# University of Dhaka

# DU_Oblivion

Farhan Ibn Shahid, Md Istahak Islam, Soyeb Pervez Jim

2025-01-06

# Contest (1)

### template.cpp
24 lines
```cpp
#include <bits/stdc++.h>
using namespace std;

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

void solve() {}
int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
    int t = 1;
    cin >> t;
    while (t--) {
        solve();
    }
#ifdef ONPC
    cerr << endl
         << "finished in " << clock() * 1.0 / CLOCKS_PER_SEC << "
            sec" << endl;
#endif
}
```

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

### stress.sh
10 lines
```bash
#!/bin/bash
[ "$#" -ne 3 ] && echo "Usage: $0 test_file brute_file
    mycode_file" && exit 1
g++ -O2 $1 -o test && g++ -O2 $2 -o brute && g++ -O2 $3 -o
    mycode
for i in {1..10000}; do
  ./test > tests.txt
  ./brute < tests.txt > correct.txt
  ./mycode < tests.txt > myans.txt
  diff -q correct.txt myans.txt >/dev/null || { echo -e "\e[31
    mTest $i: WA\e[0m"; cat tests.txt; break; }
  echo -e "\e[32mTest $i: AC\e[0m"
done
```

### interactiveStress.py
19 lines
```python
import subprocess, random
def generate_permutation(n): return random.sample(range(1, n +
    1), n)
def handle_queries(hidden, n, max_q=6666):
    process = subprocess.Popen(["./solve"], stdin=subprocess.
        PIPE, stdout=subprocess.PIPE, text=True)
    process.stdin.write(f"{n}\n"); process.stdin.flush()
    for _ in range(max_q):
        query = process.stdout.readline().strip().split()
        if query[0] == "1":
            print("Correct!" if list(map(int, query[1:])) ==
                hidden else "Wrong!")
            break
        matches = sum(p == h for p, h in zip(map(int, query
            [1:]), hidden))
        process.stdin.write(f"{matches}\n"); process.stdin.
            flush()
    else: print("Query limit exceeded!")
    process.terminate()

n = 1000
hidden_permutation = generate_permutation(n)
print("Hidden permutation:", hidden_permutation)
handle_queries(hidden_permutation, n)
```

### optimization.cpp
54 lines
```cpp
// GNU Pbds
#include <ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using omap = tree<T, R, less<
    T>, rb_tree_tag, tree_order_statistics_node_update>;
// unordered_map
struct chash{
    size_t operator()(const pair<int,int>&x)const{
        return hash<long long>()(((long long)x.first)^(((long long)
            x.second)<<32));
    }
};
struct chash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::
            now().time_since_epoch().count();
```

```cpp
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<ii, int, chash> cnt;
// set custom operator
struct comp {
    bool operator()(const int& a, const int& b) const {
        return a < b;
    }
};
set<int, comp> st;
//Random Number
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
int x = rng() % 495;
//Running time
clock_t st = clock();
double t = (clock() - st) / (1.0 * CLOCKS_PER_SEC);
string line;  getline(cin, line);
istringstream iss;
string word;
while (iss >> word) {
    cout << word << "\n";
}
merge(t[v].begin(), t[v].end(), t[w].begin(), t[w].end(),
    back_inserter(t[u]));
#pragma GCC optimize("O3", "unroll-loops")
#pragma GCC target("avx2", "popcnt")
ulimit -s 65532
//Rollback
vector<pair<int*, int>> cng[N];
cng[i].push_back({&x, x});
for (auto [x, y]: cng[i]) {
    *x = y;
}
```

### runcpp.sh

```bash
#!/bin/bash
# chmod +x runcpp.sh
runcpp() {
    if [ "$#" -ne 1 ]; then
        echo "Usage: runcpp <cpp_file>"
        return 1
    fi
    g++ -std=c++17 -o program_name "$1" && ./program_name <
        input.in > output.in
    # Check if the output matches the expected output
    if diff -q output.in expected_output.in >/dev/null; then
        echo "Output matches expected output."
    else
        echo "Output does not match expected output."
        echo "Differences:"
        diff output.in expected_output.in
    fi
}
# Call the function if script is run
if [ "$#" -eq 1 ]; then
    runcpp "$1"
else
    echo "Please provide a C++ file."
fi
```

# Mathematics (2)

## 2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable $x_i$ is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where $A_i'$ is $A$ with the $i$'th column replaced by $b$.

## 2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots $r$ become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

## 2.3 Trigonometry

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

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

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

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

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

## 2.4 Geometry

### 2.4.1 Triangles
Side lengths: $a, b, c$

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

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

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

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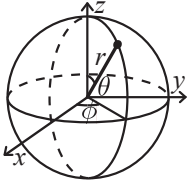


$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \text{acos}(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \text{atan2}(y, x) \end{aligned}$$

## 2.5   Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1 + \tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

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

## 2.6   Sums

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

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

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

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

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

## 2.7   Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$

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

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

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

## 2.8   Probability theory

Let $X$ be a discrete random variable with probability $p_X(x)$ of assuming the value $x$. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where $\sigma$ is the standard deviation. If $X$ is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent $X$ and $Y$,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

### 2.8.1   Discrete distributions
### Binomial distribution

The number of successes in $n$ independent yes/no experiments, each which yields success with probability $p$ is $\mathrm{Bin}(n,p)$, $n = 1, 2, \ldots, 0 \leq p \leq 1$.

$$p(k) = \binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu = np, \ \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small $p$.

Some properties:

$$\binom{n}{k} = \binom{n}{n-k} \qquad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} \qquad \sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\sum_{m=0}^n \binom{m}{k} = \binom{n+1}{k+1} \qquad \sum_{k=0}^m \binom{n+k}{k} = \binom{n+m-1}{m}$$

$$\sum_{k=0}^n k\binom{n}{k} = n2^{n-1} \qquad \sum_{k=0}^n \binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{k=0}^n \binom{n-k}{k} = F_{n+1} \qquad \sum_{k=0}^n (-1)^k\binom{n}{k} = 0$$

$$\sum_{i=0}^k \binom{m}{i}\binom{n}{k-i} \quad = \binom{m+n}{k}$$

# Data structures (3)

### 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:** $\mathcal{O}(\log N)$
0f4bdb, 19 lines

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

### SparseTable.h
**Description:** returns the gcd of the range [L, R].
**Usage:** `SparseTable<int> st(n); st.build(arr), st.query(L, R);`
Initial capacity must be a power of 2 (if provided).
**Time:** $\mathcal{O}(nlogn)$ to build, $\mathcal{O}(1)$ query
dc4d4d, 31 lines

```cpp
template <typename T> struct SparseTable {
  vector<vector<T>> st;
  vector<int> log;
  SparseTable(int n) {
    int maxLog = log2(n) + 1;
    st.assign(n, vector<T>(maxLog));
    log.assign(n + 1, 0);
    for (int i = 2; i <= n; i++) {
      log[i] = log[i / 2] + 1;
    }
  }
  void build(vector<T> &arr) {
    int n = arr.size();
    int maxLog = log2(n) + 1;
    for (int i = 0; i < n; i++) {
      st[i][0] = arr[i];
    }
    for (int j = 1; (1 << j) <= n; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        st[i][j] = fn(st[i][j - 1], st[i + (1 << (j - 1))][j -
          1]);
      }
    }
  }
  inline T fn(T a, T b) { return __gcd(a, b); }
  inline T query(int L, int R) {
    if (R < L)
      return 0;
    int j = log[R - L + 1];
    return fn(st[L][j], st[R - (1 << j) + 1][j]);
  }
};
```

### LazySegmentTree.h
**Description:** Segment tree with lazy propagation
**Usage:** `st.query(1, 1, n, l, r), st.upd(1, 1, n, l, r, val);`
**Time:** $\mathcal{O}(\log N)$
d6933c, 57 lines

```cpp
const int N = 1e5 + 9;
```

```cpp
int a[N];
struct segment_tree {
#define lc (n << 1)
#define rc ((n << 1) | 1)
  long long t[4 * N], lazy[4 * N];
  segment_tree() {
    memset(t, 0, sizeof t);
    memset(lazy, 0, sizeof lazy);
  }
  inline void push(int n, int b, int e) {
    if (lazy[n] == 0)
      return;
    t[n] = t[n] + lazy[n] * (e - b + 1);
    if (b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline long long combine(long long a, long long b) { return a
      + b; }
  inline void pull(int n) { t[n] = t[lc] + t[rc]; }
  void build(int n, int b, int e) {
    lazy[n] = 0;
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, long long v) {
    push(n, b, e);
    if (j < b || e < i)
      return;
    if (i <= b && e <= j) {
      lazy[n] = v; // set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j)
      return 0; // return null
    if (i <= b && e <= j)
      return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid + 1,
        e, i, j));
  }
};
```

