sorted.begin() +
            1;
        en = lower_bound(sorted.begin(), sorted.end(), sweep[j].y2) - sorted.begin();
        update(1, 1, sz_ - 1);
    }
}
int main() {
//    freopen("in.in","r",stdin);
    int n;
    scanf("%d", &n);
    for(int j = 1; j <= n; j++) {
        int a, b, c, d;
        scanf("%d %d %d %d", &a, &b, &c, &d);
        Rect.push_back(Rectangle(a, b, c, d));
    }
    compressncalc();
    Sweep();
    cout << ans << endl;
}
```

7 Graphs

7.1 Ariculation Point

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

void dfs(int node, int par){
    dfsn[node] = low[node] = ++timer;
    int kam = 0;
    for(auto i: adj[node]){
        if(i == par) continue;
        if(dfsn[i] == 0){
            kam++;
            dfs(i, node);
            low[node] = min(low[node], low[i]);
            if(dfsn[node] <= low[i] && par != 0) ar_point[node] = 1;
        }
        else low[node] = min(low[node], dfsn[i]);
    }
    if(par == 0 && kam > 1) ar_point[node] = 1;
}
int main(){
    // Input
    for(int i = 1; i <= n; i++){
        if(dfsn[i] == 0) dfs(i, 0);
    }
    int c = 0;
    for(int i = 1; i <= n; i++){
        if(ar_point[i]) c++;
    }
    cout << c << '\n';
}
```

7.2 Bridges Tree and Diameter

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

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


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

```
                if(cur == node) break;
            }
        }
    }

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

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

        return 0;
    }
```

7.3 Dinic With Scalling

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

7.4 Gomory Hu

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

#include "PushRelabel.h"

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

7.5 Kosaraju

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

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

const int N = 1e5+5;

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

```cpp
    int c = 0;
    while(stk.size()){
        auto cur = stk.top();
        stk.pop();
        if(!vis[cur])
            rdfs(cur,c++);

    }
    return c;
}
```

7.6 Maximum Clique

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

7.7 HopcraftKarp matching (Bipartite)

```cpp
// Hopcroft-Karp's (1-based)
// Complexity: O(m * sqrt{n})
struct graph {
    int L, R;
    vector<vector<int> > adj;

    graph(int l, int r) : L(l), R(r), adj(l + 1) {}

    void add_edge(int u, int v) {
        adj[u].push_back(v + L);
    }

    int maximum_matching() {
        vector<int> mate(L + R + 1, -1), level(L + 1);
        function<bool(void)> levelize = [&]() {
            queue<int> q;
            for (int i = 1; i <= L; i++) {
                level[i] = -1;
                if (mate[i] < 0)
                    q.push(i), level[i] = 0;
            }
            while (!q.empty()) {
                int node = q.front();
                q.pop();
                for (auto i: adj[node]) {
                    int v = mate[i];
                    if (v < 0)
                        return true;
                    if (level[v] < 0) {
                        level[v] = level[node] + 1;
                        q.push(v);
                    }
                }
            }
            return false;
        };
        function<bool(int)> augment = [&](int node) {
            for (auto i: adj[node]) {
                int v = mate[i];
                if (v < 0 || (level[v] > level[node] && augment(v))) {
                    mate[node] = i;
                    mate[i] = node;
                    return true;
                }
            }
            return false;
        };
        int match = 0;
        while (levelize())
            for (int i = 1; i <= L; i++)
                if (mate[i] < 0 && augment(i))
                    match++;
        return match;
    }
};
```

