



Politecnico di Torino

# OverkillaTO

Nima Naderi Ghotbodini, Jiakai Hu, Pasquale Bianco

ICPC-SWERC 2025/26

Novemeber 21-23, 2025

## 1 Contest

## 2 Mathematics

## 3 Data structures

## 4 Numerical

## 5 Number theory

## 6 Combinatorial

## 7 Graph

## 8 Geometry

## 9 Strings

## 10 Various

# Contest (1)

### template.cpp

20 lines

```
// #pragma GCC optimize("O2")
#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()
#define MSB(x) (63 - __builtin_clzll(ll(x)))
#define bit(x, b) ((x >> b) & 1)
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
template<typename T>
using vec = vector<T>

int32_t main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin.exceptions(cin.failbit);
}

//bs...Find_first() ...Find_next(i) .set() .reset() .count()
```

### .bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
-fsanitize=undefined,address'
```

### gen.cpp

6 lines

```
ll rand(ll l, ll r) { return l + rand() % (r - l + 1); }
int32_t main(int argc, char* argv[]){
    srand(atoi(argv[1]));
    ll n = rand(2, 20); //input size
    cout << "TEST" << '\n';
}
```

### tester.sh

11 lines

```
g++ sol.cpp -std=c++17 -o main
g++ naive.cpp -std=c++17 -o naive
g++ gen.cpp -std=c++17 -o gen
for((i = 1; ; ++i)); do
    echo test case: $i
    ./gen $i > input
    ./naive < input > output1
    ./main < input > output2
    diff -w output1 output2 || break
    echo passed!
done
```

### .vimrc

20 lines

```
set autoindent
set cindent
set smartindent
set nu
set ts=4
set sw=4
set tabstop=4
set hlsearch
```

```
inoremap ( ()<left>
inoremap { {}<left>
inoremap [ []<left>
inoremap , ''<left>
inoremap " ""<left>
```

```
syntax on
" colorscheme habamax
colorscheme darkblue
" colorscheme zaibatsu
```

### 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 -c6
```

### flags.txt

4 lines

```
// useful g++ flags
// Wshadow for variable shadowing warning, _GLIBCXX_ASSERTIONS
// for OOB/bad iterator errors
-Wall -Wextra -Wshadow -D_GLIBCXX_DEBUG -D_GLIBCXX_ASSERTIONS -
fdiagnostics-color=always
```

### troubleshoot.txt

52 lines

#### Pre-submit:

Write a few simple test cases if sample is not enough.  
Are time limits close? If so, generate max cases.  
Is the memory usage fine?  
Could anything overflow?  
Make sure to submit the right file.

#### Wrong answer:

Print your solution! Print debug output, as well.  
Are you clearing all data structures between test cases?  
Can your algorithm handle the whole range of input?  
Read the full problem statement again.  
Do you handle all corner cases correctly?  
Have you understood the problem correctly?  
Any uninitialized variables?  
Any overflows?

Confusing N and M, i and j, etc.?  
Are you sure your algorithm works?

What special cases have you not thought of?  
Are you sure the STL functions you use work as you think?  
Add some assertions, maybe resubmit.  
Create some testcases to run your algorithm on.  
Go through the algorithm for a simple case.  
Go through this list again.  
Explain your algorithm to a teammate.  
Ask the teammate to look at your code.  
Go for a small walk, e.g. to the toilet.  
Is your output format correct? (including whitespace)  
Rewrite your solution from the start or let a teammate do it.

Runtime error:  
Have you tested all corner cases locally?  
Any uninitialized variables?  
Are you reading or writing outside the range of any vector?  
Any assertions that might fail?  
Any possible division by 0? (mod 0 for example)  
Any possible infinite recursion?  
Invalidated pointers or iterators?  
Are you using too much memory?  
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:  
Do you have any possible infinite loops?  
What is the complexity of your algorithm?  
Are you copying a lot of unnecessary data? (References)  
How big is the input and output? (consider scanf)  
Avoid vector, map. (use arrays/unordered\_map)  
What do your teammates think about your algorithm?

Memory limit exceeded:  
What is the max amount of memory your algorithm should need?  
Are you clearing all data structures between test cases?

# Mathematics (2)

## 2.1 Geometry

### 2.1.1 Triangles

Side lengths:  $a, b, c$

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

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

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

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

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):

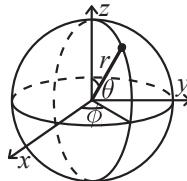
$$s_a = \sqrt{bc \left[ 1 - \left( \frac{a}{b+c} \right)^2 \right]}$$

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

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

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

## 2.1.2 Spherical coordinates



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \arccos(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \arctan(y/x) \end{aligned}$$

## 2.2 Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

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

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

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

## First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability  $p$  is  $Fs(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

## Poisson distribution

The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $Po(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

### 2.2.1 Continuous distributions

## Data structures (3)

### OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the  $n$ 'th element, and finding the index of an element. To get a map, change `null_type`.  
**Time:**  $\mathcal{O}(\log N)$

d41d8c, 22 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

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

```
##include <ext/pb_ds/assoc_container.hpp>
##include <ext/pb_ds/tree_policy.hpp> //commented for ump
#define ordered_set ll tree<ll, null_type,less<ll>, rb_tree_tag,
tree_order_statistics_node_update>
template <typename T> using ordered_set = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;
```

```
unordered_map<ll, tree<pair<ll, ll>, null_type, less<pair<ll,
ll>>, rb_tree_tag, tree_order_statistics_node_update>> ump
;
```

```
void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

### HashMap.h

**Description:** Hash map with mostly the same API as `unordered_map`, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d41d8c, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 114018 * acos(0) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
```

```
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{{},{}},1<<16});
```

### SegmentTree.h

**Description:** Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying `T`, `f` and `unit`.

Time:  $\mathcal{O}(\log N)$ 

d41d8c, 19 lines

```
struct Tree {
    typedef int T;
    static constexpr T unit = INT_MIN;
    T f(T a, T b) { return max(a, b); } // (any associative fn)
    vector<T> s; int n;
    Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
    }
    T query(int b, int e) { // query [b, e)
        T ra = unit, rb = unit;
        for (b += n, e += n; b < e; b /= 2, e /= 2) {
            if (b % 2) ra = f(ra, s[b++]);
            if (e % 2) rb = f(s[--e], rb);
        }
        return f(ra, rb);
    }
}
```

```
}
```

```
return f(ra, rb);
}
```

### LazySegmentTree.h

**Description:** Segment tree with ability to add or set values of large intervals, and compute max of intervals.

Time:  $\mathcal{O}(N \log N)$ 

d41d8c, 34 lines

```
#define lc (id * 2)
#define rc (id * 2 + 1)
#define ln (e - s + 1)
#define md ((s + e) / 2)
#define dm ((s + e) / 2 + 1)
const ll MXS = MXN * 4 + 10;
ll n, q, A[MXN], seg[MXS];
ll Max[MXS], Min[MXS], lazySet[MXS], lz[MXS];
void Build(ll id = 1, ll s = 1, ll e = n) {
    if(ln == 1){ /* something */ return; }
    Build(lc, s, md), Build(rc, dm, e);
    // merge
}
void Shift(ll id = 1, ll s = 1, ll e = n) {
    if(!lz[id]) return;
    //apply lazy
    if(ln > 1){ lz[lc] += lz[id], lz[rc] += lz[id]; }
    lz[id] = 0;
}
void Upd(ll l, ll r, ll x, ll id = 1, ll s = 1, ll e = n) {
    Shift(id, s, e);
    if(e < l || s > r) return;
    if(l <= s && e <= r) /*Max[id] == Min[id]*/ {
        lz[id] = x; Shift(id, s, e); return;
    }
    Upd(l, r, x, lc, s, md), Upd(l, r, x, rc, dm, e);
    // merge
}
ll Get(ll l, ll r, ll id = 1, ll s = 1, ll e = n) {
    Shift(id, s, e);
    if(e < l || s > r) return 0; // Null
    if(l <= s && e <= r) return seg[id];
    return Get(l, r, lc, s, md) + Get(l, r, rc, dm, e);
}
```

### PersistantSegTree.h

**Description:** Persistant segtree, with having times of queries

Time:  $\mathcal{O}(N \log N)$ 

d41d8c, 47 lines

```
const int LOG = 17;
const int MXS = MXN * 4 * LOG;
struct DATA{
    int ans;
    DATA(){ ans = 0; }
};
DATA Merge(const DATA& a, const DATA& b){}
ll n, ts, ort, Root[MXN]; //MXN refers to different time slots
DATA seg[MXS]; int LC[MXS], RC[MXS];
ll New(ll id){
    ll nd = ++ts;
    LC[nd] = LC[id], RC[nd] = RC[id], seg[nd] = seg[id];
    return nd;
}
ll Build(ll s = 1, ll e = n) {
    ll nd = New(0);
    if(ln == 1){ seg[nd].sz = 1; return nd; }
    LC[nd] = Build(s, md), RC[nd] = Build(dm, e);
    seg[nd] = Merge(seg[LC[nd]], seg[RC[nd]]);
    return nd;
}
```

```

}
11 Upd(11 p, 11 id = ort, 11 s = 1, 11 e = n){
    if(ln == 1){ /* update node part */
        seg[nd].ans = 1; return nd;
    }
    if(p <= md) LC[nd] = Upd(p, LC[id], s, md);
    else RC[nd] = Upd(p, RC[id], dm, e);
    seg[nd] = Merge(seg[LC[nd]], seg[RC[nd]]);
    return nd;
}

DATA Get(11 l, 11 r, 11 id, 11 s = 1, 11 e = n){
    if(l <= s && e <= r) return seg[id];
    if(r <= md) return Get(l, r, LC[id], s, md);
    if(l >= dm) return Get(l, r, RC[id], dm, e);
    return Merge(Get(l, r, LC[id], s, md), Get(l, r, RC[id], dm, e));
}

int main(){
    ort = Root[1] = Build(); //Build the base
    vector<11> Qry[MXN]; rep(i, 1, n){
        for(auto X: Qry[i]) ort = Upd(X);
        Root[i] = ort; //Save root at spesific times
    }
    // Solution: Binary search on the timeline
    //if(Get(l, r, Root[mid]).ans >= k)
    //Idea: Somtimes Binary search on segment tree is possible
}

```

**2FenInsteadOfSegment.h**

**Description:** Does advanced queries with two fenwicks  
**Time:**  $\mathcal{O}(\log N)$ .

d41d8c, 21 lines

```

11 Fen[2][MXN];
void Add(int id, int p, int x){
    for(; p < MXN; p += p & -p) Fen[id][p] += x;
}

11 Ask(int id, int p){
    11 niw = 0;
    for(; p; p -= p & -p) niw += Fen[id][p];
    return niw;
}

void UpdFen(int l, int r, int x){
    Add(0, l, x);
    Add(0, r + 1, -x);
    Add(1, l, x * (l - 1));
    Add(1, r + 1, -x * r);
}

11 Get(int r){
    return Ask(0, r) * r - Ask(1, r);
}

11 GetFen(int l, int r){
    return Get(r) - Get(l - 1);
}

```

**Matrix.h**

**Description:** Basic operations on square matrices.  
**Usage:** Matrix M = Matrix(2, 1)

MAX.M[1][0]

d41d8c, 65 lines

```

const 11 MXZ = 100 + 10;
struct Matrix{
    int n, m; 11 M[MXZ][MXZ];
    Matrix(int _n, int _m, 11 num = 0){
        n = _n, m = _m;
        if(num == -1)
            for(int i = 0; i < n; i++)
                for(int j = 0; j < m; j++) M[i][j] = (i == j);
        else
    }
}