## PersistentSegtree.h
**Description:** PresistentSegment Tree
<span style="float:right">792203, 26 lines</span>

```cpp
struct Node{
  ll val; Node *l,*r;
  Node(ll x):val(x),l(nullptr),r(nullptr){}
  Node(Node *ll,Node *rr):l(ll),r(rr){
    val=0; if(l)val+=l->val; if(r)val+=r->val;
  }
  Node(Node *cp):val(cp->val),l(cp->l),r(cp->r){}
```

```cpp
};
int n,cnt=1; ll a[N]; Node *roots[N];
Node *build(int l=1,int r=n){
  if(l==r) return new Node(a[l]);
  int mid=(l+r)/2;
  return new Node(build(l,mid),build(mid+1,r));
}
Node *update(Node *node,int val,int pos,int l=1,int r=n){
  if(l==r) return new Node(val);
  int mid=(l+r)/2;
  if(pos>mid) return new Node(node->l,update(node->r,val,pos,
      mid+1,r));
  else return new Node(update(node->l,val,pos,l,mid),node->r);
}
ll query(Node *node,int a,int b,int l=1,int r=n){
  if(l>b||r<a) return 0;
  if(l>=a&&r<=b) return node->val;
  int mid=(l+r)/2;
  return query(node->l,a,b,l,mid)+query(node->r,a,b,mid+1,r);
}
```

## UnionFind.h
**Description:** Disjoint-set data structure.
**Time:** $\mathcal{O}(\alpha(N))$
<span style="float:right">7aa27c, 14 lines</span>

```cpp
struct UF {
  vi e;
  UF(int n) : e(n, -1) {}
  bool sameSet(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

## UnionFindRollback.h
**Description:** Disjoint-set data structure with undo. If undo is not needed,
skip st, time() and rollback().
**Usage:** int t = uf.time(); ...; uf.rollback(t);
**Time:** $\mathcal{O}(\log(N))$
<span style="float:right">de4ad0, 21 lines</span>

```cpp
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

## SubMatrix.h
**Description:** Calculate submatrix sums quickly, given upper-left and lower-
right corners (half-open).
**Usage:** SubMatrix<int> m(matrix);
m.sum(0, 0, 2, 2); // top left 4 elements
**Time:** $\mathcal{O}(N^2 + Q)$
<span style="float:right">c59ada, 13 lines</span>

```cpp
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

## Matrix.h
**Description:** Basic operations on square matrices.
**Usage:** Matrix<int, 3> A;
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;
<span style="float:right">c43c7d, 26 lines</span>

```cpp
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

## 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 ("con-
vex hull trick").
**Time:** $\mathcal{O}(\log N)$
<span style="float:right">8ec1c7, 29 lines</span>

```cpp
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll 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(ll k, ll 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));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

### Triebit.h
**Description:** Trie bit
<span style="float:right">e703bf, 78 lines</span>

```
struct Trie {
  static const int B = 31;
  struct node {
    node *nxt[2];
    int sz;
    node() {
      nxt[0] = nxt[1] = NULL;
      sz = 0;
    }
  } *root;
  Trie() { root = new node(); }
  void insert(int val) {
    node *cur = root;
    cur->sz++;
    for (int i = B - 1; i >= 0; i--) {
      int b = val >> i & 1;
      if (cur->nxt[b] == NULL)
        cur->nxt[b] = new node();
      cur = cur->nxt[b];
      cur->sz++;
    }
  }
  int query(int x, int k) {
    // number of values s.t. val ^ x < k
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      if (cur == NULL)
        break;
      int b1 = x >> i & 1, b2 = k >> i & 1;
      if (b2 == 1) {
        if (cur->nxt[b1])
          ans += cur->nxt[b1]->sz;
        cur = cur->nxt[!b1];
      } else
        cur = cur->nxt[b1];
    }
    return ans;
  }
  int get_max(int x) { // returns maximum of val ^ x
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur->nxt[!k])
```

```
        cur = cur->nxt[!k], ans <<= 1, ans++;
      else
        cur = cur->nxt[k], ans <<= 1;
    }
    return ans;
  }
  int get_min(int x) { // returns minimum of val ^ x
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur->nxt[k])
        cur = cur->nxt[k], ans <<= 1;
      else
        cur = cur->nxt[!k], ans <<= 1, ans++;
    }
    return ans;
  }
  void del(node *cur) {
    for (int i = 0; i < 2; i++)
      if (cur->nxt[i])
        del(cur->nxt[i]);
    delete (cur);
  }
};
// Next Combination Mask
int next_combs_mask(int mask) {
  int lsb = -mask & mask;
  return (((mask + lsb) ^ mask) / (lsb << 2)) | (mask + lsb);
}
// Iterate over submask in decreasing order
for (int s = mask; s > 0; s = (s - 1) & mask) {
}
```

### FenwickTree.h
**Description:** Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
**Time:** Both operations are $\mathcal{O}(\log N)$.
<span style="float:right">2925b4, 26 lines</span>

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  void upd(int l, int r, ll dif) {
    upd(l, dif);
    upd(r + 1, -dif);
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

### 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" <span style="float:right">157f07, 22 lines</span>

```
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);
  }
  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;
  }
};
```

### 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:** $\mathcal{O}(N\sqrt{Q})$
<span style="float:right">a12ef4, 47 lines</span>

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
```

```
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) rep(end,0,2) {
        int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                    else { add(c, end); in[c] = 1; } a = c; }
        while (!(L[b] <= L[a] && R[a] <= R[b]))
            I[i++] = b, b = par[b];
        while (a != b) step(par[a]);
        while (i--) step(I[i]);
        if (end) res[qi] = calc();
    }
    return res;
}
```

# Numerical (4)

## 4.1    Polynomials and recurrences

### Polynomial.h

<div align="right">c9b7b0, 17 lines</div>

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

### PolyRoots.h

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

<div align="right">"Polynomial.h"    b00bfe, 23 lines</div>

```
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;
}
```

## 4.2    Optimization

### 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.

<div align="right">4756fc, 7 lines</div>

```
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;
}
```

### Simplex.h

**Description:** Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. 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:** $\mathcal{O}\left(NM * \#pivots\right)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}\left(2^n\right)$ in the general case.

<div align="right">aa8530, 62 lines</div>

```
typedef double T; // long double, Rational, double + mod<P>...
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:** $\mathcal{O}\left(N^3\right)$

<div align="right">bd5cec, 15 lines</div>

```
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;
}
```

### 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:** $\mathcal{O}\left(n^3\right)$

<div align="right">ebfff6, 32 lines</div>

```
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];
```

```cpp
      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;
}
```

## MatrixExpo.h
**Description:** Matrix Exponentiation

<span style="float:right">f7a88b, 33 lines</span>

```cpp
using row = vector<int>;
using matrix = vector<row>;
matrix unit_mat(int n) {
  matrix I(n, row(n));
  for (int i = 0; i < n; ++i){
    I[i][i] = 1;
  }
  return I;
}
matrix mat_mul(matrix a, matrix b) {
  int m = a.size(), n = a[0].size();
  int p = b.size(), q = b[0].size();
  // assert(n==p);
  matrix res(m, row(q));
  for (int i = 0; i < m; ++i){
    for (int j = 0; j < q; ++j){
      for (int k = 0; k < n; ++k){
        res[i][j] = (res[i][j] + a[i][k]*b[k][j]) % mod;
      }
    }
  }
  return res;
}
matrix mat_exp(matrix a, int p) {
  int m = a.size(), n = a[0].size(); // assert(m==n);
  matrix res = unit_mat(m);
  while (p) {
    if (p&1)  res = mat_mul(a, res);
    a = mat_mul(a, a);
    p >>= 1;
  }
  return res;
}
```

## Gauss.h
**Description:** Gauss

<span style="float:right">36bf8e, 60 lines</span>

```cpp
ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}
pair <int, ld> gaussJordan (int n, int m, ld eq[N][N], ld res[N
    ]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
```

```cpp
    for (int k = i; k < n; ++k) if (fabs(eq[k][j]) > fabs(eq[
        piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x * eq[i][l];
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[
        pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] * res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanModulo (int n, int m, int eq[N][N],
    int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] > eq[piv][j]) piv
        = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det = (ll) det * eq
        [i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) % mod;
      for (int l = j; l <= m; ++l) if (eq[i][l]) eq[k][l] = (eq
          [k][l] + mod_sq - x * eq[i][l]) % mod;
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[
        pos[i]][m] * bigMod(eq[pos[i]][i], -1, mod) % mod;
  }
  for (int i = 0; i < n; ++i) {
    ll cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += (ll) eq[i][j] * res[j],
        cur %= mod;
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
```

## Xorbasis.h
**Description:** Xor basis

<span style="float:right">9b7a52, 13 lines</span>

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

}
}

# 4.4   Fourier transforms

## FastFourierTransform.h
**Description:** Returns coefficient of multiplication of two polynomials

<span style="float:right">ac69ab, 46 lines</span>

```cpp
const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w *
          wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<ll> &a, vector<ll> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a
      );
  while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

## FastFourierTransformMod.h
**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (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"

<span style="float:right">b82773, 22 lines</span>

```cpp
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
```

```
rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
fft(L), fft(R);
rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
}
fft(outl), fft(outs);
rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
}
return res;
}
```

## 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. 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, mod).

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

"../number-theory/ModPow.h"                         ced03d, 35 lines
```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
        n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n)
        out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

# 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.

"euclid.h"                                           35bfea, 18 lines
```
const ll mod = 17; // change to something else
struct Mod {
    ll x;
    Mod(ll xx) : x(xx) {}
    Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
    Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
    Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
    Mod operator/(Mod b) { return *this * invert(b); }
    Mod invert(Mod a) {
        ll x, y, g = euclid(a.x, mod, x, y);
        assert(g == 1); return Mod((x + mod) % mod);
    }
    Mod operator^(ll e) {
        if (!e) return Mod(1);
        Mod r = *this ^ (e / 2); r = r * r;
        return e&1 ? *this * r : r;
    }
};
```

### ModInverse.h

**Description:** Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.

                                                     6f684f, 3 lines
```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

