Université de Montpellier

UMontpellier-B

Denil Bouafia, Julia Pham Ba Nien, Lysandre Terrisse

2024-11-29

1 **Contest** 1

2 **Mathematics** 1

3 **Data structures** 2

4 **Numerical** 4

5 **Number theory** 7

6 **Combinatorial** 8

7 **Dynamic programming** 10

8 **Graph** 11

9 **Trees** 16

10 **Geometry** 17

11 **Strings** 22

12 **Various** 24

Contest (1)

template.cpp — 25 lines

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
using namespace std;
#define INF 0x3f3f3f3f
#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;
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
     count());

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
    vi cord;
    cord.erase(unique(cord.begin(),cord.end()),cord.end());  //
         supprime doublons
    ll k = rng(); // random 64 bits integer
    random_device dev;
    mt19937 rng(dev());
    uniform_int_distribution<mt19937::result_type> dist6(1,6);
        // distribution in range [1, 6]
    cout << fixed << setprecision(6) << dist6(rng) << endl;
}
```

.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 = <>
```

hash.sh — 3 lines

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

hashpy.sh — 2 lines

```
# Usage : cat fichier.py | hashpy.sh
cat - | grep -vE '^\s*(#|$)' | tr -d '[:space:]' | md5sum | cut
     -c-6
```

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 Vieta's Formulas

For a polynomial of degree n:

$$\frac{-b}{a} = \text{sum of roots}, \quad (-1)^n \cdot \frac{k}{a} = \text{product of roots}.$$

Here: - k is the coefficient of the constant term, - a is the leading coefficient, - b is the coefficient of the term of degree $n-1$.

2.3 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. $a_n = (d_1 n + d_2) r^n$.

2.4 Trigonometry

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

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

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

$$\begin{aligned} a\cos x + b \sin x &= r\cos(x - \phi) \\ a \sin x + b \cos x &= r \sin(x + \phi) \end{aligned}$$

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

2.5 Geometry

2.5.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

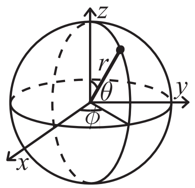
2.5.2 Quadrilaterals

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

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

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

2.5.3 Spherical coordinates



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

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 Additional Concepts

2.7.1 Divisors and Sum of Divisors

Number of Divisors: If $N = a^p \times b^q \times \cdots \times c^r$, the total number of divisors is:

$$(p+1)\times(q+1)\times\cdots\times(r+1).$$

Sum of Divisors: For $N = a^p \times b^q \times \cdots \times c^r$, the sum of divisors is:

$$\mathrm{SumDiv}(N) = \frac{a^{p+1}-1}{a-1}\cdot\frac{b^{q+1}-1}{b-1}\cdots\frac{c^{r+1}-1}{c-1}.$$

2.7.2 Sum of an Infinite Geometric Series

The sum of an infinite geometric series of the form:

$$1 + r + r^2 + r^3 + \ldots = \frac{1}{1-r}, \quad \text{for } |r| < 1.$$

2.7.3 Sum of First n Terms of an AP

The sum of the first n terms (S_n) is:
$S_n = \frac{n}{2}\cdot(2a + (n-1)d)$

Alternatively, if the last term (l) is known: $S_n = \frac{n}{2}\cdot(a+l)$ Here: - S_n = sum of the first n terms, - a = first term, - d = common difference, - n = number of terms, - l = last term.

2.8 Combinatorics

Combinations

Without Repetition: C_n^k With Repetition: $K_n^k = \frac{(n+k-1)!}{k!\cdot(n-1)!}$

Arrangements

Without Repetition: $A_n^k = \frac{n!}{(n-k)!}$ With Repetition: n^k

Distributing Identical Objects into Boxes

With empty boxes allowed: C_{n+k-1}^k With no empty box: C_{n-1}^{k-1}

2.8.1 Coefficients binomiaux

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k=0 \text{ ou } k=n, \\ \binom{n-1}{k-1}+\binom{n-1}{k} & \text{si } 1 \le k \le n-1. \end{cases}$$