```

**2FenInsteadOfSegment Matrix LineContainer FenwickTree XorHashingFenwick**

```

        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++) M[i][j] = num;
    }

    void Print(bool f = 0){
        cerr << "=====N.N=====\\n";
        if(f) cerr << "Size : " << n << ' ' << m << '\\n';
        for(int i = 0; i < n; i++, cerr << '\\n'){
            for(int j = 0; j < m; j++) cerr << M[i][j] << ' ';
        }
        cerr << "=====N.N=====\\n";
    }

    Matrix operator + (const Matrix &T){
        Matrix R = Matrix(n, m);
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                R.M[i][j] = (M[i][j] + T.M[i][j]) % Mod;
        return R;
    }

    Matrix operator * (const Matrix &T){
        Matrix R = Matrix(n, T.m);
        if(m != T.n){
            cerr << "Cannot * Matrices !" << '\\n'; return R;
        }

        for(int i = 0; i < n; i++){
            for(int j = 0; j < T.m; j++){
                for(int k = 0; k < m; k++){
                    11 now = (Ok(M[i][k]) * Ok(T.M[k][j])) %
                        Mod;
                    R.M[i][j] = (R.M[i][j] + now) % Mod;
                }
            }
        }
        return R;
    }

    Matrix operator ^ (const 11 t){
        Matrix R = Matrix(n, m, -1);
        if(n != m){
            cerr << "Cannot ^ Matrice with n != m !" << '\\n'; return R;
        }

        if(t == 0) return R;
        Matrix T = Matrix(n, m);
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                T.M[i][j] = M[i][j];

        11 p = t;
        while(p){
            if(p & 1LL) R = R * T;
            T = (T * T), p /= 2;
        }
        return R;
    }

    11 Fib(11 t){
        if(t == 1) return 1; Matrix M(2, 1), C(2, 2, 1);
        M.M[0][0] = M.M[1][0] = 1; C.M[1][1] = 0;
        C = (C ^ (t - 2)); M = (C * M); return M.M[0][0];
    }
}

```

**LineContainer.h**

**Description:** Container where you can add lines of the form  $kx+m$ , and query maximum values at points x. Useful for dynamic programming ("convex hull trick").

**Time:**  $\mathcal{O}(\log N)$

d41d8c, 30 lines

```

struct Line {
    mutable 11 k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(11 x) const { return p < x; }
}

```

```

};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const 11 inf = LLONG_MAX;
    11 div(11 a, 11 b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if(y == end()) return x->p = inf, 0;
        if(x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(11 k, 11 m) {
        auto z = insert({k, m, 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));
    }
    11 query(11 x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

```

**FenwickTree.h**

**Description:** Computes partial sums

**Time:**  $\mathcal{O}(\log N)$

```

11 Fen[MXN]; //Be careful with Mod
void Upd(11 p, 11 x){
    for(; p < MXN; p += p & -p) Fen[p] += x;
} // Call fill(Fen, Fen + n + 5, 0) in main
void Upd(11 l, 11 r, 11 x){
    Upd(l, x), Upd(r + 1, -x);
}

11 Get(11 p){
    11 s = 0; for(; p; p -= p & -p) s += Fen[p]; return s;
}

11 Get(11 l, 11 r){
    return (r < l ? 0 : Get(r) - Get(l - 1));
}

11 Find(11 k){
    11 ans = 0;
    for(int i = LOG; ~i; i--){
        if(ans + (1LL << i) < MXN && Fen[ans + (1LL << i)] >= k)
            ans += (1LL << i), k -= Fen[ans];
    }
    return ans + 1;
}

```

**XorHashingFenwick.h**

**Description:** Random number generators and fenwick on it with xor

```

const 11 MXN = 3e5 + 10;
const 11 MXK = 32; //16
//Random number generation (64 bits is safe):
// mt19937_64 rng(chrono::steady_clock::now().time_since_epoch()
// .count());
// mt19937 Rnd(chrono::high_resolution_clock::now().
// time_since_epoch().count());
// mt19937 rng(time(0)); // mt19937 mrand(19260817);
// srand(time(0)); srand(time(NULL));
// ll uid(ll a, ll b) { return uniform_int_distribution<ll>(a,
// b)(Rnd); }

```

```

typedef unsigned long long u64;
const valarray<u64> zero(0ull, MXK);

static uint64_t hsh(uint64_t x){
    // http://xorshift.di.unimi.it/splitmix64.c
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbff58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb13311leb;
    return x ^ (x >> 31);
}

valarray<u64> to_arr(u64 x){
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::
        now().time_since_epoch().count();
    x += FIXED_RANDOM;
    valarray<u64> ans(0ull, MXK);
    for(ll i = 0; i < MXK; i++)
        ans[i] = hsh(x + i * i * i + i) % (1ull << 32);
    return ans;
}

valarray<u64> Fen[MXN];
inline void initFen(){ //boundry can be n itself
    for(int i = 0; i < MXN; i++) Fen[i] = zero;
}
void Upd(ll p, valarray<u64> x){
    for(); p < MXN; p += p & -p) Fen[p] += x;
}
valarray<u64> Get(ll p){
    valarray<u64> s(0ull, MXK); for(; p; p -= p & -p) s += Fen[
        p]; return s;
}

ll n;
valarray<u64> A[MXN];

struct ValarrayComparator {
    bool operator()(const valarray<u64>& a, const valarray<u64>
        & b) const {
        assert(a.size() == MXK && b.size() == MXK);
        for(size_t i = 0; i < MXK; ++ i){
            if(a[i] != b[i]) return a[i] < b[i];
        }
        return false;
    }
}; //Check mod k for understanding if it appeared kt times
map<valarray<u64>, ll, ValarrayComparator> mp;

int32_t main(){
    initFen();
    cin >> n;
    for(int i = 1; i <= n; i++){
        ll x; cin >> x;
        A[i] = to_arr(x); //auto y = ..
    }
}

```

## FenwickTree2d.h

**Description:** Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).

**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

"FenwickTree.h" d41d8c, 22 lines

```

struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }
}

```

## FenwickTree2d RMQ MoQueries PolyArithmetic

```

void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
}
int ind(int x, int y) {
    return (int)lower_bound(all(ys[x]), y) - ys[x].begin();
}
void update(int x, int y, ll dif) {
    for (x < sz(ys); x |= x + 1)
        ft[x].update(ind(x, y), dif);
}
ll query(int x, int y) {
    ll sum = 0;
    for (x; x &= x - 1)
        sum += ft[x-1].query(ind(x-1, y));
    return sum;
}

```

## RMQ.h

**Description:** Range Minimum Queries on an array (constant time). d41d8c, 19 lines

```

int rmq[LOG][MXN], lg[MXN];
inline void BuildRmq(){
    for(int i = 1; i <= n; i++) rmq[0][i] = A[i];
    for(int j = 1; j < LOG; j++){
        for(int i = 1; i <= n; i++){
            if(i < (1LL << j)) continue;
            rmq[j][i] = max(rmq[j - 1][i], rmq[j - 1][i - (1LL
                << (j - 1))]);
        }
    }
    inline int Max(int l, int r){
        if(l == r) return rmq[0][l];
        if(r < l) swap(l, r);
        return max(rmq[lg[r - 1 + 1]][r], rmq[lg[r - 1 + 1]][l + (1
            LL << lg[r - 1 + 1]) - 1]);
    }
    inline void InitLog(){
        for(int i = 0; (1LL << i) < MXN; i++) lg[(1LL << i)] = i;
        for(int i = 1; i < MXN; i++) lg[i] = max(lg[i - 1],
            lg[i]);
    } //Call InitLog() & BuildRmq();
}

```

## MoQueries.h

**Description:** Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge (a,c) and remove the initial add call (but keep in). **Time:**  $\mathcal{O}(N\sqrt{Q})$  d41d8c, 23 lines

```

bool CMP(int x, int y){
    if(Ql[x] / SQR == Ql[y] / SQR) return (Qr[x] < Qr[y]);
    return (Ql[x] / SQR < Ql[y] / SQR);
}
bool Cmp(int x, int y){
    if(Ql[x] / SQR == Ql[y] / SQR) return ((Ql[x] / SQR) % 2) ^
        (Qr[x] < Qr[y]);
    return (Ql[x] / SQR < Ql[y] / SQR);
}
void Add(ll x){ x = A[x]; /* ... */ }
void Ers(ll x){ x = A[x]; /* ... */ }
int main(){
    for(int i = 1; i <= q; i++) cin >> Ql[i] >> Qr[i], vec.
        push_back(i);
    sort(vec.begin(), vec.end(), Cmp);
    ll Ml = 1, Mr = 0;
    for(auto id : vec){
        int l = Ql[id], r = Qr[id];
        while(Mr < r) Mr ++, Add(Mr);
        while(l < Ml) Ml --, Add(Ml);
        while(Mr > r) Ers(Mr), Mr --;
        while(Ml < l) Ers(Ml), Ml++;
    }
    return 0;
}

```

## Numerical (4)

### 4.1 Polynomials and recurrences

#### PolyArithmetic.h

**Description:** Inputs must be in [0, mod).

**Time:**  $\mathcal{O}(N \log N)$  for more complex operations (high constant),  $\mathcal{O}(N)$  for simple operations d41d8c, 66 lines

```

using poly = vector<Z>;
void fix(poly &a) {
    while (a.size() and a.back().x == 0) a.pop_back();
}
poly trunc(poly a, int k) {
    poly f = a;
    f.resize(k);
    return f;
}
poly add(poly a, poly b) {
    int n = max(a.size(), b.size());
    poly res(n);
    for (int i = 0; i < (int)a.size(); i++) res[i] += a[i];
    for (int i = 0; i < (int)b.size(); i++) res[i] += b[i];
    return res;
}
poly sub(poly a, poly b) {
    int n = max(a.size(), b.size());
    poly res(n);
    for (int i = 0; i < (int)a.size(); i++) res[i] += a[i];
    for (int i = 0; i < (int)b.size(); i++) res[i] -= b[i];
    return res;
}
poly mult(poly a, poly b) {
    fix(a), fix(b);
    return conv(a, b);
}
poly deriv(poly a) {
    if (a.empty()) return a;
    poly res(a.size() - 1);
    for (int i = 0; i < a.size() - 1; ++i) res[i] = a[i + 1] *
        (i+1);
    return res;
}
poly integr(poly a) {
    poly res(a.size() + 1);
    for (int i = 0; i < a.size(); ++i) res[i + 1] = a[i] / (i +
        1);
    return res;
}
poly inv(poly a, int m) {
    poly x{Z(1)/a[0]};
    int k = 1;
}
```

```

while (k < m) k *= 2, x = trunc(mult(x, (sub(poly{2}, mult(
    trunc(a, k), x)))), k);
return trunc(x, m);
}

poly log(poly a, int m) {
    poly res = mult(deriv(a), inv(a, m));
    res = trunc(integr(res), m);
    return res;
}

poly exp(poly a, int m) {
    poly x{1};
    int k = 1;
    while (k < m) k *= 2, x = trunc(mult(x, add(sub(poly{1},
        log(x, k)), trunc(a, k))), k);
    return trunc(x, m);
}
// for pow shift until first non zero coeff, then reshift back,
p^x = exp(x * log p)

```

**PolyRoots.h****Description:** Finds the real roots to a polynomial.**Usage:** polyRoots({{2,-3,1}},-1e9,1e9) // solve  $x^2-3x+2 = 0$ **Time:**  $\mathcal{O}(n^2 \log(1/\epsilon))$ **Polynomial.h** d41d8c, 23 lines

```

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;
}

```

**PolyInterpolate.h****Description:** Given  $n$  points  $(x[i], y[i])$ , computes an  $n-1$ -degree polynomial  $p$  that passes through them:  $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$ . For numerical precision, pick  $x[k] = c * \cos(k/(n-1) * \pi)$ ,  $k = 0 \dots n-1$ .**Time:**  $\mathcal{O}(n^2)$ 

d41d8c, 13 lines

```

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;
}

```

} BerlekampMassey.h

**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $2n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size  $\leq n$ .