### ModLog.h

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

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

                                                     c040b8, 11 lines
```
ll modLog(ll a, ll b, ll m) {
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = f = e * a % m) != b % m)
        A[e * b % m] = j++;
    if (e == b % m) return j;
    if (__gcd(m, e) == __gcd(m, b))
        rep(i,2,n+2) if (A.count(e = e * f % m))
            return n * i - A[e];
    return -1;
}
```

### ModMulLL.h

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

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

                                                     bbbd8f, 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 >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

### ModSqrt.h

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

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

"ModPow.h"                                           19a793, 24 lines
```
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    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 smaller than LIM.

**Time:** LIM=1e9 ≈ 1.5s

                                                     6b2912, 20 lines
```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
    vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
    vector<pii> cp;
    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
        cp.push_back({i, i * i / 2});
        for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
    }
    for (int L = 1; L <= R; L += S) {
        array<bool, S> block{};
        for (auto &[p, idx] : cp)
            for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
        rep(i,0,min(S, R - L))
            if (!block[i]) pr.push_back((L + i) * 2 + 1);
    }
    for (int i : pr) isPrime[i] = 1;
    return pr;
}
```

### 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 $a^b \bmod c$.

"ModMulLL.h"                                          60dcd1, 12 lines
```
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {   // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
```

```
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

## Factor.h

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

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

"ModMulLL.h", "MillerRabin.h"         d8d98d, 18 lines

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

## 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 $a \pmod{b}$.

33ba8f, 5 lines

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

### 5.3.1 Chinese Remainder Theorem

Let $m = m_1 \cdot m_2 \cdots m_k$, where $m_i$ are pairwise coprime. In addition to $m_i$, we are also given a system of congruences

$$\begin{cases} a & \equiv & a_1 \pmod{m_1} \\ a & \equiv & a_2 \pmod{m_2} \\ & \vdots & \\ a & \equiv & a_k \pmod{m_k} \end{cases}$$

where $a_i$ are some given constants. CRT will give the unique solution modulo $m$.

### 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 $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
**Time:** $\log(n)$

"euclid.h"         04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
```

---

```
  if (n > m) swap(a, b), swap(m, n);
  ll 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;
}
```

## CRT2.h

**Description:** Chinese Remainder Theorem.
**Time:** $\mathcal{O}(n)$ here $n$ is the number of congruences.

c52f3f, 17 lines

```
struct Congruence {
  ll a, m;
};
ll CRT(vector<Congruence> const &congruences) {
  ll M = 1;
  for (auto const &congruence : congruences) {
    M *= congruence.m;
  }
  ll solution = 0;
  for (auto const &congruence : congruences) {
    ll a_i = congruence.a;
    ll M_i = M / congruence.m;
    ll N_i = mod_inv(M_i, congruence.m);
    solution = (solution + a_i * M_i % M * N_i) % M;
  }
  return solution;
}
```

### 5.3.2 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

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

### Diophantine.h

**Description:** Provides any solution of $ax + by = c$
**Time:** $\mathcal{O}(\log(n))$

"euclid.h"         7b0328, 8 lines

```
bool find_any_solution(int a, int b, int c, int &x0, int &y0,
    int &g) {
  g = euclid(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g, y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

### phiFunction.h

**Description:** Euler's $\phi$ function is defined as $\phi(n) := \#$ of positive integers $\le n$ 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 $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1}...(p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$. $\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
**Euler's thm:** $a, n$ coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
**Fermat's little thm**: $p$ prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p} \; \forall a$.
**Time:** $\mathcal{O}(\log \log n)$ and $\mathcal{O}(\sqrt{n})$ for the second version.

3998a6, 21 lines

---

```
const int LIM = 5000000;
int phis[LIM];

void calculatePhi() {
  rep(i, 0, LIM) phis[i] = i & 1 ? i : i / 2;
  for (int i = 3; i < LIM; i += 2)
    if (phis[i] == i)
      for (int j = i; j < LIM; j += i)
        phis[j] -= phis[j] / i;
}
int phi(int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      result -= result / i;
    }
  }
  if (n > 1) result -= result / n;
  return result;
}
```

## 5.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

## 5.5 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

## 5.6 Fibonacchi

Fibonacci numbers are defined by $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$. Again, $F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} \approx \frac{\phi^n}{\sqrt{5}}$, where $\phi = \frac{1+\sqrt{5}}{2}$ and $\hat{\phi} = \frac{1-\sqrt{5}}{2}$. Some important properties of Fibonacci numbers:

$$F_{n-1}F_{n+1} - F_n^2 = (-1)^n \qquad F_{n+k} = F_{k-1}F_n + F_k F_{n+1}$$

$$F_{2n} = F_n(F_{n-1} + F_{n+1}) \qquad F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$n|m \Leftrightarrow F_n|F_m \qquad \gcd(F_m, F_n) = F_{\gcd(m,n)}$$

$$F_A F_B = F_{k+1}F_A^2 + F_k F_A F_{A-1} \qquad \sum_{i=0}^{n} F_i^2 = F_{n+1}F_n$$

$$\sum_{i=0}^{n} F_i F_{i+1} = F_{n+1}^2 - (-1)^n \qquad \sum_{i=0}^{n} F_i F_{i-1} = \sum_{i=0}^{n-1} F_i F_{i+1}$$

$$\sum_{0 \le k \le n} \binom{n-k}{k} = F_{n+1}$$

$$\gcd(F_n, F_{n+1}) = \gcd(F_n, F_{n+2}) \qquad = \gcd(F_{n+1}, F_{n+2}) = 1$$

## Fibonacchi.h

**Description:** Fast doubling Fibonacci algorithm. Returns F(n) and F(n+1).
**Time:** $\mathcal{O}(\log n)$

<div align="right">f4c71e, 11 lines</div>

```cpp
pair<int, int> fib(int n) {
  if (n == 0)
    return {0, 1};
  auto p = fib(n >> 1);
  int c = p.first * (2 * p.second - p.first);
  int d = p.first * p.first + p.second * p.second;
  if (n & 1)
    return {d, c + d};
  else
    return {c, d};
}
```

## 5.7 Primitive Roots

In modular arithmetic, a number $g$ is called a 'primitive root modulo n' if every number coprime to $n$ is congruent to a power of $g$ modulo $n$. Mathematically, $g$ is a 'primitive root modulo n' if and only if for any integer $a$ such that $\gcd(a, n) = 1$, there exists an integer $k$ such that:

$$g^k \equiv a \pmod{n}.$$

$k$ is then called the 'index' or 'discrete logarithm' of $a$ to the base $g$ modulo $n$. $g$ is also called the 'generator' of the multiplicative group of integers modulo $n$.

In particular, for the case where $n$ is a prime, the powers of primitive root runs through all numbers from 1 to $n-1$.

## PrimitiveRoot.h

**Description:** Primitive root of a prime number.
**Time:** $\mathcal{O}(sqrt(n) * log(n))$

<div align="right">c6d472, 28 lines</div>

```cpp
int powmod (int a, int b, int p) {
  int res = 1;
  while (b)
    if (b & 1)
      res = int (res * 1ll * a % p),  --b;
    else
      a = int (a * 1ll * a % p),  b >>= 1;
  return res;
}
int generator (int p) {
  vector<int> fact;
  int phi = p-1,  n = phi;
  for (int i=2; i*i<=n; ++i)
    if (n % i == 0) {
      fact.push_back (i);
      while (n % i == 0)
        n /= i;
    }
  if (n > 1)
    fact.push_back (n);
  for (int res=2; res<=p; ++res) {
    bool ok = true;
    for (size_t i=0; i<fact.size() && ok; ++i)
      ok &= powmod (res, phi / fact[i], p) != 1;
    if (ok)  return res;
  }
  return -1;
}
```

## 5.8 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 $n < 1e19$.

## 5.9 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

## Mobius.h

**Description:** The sieve of Eratosthenes computes the smallest prime factor for each number and the mobius function computes the mobius function for each number.
**Time:** $\mathcal{O}(nloglogn)$ for sieve and $\mathcal{O}(n)$ for mobius

<div align="right">a7a1e2, 25 lines</div>

```cpp
ll const N = 1e7;
vector<ll> spf(N), mobius(N);
void sieve() {
  for (ll i = 2; i < N; i++) {
    if (spf[i] == 0) {
      for (ll j = i; j < N; j += i) {
        if (spf[j] == 0) {
          spf[j] = i;
        }
      }
    }
  }
}
void mobius() {
  for (ll i = 1; i < N; i++) {
    if (i == 1)
      mobius[i] = 1;
    else {
      if (spf[i / spf[i]] == spf[i])
        mobius[i] = 0;
      else
        mobius[i] = -1 * mobius[i / spf[i]];
    }
  }
}
```

### 5.9.1 GCD and LCM

if $m$ is any integer, then $\gcd(a + m \cdot b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if $a_1$ and $a_2$ are relatively prime, then
$\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \mathrm{lcm}(b, c)) = \mathrm{lcm}(\gcd(a, b), \gcd(a, c))$.
$\mathrm{lcm}(a, \gcd(b, c)) = \gcd(\mathrm{lcm}(a, b), \mathrm{lcm}(a, c))$.