7.8 Hungarian Weighted matching (Bipartite)

```cpp
// Weighted Bipartite matching N^2 * M
// note that n must be <= m so in case in your problem n >= m, just swap
// also note this void set(int x, int y, ll v){a[x+1][y+1]=v;}
// the algorithim assumes you're using 0-index but it's using 1-based
struct Hungarian {
    const ll INF = 1000000000000000000; ///10^18
    int n,m;
    vector<vector<ll> > a;
    vector<ll> u,v;vector<int> p,way;
    Hungarian(int n, int m):
    n(n),m(m),a(n+1,vector<ll>(m+1,INF-1)),u(n+1),v(m+1),p(m+1),way(m+1){}
    void set(int x, int y, ll v){a[x+1][y+1]=v;}
    ll assign(){
        for(int i = 1; i <= n; i++){
            int j0=0;p[0]=i;
            vector<ll> minv(m+1,INF);
            vector<char> used(m+1,false);
            do {
                used[j0]=true;
                int i0=p[j0],j1;ll delta=INF;
                for(int j = 1; j <= m; j++)if(!used[j]){
                    ll cur=a[i0][j]-u[i0]-v[j];
                    if(cur<minv[j])minv[j]=cur,way[j]=j0;
                    if(minv[j]<delta)delta=minv[j],j1=j;
                }
                for(int j = 0; j <= m; j++)
                    if(used[j])u[p[j]]+=delta,v[j]-=delta;
                    else minv[j]-=delta;
                j0=j1;
            } while(p[j0]);
            do {
                int j1=way[j0];p[j0]=p[j1];j0=j1;
            } while(j0);
        }
        return -v[0];
    }
    vector<int> restoreAnswer() {   ///run it after assign
        vector<int> ans (n+1);
        for (int j=1; j<=m; ++j)
            ans[p[j]] = j;
        return ans;
    }
};
```

7.9 MinCostMaxFlow

```cpp
/*
    Notes:
        make sure you notice the #define int ll
        focus on the data types of the max flow everythign inside is integer
        addEdge(u,v,cap,cost)
        note that for min cost max flow the cost is sum of cost * flow over all edges
*/
struct Edge {
    int to;
    int cost;
    int cap, flow, backEdge;
};
struct MCMF {
    const int inf = 1000000010;
    int n;
    vector<vector<Edge>> g;
    MCMF(int _n) {
        n = _n + 1;
        g.resize(n);
    }
    void addEdge(int u, int v, int cap, int cost) {
```

```
        Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
        Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
        g[u].push_back(e1);
        g[v].push_back(e2);
    }
    pair<int, int> minCostMaxFlow(int s, int t) {
        int flow = 0;
        int cost = 0;
        vector<int> state(n), from(n), from_edge(n);
        vector<int> d(n);
        deque<int> q;
        while (true) {
            for (int i = 0; i < n; i++)
                state[i] = 2, d[i] = inf, from[i] = -1;
            state[s] = 1;
            q.clear();
            q.push_back(s);
            d[s] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop_front();
                state[v] = 0;
                for (int i = 0; i < (int) g[v].size(); i++) {
                    Edge e = g[v][i];
                    if (e.flow >= e.cap || (d[e.to] <= d[v] + e.cost))
                        continue;
                    int to = e.to;
                    d[to] = d[v] + e.cost;
                    from[to] = v;
                    from_edge[to] = i;
                    if (state[to] == 1) continue;
                    if (!state[to] || (!q.empty() && d[q.front()] > d[to]))
                        q.push_front(to);
                    else q.push_back(to);
                    state[to] = 1;
                }
            }
            if (d[t] == inf) break;
            int it = t, addflow = inf;
            while (it != s) {
                addflow = min(addflow,
                              g[from[it]][from_edge[it]].cap
                              - g[from[it]][from_edge[it]].flow);
                it = from[it];
            }
            it = t;
            while (it != s) {
                g[from[it]][from_edge[it]].flow += addflow;
                g[it][g[from[it]][from_edge[it]].backEdge].flow -= addflow;
                cost += g[from[it]][from_edge[it]].cost * addflow;
                it = from[it];
            }
            flow += addflow;
        }
        return {cost, flow};
    }
};
```