**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}

**Time:**  $\mathcal{O}(N^2)$

../number-theory/ModPow.h

d41d8c, 20 lines

```

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;
}

```

LinearRecurrence.h

**Description:** Generates the  $k$ 'th term of an  $n$ -order linear recurrence  $S[i] = \sum_j S[i-j-1]tr[j]$ , given  $S[0 \dots \geq n-1]$  and  $tr[0 \dots n-1]$ . Faster than matrix multiplication. Useful together with Berlekamp-Massey.

**Usage:** linearRec({0, 1}, {1, 1}, k) //  $k$ 'th Fibonacci number

**Time:**  $\mathcal{O}(n^2 \log k)$

d41d8c, 26 lines

```

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 /= 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;
}

```

**4.2 Optimization****GoldenSectionSearch.h****Description:** Finds the argument minimizing the function  $f$  in the interval  $[a, b]$  assuming  $f$  is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is  $\epsilon$ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.**Usage:** double func(double x) { return 4+x+.3\*x\*x; }**Time:**  $\mathcal{O}(\log((b-a)/\epsilon))$ 

d41d8c, 14 lines

```

double gss(double a, double b, double (*f)(double)) {
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}

```

**HillClimbing.h****Description:** Poor man's optimization for unimodal functions

d41d8c, 14 lines

```

typedef array<double, 2> P;

template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}

```

**Integrate.h****Description:** Simple integration of a function over an interval using Simpson's rule. The error should be proportional to  $h^4$ , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

d41d8c, 7 lines

```

template<class F>
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
    rep(i,1,n*2)
        v += f(a + i*h) * (i&1 ? 4 : 2);
    return v * h / 3;
}

```

**IntegrateAdaptive.h****Description:** Fast integration using an adaptive Simpson's rule.**Usage:** double sphereVolume = quad(-1, 1, [&](double x) {
 return quad(-1, 1, [&](double y) {
 return quad(-1, 1, [&](double z) {
 return x\*x + y\*y + z\*z < 1; });
 });
});

d41d8c, 15 lines

```

typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>

```

```
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}

template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

## Simplex.h

**Description:** Solves a general linear maximization problem: maximize  $c^T x$  subject to  $Ax \leq b$ ,  $x \geq 0$ . Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of  $c^T x$  otherwise. The input vector is set to an optimal  $x$  (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that  $x = 0$  is viable.

**Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}};

vd b = {1,1,-4}, c = {-1,-1}, x;

T val = LPSolver(A, b, c).solve(x);

**Time:**  $\mathcal{O}(NM * \#pivots)$ , where a pivot may be e.g. an edge relaxation.  $\mathcal{O}(2^n)$  in the general case.

d41d8c, 68 lines

```
typedef double T; // long double, Rational, double + modP>...
typedef vector<T> vd;
typedef vector<vd> vvd;
```

```
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
```

```
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
```

```
struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
            rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
            rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
            rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m+1][n] = 1;
    }

    void pivot(int r, int s) {
        T *a = D[r].data(), inv = 1 / a[s];
        rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
            T *b = D[i].data(), inv2 = b[s] * inv;
            rep(j,0,n+2) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
        rep(j,0,n+2) if (j != s) D[r][j] *= inv;
        rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }
}
```

```
bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] > -eps) return true;
        int r = -1;
        rep(i,0,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
```

```
< MP(D[r][n+1] / D[r][s], B[r])) r = i;
        if (r == -1) return false;
        pivot(r, s);
    }

    T solve(vd &x) {
        int r = 0;
        rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
        if (D[r][n+1] < -eps) {
            pivot(r, n);
            if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
            rep(i,0,m) if (B[i] == -1) {
                int s = 0;
                rep(j,1,n+1) ltj(D[i]);
                pivot(i, s);
            }
        }
        bool ok = simplex(1); x = vd(n);
        rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
        return ok ? D[m][n+1] : inf;
    }
}
```

## 4.3 Matrices

### Determinant.h

**Description:** Calculates determinant of a matrix. Destroys the matrix.

**Time:**  $\mathcal{O}(N^3)$

d41d8c, 15 lines

```
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;
}
```

### IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

**Time:**  $\mathcal{O}(N^3)$

d41d8c, 18 lines

```
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;
    }
}
```

### SolveLinear.h

**Description:** Solves  $A * x = b$ . If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in  $A$  and  $b$  is lost.

**Time:**  $\mathcal{O}(n^2m)$

d41d8c, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
```

```
int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);
```

```
rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
        if ((v = fabs(A[r][c])) > bv)
            br = r, bc = c, bv = v;
    if (bv <= eps) {
        rep(j,i,n) if (fabs(b[j]) > eps) return -1;
        break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
        double fac = A[j][i] * bv;
        b[j] -= fac * b[i];
        rep(k,i+1,m) A[j][k] -= fac * A[i][k];
    }
    rank++;
}

x.assign(m, 0);
for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
}
return rank; // (multiple solutions if rank < m)
```

### SolveLinear2.h

**Description:** To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

```
"SolveLinear.h"
d41d8c, 7 lines
rep(j,0,n) if (j != i) // instead of rep(j, i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:;
```

### SolveLinearBinary.h

**Description:** Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .

**Time:**  $\mathcal{O}(n^2m)$

d41d8c, 34 lines

typedef bitset<1000> bs;

```
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
```

```

if (br == n) {
    rep(j,i,n) if(b[j]) return -1;
    break;
}
int bc = (int)A[br]._Find_next(i-1);
swap(A[i], A[br]);
swap(b[i], b[br]);
swap(col[i], col[bc]);
rep(j,0,n) if (A[j][i] != A[j][bc]) {
    A[j].flip(i); A[j].flip(bc);
}
rep(j,i+1,n) if (A[j][i]) {
    b[j] ^= b[i];
    A[j] ^= A[i];
}
rank++;
}

x = bs();
for (int i = rank; i--) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
}
return rank; // (multiple solutions if rank < m)
}

```

## MatrixInverse.h

**Description:** Invert matrix  $A$ . Returns rank; result is stored in  $A$  unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set  $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$  where  $A^{-1}$  starts as the inverse of  $A$  mod p, and k is doubled in each step.

**Time:**  $\mathcal{O}(n^3)$

d41d8c, 35 lines

```

int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i;
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
        swap(col[i], col[c]);
        double v = A[i][i];
        rep(j,i+1,n)
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k,i+1,n) A[j][k] -= f*A[i][k];
            rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
}

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
}

rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
}

```

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

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

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

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

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  $\text{tr}$  nor the check for  $\text{diag}[i] == 0$  is needed.

**Time:**  $\mathcal{O}(N)$

d41d8c, 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] = 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;
}

```

## 4.4 Fourier transforms

## FastFourierTransform.h

**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.

**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

d41d8c, 35 lines

```

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% faster if 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]; // (25% faster if hand-rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i,0,sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}

```

## FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod}]$ .

**Time:**  $\mathcal{O}(N \log N)$ , where  $N = |A| + |B|$  (twice as slow as NTT or FFT)

"FastFourierTransform.h"

```

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) / li;
    }
    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;
}

```

## NumberTheoreticTransform.h

**Description:** ntt(a) computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(mod-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^a b + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod.  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod}]$ .

**Time:**  $\mathcal{O}(N \log N)$

"..../number-theory/ModPow.h"

```

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> v1;
void ntt(v1& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static v1 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];
            ai += a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z) >= mod ? z - mod : z;
        }
}
v1 conv(const v1 &a, const v1 &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);
    v1 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};
}
```

## FastSubsetTransform.h

**Description:** Transform to a basis with fast convolutions of the form  $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$ , where  $\oplus$  is one of AND, OR, XOR. The size of  $a$  must be a power of two.

**Time:**  $\mathcal{O}(N \log N)$

d41d8c, 16 lines

```
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
                pii(u + v, u - v); // XOR
        }
        if (inv) for (int& x : a) x /= sz(a); // XOR only
    }
    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;
    }
}
```

## Number theory (5)

### 5.1 Modular arithmetic

#### ModularArithmetic.h

**Description:** Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

d41d8c, 22 lines

```
const ll mod = 998244353;
struct mint {
    ll x;
```

```
    mint() : x(0) {}
    mint(ll xx) : x(xx) {}
    mint operator+(mint b) { return mint((x + b.x) % mod); }
    mint operator-(mint b) { return mint((x - b.x + mod) % mod); }
    mint operator*(mint b) { return mint((x * b.x) % mod); }
    mint operator/(mint b) { return *this * invert(b); }
    // mint operator+=(mint b) { return *this = *this + b; } // can be ignored if not necessary
    // mint operator-=(mint b) { return *this = *this - b; }
    // mint operator*=(mint b) { return *this = *this * b; }
    // mint operator/=(mint b) { return *this = *this / b; }
    mint invert(mint a) { // euclid for non prime mod
        return a ^ (mod - 2);
    }
    mint operator^(ll e) {
        if (!e) return mint(1);
        mint r = *this ^ (e / 2); r = r * r;
        return e & 1 ? *this * r : r;
    }
};
```

#### ModInverseAndExtendedGCD.h

**Description:** inverse of x mod Mod iff gcd(Mod, x) = 1

d41d8c, 26 lines

```
ll gcd(ll x, ll y){
    return (!y ? x : gcd(y, x % y));
}
// Extended euclidean algorithm - Bezout's identity
tuple<ll, ll, ll> extended_gcd(ll a, ll b){
    if(b == 0) return {a, 1, 0};
    auto [gcd, x1, y1] = extended_gcd(b, a % b);
    return {gcd, y1, x1 - (a / b) * y1};
}
ll inv(ll x, ll Mod){ // return 0 if doesn't exist
    auto [g, ix, _] = extended_gcd(x, Mod);
    if (g != 1) return 0;
    return (ix % Mod + Mod) % Mod;
}
void init() //inverse of x mod Mod
{
    ll p = 2, m = Mod; phi = m;
    while(p * p <= m){
        if(m % p == 0){
            while(m % p == 0) m /= p;
            phi -= phi / p;
        }
        p++;
    }
    if(m > 1) phi -= phi / m;
    ll x = 10, inv_x = power(x, phi - 1);
}
```

#### ModLog.h

**Description:** Returns the smallest  $x > 0$  s.t.  $a^x \equiv b \pmod{m}$ , or -1 if no such  $x$  exists. modLog(a,1,m) can be used to calculate the order of  $a$ .

**Time:**  $\mathcal{O}(\sqrt{m})$

d41d8c, 11 lines

```
11 modLog(ll a, ll b, ll m) {
    ll n = (11) 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 % 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;
}
```

#### ModSum.h

**Description:** Sums of mod'ed arithmetic progressions.

$\text{modsum}(t_0, c, k, m) = \sum_{i=0}^{t_0-1} (ki + c) \% m$ . divsum is similar but for floored division.

**Time:**  $\log(m)$ , with a large constant.

d41d8c, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
```

```
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);
}
```

11 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);
}
```

#### ModMulLL.h

**Description:** Calculate  $a \cdot b \pmod{c}$  (or  $a^b \pmod{c}$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .

**Time:**  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow

d41d8c, 11 lines

```
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 >= (11)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;
}
```

#### ModSqrt.h

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 \equiv a \pmod{p}$  ( $-x$  gives the other solution).

**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

"ModSqrt.h"

```
11 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;
    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;
    }
}
```

## 5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit.