For non-negative integers $a$ and $b$, where $a$ and $b$ are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

$$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$$

$$\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$$

$$\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n} \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^{n} \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^{n} \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k, n)} - 1, \text{ for } n > 1$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \gcd(i, j) = \sum_{d=1}^{n} \phi(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^{n} \phi(i) i^2$$

$$F(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathrm{lcm}(i, j) =$$

$$\sum_{l=1}^{n} \left( \frac{\left(1 + \lfloor \frac{n}{l} \rfloor\right) \left(\lfloor \frac{n}{l} \rfloor\right)}{2} \right)^2 \sum_{d|l} \mu(d) ld$$

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

# 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.h

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

<div align="right">044568, 7 lines</div>

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

### 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.

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

### 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 $g$ ($g.x = x$).

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.

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

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

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

Partition.h
**Description:** Finds partition of a number.
**Time:** $\mathcal{O}\left(n\sqrt{n}\right)$                                    a2eb0c, 12 lines
```
for (int i = 1; i <= n; ++i) {
  pent[2 * i - 1] = i * (3 * i - 1) / 2;
```

```
  pent[2 * i] = i * (3 * i + 1) / 2;
}
p[0] = 1;
for (int i = 1; i <= n; ++i) {
  p[i] = 0;
  for (int j = 1, k = 0; pent[j] <= i; ++j) {
    if (k < 2) p[i] = add(p[i], p[i - pent[j]]);
    else p[i] = sub(p[i], p[i - pent[j]]); ++k, k &= 3;
  }
}
```

### 6.2.2  Lucas' Theorem

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

- $\binom{m}{n}$ is divisible by p if and only if at least one of the base-$p$ digits of $n$ is greater than the corresponding base-$p$ digit of $m$.

- The number of entries in the $n$th row of Pascal's triangle that are not divisible by $p = \prod_{i=0}^{k}(n_i + 1)$

- All entries in the $(p^k - 1)th$ row are not divisble by $p$.

- $\binom{n}{m} \equiv \lfloor \frac{n}{p} \rfloor (\mod p)$

### 6.2.3  Binomials

multinomial.h
**Description:** Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$.                                    a0a312, 5 lines
```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
  return c;
}
```

### 6.2.4  2nd Kaplansky's Lemma

The number of ways of selecting $k$ objects, no two consecutive, from $n$ labelled objects arrayed in a circle is $\frac{n}{k}\binom{n-k-1}{k-1} = \frac{n}{n-k}\binom{n-k}{k}$

### 6.2.5  Distinct Objects into Distinct Bins

- $n$ distinct objects into $r$ distinct bins $= r^n$

- Among $n$ distinct objects, exactly $k$ of them into r distincts bins $= \binom{n}{k} r^k$

- $n$ distinct objects into $r$ distinct bins such that each bin contains at least one object $= \sum_{i=0}^{r}(-1)^i\binom{r}{i}(r-i)^n$

## 6.3  General purpose numbers

### 6.3.1  Bernoulli numbers

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

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.

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

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

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

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

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

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

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

### 6.3.4  Stirling numbers of the second kind

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

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

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

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

Stirling2nd.h
**Description:** Stirling number of the second kind.
**Time:** $\mathcal{O}\left(k \log n\right)$                                    6f78c8, 10 lines
```
ll get_sn2(int n, int k) {
  ll sn2 = 0;
  for (int i = 0; i <= k; ++i) {
    ll now = nCr(k, i) * powmod(k - i, n, mod) % mod;
    if (i & 1) now = now * (mod - 1) % mod;
    sn2 = (sn2 + now) % mod;
  }
  sn2 = sn2 * ifact[k] % mod;
  return sn2;
}
```

### 6.3.5  Bell numbers

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

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

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

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

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

# Graph (7)

## 7.1 Fundamentals

BellmanFord.h
**Description:** Calculates shortest paths from $s$ in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
**Time:** $\mathcal{O}(VE)$
830a8f, 21 lines
```cpp
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;
  }
}
```

## 7.2 Network flow

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.
58385b, 71 lines
```cpp
#include <bits/extc++.h>
```

```cpp
const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> 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, ll 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; ll di;
    __gnu_pbds::priority_queue<pair<ll, 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 (edge& e : ed[s]) if (!seen[e.to]) {
        ll 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, ll> maxflow(int s, int t) {
    ll totflow = 0, 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
  }
};
```

Dinic.h
**Description:** Flow algorithm with complexity $O(VE \log U)$ where $U = \max |\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.
d7f0f1, 42 lines
```cpp
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void addEdge(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

## 7.3 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:** $\mathcal{O}(\sqrt{V}E)$
953d6f, 39 lines
```cpp
struct HopcroftKarp {
  int n, m;
  vector<int> l, r, lv, ptr;
  vector<vector<int>> adj;
  HopcroftKarp(int _n, int _m) : n(_n), m(_m), adj(n + m) {}
  void addEdge(int u, int v) { adj[u].emplace_back(v + n); }
  void bfs() {
    lv.assign(n, -1); queue<int> q;
```

```cpp
        for (int i = 0; i < n; ++i) if (l[i] == -1) lv[i] = 0,
            q.push(i);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v : adj[u]) if (r[v] != -1 && lv[r[v]] ==
                -1) {
                lv[r[v]] = lv[u] + 1; q.push(r[v]);
            }
        }
    }
    bool dfs(int u) {
        for (int &i = ptr[u]; i < adj[u].size(); ++i) {
            int v = adj[u][i];
            if (r[v] == -1 || (lv[r[v]] == lv[u] + 1 && dfs(r[v
                ]))) {
                l[u] = v, r[v] = u; return true;
            }
        }
        return false;
    }
    int maxMatching() {
        int match = 0;
        l.assign(n + m, -1), r.assign(n + m, -1);
        while (true) {
            ptr.assign(n, 0); bfs(); int cnt = 0;
            for (int i = 0; i < n; ++i) if (l[i] == -1 && dfs(i
                )) cnt++;
            if (cnt == 0) break; match += cnt;
        }
        return match;
    }
    void printMatching() {
        for (int i = 0; i < n; ++i) if (l[i] != -1) cout << l[i
            ] - n + 1 << " ";
    }
};
```

### 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:** $\mathcal{O}\left(N^2 M\right)$

```cpp
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], j1, 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], j1 = j;
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
```

```cpp
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}
```

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

```cpp
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);
}
```

### ArticulationPoint.h
**Description:** Finding articulation points in a graph.

```cpp
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N), ap;
void dfs(int u, int p) {
    tin[u] = low[u] = t++;
    int is_ap = 0, child = 0;
    for (int v : adj[u]) {
        if (v != p) {
            if (tin[v] != -1) {
                low[u] = min(low[u], tin[v]);
            } else {
                child++;
                dfs(v, u);
                if (tin[u] <= low[v]) is_ap = 1;
                low[u] = min(low[u], low[v]);
            }
        }
    }
    if ((p != -1 or child > 1) and is_ap)
        ap.push_back(u);
}
dfs(0, -1);
```

### Bridge.h
**Description:** Finds all the bridges in a graph.

```cpp
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    bool parent_skipped = false;
    for (int to : adj[v]) {
        if (to == p && !parent_skipped) {
            parent_skipped = true;
            continue;
        }
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
```

### BiconnectedComponents.h
**Description:** Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
**Usage:** int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
**Time:** $\mathcal{O}\left(E + V\right)$

```cpp
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, top = me;
    for (auto [y, e] : ed[at]) if (e != par) {
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */ }
        }
    }
    return top;
}
template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

## 2sat.h

**Description:** Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ 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.

5f9706, 50 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), gr(2*n) {}
  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }
  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

## EulerWalk.h

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

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

780b64, 15 lines

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

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

e210e2, 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    Trees

### BinaryLifting.h

**Description:** Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.

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

bfce85, 23 lines

```
vector<vi> treeJump(vi& P){
  int on = 1, d = 1;
  while(on < sz(P)) on *= 2, d++;
  vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
}
int jmp(vector<vi>& tbl, int nod, int steps){
  rep(i,0,sz(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}
int lca(vector<vi>& tbl, vi& depth, int a, int b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(tbl, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = sz(tbl); i--;) {
    int c = tbl[i][a], d = tbl[i][b];
    if (c != d) a = c, b = d;
  }
  return tbl[0][a];
}
```

### LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

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

"../data-structures/RMQ.h"

0f62fb, 20 lines

```
struct LCA {
  int T = 0;
  vi time, path, ret;
  RMQ<int> rmq;

  LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
  void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(time[v]);
      dfs(C, y, v);
    }
  }
  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

### DsuOnTree.h

**Description:** Dsu on tree

e02277, 47 lines

```
void dfs(int u, int p) {
  node[tt] = u;
  tin[u] = tt++, sz[u] = 1, hc[u] = -1;
  for (auto v : adj[u]) {
    if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if (hc[u] == -1 or sz[hc[u]] < sz[v]) {
        hc[u] = v;
      }
    }
  }
  tout[u] = tt - 1;
}
```