7.10 Push Relabel Max Flow

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

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

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

    void addEdge(int from, int to, int cap) {
        g[from].push_back(edge(from, to, cap, 0, g[to].size()));
        if (from == to)
            g[from].back().index++;
        g[to].push_back(edge(to, from, 0, 0, g[from].size() - 1));
    }
    void enqueue(int v) {
        if (!active[v] && excess[v] > 0) {
            active[v] = true;
            Q.push(v);
        }
    }
    void push(edge &e) {
        int amt = (int) min(excess[e.from], (long long) e.cap - e.flow);
        if (height[e.from] <= height[e.to] || amt == 0)
            return;
        e.flow += amt;
        g[e.to][e.index].flow -= amt;
        excess[e.to] += amt;
        excess[e.from] -= amt;
        enqueue(e.to);
    }
    void relabel(int v) {
        count[height[v]]--;
        int d = 2 * n;
        for (auto &it: g[v]) {
            if (it.cap - it.flow > 0)
                d = min(d, height[it.to] + 1);
        }
        height[v] = d;
        count[height[v]]++;
        enqueue(v);
    }
    void gap(int k) {
        for (int v = 0; v < n; v++) {
            if (height[v] < k)
                continue;
            count[height[v]]--;
            height[v] = max(height[v], n + 1);
            count[height[v]]++;
            enqueue(v);
        }
    }
    void discharge(int v) {
        for (int i = 0; excess[v] > 0 && i < g[v].size(); i++)
            push(g[v][i]);
        if (excess[v] > 0) {
            if (count[height[v]] == 1)
                gap(height[v]);
            else
                relabel(v);
        }
    }
    long long max_flow(int source, int dest) {
        count[0] = n - 1;
        count[n] = 1;
        height[source] = n;
        active[source] = active[dest] = 1;
        for (auto &it: g[source]) {
            excess[source] += it.cap;
            push(it);
        }
        while (!Q.empty()) {
            int v = Q.front();
            Q.pop();
            active[v] = false;
            discharge(v);
        }
        long long max_flow = 0;
        for (auto &e: g[source])
            max_flow += e.flow;

        return max_flow;
    }
};
```

7.11 Prufer Code

```
const int N = 3e5 + 9;
/*
prufer code is a sequence of length n-2 to uniquely determine a labeled tree with n vertices
Each time take the leaf with the lowest number and add the node number the leaf is connected to
the sequence and remove the leaf. Then break the algo after n-2 iterations
*/
//0-indexed
int n;
vector<int> g[N];
int parent[N], degree[N];

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

```cpp
}

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

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

7.12 Tarjan Algo

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

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

    return 0;
}
```

8 NumberTheory

8.1 ModSum (Sum Of floored division)

```cpp
// log(m), with a large constant.
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

// return sum_{i=0}^{to-1} floor((ki + c) / m) (mod 2^64)
ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
// return sum_{i=0}^{to-1} (ki+c) % m
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

8.2 ModMulLL

```cpp
// Calculate a^b % c and a*b % c
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

8.3 ModSqrt Finds x s.t $x^2 = a \mod p$

```cpp
// Description: Finds x s.t. x^2 = a mod p
// Time: O(log^2 p) worst case, O(log p) for most p
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    /// find a non-square mod p
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = modpow(a, (s + 1) / 2, p);
    ll b = modpow(a, s, p), g = modpow(n, s, p);
    for (;; r = m) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

8.4 MillerRabin Primality check

```cpp
#include "ModMulLL.h"
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
```

```
    s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {   // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

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

```
"ModMulLL.cpp", "MillerRabin.cpp"
ull pollard(ull n) {
    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
```

8.6 Primitive Roots

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

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

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

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

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

```
// This program finds all numbers x such that x^k = a (mod n)
vector<int> discrete_root(int n, int k, int a) {
    if (a == 0)
        return {0};

    int g = primitive_root(n);
    // Baby-step giant-step discrete logarithm algorithm
    int sq = (int) sqrt(n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i - 1] = {powmod(g, i * sq * k % (n - 1), n), i};
    sort(dec.begin(), dec.end());
    int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = powmod(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) return {};

    int delta = (n - 1) / __gcd(k, n - 1);
    vector<int> ans;
    for (int cur = any_ans % delta; cur < n - 1; cur += delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

8.9 Totient function