```
11 pt; vector<ll> lpf, Prm, prm;
inline void Sieve(11 m){ //maximum value in inputs
    lpf.assign(m + 1, 0);
    for(int i = 2; i <= m; i ++){
        if(!lpf[i]) Prm.push_back(lpf[i] = i);
        for(int p : Prm){
            if(p > lpf[i] || p * i > m) break;
            lpf[p * i] = p;
        }
    }
    pt = Prm.size();
}
inline void factorize(11 num){
    prm.clear();
    for(int x = num, p = lpf[x]; x > 1; prm.push_back(p), p =
        lpf[x]){
        while(x % p == 0) x /= p;
    }
}
inline void GetPresentPrimes(){
    Prm.clear();
    for(int i = 1; i <= n; i ++){
        factorize(A[i]);
        for(auto X : prm) Prm.push_back(X);
    }
    sort(Prm.begin(), Prm.end());
    Prm.resize(pt = (unique(Prm.begin(), Prm.end()) - Prm.begin
        ()));
}
inline int GetPrmId(11 x){
    return lower_bound(Prm.begin(), Prm.end(), x) - Prm.begin()
        + 1;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.Time: 7 times the complexity of  $a^b \pmod{c}$ .

```
"ModMullL.h"
bool isPrime(11 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;
}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -&gt; {11, 19, 11}).

Time:  $\mathcal{O}(n^{1/4})$ , less for numbers with small factors.

```
"ModMullL.h", "MillerRabin.h"
ull pollard(11 n) {
    auto f = [n](11 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);
        else
```

```
        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(11 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;
}
```

## 5.3 Divisibility

euclid.h

Description: Finds two integers  $x$  and  $y$ , such that  $ax + by = \gcd(a, b)$ . If you just need  $\gcd$ , use the built in  $\_gcd$  instead. If  $a$  and  $b$  are coprime, then  $x$  is the inverse of  $a \pmod{b}$ .

```
d41d8c, 5 lines
11 euclid(11 a, 11 b, 11 &x, 11 &y) {
    if (!b) return x = 1, y = 0, a;
    11 d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
}
```

CRT.h

Description: Chinese Remainder Theorem.

crt(a, m, b, n) computes  $x$  such that  $x \equiv a \pmod{m}$ ,  $x \equiv b \pmod{n}$ . If  $|a| < m$  and  $|b| < n$ ,  $x$  will obey  $0 \leq x < \text{lcm}(m, n)$ . Assumes  $mn < 2^{62}$ .Time:  $\log(n)$ 

```
"euclid.h"
11 crt(11 a, 11 m, 11 b, 11 n) {
    if (n > m) swap(a, b), swap(m, n);
    11 x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

### 5.3.1 Bézout's identity

For  $a \neq b \neq 0$ , then  $d = \gcd(a, b)$  is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If  $(x, y)$  is one solution, then all solutions are given by

$$\left( x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)} \right), \quad k \in \mathbb{Z}$$

phiFunction.h

Description: Euler's  $\phi$  function is defined as  $\phi(n) := \#$  of positive integers  $\leq n$  that are coprime with  $n$ .  $\phi(1) = 1$ ,  $p$  prime  $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$ ,  $m, n$  coprime  $\Rightarrow \phi(mn) = \phi(m)\phi(n)$ . If  $n = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$  then  $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} \dots (p_r - 1)p_r^{k_r - 1}$ .  $\phi(n) = n \cdot \prod_{p|n} (1 - 1/p)$ . $\sum_{d|n} \phi(d) = n$ ,  $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2$ ,  $n > 1$ Euler's thm:  $a, n$  coprime  $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$ .Fermat's little thm:  $p$  prime  $\Rightarrow a^{p-1} \equiv 1 \pmod{p} \forall a$ .

```
d41d8c, 8 lines
const int LIM = 5000000;
int phi[LIM];
void calculatePhi() {
```

```
rep(i, 0, LIM) phi[i] = i&1 ? i : i/2;
for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

## 5.4 Fractions

ContinuedFractions.h

Description: Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ .For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ ) If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ 's eventually become cyclic.Time:  $\mathcal{O}(\log N)$ 

```
d41d8c, 21 lines
typedef double d; // for  $N \sim 1e7$ ; long double for  $N \sim 1e9$ 
pair<ll, ll> approximate(d x, 11 N) {
    11 LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
    for (;;) {
        11 lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
            a = (11)floor(y), b = min(a, lim),
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If  $b > a/2$ , we have a semi-convergent that gives us a
            // better approximation; if  $b = a/2$ , we *may* have one.
            // Return {P, Q} here for a more canonical approximation.
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > 3*N) {
            return {NP, NQ};
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
}
```

FracBinarySearch.h

Description: Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0, 1]$  such that  $f(p/q)$  is true, and  $p, q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`Time:  $\mathcal{O}(\log(N))$ 

```
d41d8c, 25 lines
struct Frac { 11 p, q; };

template<class F>
Frac fracBS(F f, 11 N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        11 adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >>= si) {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !adv;
    }
    return dir ? hi : lo;
}
```

## 5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

## 5.6 Primes

$p = 962592769$  is such that  $2^{21} \mid p - 1$ , which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power  $p^a$ , except for  $p = 2, a > 2$ , and there are  $\phi(\phi(p^a))$  many. For  $p = 2, a > 2$ , the group  $\mathbb{Z}_{2^a}^\times$  is instead isomorphic to  $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$ .

## 5.7 Estimates

$$\sum_{d|n} d = O(n \log \log n).$$

The number of divisors of  $n$  is at most around 100 for  $n < 5e4$ , 500 for  $n < 1e7$ , 2000 for  $n < 1e10$ , 200 000 for  $n < 1e19$ .

## 5.8 Möbius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Möbius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:

$$\sum_{d|n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m)g(\lfloor \frac{n}{m} \rfloor)$$

## Combinatorial (6)

### 6.1 Permutations

#### 6.1.1 Factorial

$n$	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
$n$	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
$n$	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

## IntPerm nCr

IntPerm.h  
**Description:** Permutation  $\rightarrow$  integer conversion. (Not order preserving.) Integer  $\rightarrow$  permutation can use a lookup table.  
**Time:**  $\mathcal{O}(n)$  d41d8c, 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;
    return r;
}
```

### 6.1.2 Cycles

Let  $g_S(n)$  be the number of  $n$ -permutations whose cycle lengths all belong to the set  $S$ . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

### 6.1.3 Derangements

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

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

### 6.1.4 Burnside's lemma

Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  are the elements fixed by  $g$  ( $g \cdot x = x$ ).

If  $f(n)$  counts “configurations” (of some sort) of length  $n$ , we can ignore rotational symmetry using  $G = \mathbb{Z}_n$  to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

## 6.2 Partitions and subsets

### 6.2.1 Partition function

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

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

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

$n$	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	$\sim 2e5$	$\sim 2e8$

### 6.2.2 Lucas' Theorem

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

### 6.2.3 Binomials

nCr.h  
**Description:** Computes  $\binom{n}{k}$ . d41d8c, 26 lines

```
ll power(ll a, ll b) {
    return (!b ? 1 : power(a * a % Mod, b / 2) * (b & 1LL ? a : 1) % Mod);
}
ll inv(ll x) { return power(x, Mod - 2); }
void mkay(ll &x) {
    if(x < 0) x += Mod;
    if(x >= Mod) x -= Mod;
}
ll F[MXN], I[MXN];
ll nCr(ll n, ll r) {
    if(r < 0 || r > n) return 0;
    return F[n] * I[r] % Mod * I[n - r] % Mod;
}
inline void Init() {
    F[0] = I[0] = 1;
    for(int i = 1; i < MXN; i++) F[i] = F[i - 1] * i % Mod;
    I[MXN - 1] = inv(F[MXN - 1]);
    for(int i = MXN - 2; i; --i) I[i] = I[i + 1] * (i + 1) % Mod;
}
// Call Init(); in main
ll NcR(ll n, ll r) { //Iterative version
    if(r < 0 || r > n) return 0;
    ll ans = 1;
    for(ll i = n; i > n - r; --i) ans = ans * i % Mod;
    for(ll i = 2; i <= r; i++) ans = ans * inv(i) % Mod;
    return ans;
}
```

## 6.3 General purpose numbers

### 6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).  $B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

$$\sum_{i=1}^n n^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

### 6.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

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

$$\begin{aligned}c(8, k) &= 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\c(n, 2) &= 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots\end{aligned}$$

### 6.3.3 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 t.  $\pi(j) \geq j$ ,  $k$   $j$ :s t.  $\pi(j) > j$ .

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

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

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

### 6.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

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

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

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

### 6.3.5 Bell numbers

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

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

### 6.3.6 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  $d_i$ :  $(n-2)! / ((d_1-1)! \cdots (d_n-1)!)$

### 6.3.7 Catalan numbers

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

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

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

- sub-diagonal monotone paths in an  $n \times n$  grid.
- strings with  $n$  pairs of parenthesis, correctly nested.
- binary trees 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 (7)

### 7.1 Fundamentals

Dijkstra.h

Description: single/multisource dijkstra

d41d8c, 23 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;
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than  $n$  – nodes reachable from cycles will not be returned.

Time:  $\mathcal{O}(|V| + |E|)$

d41d8c, 14 lines

```
vi topoSort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority_queue for lexic. largest ans.
    rep(i, 0, sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) {
        int i = q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x);
    }
    return ret;
}
```

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:  $\mathcal{O}(VE)$

d41d8c, 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  $m[i][j] = \text{inf}$  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:  $\mathcal{O}(N^3)$

d41d8c, 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;
}
```

### 7.2 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time:  $\mathcal{O}(V^2 \sqrt{E})$

d41d8c, 48 lines

```
struct PushRelabel {
    struct Edge {
        int dest, back;
        ll f, c;
    };
    vector<vector<Edge>> g;
    vector<ll> ec;
    vector<Edge*> cur;
    vector<vi> hs; vi H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

    void addEdge(int s, int t, ll cap, ll rcap=0) {
        if (s == t) return;
        g[s].push_back({t, sz(g[t]), 0, cap});
        g[t].push_back({s, sz(g[s])-1, 0, rcap});
    }

    void addFlow(Edge& e, ll f) {
        Edge &back = g[e.dest][e.back];
        if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
        e.f += f; e.c -= f; ec[e.dest] += f;
        back.f -= f; back.c += f; ec[back.dest] -= f;
    }

    ll calc(int s, int t) {
        int v = sz(g); H[s] = v; ec[t] = 1;
        vi co(2*v); co[0] = v-1;
```