```cpp
void dsu(int u, int p, int keep) {
  for (int v : adj[u]) {
    if (v != p and v != hc[u]) {
      dsu(v, u, 0);
    }
  }
  if (hc[u] != -1) {
    dsu(hc[u], u, 1);
  }
  for (auto v : adj[u]) {
    if (v != p and v != hc[u]) {
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // get ans in case of ans is related to simple path or
            pair
      }
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // Add contribution of node w
      }
    }
  }
  // Add contribution of node u
  // get ans in case ans is related to subtree
  if (!keep) {
    for (int i = tin[u]; i <= tout[u]; ++i) {
      int w = node[i];
      // remove contribution of node w
    }
    // Data structure in initial state (empty contribution)
  }
}
dfs(0, 0);
dsu(0, 0, 0);
```

## Vtree.h
**Description:** Virtual tree
**Time:** $\mathcal{O}(N \log N)$
<div align="right">3a46a6, 69 lines</div>

```cpp
const int LG=19;
vector<vector<int>> adj(N), new_adj(N);
vector<int> need;
int anc[N][LG], par[N], _time=0, in[N], out[N], h[N], stk[N],
    imp[N], ans=0;
void dfs_pre(int ind, int p, int hi=0){
    in[ind]=_time++;
    anc[ind][0]=p;
    par[ind]=p;
    h[ind]=hi;
    for(auto i:adj[ind]) if(i!=p) dfs_pre(i,ind,hi+1);
    out[ind]=_time++;
}
bool is_anc(int ind, int anc){
    return in[ind]>=in[anc] and out[ind]<=out[anc];
}
int get_lca(int ind1, int ind2){
    if(is_anc(ind1,ind2)) return ind2;
    if(is_anc(ind2,ind1)) return ind1;
    for(int i=LG-1;i>=0;i--){
        if(!is_anc(ind1,anc[ind2][i])) ind2=anc[ind2][i];
    }
    return anc[ind2][0];
}
void add_edge(int ind1, int ind2){
    if(ind1!=ind2){
        new_adj[ind1].push_back(ind2);
        new_adj[ind2].push_back(ind1);
    }
}
```

```cpp
void buildTree(vector<int> &nodes){
    if(nodes.size()<=1) return;
    sort(nodes.begin(),nodes.end(),[](int x,int y){ return in[x
        ]<in[y]; });
    int root=get_lca(nodes[0],nodes.back()),sz=nodes.size(),ptr
        =0;
    ptr=0,stk[ptr++]=root;
    for(int i=0;i<sz;++i){
        int u=nodes[i],lca=get_lca(u,stk[ptr-1]);
        if(lca==stk[ptr-1]) stk[ptr++]=u;
        else{
            while(ptr>1 and h[stk[ptr-2]]>=h[lca]) add_edge(stk
                [ptr-2],stk[ptr-1]),--ptr;
            if(stk[ptr-1]!=lca){
                add_edge(lca,stk[--ptr]);
                stk[ptr++]=lca,nodes.emplace_back(lca);
            }
            stk[ptr++]=u;
        }
    }
    if(find(nodes.begin(),nodes.end(),root)==nodes.end()) nodes
        .emplace_back(root);
    for(int j=0;j+1<ptr;++j) add_edge(stk[j],stk[j+1]);
    sort(nodes.begin(),nodes.end(),[](int x,int y){ return in[x
        ]<in[y]; });
}
int find_ans(int ind, int p){
    int now=0;
    if(imp[ind] and imp[p] and par[ind]==p and p!=ind){
        ans=-1e6;
        return 0;
    }
    for(auto i:new_adj[ind]){
        if(i!=p) now+=find_ans(i,ind);
    }
    if(imp[ind]){
        ans+=(now);
        return 1;
    }
    else{
        if(now>1) { ans++; return 0; }
        else if(now==1) return 1;
        else return 0;
    }
}
```

## HLD.h
**Description:** Heavy Light Decomposition
<div align="right">4daed2, 41 lines</div>

```cpp
int query(int a, int b) {
  int res = 0;
  for (; head[a] != head[b]; b = parent[head[b]]) {
    if (depth[head[a]] > depth[head[b]])
      swap(a, b);
    int cur_path_max = segquery(1, 0, n - 1, pos[head[b]], pos[
        b]);
    res = MAX(res, cur_path_max);
  }
  if (depth[a] > depth[b])  swap(a, b);
  int cur_path_max = segquery(1, 0, n - 1, pos[a] + 1, pos[b]);
  res = MAX(res, cur_path_max);
  return res;
}
int dfs(int ind, int p) {
  int sz = 1;
  int mx_sz = 0;
  parent[ind] = p;
  depth[ind] = depth[p] + 1;
  for (auto [x, y] : adj[ind]) {
```

```cpp
    if (x == p)
      continue;
    int c_size = dfs(x, ind);
    edgVal[x] = y;
    sz += c_size;
    if (c_size > mx_sz) {
      heavy[ind] = x;
      mx_sz = c_size;
    }
  }
  return sz;
}
void decompose(int ind, int h) {
  head[ind] = h;
  pos[ind] = cur_pos++;
  euler.push_back(ind);
  if (heavy[ind] != -1) decompose(heavy[ind], h);
  for (auto [x, y] : adj[ind]) {
    if (x != heavy[ind] and x != parent[ind])
      decompose(x, x);
  }
}
```

## CentroidDecomp.h
**Description:** Centroid decompose
<div align="right">a3ba5c, 26 lines</div>

```cpp
void calc_sz(int u, int p) {
  sz[u] = 1;
  for (auto v : adj[u]) {
    if (v != p and !is_cen[v]) {
      calc_sz(v, u);
      sz[u] += sz[v];
    }
  }
}
int get_cen(int u, int p, int n) {
  for (auto v : adj[u]) {
    if (v != p and !is_cen[v] and 2 * sz[v] > n)
      return get_cen(v, u, n);
  }
  return u;
}
void decompose(int u = 0, int p = -1, int d = 0) {
  calc_sz(u, p);
  int c = get_cen(u, p, sz[u]);
  is_cen[c] = 1, cpar[c] = p, cdep[c] = d;
  for (int v : adj[c]) {
    if (!is_cen[v])
      decompose(v, c, d + 1);
  }
}
decompose();
```

## 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 $\mathcal{O}(\log N)$.
<div align="right">0fb462, 90 lines</div>

```cpp
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 c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != 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;
  }
};
```

## DPOnTree.h
**Description:** DPonTree
<span>0cbfdd, 41 lines</span>

```
void dfs(int ind, int p)
{
  int tmp = 0;
  int s=1;
  for(auto i: adj[ind])
  {
    if(i!=p)
    {
      dfs(i, ind);
      tmp += (down[i]+sz[i]);
      s+=sz[i];
    }
  }
  sz[ind] = s;
  down[ind] = tmp;
}
void dfs2(int ind, int p)
{
  if(ind!=p)up[ind] += (up[p] + n - sz[p]), up[ind]++;
  int tmp=0;
  for(auto i: adj[ind])
  {
    if(i!=p)
    {
      up[i] += tmp;
      tmp += (down[i]+2LL*sz[i]);
    }
  }
  reverse(adj[ind].begin(), adj[ind].end());
  tmp = 0;
  for(auto i: adj[ind])
  {
    if(i!=p)
    {
      up[i] +=tmp;
      tmp += (down[i]+2LL*sz[i]);
    }
  }
  ans[ind] = down[ind] + up[ind];
  for(auto i: adj[ind])if(i!=p)dfs2(i, ind);
}
```

## 7.7 Math

### 7.7.1 Number of Spanning Trees
Create an $N \times N$ matrix mat, and for each edge $a \to b \in G$, do mat[a][b]--, mat[b][b]++ (and mat[b][a]--, mat[a][a]++ if $G$ is undirected). Remove the $i$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).

### 7.7.2 Erdős–Gallai theorem
A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

# 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.)
<span>47ec0a, 28 lines</span>

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

### lineDistance.h
**Description:**
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.
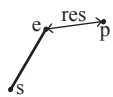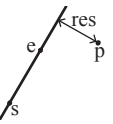"Point.h"
<span>f6bf6b, 4 lines</span>

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

### SegmentDistance.h
**Description:**
Returns the shortest distance between point p and the line segment from point s to e.

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

"Point.h" 5c88f4, 6 lines

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

### SegmentIntersection.h
**Description:**
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.
**Usage:** `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

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

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

### lineIntersection.h
**Description:**
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.
**Usage:** `auto res = lineInter(s1,e1,s2,e2);`
`if (res.first == 1)`
`cout << "intersection point at " << res.second << endl;`

"Point.h" a01f81, 8 lines

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

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

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

"Point.h" 3af81c, 9 lines

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

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

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

"Point.h" c597e8, 3 lines

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

### linearTransformation.h
**Description:**

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

"Point.h" 03a306, 6 lines

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

### Angle.h
**Description:** A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.
**Usage:** `vector<Angle> v = {w[0], w[0].t360() ...};` // sorted
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i`

0f0602, 35 lines

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

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
```

```
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 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" 84d6d3, 11 lines

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

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

"Point.h" b0153d, 13 lines

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

### CirclePolygonIntersection.h
**Description:** Returns the area of the intersection of a circle with a ccw polygon.
**Time:** $\mathcal{O}(n)$

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

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