```
void phi_1_to_n(int n) {
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

8.10 CRT and EGCD

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

    return g ;
}
ll de(ll a, ll b, ll c, ll &x, ll &y) {
    ll g = extended(abs(a), abs(b), x, y);
    if(c % g) return -1;
    x *= c / g;
    y *= c / g;
    if(a < 0)x = -x;
    if(b < 0)y = -y;
    return g;
}
pair<ll, ll> CRT(vector<ll> r, vector<ll> m) {
    ll r1 = r[0], m1 = m[0];
    for(int i = 1; i < r.size(); i++) {
        ll r2 = r[i], m2 = m[i];
        ll x0, y0;
        ll g = de(m1, -m2, r2 - r1, x0, y0);
        if(g == -1) return {-1, -1} ;
        x0 %= m2;
        ll nr = x0 * m1 + r1;
        ll nm = m1 / g * m2;
        r1 = (nr % nm + nm) % nm;
        m1 = nm;
    }
    return {r1, m1};
}
```

8.11 Xor With Gauss

```
void insertVector(int mask) {
    for (int i = d - 1; i >= 0; i--) {
        if ((mask & 1 << i) == 0) continue;
        if (!basis[i]) {
            basis[i] = mask;
            return;
        }
        mask ^= basis[i];
    }
}
```

8.12 Josephus

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

9 Strings

9.1 Aho-Corasick Mostafa

```
struct AC_FSM {
#define ALPHABET_SIZE 26

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

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

    vector<Node> a;

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

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

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

9.2 KMP Anany

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

9.3 Manacher Kactl

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

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

// If the size of palindrome centered at i is x, then d2[i] stores x/2
vector<int> d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
        k++;
    }
    d2[i] = k--;
    if (i + k > r) {
        l = i - k - 1;
        r = i + k ;
    }
}
```

9.4 Suffix Array Kactl

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

        you should also print the suffix array and lcp at the beginning of the contest
        to clarify this stuff
    */
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
```

```
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                    s[i + k] == s[j + k]; k++);
    }
};
```

9.5 Suffix Automaton Mostafa

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

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

    int last, sz;
    vector<node> v;

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

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

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

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

9.6 Zalgo Anany

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

9.7 lexicographically smallest rotation of a string

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    }
    return a;
}
```

10 Trees

10.1 Centroid Decomposition

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

    int ans = 0;

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

    return ans;
}
```

10.2 Dsu On Trees

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

    ///modify node to data structure

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

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

10.3 Heavy Light Decomposition (Along with Euler Tour)

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

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

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

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

10.4 Mo on Trees

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

11 Numerical

11.1 Lagrange Polynomial

```cpp
class LagrangePoly {
public:
    LagrangePoly(std::vector<long long> _a) {
        //f(i) = _a[i]
        //interpola o vetor em um polinomio de grau y.size() - 1
        y = _a;
        den.resize(y.size());
        int n = (int) y.size();
        for(int i = 0; i < n; i++) {
            y[i] = (y[i] % MOD + MOD) % MOD;
            den[i] = ifat[n - i - 1] * ifat[i] % MOD;
            if((n - i - 1) % 2 == 1) {
                den[i] = (MOD - den[i]) % MOD;
            }
        }
    }

    long long getVal(long long x) {
        int n = (int) y.size();
        x = (x % MOD + MOD) % MOD;
        if(x < n) {
            //return y[(int) x];
        }
        std::vector<long long> l, r;
        l.resize(n);
        l[0] = 1;
        for(int i = 1; i < n; i++) {
            l[i] = l[i - 1] * (x - (i - 1) + MOD) % MOD;
        }
        r.resize(n);
        r[n - 1] = 1;
        for(int i = n - 2; i >= 0; i--) {
            r[i] = r[i + 1] * (x - (i + 1) + MOD) % MOD;
        }
        long long ans = 0;
        for(int i = 0; i < n; i++) {
            long long coef = l[i] * r[i] % MOD;
            ans = (ans + coef * y[i] % MOD * den[i]) % MOD;
```

```
        }
        return ans;
    }

private:
    std::vector<long long> y, den;
};
```