```

rep(i,0,v) cur[i] = g[i].data();
for (Edge& e : g[s]) addFlow(e, e.c);

for (int hi = 0;;) {
    while (hs[hi].empty()) if (!hi--) return -ec[s];
    int u = hs[hi].back(); hs[hi].pop_back();
    while (ec[u] > 0) // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
            H[u] = 1e9;
            for (Edge& e : g[u]) if (e.c && H[e.u] > H[e.dest]+1)
                H[e.u] = H[e.dest]+1, cur[u] = &e;
            if (++co[H[u]], !--co[hi] && hi < v)
                rep(i,0,v) if (hi < H[i] && H[i] < v)
                    --co[H[i]], H[i] = v + 1;
            hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
            addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
}

bool leftOfMinCut(int a) { return H[a] >= sz(g); }

```

## MinCostMaxFlow.h

**Description:** Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

**Time:**  $\mathcal{O}(FE \log(V))$  where F is max flow.  $\mathcal{O}(VE)$  for setpi.[d41d8c](#), 82 lines

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

```

typedef double EDGE_W;
const EDGE_W INF = numeric_limits<EDGE_W>::max() / 4;
struct MCMF {
    struct edge {
        int from, to, rev;
        ll cap;
        EDGE_W cost;
        ll flow;
    };
    int N;
    vector<vector<edge>> ed;
    vi seen;
    vector<EDGE_W> dist, pi;
    vector<edge*> par;

    MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N)
    {}

    void addEdge(int from, int to, ll cap, EDGE_W cost) {
        if (from == to) return;
        ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
        ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
    }

    void path(int s) {
        fill(all(seen), 0);
        fill(all(dist), INF);
        dist[s] = 0; EDGE_W di;

        __gnu_pbds::priority_queue<pair<EDGE_W, int>> q;
        vector<decltype(q)::point_iterator> its(N);
        q.push({ 0, s });

        while (!q.empty()) {
            s = q.top().second; q.pop();
            seen[s] = 1; di = dist[s] + pi[s];

            for (auto e : graph[s])
                if (e.cap - e.flow > 0 && val < dist[e.to]) {
                    dist[e.to] = val;
                    par[e.to] = &e;
                    if (its[e.to] == q.end())
                        its[e.to] = q.push({ -dist[e.to], e.to });
                }
            }
        }
    }
};

class MinCostMaxFlow {
public:
    MinCostMaxFlow() {
        setpi(0);
        for (int i = 0; i < N; i++) adj[i].clear();
    }

    void addEdge(int u, int v, ll cp, EDGE_W cost) {
        adj[u].pb({v, cp, 0});
        adj[v].pb({u, cp, cost});
    }

    ll maxflow(int s, int t) {
        ll totflow = 0;
        EDGE_W totcost = 0;
        while (path(s))
            totflow += dist[t];
        return totflow;
    }

    void setpi(int s) {
        fill(all(pi), INF); pi[s] = 0;
        int it = N, ch = 1, ll v;
        while (ch-- && it--)
            rep(i,0,N) if (pi[i] != INF)
                for (edge& e : ed[i]) if (e.cap)
                    if ((v = pi[i] + e.cost) < pi[e.to])
                        pi[e.to] = v, ch = 1;
            assert(it >= 0); // negative cost cycle
    }
};

```

## MinCostMaxFlow EdmondsKarp MaxFlowNNG

```

for (edge& e : ed[s]) if (!seen[e.to]) {
    EDGE_W val = di - pi[e.to] + e.cost;
    if (e.cap - e.flow > 0 && val < dist[e.to]) {
        dist[e.to] = val;
        par[e.to] = &e;
        if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
    }
}
else
    q.modify(its[e.to], { -dist[e.to], e.to });
}
}
rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}

pair<ll, EDGE_W> maxflow(int s, int t) {
    ll totflow = 0;
    EDGE_W totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);

        totflow += fl;
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl;
        }
    }
    rep(i,0,N) for (edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
}

// If some costs can be negative, call this before maxflow:
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1, ll v;
    while (ch-- && it--)
        rep(i,0,N) if (pi[i] != INF)
            for (edge& e : ed[i]) if (e.cap)
                if ((v = pi[i] + e.cost) < pi[e.to])
                    pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
}

```

## EdmondsKarp.h

**Description:** Flow algorithm with guaranteed complexity  $\mathcal{O}(VE^2)$ . To get edge flow values, compare capacities before and after, and take the positive values only.

```
d41d8c, 35 lines
template<class T> T edmondsKarp(vector<unordered_map<int, T>>& graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
    vi par(sz(graph)), q = par;

    for (;;) {
        fill(all(par), -1);
        par[source] = 0;
        int ptr = 1;
        q[0] = source;

        rep(i,0,ptr) {
            int x = q[i];
            for (auto e : graph[x]) {

```

```

if (par[e.first] == -1 && e.second > 0) {
    par[e.first] = x;
    q[ptr++] = e.first;
    if (e.first == sink) goto out;
}
}
return flow;
out:
T inc = numeric_limits<T>::max();
for (int y = sink; y != source; y = par[y])
    inc = min(inc, graph[par[y]][y]);
flow += inc;
for (int y = sink; y != source; y = par[y]) {
    int p = par[y];
    if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
    graph[y][p] += inc;
}
}

```

## MaxFlowNNG.h

**Description:** Maxflow

**Time:**  $\mathcal{O}(FE \log(V))$  where F is max flow.  $\mathcal{O}(VE)$  for setpi.[d41d8c](#), 73 lines

```
struct MaxFlow {
    struct Edge {
        int fr, to; ll cp, fl;
    };
    int n, src, snk; ll D;
    vector<Edge> E;
    vector<int> adj[maxn];
    int cnt[maxn]; ll dis[maxn];
    queue<int> qu;
    void init(int sz, int v1, int v2) {
        E.clear();
        n = sz; src = v1; snk = v2;
        for (int i = 0; i < n; i++) adj[i].clear();
    }
    void addEdge(int u, int v, ll cp) {
        adj[u].pb({v, cp, 0});
        adj[v].pb({u, 0, 0});
    }
    bool bfs() {
        int v = src;
        fill(dis, dis + n, oo); fill(cnt, cnt + n, 0);
        dis[v] = 0; qu.push(v);
        while (!qu.empty()) {
            int v = qu.front(); qu.pop();
            for (int j : adj[v]) {
                int u = E[j].to; ll cp = E[j].cp, fl = E[j].fl, rm = cp - fl;
                if (rm < D) continue;
                if (dis[v] + 1 < dis[u]) {
                    dis[u] = dis[v] + 1;
                    qu.push(u);
                }
            }
        }
        return (dis[snk] < oo);
    }
    ll dfs(int v, ll f) {
        if (f < D) f = 0;
        if (v == snk || f == 0) return f;
        ll res = 0;
        for (int i = adj[v].begin(); i != adj[v].end(); i++)
            if (adj[v][i].cp - adj[v][i].fl > 0) {
                int j = adj[v][i].to;
                if (dis[j] < oo) {
                    ll nf = min(f, adj[v][i].cp - adj[v][i].fl);
                    adj[v][i].fl += nf;
                    adj[j][i].cp -= nf;
                    res += nf;
                    if (dfs(j, f - nf) == 0) {
                        adj[v][i].fl -= nf;
                        adj[j][i].cp += nf;
                        break;
                    }
                }
            }
        return res;
    }
};

```

```

int u = E[j].to; ll cp = E[j].cp, fl = E[j].fl, rm = cp -
    fl;
if (dis[v] + 1 != dis[u]) continue;

int x = dfs(u, min(f, rm));
res += x; f -= x;
E[j].fl += x; E[j ^ 1].fl -= x;
if (f == 0) break;
}
return res;
}
ll dinic() {
    ll res = 0; D = 1;
    while (D > 0) {
        while (bfs()) res += dfs(src, INF);
        D /= 2;
    }
    return res;
}
MaxFlow A;
int main() {
    A.init(c1 + c2 + 2, c1 + c2, c1 + c2 + 1);
    for (int i = 0; i < c1; i++) A.addEdge(A.src, i, 1);
    for (int i = c1; i < c1 + c2; i++) A.addEdge(i, A.snk, 1);
    for (int i = 0; i < n; i++) {
        int u = col1[i], v = c1 + col2[i];
        ind[i] = len(A.E); A.addEdge(u, v, 1);
    }
    cout << n - A.dinic() << endl;
    return 0;
}

```

## 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:**  $\mathcal{O}(V^3)$  d41d8c, 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) { //  $O(V^2) \rightarrow O(E \log V)$  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"                                         d41d8c, 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;
}

```

## 7.3 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:**  $\mathcal{O}(\sqrt{VE})$  d41d8c, 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:**  $\mathcal{O}(VE)$  d41d8c, 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"                                         d41d8c, 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;
}

```

### WeightedMatching.h

**Description:** Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires  $N \leq M$ .

**Time:**  $\mathcal{O}(N^2M)$  d41d8c, 31 lines

```

pair<int, vi> hungarian(const vector<vi> &a) {
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i, 1, n) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], jl, delta = INT_MAX;
            rep(j, 1, m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], jl = j;
            }
            rep(j, 0, m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = jl;
        } while (p[j0]);
        while (j0) { // update alternating path
            int jl = pre[j0];
            p[j0] = p[jl], j0 = jl;
        }
    }
    rep(j, 1, m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}

```

## GeneralMatching.h

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

```

.../numerical/MatrixInverse-mod.h" d41d8c, 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(i, 0, M/2) {
        rep(j, 0, M) if (has[i])
            rep(i, 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[fj] = 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;
}

```

## 7.4 DFS algorithms

## SCC.h

**Description:** Finds strongly connected components in a directed graph. If vertices  $u, v$  belong to the same component, we can reach  $u$  from  $v$  and vice versa.

**Usage:** `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.

**Time:**  $\mathcal{O}(E + V)$

d41d8c, 24 lines

```

vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ?: dfs(e, g, f));

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps;
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    }
    return val[j] = low;
}

template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    rep(i, 0, n) if (comp[i] < 0) dfs(i, g, f);
}

```

## BiconnectedComponents.h

**Description:** finds biconnected components (connected after vertex removal)

```

const int N = 5e5 + 1;

bitset<N> arti;
vector<int> g[N], st, comp[N];
int n, ptr, ncc, in[N], low[N], id[N];

void dfs(int u, int from = -1) {
    in[u] = low[u] = ++ptr;
    st.emplace_back(u);
    for (int v : g[u]) if (v ^ from) {
        if (!in[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= in[u]) {
                arti[u] = ~from or in[v] > in[u] + 1;
                comp[+ncc].emplace_back(u);
            }
            while (comp[ncc].back() ^ v) {
                comp[ncc].emplace_back(st.back());
                st.pop_back();
            }
        }
    }
}

```

```

        }
    }
    else low[u] = min(low[u], in[v]);
}

void bcc() {
    for (int i = 0; i < n; i++) {
        if (!in[i]) {
            dfs(i);
            if (g[i].empty()) comp[+ncc].push_back(i); // COMPS NUMBERED 1..ncc INCLUDED
        }
    }
}

// BLOCK CUT TREE
// vector<int> tree[N];
// void buildTree() {
//     ptr = 0;
//     for (int i = 0; i < n; ++i) { // index nodes properly
//         here
//             if (arti[i]) id[i] = ++ptr;
//         }
//         for (int i = 1; i <= ncc; ++i) {
//             int x = ++ptr;
//             for (int u : comp[i]) {
//                 if (arti[u]) tree[x].emplace_back(id[u]), tree[id[u]].emplace_back(x);
//                 else id[u] = x;
//             }
//         }
//     }
// }

```

## TwoEdgeConnectedComponents.h

**Description:** does dfs to find bridges, then removes them and does dfs to find 2CCs