```
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

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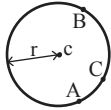


"Point.h"                                                                    1caa3a, 9 lines
```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
    abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

## MinimumEnclosingCircle.h
**Description:** Computes the minimum circle that encloses a set of points.
**Time:** expected $\mathcal{O}(n)$

"circumcircle.h"                                                            09dd0a, 17 lines
```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

# 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 in = inPolygon(v, P{3, 3}, false);
**Time:** $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h"                              2bf504, 11 lines
```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

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

"Point.h"                                                                    f12300, 6 lines
```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

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

"Point.h"                                                                    9706dc, 9 lines
```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

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



"Point.h", "lineIntersection.h"                                            f2b7d4, 13 lines
```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```
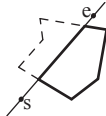
## MinkowskiSum.h
**Description:** Returns the summation polygon of two polygons.
**Time:** $\mathcal{O}(n + m)$

"Point.h"                                                                    c66dc3, 26 lines
```
void reorder_polygon(vector<pt> & 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());
}

vector<pt> minkowski(vector<pt> P, vector<pt> Q){
    // the first vertex must be the lowest
    reorder_polygon(P); reorder_polygon(Q);
    // we must ensure cyclic indexing
    P.push_back(P[0]); P.push_back(P[1]);
    Q.push_back(Q[0]); Q.push_back(Q[1]);
    // main part
```

```
    vector<pt> ret;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() - 2){
        ret.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
        i += cross >= 0 && i < P.size() - 2;
        j += cross <= 0 && j < Q.size() - 2;
    }
    return ret;
}
```

## 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"                                                                    ddbd5e, 32 lines
```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts, bool allow_collinear =
    false) {
  if (sz(pts) <= 1)
    return pts;
  sort(all(pts));
  pts.erase(unique(all(pts), [&](const P &a, const P &b) {
        return a == b; }),
        pts.end());
  vector<P> hull;
  auto condition = [&](const P &a, const P &b, const P &c) ->
    bool {
    ll cross = a.cross(b, c);
    if (allow_collinear)
      return cross < 0;
    else
      return cross <= 0;
  };
  for (const P &p : pts) {
    while (sz(hull) >= 2 &&
        condition(hull[sz(hull) - 2], hull[sz(hull) - 1], p)
        )
      hull.pop_back();
    hull.push_back(p);
  }
  int lower_size = sz(hull) + 1;
  for (int i = sz(pts) - 1; i >= 0; --i) {
    const P &p = pts[i];
    while (sz(hull) >= lower_size &&
        condition(hull[sz(hull) - 2], hull[sz(hull) - 1], p)
        )
      hull.pop_back();
    hull.push_back(p);
  }
  hull.pop_back();
  return hull;
}
```

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

"Point.h"                                                                    c571b8, 12 lines
```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
```

```
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  }
  return res.second;
}
```

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

<u>"Point.h", "sideOf.h", "OnSegment.h"</u>     71446b, 14 lines

```
typedef Point<ll> P;

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

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

<u>"Point.h"</u>     7cf45b, 39 lines

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

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

```
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

## 8.4   Misc. Point Set Problems

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

<u>"Point.h"</u>     ac41a6, 17 lines

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

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

<u>"Point.h"</u>     df6f59, 23 lines

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

### SweepLine.h
**Description:** Returns any intersecting segments, or -1, -1 if none exist.
**Time:** $\mathcal{O}(N \log N)$

    4709c6, 79 lines

```
const double EPS = 1E-9;
struct pt {
  double x, y;
};
struct seg {
  pt p, q;
  int id;
  double get_y(double x) const {
    if (abs(p.x - q.x) < EPS)
      return p.y;
    return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
  }
};
bool intersect1d(double l1, double r1, double l2, double r2) {
  if (l1 > r1)
    swap(l1, r1);
  if (l2 > r2)
    swap(l2, r2);
  return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt &a, const pt &b, const pt &c) {
  double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a
        .x);
  return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg &a, const seg &b) {
  return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
         intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
         vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
         vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg &a, const seg &b) {
  double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
  return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
  double x;
  int tp, id;
  event() {}
  event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
  bool operator<(const event &e) const {
    if (abs(x - e.x) > EPS)
      return x < e.x;
    return tp > e.tp;
  }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
  return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) { return ++it; }
pair<int, int> solve(const vector<seg> &a) {
  int n = (int)a.size();
  vector<event> e;
  for (int i = 0; i < n; ++i) {
    e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
    e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
  }
  sort(e.begin(), e.end());
  s.clear();
  where.resize(a.size());
  for (size_t i = 0; i < e.size(); ++i) {
    int id = e[i].id;
    if (e[i].tp == +1) {
      set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev
          (nxt);
      if (nxt != s.end() && intersect(*nxt, a[id]))
```

```
        return make_pair(nxt->id, id);
      if (prv != s.end() && intersect(*prv, a[id]))
        return make_pair(prv->id, id);
      where[id] = s.insert(nxt, a[id]);
    } else {
      set<seg>::iterator nxt = next(where[id]), prv = prev(
          where[id]);
      if (nxt != s.end() && prv != s.end() && intersect(*nxt, *
          prv))
        return make_pair(prv->id, nxt->id);
      s.erase(where[id]);
    }
  }
  return make_pair(-1, -1);
}
```

## DelaunayTriangulation.h
**Description:** Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.
**Time:** $\mathcal{O}\left(n^2\right)$

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

## 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:** $\mathcal{O}\left(n \log n\right)$

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

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

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

```
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

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

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

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

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

## 8.5 3D

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

```
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.

```
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:** $\mathcal{O}\left(n^2\right)$

```
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) f1 ($\phi_1$) and f2 ($\phi_2$) from x axis and zenith angles (latitude) t1 ($\theta_1$) and t2 ($\theta_2$) 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*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines
```
double sphericalDistance(double f1, double t1,
        double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
    double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
    double dz = cos(t2) - cos(t1);
    double d = sqrt(dx*dx + dy*dy + dz*dz);
    return radius*2*asin(d/2);
}
```

# Strings (9)

### Trie.h
**Description:** Trie

60bd88, 59 lines
```
struct Trie {
    const int A = 26;
    int N;
    vector<vector<int>> next;
    vector<int> cnt;
    Trie() : N(0) { node(); }
    int node() {
        next.emplace_back(A, -1);
        cnt.emplace_back(0);
        return N++;
    }
    inline int get(char c) { return c - 'a'; }
    inline void insert(string s) {
        int cur = 0;
        for (char c : s) {
```
```
            int to = get(c);
            if (next[cur][to] == -1) next[cur][to] = node();
            cur = next[cur][to];
        }
        cnt[cur]++;
    }
    inline bool find(string s) {
        int cur = 0;
        for (char c : s) {
            int to = get(c);
            if (next[cur][to] == -1) return false;
            cur = next[cur][to];
        }
        return (cnt[cur] != 0);
    }
    // Doesn't check for existence
    inline void erase(string s) {
        int cur = 0;
        for (char c : s) {
            int to = get(c);
            cur = next[cur][to];
        }
        cnt[cur]--;
    }
    vector<string> dfs() {
        stack<pair<int, int>> st;
        string s;
        vector<string> ret;
        for (st.push({0, -1}), s.push_back('$'); !st.empty();) {
            auto [u, c] = st.top();
            st.pop(), s.pop_back(), c++;
            if (c < A) {
                st.push({u, c}), s.push_back(c + 'a');
                int v = next[u][c];
                if (~v) {
                    if (cnt[v]) ret.emplace_back(s);
                    st.push({v, -1});
                    s.push_back('$');
                }
            }
        }
        return ret;
    }
};
```

### KMP.h
**Description:** 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:** $\mathcal{O}(n)$

d4375c, 15 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:** z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
**Time:** $\mathcal{O}(n)$

ee09e2, 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 - l]);
        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;
}
```

### Manacher2.h
**Description:** Mancher

5c1b24, 34 lines
```
struct manacher {
    vector<int> p;
    void run_mancher(string s) {
        int n = s.size();
        p.assign(n, 1);
        int l = 1, r = 1;
        for (int i = 1; i < n; i++) {
            p[i] = max(0, min(r - i, p[l + r - i]));
            while (i + p[i] < n and i - p[i] >= 0 and s[i + p[i]] ==
                    s[i - p[i]])
                p[i]++;
            if (i + p[i] > r) {
                l = i - p[i], r = i + p[i];
            }
        }
    }
    void build(string s) {
        string t;
        t.push_back('#');
        for (auto c : s) {
            t.push_back(c);
            t.push_back('#');
        }
        run_mancher(t);
    }
    int getLongest(int c, int odd) {
        int pos = 2 * c + 1 + (!odd);
        return p[pos] - 1;
    }
    bool checkPalindrome(int l, int r) {
        int len = r - l + 1;
        int mxlen = getLongest((l + r) / 2, l % 2 == r % 2);
        return len <= mxlen;
    }
};
```

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

d07a42, 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:** Suffix Array

95d539, 44 lines