Somme sur n: $\sum_{n=p}^{q}\binom{n}{k} = \binom{q+1}{k+1} - \binom{p}{k+1}$

2.8.2 Principe d'inclusion-exclusion

$|\bigcup_{i\in I} A_i| = \sum_{\substack{J\subseteq I\\ J\neq\emptyset}}(-1)^{|J|-1}|\bigcap_{j\in J} A_j|$

2.9 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) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ 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 σ 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: Expectation is linear:
$\mathbb{E}(aX+bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$
For independent X and Y,
$V(aX+bY) = a^2V(X) + b^2V(Y)$

2.9.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

2.9.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$$

Data structures (3)

UnionFind.py
Description: Union Find
Time: $\mathcal{O}(\alpha(N))$

90eacc, 16 lines

```
class DSU:
    def __init__(self, n):
        self.up = list(range(n))
        self.size = [1] * n
    def find(self, x):
        if self.up[x] != x:
            self.up[x] = self.find(self.up[x])
        return self.up[x]
    def union(self, x, y):
        x, y = self.find(x), self.find(y)
        if x == y: return False
        if self.size[x] < self.size[y]:
            x, y = y, x
        self.up[y] = x
        self.size[x] += self.size[y]
        return True
```

SparseTable.py
Description: Sparse Table
Time: build $\mathcal{O}(N\log N)$, query $\mathcal{O}(1)$.

73d2ae, 16 lines

```
class SparseTable:
    def __init__(self, arr, op=min):
        self.op = op
        self.n = len(arr)
        self.h = self.n.bit_length() - 1
        self.table = [[0] * self.n for _ in range(self.h + 1)]
        self.table[0] = [a for a in arr]
        for k in range(self.h):
            nxt, prv = self.table[k + 1], self.table[k]
            l = 1 << k
            for i in range(self.n - l * 2 + 1):
                nxt[i] = op(prv[i], prv[i + l])
    def prod(self, l, r): # [l, r)
        assert 0 <= l < r <= self.n
        k = (r - l).bit_length() - 1
        return self.op(self.table[k][l], self.table[k][r - (1
            << k)])
```

SegmentTree.py
Description: SegmentTree
Time: $\mathcal{O}(N\log N)$

8903d0, 43 lines

```
def update(p, val):
    p += N
    seg[p] = val
    while p > 1:
```

```
        seg[p>>1] = seg[p] + seg[p^1]
        p >>= 1
def query(l, r): # [l, r)
    res = 0
    l += N; r += N
    while l < r:
        if l&1: res += seg[l]; l += 1
        if r&1: r -= 1; res += seg[r]
        l >>= 1; r >>= 1
    return res
N = 1 << n.bit_length()
seg = [0] * (2 * N)
for i in range(n):
    seg[N + i] = a[i]
for k in range(N - 1, 0, -1):
    seg[k] = seg[k<<1] + seg[k<<1|1]

# supprimer ligne -> for k in range(N - 1, 0, -1):
def update(l, r, val): # [l, r)
    l += N; r += N
    while l < r:
        if l&1: seg[l] += val; l += 1
        if r&1: r -= 1; seg[r] += val
        l >>= 1; r >>= 1
def query(p):
    res = 0
    p += N
    while p > 0:
        res += seg[p]
        p >>= 1
    return res

# reduce the complexity from O(N log N) to O(N) to get all
    values.
# works only in case the order of modifications on a single
    element doesn't affect the result.
def push():
    for i in range(1, N):
        seg[i<<1] += seg[i]
        seg[i<<1|1] += seg[i]
        seg[i] = 0
```

Fenwick.py

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

d7d46f, 62 lines

```
def update(pos, v):
    while pos < len(T):
        T[pos] += v
        pos |= pos + 1
def _query(pos):
    r = 0
    while pos:
        r += T[pos-1]
        pos &= pos-1
    return r
def query(a, b): # [l, r)
    return _query(b) - _query(a)
T = list(X)
for i in range(n):
    j = i | (i+1)
    if j < n: T[j] += T[i]

def update_(i, val):
    i = i + 1
    while i < len(T):
        T[i] += val
        i += i & (-i)
def update(l, r, val): # [l, r)
    update_(l, val)
    update_(r, -val)  # T length n+1
def query(i):
    res = 0
    i = i + 1
    while i > 0:
        res += T[i]
        i -= i & (-i)
    return res

# 1-indexed
def add(b, idx, v):
    while idx <= n:
        b[idx] += v
        idx += idx & -idx
def range_add(l, r, v):
    j = l
    while j <= n:
        B1[j] += v
        B2[j] += v * (l - 1)
        j += j & -j
    j = r + 1
    while j <= n:
        B1[j] -= v
        B2[j] -= v * r
        j += j & -j
def _query(b, idx):
    total = 0
    while idx > 0:
        total += b[idx]
        idx -= idx & -idx
    return total
def prefix_sum(idx):
    return _query(B1, idx) * idx - _query(B2, idx)
def range_sum(l, r):
    return prefix_sum(r) - prefix_sum(l - 1)
B1, B2 = [0] * (n + 1), [0] * (n + 1)
for i, v in enumerate(X, 1):
    range_add(i, i, v)
```