```

struct LowLink{
    int n, pos = 0;
    vi ord, low, par, blg;
    vector<vi> G, C;

    LowLink(vector<vi>& adj) : n(sz(adj)), ord(n,-1), low(n),
        par(n,-1), blg(n,-1), G(adj) {}

    bool bridge(int u, int v) {
        if (ord[u] > ord[v]) swap(u,v);
        return ord[u] < low[v];
    }

    void dfs(int v){
        ord[v] = low[v] = pos++;
        int cnt = 0;
        for(int u : G[v]){
            if(u == par[v] && cnt == 0) cnt++;
            else if(~ord[u]) low[v] = min(low[v], ord[u]);
            else {
                par[u] = v;
                dfs(u);
                low[v] = min(low[v], low[u]);
            }
        }
    }

    void fill_comp(int v){
        C[blg[v]].emplace_back(v);
        for (int u : G[v]) {
            if (~blg[u] || bridge(u,v)) continue;

```

```

    blg[u] = blg[v];
    fill_comp(u);
}

void add_comp(int v, int &k) {
    if(~blg[v]) return;
    blg[v] = k++;
    C.emplace_back();
    fill_comp(v);
}

int build() {
    for(int i=0; i<n; i++) if(ord[i] < 0) dfs(i);

    int k = 0;
    for(int i=0; i<n; i++) add_comp(i, k);
    return k;
}

```

## 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  $(a \mid\mid b) \&\& (\neg a \mid\mid c) \&\& (\neg d \mid\mid b) \&\& \dots$  becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ( $\sim x$ ).

**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.

d41d8c, 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]);
    }

```

## 2sat EulerWalk EdgeColoring MaximalCliques MaximumClique

```

    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:**  $\mathcal{O}(V + E)$

d41d8c, 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()};
    }
}

```

## 7.5 Coloring

## EdgeColoring.h

**Description:** Given a simple, undirected graph with max degree  $D$ , computes a  $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. ( $D$ -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

**Time:**  $\mathcal{O}(NM)$

d41d8c, 31 lines

```

vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])

```

```

            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
            while (adj[fan[i]][d] != -1) {
                int left = fan[i], right = fan[++i], e = cc[i];
                adj[u][e] = left;
                adj[left][e] = u;
                adj[right][e] = -1;
                free[right] = e;
            }
            adj[u][d] = fan[i];
            adj[fan[i]][d] = u;
            for (int y : {fan[0], u, end})
                for (int& z = free[y] = 0; adj[y][z] != -1; z++);
            }
            rep(i, 0, sz(eds))
                for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
            return ret;
        }
}

```

## 7.6 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:**  $\mathcal{O}(3^{n/3})$ , much faster for sparse graphs

d41d8c, 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).FindFirst();
    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.

d41d8c, 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.

## 7.7 Trees

### LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree

**Time:**  $\mathcal{O}(N \log N + Q)$

```

.../data-structures/RMQ.h"
d41d8c, 30 lines
11 Jad[LOG][MXN], dis[MXN];
void prep(ll u, ll par){
    Jad[0][u] = par;
    for(int i = 1; i < LOG; i++) {
        Jad[i][u] = Jad[i - 1][Jad[i - 1][u]];
    }
    for(auto v : adj[u]){
        if(v == par) continue;
        dis[v] = dis[u] + 1;
        prep(v, u);
    }
}
11 K_Jad(ll u, ll k){
    for(int i = 0; i < LOG; i++){
        if((k >> i) & 1LL) u = Jad[i][u];
    }
    return u;
}
11 LCA(ll u, ll v){
    if(dis[v] < dis[u]) swap(u, v);
    v = K_Jad(v, dis[v] - dis[u]);
    if(u == v) return u;
    for(int i = LOG - 1; ~i; i--){
        if(Jad[i][u] != Jad[i][v]) u = Jad[i][u], v = Jad[i][v];
    }
    return Jad[0][u];
}
inline ll Distance(ll u, ll v){
    return dis[u] + dis[v] - 2 * dis[LCA(u, v)];
}

```

```
}
```

### CompressTree.h

**Description:** Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most  $|S| - 1$ ) pairwise LCA's and compressing edges. Returns a list of (par, orig\_index) representing a tree rooted at 0. The root points to itself.

**Time:**  $\mathcal{O}(|S| \log |S|)$

[LCA.h](#) d41d8c, 21 lines

```

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}

```

### HLD.h

**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most  $\log(n)$  light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS\_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

**Time:**  $\mathcal{O}((\log N)^2)$

```

d41d8c, 41 lines
void prep(ll u, ll par){
    sub[u]++, Par[u] = par;
    for(auto v : adj[u]){
        if(v == par) continue;
        dis[v] = dis[u] + 1, prep(v, u);
        sub[u] += sub[v];
        if(sub[v] > sub[hvs[u]]) hvs[u] = v;
    }
}
void dfs(ll u, ll par, ll head){
    Stm[u] = ++timer, Tree[timer] = u;
    hd[u] = head;
    if(hvs[u]) dfs(hvs[u], u, head);
    for(auto v : adj[u]){
        if(v == par || v == hvs[u]) continue;
        dfs(v, u, v);
    }
    Ftm[u] = timer;
}
void Qry(ll u, ll v){ //u is ancestor of v
    ll ans = 0; while(hd[v] != hd[u]){
        ans += Get(Stm[hd[v]], Stm[v]);
        v = Par[hd[v]];
    }
    ans += Get(Stm[u], Stm[v]); return ans;
}
ll Geq(ll u){ //root to u query:
    ll ans = 0; while(u{

```

```

        ans += Get(Stm[hd[u]], Stm[u]);
        u = Par[hd[u]];
    }
    return ans;
}

```

```

11 LCA(ll u, ll v){
    while(hd[u] != hd[v]){
        if(dis[hd[u]] > dis[hd[v]]) swap(u, v);
        v = Par[hd[v]];
    }
    if(dis[u] > dis[v]) swap(u, v);
    return u;
}

```

### LinkCutTree.h

**Description:** Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

**Time:** All operations take amortized  $\mathcal{O}(\log N)$ .

```

d41d8c, 90 lines
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p) == p) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            y->c[h ^ 1] = x;
        }
        z->c[i ^ 1] = this;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (pushFlip(); p; ) {
            if (p->p) p->p->pushFlip();
            p->pushFlip(); pushFlip();
            int cl = up(), c2 = p->up();
            if (c2 == -1) p->rot(cl, 2);
            else p->p->rot(c2, cl != c2);
        }
    }
    Node* first() {
        pushFlip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
}
struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}
}

```

```

void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
}

void cut(int u, int v) { // remove an edge (u, v)
    Node* x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
        x->c[0] = top->p = 0;
        x->fix();
    }
}

bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
}

void makeRoot(Node* u) {
    access(u);
    u->splay();
    if (u->c[0]) {
        u->c[0]->p = 0;
        u->c[0]->flip ^= 1;
        u->c[0]->pp = u;
        u->c[0] = 0;
        u->fix();
    }
}

Node* access(Node* u) {
    u->splay();
    while ((Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp;
            pp->c[1] = u; pp->fix(); u = pp;
        }
    }
    return u;
}
};

DirectedMST.h

```

**Description:** Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

**Time:**  $\mathcal{O}(E \log V)$

```

.../data-structures/UnionFindRollback.h" d41d8c, 60 lines
struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

```

```

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int>, vector<Edge>> cycs;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
    }

    for (auto& [u, t, comp] : cycs) { // restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    rep(i, 0, n) par[i] = in[i].a;
    return {res, par};
}

```

## 7.8 Math

### 7.8.1 Number of Spanning Trees

Create an  $N \times N$  matrix mat, and for each edge  $a \rightarrow b \in G$ , do  $\text{mat}[a][b]--$ ,  $\text{mat}[b][b]++$  (and  $\text{mat}[b][a]--$ ,  $\text{mat}[a][a]++$  if  $G$  is undirected). Remove the  $i$ th 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).

### 7.8.2 Erdős–Gallai theorem

A simple graph with node degrees  $d_1 \geq \dots \geq d_n$  exists iff  $d_1 + \dots + d_n$  is even and for every  $k = 1 \dots n$ ,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

## Geometry (8)

### 8.1 Geometric primitives

Point.h

**Description:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)  
d41d8c, 31 lines

```

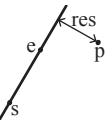
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    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()); }
    T distsq(P p) const { return (x-p.x)*(x-p.x)+(y-p.y)*(y-p.y); }
    double dist(P p) const { return sqrt(this->distsq(p)); }
    // 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(); }
    P _rot(P p, double a) const { // helper
        return P(p.x*cos(a)-p.y*sin(a), p.x*sin(a)+p.y*cos(a));
    } // returns point rotated 'a' radians ccw around origin 'r'
    P rotate(double a, P r) { return r + _rot(*this-r, 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" d41d8c, 4 lines



```

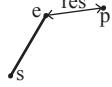
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double)(b-a).cross(p-a)/(b-a).dist();
}

```

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" d41d8c, 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" d41d8c, 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" d41d8c, 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" d41d8c, 9 lines

```
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 > 1) - (a < -1);
}
```



## 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>.

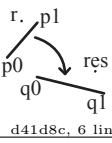
"Point.h" d41d8c, 3 lines

```
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" d41d8c, 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. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" d41d8c, 16 lines

```
// projection of point p on line ax + by + c = 0
template<class P> P projOnLine(P p, dbl a, dbl b, dbl c) {
    double d = (a * p.x + b * p.y + c) / (a*a + b*b);
    return {p.x - a * d, p.y - b * d};
}
```

```
// projection of point p on line defined by points a and b
template<class P> P projOnLine(P p, P a, P b) {
    P ab = b - a;
    double t = (p - a).dot(b-a) / ab.dot(b-a);
    return a + ab * t;
}
```

```
// reflection of point p over line defined by points a and b
template<class P> P reflectOverLine(P p, P a, P b) {
    P proj = projOnLine(p, a, b);
    return proj * 2 - p;
}
```

## Angle.h

**Description:** functions for angles in 2D geometry.

d41d8c, 14 lines

```
// smaller angle in B in radians [0, pi)
const double PI = acos(-1.0);
template<class P> double angle3pt(P a, P b, P c) {
    double dotp = (a-b).dot(c-b);
    double crsp = (a-b).cross(c-b);
    return fabs(atan2(crsp, dotp)); // [0, pi]
}
```

// comparator sorting angles ccw wrt to (0,0)

```
template<class P> bool half(const P&p) {
    return (p.y > 0 || (p.y == 0 && p.x > 0));
}
template<class P> bool cmp(const P&a, const P&b) {
    bool ha = half(a), hb = half(b);
    return (ha != hb) ? ha > hb : a.cross(b) > 0;
}
```

## 8.2 Circles

### CircleIntersection.h

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

"Point.h" d41d8c, 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" d41d8c, 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;
}
```

## CirclePolygonIntersection.h

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

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

".../content/geometry/Point.h" d41d8c, 19 lines

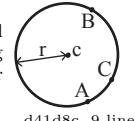
```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        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 circumcircle 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"



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"` d41d8c, 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, i, 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};
}
```

## 8.3 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 inPolygon(v, P{3, 3}, false);`

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

`"Point.h", "OnSegment.h", "SegmentDistance.h"` d41d8c, 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*q[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"` d41d8c, 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"` d41d8c, 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"`

```
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;
}
```



d41d8c, 13 lines

### Minkowski.h

**Description:** Minkowski sum of two convex polygons. Make sure both polygons have at least 2 points!!!!

**Time:**  $\mathcal{O}(n+m)$

```
"Point.h" d41d8c, 20 lines
template<class T> void reorder_polygon(vector<T> &P) {
    size_t pos = 0;
    for (size_t i = 1; i < P.size(); i++) {
        if (P[i].y < P[pos].y ||
            (P[i].y == P[pos].y && P[i].x < P[pos].x))
            pos = i;
    } rotate(P.begin(), P.begin() + pos, P.end());
}

template<class T> vector<T> Mink(vector<T> P, vector<T> Q) {
    reorder_polygon(P); reorder_polygon(Q);
    P.push_back(P[0]); P.push_back(P[1]);
    Q.push_back(Q[0]); Q.push_back(Q[1]);
    vector<T> result; size_t i = 0, j = 0;
    while (i < P.size() - 2 || j < Q.size() - 2) {
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
        if (cross >= 0 && i < P.size() - 2) ++i;
        if (cross <= 0 && j < Q.size() - 2) ++j;
    } return result;
}
```

### 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"` d41d8c, 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-->0; 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"` d41d8c, 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"` d41d8c, 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;
        if (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" d41d8c, 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) {
```



d41d8c, 13 lines

```

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;
}

```

## 8.4 Misc. Point Set Problems

### ClosestPair.h

Description: Finds the closest pair of points.