```cpp
void count_sort(vector<pli> &b, int bits) {
  int mask = (1 << bits) - 1;
  rep(it, 0, 2) {
    int shift = it * bits;
    vi q(1 << bits), w(sz(q) + 1);
    rep(i, 0, sz(b)) q[(b[i].first >> shift) & mask]++;
    partial_sum(q.begin(), q.end(), w.begin() + 1);
    vector<pli> res(sz(b));
    rep(i, 0, sz(b)) res[w[(b[i].first >> shift) & mask]++] = b
        [i];
    swap(b, res);
  }
}
struct SuffixArray {
  vi a; string s;
  SuffixArray(const string &str) : s(str + '\0') {
    int N = sz(s), q = 8;
    while ((1 << q) < N) q++;
    vector<pli> b(N);
    a.resize(N);
    rep(i, 0, N) b[i] = {s[i], i};
    for (int moc = 0;; moc++) {
      count_sort(b, q);
      rep(i, 0, N) a[b[i].second] = (i && b[i].first == b[i -
          1].first) ? a[b[i - 1].second] : i;
      if ((1 << moc) >= N) break;
      rep(i, 0, N) {
        b[i] = {(ll)a[i] << q, i + (1 << moc) < N ? a[i + (1 <<
            moc)] : 0};
        b[i].second = i;
      }
    }
    rep(i, 0, N) a[i] = b[i].second;
  }
  vi lcp() {
    int n = sz(a), h = 0;
    vi inv(n), res(n);
    rep(i, 0, n) inv[a[i]] = i;
    rep(i, 0, n) if (inv[i]) {
      int p0 = a[inv[i] - 1];
      while (s[i + h] == s[p0 + h]) h++;
      res[inv[i]] = h;
      if (h) h--;
    }
    return res;
  }
};
```

## SuffixAuto.h
**Description:** Suffix Automaton
**Time:** $\mathcal{O}(N)$

925841, 46 lines

```cpp
char s[N], p[N];
map <char, int> to[N << 1];
int len[N << 1], link[N << 1], sz, last;

inline void init() {
  len[0] = 0, link[0] = -1, sz = 1, last = 0, to[0].clear();
}

void feed (char c) {
  int cur = sz++, p = last;
  len[cur] = len[last] + 1, link[cur] = 0, to[cur].clear();
  while (~p and !to[p].count(c)) to[p][c] = cur, p = link[p];
  if (~p) {
    int q = to[p][c];
    if (len[q] - len[p] - 1) {
```

```cpp
    int r = sz++;
    len[r] = len[p] + 1, to[r] = to[q], link[r] = link[q];
    while (~p and to[p][c] == q) to[p][c] = r, p = link[p];
    link[q] = link[cur] = r;
  } else link[cur] = q;
  } last = cur;
}

bool run() {
  int m = strlen(p);
  for (int i = 0, u = 0; i < m; ++i) {
    if (!to[u].count(p[i])) return 0;
    u = to[u][p[i]];
  } return 1;
}

int main() {
  int t; cin >> t;
  while (t--) {
    scanf("%s", s);
    int n = strlen(s);
    init();
    for (int i = 0; i < n; ++i) feed(s[i]);
    int q; scanf("%d", &q);
    while (q--) {
      scanf("%s", p);
      puts(run() ? "y" : "n");
    }
  }
  return 0;
}
```

## Hashing.h
**Description:** Self-explanatory methods for string hashing.( Arithmetic mod $2^{64} - 1$. 2x slower than mod $2^{64}$ and more code, but works on evil test data (e.g. Thue-Morse, where ABBA... and BAAB... of length $2^{10}$ hash the same mod $2^{64}$). "typedef ull H;" instead if you think test data is random, or work mod $10^9 + 7$ if the Birthday paradox is not a problem.)

2d2a67, 36 lines

```cpp
typedef uint64_t ull;
struct H {
  ull x; H(ull x=0) : x(x) {}
  H operator+(H o) { return x + o.x + (x + o.x < x); }
  H operator-(H o) { return *this + ~o.x; }
  H operator*(H o) { auto m = (__uint128_t)x * o.x;
    return H((ull)m) + (ull)(m >> 64); }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)
struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(sz(str)) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
```

```cpp
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}
H hashString(string& s){H h{}; for(char c:s) h=h*C+c; return h;}
```

## AhoCorasick.h
**Description:** Aho Corasick

6733c5, 56 lines

```cpp
struct AC {
  int N, P;
  const int A = 26;
  vector <vector <int>> next;
  vector <int> link, out_link;
  vector <vector <int>> out;
  AC(): N(0), P(0) {node();}
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  inline int get (char c) {
    return c - 'a';
  }
  int add_pattern (const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    }
    out[u].push_back(P);
    return P++;
  }
  void compute() {
    queue <int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front(); q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) next[u][c] = next[link[u]][c];
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] = out[link[v]].empty() ? out_link[link[v
              ]] : link[v];
          q.push(v);
        }
      }
    }
  }
  int advance (int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)];
    return u;
  }
  void match (const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v]) {
        for (auto p : out[v]) cout << "match " << p << endl;
      }
    }
  }
};
```

# Various (10)

## 10.1 Misc. algorithms

### LIS.h
**Description:** Compute indices for the longest increasing subsequence.
**Time:** $\mathcal{O}(N \log N)$
2932a0, 17 lines

```cpp
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;
}
```

## 10.2 Dynamic programming

### DivideAndConquerDP.h
**Description:** Given $a[i] = \min_{lo(i) \le k < hi(i)} (f(i,k))$ where the (minimal) optimal $k$ increases with $i$, computes $a[i]$ for $i = L..R - 1$.
**Time:** $\mathcal{O}((N + (hi - lo)) \log N)$
d38d2b, 18 lines

```cpp
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+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

## 10.3 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

### 10.3.1 Bit hacks

- `x & -x` is the least bit in x.

- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of m (except m itself).

- `c = x&-x, r = x+c; (((r^x) >> 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))`

```cpp
  if (i & 1 << b) D[i] += D[i^(1 << b)];
```
computes all sums of subsets.

### 10.3.2 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 $a$ $(\text{mod } b)$ in the range $[0, 2b)$.
751a02, 8 lines

```cpp
typedef unsigned long long ull;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(-1ULL / b) {}
  ull reduce(ull a) { // a % b + (0 or b)
    return a - (ull)((__uint128_t(m) * a) >> 64) * b;
  }
};
```

## 10.4 Miscellaneous

### SOSDP.h
**Description:** SOS DP
60df40, 10 lines

```cpp
vector<vector<int>> dp(1 << n, vector<int>(n));
vector<int> sos(1 << n);
for (int mask = 0; mask < (1 << n); mask++) {
  dp[mask][-1] = a[mask];
  for (int x = 0; x < n; x++) {
    dp[mask][x] = dp[mask][x - 1];
    if (mask & (1 << x)) { dp[mask][x] += dp[mask - (1 << x)][x
        - 1]; }
  }
  sos[mask] = dp[mask][n - 1];
}
```

### submaskiterate.h
**Description:** Submask iterate
d4afb8, 3 lines

```cpp
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
  ... s and m ...
```

### nCrNotP.h
**Description:** Finds $nCr$ modulo a number that is not necessarily prime. Its good when $m$ is small and not fixed.
**Time:** $\mathcal{O}(m \log m)$
"../number-theory/CRT.h", "../number-theory/ModPow.h"
5815f4, 32 lines

```cpp
int F[1000002] = {1}, p, e, pe;
ll lg(ll n, int p) {
  ll r = 0;
  while (n) n /= p, r += n;
  return r;
}
ll f(ll n) {
  if (!n) return 1;
  return modpow(F[pe], n / pe, pe) * (F[n % pe] * f(n / p) % pe
      ) % pe;
}
```

```cpp
ll ncr(ll n, ll r) {
  ll c;
  if ((c = lg(n, p) - lg(r, p) - lg(n - r, p)) >= e)
    return 0;
  for (int i = 1; i <= pe; i++)
    F[i] = F[i - 1] * (i % p == 0 ? 1 : i) % pe;
  return (f(n) * modpow(p, c, pe) % pe) *
    modpow(f(r) * f(n - r), pe - (pe / p) - 1, pe) % pe;
}
ll ncr(ll n, ll r, ll m) {
  ll a0 = 0, m0 = 1;
  for (p = 2; m != 1; p++) {
    e = 0, pe = 1;
    while (m % p == 0)
      m /= p, e++, pe *= p;
    if (e) {
      a0 = crt(a0, m0, ncr(n, r), pe);
      m0 = m0 * pe;
    }
  }
  return a0;
}
```

# Fahim Vai's Notes (11)

## 11.1 Coloring

- The number of labeled undirected graphs with $n$ vertices, $G_n = 2^{\binom{n}{2}}$

- The number of labeled directed graphs with $n$ vertices, $G_n = 2^{n(n-1)}$

- The number of connected labeled undirected graphs with $n$ vertices, $C_n = 2^{\binom{n}{2}} - \frac{1}{n} \sum_{k=1}^{n-1} k \binom{n}{k} 2^{\binom{n-k}{2}} C_k = 2^{\binom{n}{2}} - \sum_{k=1}^{n-1} \binom{n-1}{k-1} 2^{\binom{n-k}{2}} C_k$

- The number of k-connected labeled undirected graphs with $n$ vertices, $D[n][k] = \sum_{s=1}^{n} \binom{n-1}{s-1} C_s D[n-s][k-1]$

- Cayley's formula: the number of trees on $n$ labeled vertices = the number of spanning trees of a complete graph with $n$ labeled vertices = $n^{n-2}$

- Number of ways to color a graph using k color such that no two adjacent nodes have same color

-- Complete graph = $k(k-1)(k-2)...(k-n+1)$

-- Tree = $k(k-1)^{n-1}$

-- Cycle = $(k-1)^n + (-1)^n(k-1)$

- Number of trees with $n$ labeled nodes: $n^{n-2}$

## 11.2 Lucas Number

Number of edge cover of a cycle graph $C_n$ is $L_n$

$L(n) = L(n-1) + L(n-2); L(0) = 2, L(1) = 1$

## 11.3 Catalan Number

$$C_{n+1} = C_0C_n + C_1C_{n-1} + C_2C_{n-2} + ... + C_nC_0$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

## 11.4 Derangement

$$D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$$

$$D_0 = 1, D_1 = 0$$

$1, 0, 1, 2, 9, 44, 265, ...$

## 11.5 Ballot Theorem

Suppose that in an election, candidate A receives a votes and candidate B receives b votes, where $a \geq kb$ for some positive integer k. Compute the number of ways the ballots can be ordered so that A maintains more than k times as many votes as B throughout the counting of the ballots.