11.2 Polynomials

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) += a[i];
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
        a.pop_back();
    }
};

// Finds the real roots to a polynomial
// O(n^2 log(1/e))
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}

// Given n points (x[i], y[i]), computes an n-1-degree polynomial that passes through
    them.
// For numerical precision pick x[k] = c * cos(k / (n - 1) * pi).
// O(n^2)
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}

// Recovers any n-order linear recurrence relation from the first 2n terms of the
    recurrence.
// Useful for guessing linear recurrences after bruteforcing the first terms.
// Should work on any field, but numerical stability for floats is not guaranteed.
// O (n^2)
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }
    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
}

// Generates the kth term of an n-order linear recurrence
// S[i] = S[i - j - 1]tr[j], given S[0..>= n - 1] and tr[0..n - 1]
// Useful together with Berlekamp-Massey.
// O(n^2 * log(k))
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);
    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };
    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;
    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }
    ll res = 0;
    rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
    return res;
}
```

12 Guide

12.1 Strings

- Longest Common Substring is easier with suffix automaton
- Problems that tell you cound stuff that appears X times or count appearnces (Use suffixr 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

12.2 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 $= \frac{\pi h^2(3r-h)}{3}$

12.3 Graph Theory

- Euler formula: $v + f = e + 2$

12.4 Joseph problem

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

1 Contest 1

2 Mathematics 1

3 Numerical 2

4 Number theory 2

5 Combinatorial 2

6 Graph 2

7 Geometry 5

Contest (1)

template.cpp — 14 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc — 3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc — 6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh — 3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

Mathematics (2)

2.1 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$a\cos x + b \sin x = r \cos(x - \phi)$$
$$a\sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2+b^2}, \phi = \operatorname{atan2}(b, a)$.

2.2 Geometry

2.2.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a+b+c}{2}$

Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc\left[1 - \left(\frac{a}{b+c}\right)^2\right]}$$

Law of sines: $\dfrac{\sin\alpha}{a} = \dfrac{\sin\beta}{b} = \dfrac{\sin\gamma}{c} = \dfrac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Law of tangents: $\dfrac{a+b}{a-b} = \dfrac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

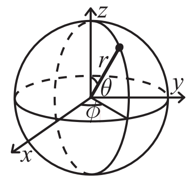
2.2.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin\theta = F \tan\theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.2.3 Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \operatorname{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \operatorname{atan2}(y, x)$$

2.3 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x = 1 + \tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.4 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.5 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

Numerical (3)

3.1 Polynomials and recurrences

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, \ldots, c_n\},$$
$$\{b_1, b_2, \ldots, b_n, 0\}, \{a_0, d_1, d_2, \ldots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

8f9fa8, 26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
}
```

Number theory (4)

4.1 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2),\ \ b = k \cdot (2mn),\ \ c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

Combinatorial (5)

5.1 Permutations

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

044568, 6 lines

```
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
    use |= 1 << x;                      // (note: minus, not ~!)
  return r;
}
```

5.1.1 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

5.2 Partitions and subsets

5.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	∼2e5	∼2e8

5.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

5.3 General purpose numbers

5.3.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k),\ c(0, 0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n, k) x^k = x(x+1)\ldots(x+n-1)$$

$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

5.3.2 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \ge j$, k j:s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

5.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

5.3.4 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

5.3.5 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

5.3.6 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2} C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (6)

6.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] =$ `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

6.2 Network flow

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

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

6.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

f612e4, 42 lines

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

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); dfsMatching(g, btoa);`
Time: $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
```