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

`"Point.h"`

d41d8c, 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 + (11)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;
}

```

### kdTree.h

Description: KD-tree (2d, can be extended to 3d)

`"Point.h"`

d41d8c, 63 lines

```

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {

```

## ClosestPair kdTree FastDelaunay

```

P pt; // if this is a leaf, the single point in it
T x0 = INF, xl = -INF, y0 = INF, yl = -INF; // bounds
Node *first = 0, *second = 0;

T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > xl ? xl : p.x);
    T y = (p.y < y0 ? y0 : p.y > yl ? yl : p.y);
    return (p(x, y) - p).dist2();
}

Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
        x0 = min(x0, p.x); xl = max(xl, p.x);
        y0 = min(y0, p.y); yl = max(yl, p.y);
    }
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), xl - x0 >= yl - y0 ? on_x : on_y);
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()});
    }
}

struct KDTree {
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
        // uncomment if we should not find the point itself:
        // if (p == node->pt) return {INF, P()};
        return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
        best = min(best, search(s, p));
    return best;
}

// find nearest point to a point, and its squared distance
// (requires an arbitrary operator< for Point)
pair<T, P> nearest(const P& p) {
    return search(root, p);
}

```

### 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"`

d41d8c, 88 lines

```

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t l11; // (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?
    l11 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(rb, B) = 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()) < 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;
}
```

## 8.5 3D

### PolyhedronVolume.h

**Description:** Magic formula for the volume of a polyhedron. Faces should point outwards.

d41d8c, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
    double v = 0;
    for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
    return v / 6;
}
```

### Point3D.h

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

d41d8c, 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;
    }
};
```

### 3dHull.h

**Description:** Computes all faces of the 3-dimension hull of a point set. \*No four points must be coplanar\*, or else random results will be returned. All faces will point outwards.

```
Time:  $\mathcal{O}(n^2)$ 
"Point3D.h"
typedef Point3D<double> P3;

struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }
    int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A)), vector<PR>(sz(A), {-1, -1});
#define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i])) {
            q = q * -1;
            F f{q, i, j, k};
            E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
            FS.push_back(f);
        };
        rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
            mf(i, j, k, 6 - i - j - k);
    };
    rep(i,4,sz(A)) {
        rep(j,0, sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();
            }
            int nw = sz(FS);
            rep(j,0,nw) {
                F f = FS[j];
                #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
                C(a, b, c); C(a, c, b); C(b, c, a);
            }
            for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
                A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
            return FS;
        };
    }
}
```

### sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude)  $f_1$  ( $\phi_1$ ) and  $f_2$  ( $\phi_2$ ) from x axis and zenith angles (latitude)  $t_1$  ( $\theta_1$ ) and  $t_2$  ( $\theta_2$ ) 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.  $dx * \text{radius}$  is then the difference between the two points in the x direction and  $d * \text{radius}$  is the total distance between the points.

```
d41d8c, 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);
}
```

}

## Strings (9)

### KMP.h

**Description:**  $\text{pi}[x]$  computes the length of the longest prefix of  $s$  that ends at  $x$ , other than  $s[0...x]$  itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

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

d41d8c, 16 lines

```
vi pi(const string& s) {
    vi p(sz(s));
    rep(i,1,sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}
```

```
vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i,sz(p)-sz(s),sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
    return res;
}
```

### Zfunc.h

**Description:**  $\text{z}[x]$  computes the length of the longest common prefix of  $s[i:]$  and  $s$ , except  $s[0] = 0$ . (abacaba -> 0010301)

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

d41d8c, 12 lines

```
vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

### Manacher.h

**Description:** For each position in a string, computes  $p[0][i] =$  half length of longest even palindrome around pos i,  $p[1][i] =$  longest odd (half rounded down).

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

d41d8c, 13 lines

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi,2> p = {vi(n+1), vi(n)};
    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
        int t = r-i+1;
        if (i < r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

## MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.  
**Usage:** `rotate(v.begin(), v.begin() + minRotation(v), v.end());`  
**Time:**  $\mathcal{O}(N)$

d41d8c, 8 lines

```
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;
}
```

## SuffixArray.h

**Description:** Builds suffix array for a string.  $sa[i]$  is the starting index of the suffix which is  $i$ 'th in the sorted suffix array. The returned vector is of size  $n+1$ , and  $sa[0] = n$ . The  $lcp$  array contains longest common prefixes for neighbouring strings in the suffix array:  $lcp[i] = lcp(sa[i], sa[i-1])$ ,  $lcp[0] = 0$ . The input string must not contain any zero bytes.  
**Time:**  $\mathcal{O}(n \log n)$

d41d8c, 23 lines

```
struct SuffixArray {
    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++);
    }
};
```

## SuffixTree.h

**Description:** Ukkonen's algorithm for online suffix tree construction. Each node contains indices  $[l, r)$  into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining  $[l, r)$  substrings. The root is 0 (has  $l = -1, r = 0$ ), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).

**Time:**  $\mathcal{O}(26N)$ 

d41d8c, 50 lines

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m+1]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q===-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
        }
    }
};
```

```
l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
v=s[p[m]]; q=l[m];
while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-1[v]; }
if (q==r[m]) s[m]=v; else s[m]=m+2;
q=r[v]-(q-r[m]); m+=2; goto suff;
```

```
}
}

SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
}

// example: find longest common substring (uses ALPHA = 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - 1[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
        best = max(best, {len, r[node] - len});
    return mask;
}
static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};
```

## Hashing.h

**Description:** Self-explanatory methods for string hashing.

d41d8c, 24 lines

```
const ll MXH = 2; const ll BASE = 257;
const ll Mod[MXH] = {(11)(1e9 + 7), (11)(1e9 + 9)};
void mkay(int th, ll &x){
    if(x > Mod[th]) x -= Mod[th];
}
ll pw[MXH][MXN], H[MXH][MXN];
void InitPow(){
    for(int th = 0; th < MXH; th ++){
        pw[th][0] = 1;
        for(int i = 1; i < MXN; i++) pw[th][i] = pw[th][i - 1] *
            BASE % Mod[th];
    }
}
void InitHash(ll &n, string &s){
    assert(s[0] == '$' && s.size() == n + 1);
    for(int th = 0; th < MXH; th ++){
        H[th][0] = 0;
        for(int i = 1; i < n; i++) {
            H[th][i] = (H[th][i - 1] * BASE + (s[i] - 'a' + 1)) % Mod
                [th];
        }
    }
}
inline ll Hash(int th, ll l, ll r){
    return (H[th][r] - H[th][l - 1] * pw[th][r - l + 1] % Mod[th]
        + Mod[th]) % Mod[th];
}
// Call InitPow(), InitHash(n, s); in main
```

## PalHashing.h

**Description:** Checking if the string is palindrome

d41d8c, 31 lines

```
ll n; string s; //! Length is important in RH!
ll pw[MXH][MXN], H[MXH][MXN], RH[MXH][MXN];
void InitPow() { //RH: ith suffix Hash, i <-> n - i + 1
    for(int th = 0; th < MXH; th ++){
        pw[th][0] = 1;
        for(int i = 1; i < MXN; i++) pw[th][i] = pw[th][i - 1] *
            BASE % Mod[th];
    }
}
void InitHash(){
    assert(s[0] == '$' && s.size() == n + 1);
    for(int th = 0; th < MXH; th ++){
        H[th][0] = RH[th][0] = 0;
        for(int i = 1; i <= n; i ++){
            H[th][i] = (H[th][i - 1] * BASE + (s[i] - 'a' + 1)) % Mod
                [th];
            RH[th][i] = (RH[th][i - 1] * BASE + (s[n - i + 1] - 'a' +
                1)) % Mod[th];
        }
    }
}
inline ll Hash(int th, ll l, ll r){
    return (H[th][r] - H[th][l - 1] * pw[th][r - l + 1] % Mod[th]
        + Mod[th]) % Mod[th];
}
inline ll RHash(int th, ll l, ll r){
    // H(l, r) -> H[l] - H[r + 1] * pw[len]
    return (RH[th][n - l + 1] - RH[th][n - (r + 1) + 1] * pw[th][
        r - l + 1] % Mod[th] + Mod[th]) % Mod[th];
}
inline ll IsPal(ll l, ll r){
    for(int th = 0; th < MXH; th ++)
        if(Hash(th, l, r) != RHash(th, l, r))
            return 0;
    return 1;
}
```

## AhoCorasick.h

**Description:** Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(–, word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.

**Time:** construction takes  $\mathcal{O}(26N)$ , where  $N$  = sum of length of patterns.

find(x) is  $\mathcal{O}(N)$ , where  $N$  = length of x. findAll is  $\mathcal{O}(NM)$ .

d41d8c, 34 lines

```
const ll SGM = 26;
int n, q, ts = 1, timer;
int Q[MXN], Ver[MXN], nxt[MXN][SGM], lps[MXN];
char s[MXN];
void Add(int id){
    scanf("%s", s); int sz = strlen(s);
    int u = 1;
    for(int h = 0; h < sz; h ++){
        if(!nxt[u][s[h] - 'a']) nxt[u][s[h] - 'a'] = ++ ts;
        u = nxt[u][s[h] - 'a'];
    }
    Ver[id] = u;
}
void Aho(){
    int L = 0, R = 0;
    for(int i = 0; i < SGM; i ++){
        if(!nxt[1][i]) nxt[1][i] = 1;
    }
}
```

```

        else{
            Q[R ++] = nxt[1][i];
            lps[nxt[1][i]] = 1;
        }
    }
    while(L < R){
        int u = Q[L++];
        adj[lps[u]].push_back(u); // Aho-tree
        for(int c = 0; c < SGM; c++){
            if(!nxt[u][c]) nxt[u][c] = nxt[lps[u]][c];
            else{
                lps[nxt[u][c]] = nxt[lps[u]][c];
                Q[R ++] = nxt[u][c];
            }
        }
    }
}

```

## Various (10)

### 10.1 Intervals

#### IntervalContainer.h

**Description:** Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

**Time:**  $\mathcal{O}(\log N)$

d41d8c, 23 lines

```

set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}

```

#### IntervalCover.h

**Description:** Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).

**Time:**  $\mathcal{O}(N \log N)$

d41d8c, 19 lines