The solution to the ballot problem is
$((a-kb)/(a+b)) * C(a+b, a)$

## 11.6 Classical Problem

$F(n, k)$ = number of ways to color n objects using exactly $k$ colors

Let $G(n, k)$ be the number of ways to color n objects using no more than $k$ colors.

Then, $F(n, k) = G(n, k) - C(k, 1) * G(n, k-1) + C(k, 2) * G(n, k-2) - C(k, 3) * G(n, k-3)...$

Determining G(n, k) :

Suppose, we are given a 1 * n grid. Any two adjacent cells can not have same color. Then, $G(n, k) = k * ((k-1)^(n-1))$

If no such condition on adjacent cells. Then, $G(n, k) = k^n$

## 11.7 Generating Function

$1/(1-x) = 1 + x + x^2 + x^3 + ...$
$1/(1-ax) = 1 + ax + (ax)^2 + (ax)^3 + ...$
$1/(1-x)^2 = 1 + 2x + 3x^2 + 4x^3 + ...$
$1/(1-x)^3 = C(2,2) + C(3,2)x + C(4,2)x^2 + C(5,2)x^3 + ...$
$1/(1-ax)^(k+1) =$
$1 + C(1+k, k)(ax) + C(2+k, k)(ax)^2 + C(3+k, k)(ax)^3 + ...$
$x(x+1)(1-x)^-3 = 1 + x + 4x^2 + 9x^3 + 16x^4 + 25x^5 + ...$
$e^x = 1 + x + (x^2)/2! + (x^3)/3! + (x^4)/4! + ...$

## 11.8 SUM

$1^4 + 2^4 + 3^4 + ... + n^4 = \frac{n(n+1)(2n+1)(3n2+3n+-1)}{30}$

$S_{(n,p)} = 1^p + 2^p + 3^p + 4^p + ... + n^a p$

$S(n, p) = \frac{1}{p+1}[(n+1)^{p+1} - 1 - \sum_{i=0}^{p-1}\binom{p+1}{i}S(n,i)]$

$1.2 + 2.3 + 3.4 + ... = \frac{1}{3}n(n+1)(n+2)$

$\sum_{i=1}^{n}f_k(i) = \frac{1}{k+1}n(n+1)(n+2)...(n+k) = \frac{1}{k+1}\frac{(n+k)!}{(n-1)!}$

$\sum_{i=0}nix^i = 1 + 2x^2 + 3x^3 + 4x^4 + 5x^5 + ... + nx^n = \frac{(x-(n+1)x^{n+1}+nx^{n+2})}{(x-1)^2}$

## 11.9 Probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

## 11.10 Matching Formula

**Normal Graph**
MM + MEC = n (excluding vertex), IS + VC = G, MIS + MVC = G

**Bipartite Graph**
MIS = n - MBM, MVC = MBM, MEC = n - MBM

## 11.11 Number Theory

- HCN: 1e6(240), 1e9(1344), 1e12(6720), 1e14(17280), 1e15(26880), 1e16(41472)
- $gcd(a, b, c, d, ...) = gcd(a, b-a, c-b, d-c, ...)$
- $gcd(a+k, b+k, c+k, d+k, ...) = gcd(a+k, b-a, c-b, d-c, ...)$
- Primitive root exists iff $n = 1, 2, 4, p^k, 2 \times p^k$, where $p$ is an odd prime.
- If primtive root exists, there are $\phi(\phi(n))$ primtive roots of $n$.
- The numbers from 1 to $n$ have in total $O(n \log \log n)$ unique prime factors.
- $x \equiv r_1 \mod m1$ and $x \equiv r_2 \mod m2$ has a solution iff $\gcd(m_1, m_2)|(r_1 - r_2)$
Solution of $x^2 \equiv a(\mod p)$:
- $ca \equiv cb \pmod{m} \iff a \equiv b \pmod{\frac{n}{\gcd(n,c)}}$
- $ax \equiv b \pmod{m}$ has a solution $\iff \gcd(a, m)|b$
- If $ax \equiv b \pmod{m}$ has a solution, then it has $gcd(a, m)$ solutions and they are separated by $\frac{m}{\gcd(a,m)}$
- $ax \equiv 1 \pmod{m}$ has a solution or $a$ is invertible $\pmod{m}$ $\iff \gcd(a, m) = 1$
- $x^2 \equiv 1 \pmod{p}$ then $x \equiv \pm 1 \pmod{p}$
- There are $\frac{p-1}{2}$ has no solution.
- There are $\frac{p-1}{2}$ has exaclty two solutions.
- When $p\%4 = 3$, $x \equiv \pm a^{\frac{p+1}{4}}$
- When $p\%8 = 5$, $x \equiv a^{\frac{p+3}{8}}$ or $x \equiv 2^{\frac{p-1}{4}}a^{\frac{p+3}{8}}$

## 11.12 Totient

- If $p$ is a prime $\phi(p^k) = p^k - p^{k-1}$
- If $a$ & $b$ are relatively prime, $\phi(ab) = \phi(a)\phi(b)$
- $\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})(1 - \frac{1}{p_3})...(1 - \frac{1}{p_k})$
- Sum of coprime to $n = n * \frac{\phi(n)}{2}$
- If $n = 2^k, \phi(n) = 2^{k-1} = \frac{n}{2}$
- For $a$ & $b$, $\phi(ab) = \phi(a)\phi(b)\frac{d}{\phi(d)}$
- $\phi(ip) = p\phi(i)$ whenever $p$ is a prime and it divides $i$
- The number of $a(1 <= a <= N)$ such that $gcd(a, N) = d$ is $\phi(\frac{n}{d})$
- If $n > 2$, $\phi(n)$ is always even
- Sum of gcd, $\sum_{i=1}^{n}gcd(i, n) = \sum_{d|n}d\phi(\frac{n}{d})$
- Sum of lcm, $\sum_{i=1}nlcm(i, n) = \frac{n}{2}(\sum_{d|n}(d\phi(d)) + 1)$
- $\phi(1) = 1$ and $\phi(2) = 1$ which two are only odd $\phi$
- $\phi(3) = 2$ and $\phi(4) = 2$ and $\phi(6) = 2$ which three are only prime $\phi$
- Find minimum n such that $\frac{\phi(n)}{n}$ is maximum- Multiple of small primes- $2 * 3 * 5 * 7 * 11 * 13 * ...$

**Mobius**

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j) = 1] = \sum_{k=1}^{n}\mu(k)\lfloor\frac{n}{k}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j) = \sum_{k=1}^{n}k\sum_{l=1}^{\lfloor\frac{n}{k}\rfloor}\mu(l)\lfloor\frac{n}{kl}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j) = \sum_{k=1}^{n}(\frac{\lfloor\frac{n}{k}\rfloor(1+\lfloor\frac{n}{k}\rfloor)}{2})^2\sum_{d|k}\mu(d)kd$$

**Tree Hashing**

$f(u) = sz[u] * \sum_{i=0}f(v) * p^i$ ;$f(v)$ are sorted $f(child) = 1$

**Permutation**
- To maximize the sum of adjacent differences of a permuation, it is necessary and sufficient to place the smallest half numbers in odd position and the greatest half numbers in even position. Or, vice versa.

**String**
- If the sum of length of some strings is $N$, there can be at most $\sqrt{N}$ distinct length.
- A Text can have at most $O(N \times \sqrt{N})$ distinct substrings that match with given patterns where the sum of the length of the given patterns is $N$.
- Period = n % (n - pi.back() == 0)? n - pi.back(): n
- The first $(period)$ cyclic rotations of a string are distinct. Further cyclic rotations repeat the previous strings.

- $S$ is a palindrome if and only if it's period is a palindrome.

**Bit**

- `(a xor b)` and `(a + b)` has the same parity
- `(a + b)` = `(a xor b)` + 2 (a & b)
- $gcd(a, b) \leq a - b \leq xor(a, b)$

**Convolution**

- Hamming Distance: Replace 0 with $-1$ - SQRT Decomposition: Find block size, B = sqrt(8 * n)