```
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}\left(N^3\right)$

"../numerical/MatrixInverse-mod.h" cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (pii pa : ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

  int r = matInv(A = mat), M = 2*N - r, fi, fj;
  assert(r % 2 == 0);

  if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
      mat[i].resize(M);
      rep(j,N,M) {
        int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
      }
    }
  } while (matInv(A = mat) != M);

  vi has(M, 1); vector<pii> ret;
  rep(it,0,M/2) {
    rep(i,0,M) if (has[i])
      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
        fi = i; fj = j; goto done;
    } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);
    has[fi] = has[fj] = 0;
    rep(sw,0,2) {
      ll a = modpow(A[fi][fj], mod-2);
      rep(i,0,M) if (has[i] && A[i][fj]) {
        ll b = A[i][fj] * a % mod;
        rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
      }
      swap(fi,fj);
    }
  }
  return ret;
}
```

6.4 DFS algorithms

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

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

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

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

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

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

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

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

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V+E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

6.5 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b0d5b1, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
```

```
        int k = 1;
        auto f = [&](int i) { return e[v.i][i]; };
        while (any_of(all(C[k]), f)) k++;
        if (k > mxk) mxk = k, C[mxk + 1].clear();
        if (k < mnk) T[j++].i = v.i;
        C[k].push_back(v.i);
      }
      if (j > 0) T[j - 1].d = 0;
      rep(k,mnk,mxk + 1) for (int i : C[k])
        T[j].i = i, T[j++].d = k;
      expand(T, lev + 1);
    } else if (sz(q) > sz(qmax)) qmax = q;
    q.pop_back(), R.pop_back();
  }
}
vi maxClique() { init(V), expand(V); return qmax; }
Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
  rep(i,0,sz(e)) V.push_back({i});
}
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

6.6 Math

6.6.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--, mat[b][b]++` (and `mat[b][a]--, mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

Geometry (7)

7.1 Geometric primitives

Point.h

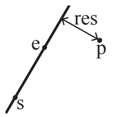
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

Description:



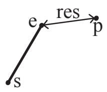
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:



Returns the shortest distance between point p and the line segment from point s to e.

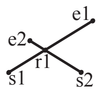
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:



If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

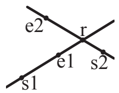
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description:



If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

Usage: `auto res = lineInter(s1,e1,s2,e2);`
`if (res.first == 1)`
`cout << "intersection point at " << res.second << endl;`

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: `bool left = sideOf(p1,p2,q)==1;`

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

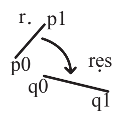
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h

Description:



Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively`
`oriented triangles with vertices at 0 and i`

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

7.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

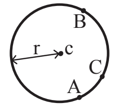
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

Description:



The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

7.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`

Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

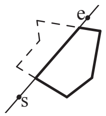
PolygonCenter.h

Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h



Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

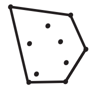
PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" 3931c6, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h



Description:
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
```

```
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

ManhattanMST.h

Description: Given N points, returns up to 4*N edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights w(p, q) = —p.x - q.x— + —p.y - q.y—. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.
Time: $\mathcal{O}(N \log N)$

"Point.h" df6f59, 23 lines

```
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
         return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
                it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
  }
  return edges;
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . . }, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" eefdf5, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

7.5 3D

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

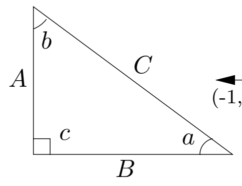
```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

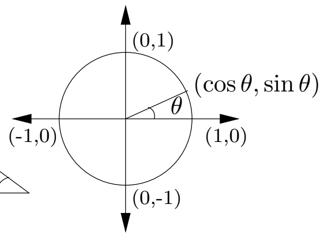
sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g\begin{vmatrix} b & c \\ e & f \end{vmatrix} - h\begin{vmatrix} a & c \\ d & f \end{vmatrix} + i\begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ - ceg - fha - ibd. \end{matrix}$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

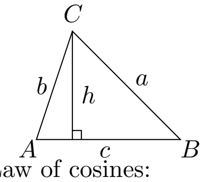
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2 \sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

... in mathematics you don't understand things, you just get used to them.
– J. von Neumann



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a+b+c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$