```

template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first;
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);

```

```

        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at]));
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);
    }
    return R;
}

```

#### ConstantIntervals.h

**Description:** Split a monotone function on  $[from, to]$  into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.

**Usage:** constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});

**Time:**  $\mathcal{O}(k \log \frac{n}{k})$

d41d8c, 19 lines

```

template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;
    if (from == to) {
        g(i, to, p);
        i = to; p = q;
    } else {
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    }
}

template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}

```

### 10.2 Misc. algorithms

#### TernarySearch.h

**Description:** Find the smallest i in  $[a, b]$  that maximizes  $f(i)$ , assuming that  $f(a) < \dots < f(i) \geq \dots \geq f(b)$ . To reverse which of the sides allows non-strict inequalities, change the  $<$  marked with (A) to  $\leq$ , and reverse the loop at (B). To minimize  $f$ , change it to  $>$ , also at (B).

**Usage:** int ind = ternSearch(0, n-1, [&](int i){return a[i];});

**Time:**  $\mathcal{O}(\log(b-a))$

d41d8c, 11 lines

```

template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)
        else b = mid+1;
    }
    rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}

```

#### LIS.h

**Description:** Compute indices for the longest increasing subsequence.

**Time:**  $\mathcal{O}(N \log N)$

d41d8c, 17 lines

```

template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;

```

```

    rep(i, 0, sz(S)) {
        // change 0 -> i for longest non-decreasing subsequence
        auto it = lower_bound(all(res), p(S[i], 0));
        if (it == res.end()) res.emplace_back(), it = res.end()-1;
        *it = {S[i], i};
        prev[i] = it == res.begin() ? 0 : (it-1)->second;
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans;
}

```

#### FastKnapsack.h

**Description:** Given  $N$  non-negative integer weights  $w$  and a non-negative target  $t$ , computes the maximum  $S \leq t$  such that  $S$  is the sum of some subset of the weights.

**Time:**  $\mathcal{O}(N \max(w_i))$

d41d8c, 16 lines

```

int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
    v[a+m-t] = b;
    rep(i, b, sz(w)) {
        u = v;
        rep(x, 0, m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0, u[x]), v[x])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}

```

### 10.3 Dynamic programming

#### KnuthDP.h

**Description:** When doing DP on intervals:  $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ , one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j-1]$  and  $p[i+1][j]$ . This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

**Time:**  $\mathcal{O}(N^2)$

#### DivideAndConquerDP.h

**Description:** Given  $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$  where the (minimal) optimal  $k$  increases with  $i$ , computes  $a[i]$  for  $i = L..R-1$ .

**Time:**  $\mathcal{O}((N + (hi - lo)) \log N)$

d41d8c, 18 lines

```

struct DP { // Modify at will:
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1;
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second, HI);
        rec(mid+1, R, best.second, HI);
    }
}

```

```
void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
```

## 10.4 Debugging tricks

- signal(SIGSEGV, [](int) { \_Exit(0); }); converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- feenableexcept(29); kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

## 10.5 Optimization tricks

`_builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

### 10.5.1 Bit hacks

- `x & ~x` is the least bit in `x`.
- for (int x = m; x; ) { `--x &= m;` ... } for (`sub=mask;` ; `sub = (sub-1)&mask`) {  
  // [Update] if (`sub == 0`) break; } loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c;`  $((r^x) \gg 2)/c$  | `r` is the next number after `x` with the same number of bits set.
- rep(b,0,K) rep(i,0,(1 << K))  
  if (`i & 1 << b`) `D[i] += D[i^(1 << b)];`  
  for (j = 0; j < MXN;) {for (mask = 0; mask < MXN)  
    if (bit(mask, j)) `dp[mask^(1<<j)]+=dp[mask];`} } computes all sums of subsets.

### 10.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

### FastMod.h

**Description:** Compute  $a\%b$  about 5 times faster than usual, where  $b$  is constant but not known at compile time. Returns a value congruent to  $a$  (mod  $b$ ) in the range  $[0, 2b)$ .  
d41d8c, 15 lines

```
typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
    ull reduce(ull a) {
        ull q = (ull)((L(m) * a) >> 64);
        ull r = a - q * b;
        return r >= b ? r - b : r;
    }
}
```

```
; FastMod f(2);
// f = FastMod(Mod); Inv[0] = Inv[1] = 1; //fast inverse
// calculation
// for(int i = 2; i < MXN; i ++){
//     Inv[i] = f.reduce(Mod - f.reduce(1ll * Mod / i * Inv[Mod
// % i]));
// }
```

### FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.  
**Usage:** `./a.out < input.txt`

**Time:** About 5x as fast as `cin/scanf`. d41d8c, 17 lines

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++];
}
```

```
int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 480;
    return a - 48;
}
```

### BumpAllocator.h

**Description:** When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation. d41d8c, 8 lines

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof buf;
    assert(s < i);
    return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

### SmallPtr.h

**Description:** A 32-bit pointer that points into BumpAllocator memory. "BumpAllocator.h" d41d8c, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof buf);
    }
    T& operator*() const { return *(T*)(buf + ind); }
    T* operator->() const { return &**this; }
    T& operator[](int a) const { return (&**this)[a]; }
    explicit operator bool() const { return ind; }
};
```

### BumpAllocatorSTL.h

**Description:** BumpAllocator for STL containers.

**Usage:** `vector<vector<int, small<int>> ed(N);` d41d8c, 14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;
```

```
template<class T> struct small {
    typedef T value_type;
```

```
small() {}
template<class U> small(const U&) {}
T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
}
void deallocate(T*, size_t) {}
```

### SIMD.h

**Description:** Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "`_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)`". Not all are described here; grep for `_mm_in` / `/usr/lib/gcc/*/4.9/include/` for more. If AVX is unsupported, try 128-bit operations, "emmmintrin.h" and `#define __SSE__` and `__MMX__` before including it. For aligned memory use `_mm_malloc(size, 32)` or `int buf[N] alignas(32)`, but prefer `loadu/storeu`. d41d8c, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "immintrin.h"
```

```
typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))
```

**High-level/specific methods:**  
`// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc`  
`// blndu_(epi8|ps|pd) (z?y:x), movemask_ep16 (hibits of bytes)`  
`// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x`  
`// sad_epu8: sum of absolute differences of u8, outputs 4xi64`  
`// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15`  
`// madd_epi16: dot product of signed i16's, outputs 8xi32`  
`// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)`  
`// permute2f128_si256(x,x,1) swaps 128-bit lanes`  
`// shuffle_epi32(x, 3x64+2*16+1*4+0) == x for each lane`  
`// shuffle_epi8(x, y) takes a vector instead of an imm`

**Methods that work with most data types (append e.g. \_epi32):**  
`// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,`  
`// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)`

```
int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }
```

```
ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmplt_epi16(vb, va), va);
        mi vp = _mm256_madd_epi16(va, vb);
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
    for (; i < n; ++i) if (a[i] < b[i]) r += a[i]*b[i]; //<- equiv
    return r;
}
```

# Techniques (A)

## techniques.txt

159 lines

Recursion  
 Divide and conquer  
     Finding interesting points in  $N \log N$   
 Algorithm analysis  
     Master theorem  
     Amortized time complexity  
 Greedy algorithm  
     Scheduling  
     Max contiguous subvector sum  
     Invariants  
     Huffman encoding  
 Graph theory  
     Dynamic graphs (extra book-keeping)  
     Breadth first search  
     Depth first search  
         \* Normal trees / DFS trees  
     Dijkstra's algorithm  
     MST: Prim's algorithm  
     Bellman-Ford  
     Konig's theorem and vertex cover  
     Min-cost max flow  
     Lovasz toggle  
     Matrix tree theorem  
     Maximal matching, general graphs  
     Hopcroft-Karp  
     Hall's marriage theorem  
     Graphical sequences  
     Floyd-Warshall  
     Euler cycles  
     Flow networks  
         \* Augmenting paths  
         \* Edmonds-Karp  
     Bipartite matching  
     Min. path cover  
     Topological sorting  
     Strongly connected components  
     2-SAT  
     Cut vertices, cut-edges and biconnected components  
     Edge coloring  
         \* Trees  
     Vertex coloring  
         \* Bipartite graphs ( $\Rightarrow$  trees)  
         \*  $3^n$  (special case of set cover)  
     Diameter and centroid  
     K'th shortest path  
     Shortest cycle  
 Dynamic programming  
     Knapsack  
     Coin change  
     Longest common subsequence  
     Longest increasing subsequence  
     Number of paths in a dag  
     Shortest path in a dag  
     Dynprog over intervals  
     Dynprog over subsets  
     Dynprog over probabilities  
     Dynprog over trees  
      $3^n$  set cover  
     Divide and conquer  
     Knuth optimization  
     Convex hull optimizations  
     RMQ (sparse table a.k.a  $2^k$ -jumps)  
     Bitonic cycle  
     Log partitioning (loop over most restricted)  
 Combinatorics

Computation of binomial coefficients  
 Pigeon-hole principle  
 Inclusion/exclusion  
 Catalan number  
 Pick's theorem  
 Number theory  
     Integer parts  
     Divisibility  
     Euclidean algorithm  
     Modular arithmetic  
         \* Modular multiplication  
         \* Modular inverses  
         \* Modular exponentiation by squaring  
     Chinese remainder theorem  
     Fermat's little theorem  
     Euler's theorem  
     Phi function  
     Frobenius number  
     Quadratic reciprocity  
     Pollard-Rho  
     Miller-Rabin  
     Hensel lifting  
     Vieta root jumping  
 Game theory  
     Combinatorial games  
     Game trees  
     Mini-max  
     Nim  
     Games on graphs  
     Games on graphs with loops  
     Grundy numbers  
     Bipartite games without repetition  
     General games without repetition  
     Alpha-beta pruning  
 Probability theory  
 Optimization  
     Binary search  
     Ternary search  
     Unimodality and convex functions  
     Binary search on derivative  
 Numerical methods  
     Numeric integration  
     Newton's method  
     Root-finding with binary/ternary search  
     Golden section search  
 Matrices  
     Gaussian elimination  
     Exponentiation by squaring  
 Sorting  
     Radix sort  
 Geometry  
     Coordinates and vectors  
         \* Cross product  
         \* Scalar product  
     Convex hull  
     Polygon cut  
     Closest pair  
     Coordinate-compression  
     Quadtrees  
     KD-trees  
     All segment-segment intersection  
 Sweeping  
     Discretization (convert to events and sweep)  
     Angle sweeping  
     Line sweeping  
     Discrete second derivatives  
 Strings  
     Longest common substring  
     Palindrome subsequences

Knuth-Morris-Pratt  
 Tries  
 Rolling polynomial hashes  
 Suffix array  
 Suffix tree  
 Aho-Corasick  
 Manacher's algorithm  
 Letter position lists  
 Combinatorial search  
     Meet in the middle  
     Brute-force with pruning  
     Best-first (A\*)  
     Bidirectional search  
     Iterative deepening DFS / A\*

Data structures  
     LCA ( $2^k$ -jumps in trees in general)  
     Pull/push-technique on trees  
     Heavy-light decomposition  
     Centroid decomposition  
     Lazy propagation  
     Self-balancing trees  
     Convex hull trick ([wcipeg.com/wiki/Convex\\_hull\\_trick](http://wcipeg.com/wiki/Convex_hull_trick))  
     Monotone queues / monotone stacks / sliding queues  
     Sliding queue using 2 stacks  
     Persistent segment tree