Fenwick2D.h

Description: 0-indexed FenwickTree2D<int> fenw(r, c); fenw.Modify(i, j, +1); fenw.Query(ib+1, jb+1)-fenw.Query(ia, jb+1)-fenw.Query(ib+1, ja)+fenw.Query(ia, ja);
Time: $\mathcal{O}(\log^2 N)$.

a7497c, 33 lines

```
template <typename T>
class FenwickTree2D {
 public:
  vector<vector<T>> fenw;
  int n, m;
  FenwickTree2D() : n(0), m(0) {}
  FenwickTree2D(int n_, int m_) : n(n_), m(m_) {
    fenw.resize(n);
    for (int i = 0; i < n; i++) {
      fenw[i].resize(m);
    }
  }
  void Modify(int i, int j, T v) {
    assert(0 <= i && i < n && 0 <= j && j < m);
    int x = i;
    while (x < n) {
      int y = j;
      while (y < m) {fenw[x][y] += v; y |= y + 1;}
      x |= x + 1;
    }
  }
  T Query(int i, int j) {
    assert(0 <= i && i <= n && 0 <= j && j <= m);
    T v{};
    int x = i;
    while (x > 0) {
      int y = j;
      while (y > 0) {v += fenw[x - 1][y - 1]; y &= y - 1;}
      x &= x - 1;
    }
    return v;
  }
};
```

PersistentSegmentTree.h

Description: Each modification of the Segment Tree we will receive a new root vertex. To quickly jump between two different versions of the Segment Tree, store this roots in an array. To use a specific version of the Segment Tree we simply call the query using the appropriate root vertex.
Time: $\mathcal{O}(\log N)$.

c4782a, 29 lines

```
struct Vertex {
    Vertex *l, *r;
    int sum;
    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(int a[], int tl, int tr) {
    if (tl == tr) return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}
int get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r) return 0;
    if (l == tl && tr == r) return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}
Vertex* update(Vertex* v, int tl, int tr, int pos, int new_val)
    {
    if (tl == tr) return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val), v
            ->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos,
            new_val));
}
```

ImplicitTreap.h

Description: Implicit Treap 0-indexed. Updates and queries [l, r). Example of use : IT treap; treap.insert(position, value); treap.query<int>(l, r+1, f()); treap.upd(l, r+1, f()); treap.get_vector(); for(auto x : treap.result_vector)
Time: $\mathcal{O}(\log N)$

d6e901, 73 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
typedef struct item * pitem;
struct item {
    int prior,val,sum=0,add=0,size=1,rev=0;
    pitem l=nullptr, r=nullptr;
    item(int val) : val(val), prior(rng()) {}};
int size(pitem p){return p?p->size:0;}
int sum(pitem p){return p?p->sum:0;}
void push(pitem t){
  if(!t){return;}
  if(t->add){
```

```
    t->val+=t->add; t->sum+=t->size*t->add;
    if(t->l){t->l->add+=t->add;}
    if(t->r){t->r->add+=t->add;}
    t->add=0;}
  if(t->rev){
        t->rev=0; swap(t->l, t->r);
        if(t->l)t->l->rev^=1;
        if(t->r)t->r->rev^=1;}}
void pull(pitem t){
    if(!t){return;}
    push(t->l),push(t->r);
    t->size=size(t->l)+size(t->r)+1;
    t->sum=t->val+sum(t->l)+sum(t->r);}
void merge(pitem &t,pitem l,pitem r){
    push(l),push(r);
  if (!l||!r) {t=l?l:r;} else
    if(l->prior>r->prior){merge(l->r,l->r,r),t=l;}
    else{merge(r->l,l,r->l),t=r;}
  pull(t);}
void split(pitem t,pitem &l,pitem &r,int val){
  if(!t)return void(l=r=nullptr);
    push(t);
  if(val>size(t->l)){
    split(t->r,t->r,r,val-size(t->l)-1), l = t;}
    else{split(t->l,l,t->l,val),r=t;}
  pull(t);}
function<void(pitem)> range_add(int v){
  return [v](pitem t) {t->add += v;};}
function<void(pitem)> reverse(){
  return [](pitem t) { t->rev ^= 1;};}
function<int(pitem)> range_sum() {
  // int return value
  return [](pitem t) {return t->sum;};}
struct IT {
  pitem root = nullptr;
    vector<int> result_vector;
  void insert(int i, int x) {
        pitem l, r; split(root, l, r, i);
        merge(l, l, new item(x)); merge(root, l, r);}
  void del(int i) {
    pitem l, r;
    split(root, l, r, i); split(r, root, r, 1);
    merge(root, l, r);}
  void upd(int l, int r, function<void(pitem)> f) { // [l, r)
    pitem a, b, c;  // a: [0, l); b: [l, r); c: [r, )
    split(root, a, b, l); split(b, b, c, r - l);
    if (b) { f(b); }
    merge(root, a, b); merge(root, root, c);}
  template<typename R>R query(int l,int r,function<R(pitem)>f){
    pitem a, b, c;  // a: [0, l); b: [l, r); c: [r, )
    split(root, a, b, l); split(b, b, c, r - l);
    assert(b); R x = f(b);
    merge(root, a, b); merge(root, root, c);
    return x;}
    void each(pitem t, const function<void(pitem)> &f) {
        if(!t) return;
        push(t); each(t->l, f); f(t); each(t->r, f);}
    void get_vector() {
        each(root,[this](pitem x){result_vector.push_back(x->
             val);});}
    void output() {
        each(root, [](pitem x){printf("(%d, %d) ", x->val, x->
             sum);});}
};
```

LineContainer.h

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

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

8ec1c7, 29 lines

```
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;
  }
};
```

MinDeque.h

Description: Deque with push_front, push_back, getmin in constant time.

Time: $\mathcal{O}(1)$.

936c82, 41 lines

```
struct minstack {
  stack<pair<int, int>> st;
  int getmin() {return st.top().second;}
  bool empty() {return st.empty();}
  int size() {return st.size();}
  void push(int x) {
    int mn = x;
    if (!empty()) mn = min(mn, getmin());
    st.push({x, mn});
  }
  void pop() {st.pop();}
  int top() {return st.top().first;}
  void swap(minstack &x) {st.swap(x.st);}
};

struct mindeque {
  minstack l, r, t;
  void rebalance() {
    bool f = false;
    if (r.empty()) {f = true; l.swap(r);}
    int sz = r.size() / 2;
    while (sz--) {t.push(r.top()); r.pop();}
    while (!r.empty()) {l.push(r.top()); r.pop();}
    while (!t.empty()) {r.push(t.top()); t.pop();}
    if (f) l.swap(r);
  }
  int getmin() {
    if (l.empty()) return r.getmin();
    if (r.empty()) return l.getmin();
    return min(l.getmin(), r.getmin());
  }
  bool empty() {return l.empty() && r.empty();}
  int size() {return l.size() + r.size();}
  void push_front(int x) {l.push(x);}
  void push_back(int x) {r.push(x);}
  void pop_front() {if (l.empty()) rebalance(); l.pop();}
  void pop_back() {if (r.empty()) rebalance(); r.pop();}
  int front() {if (l.empty()) rebalance(); return l.top();}
  int back() {if (r.empty()) rebalance(); return r.top();}
  void swap(mindeque &x) {l.swap(x.l); r.swap(x.r);}
};
```

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

de4ad0, 21 lines

```
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;
  }
};
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.

Usage: `Node* tr = new Node(v, 0, sz(v));`

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

"../various/BumpAllocator.h" 34ecf5, 50 lines

```
const int inf = 1e9;
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
```

```
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void push() {
    if (!l) {
      int mid = lo + (hi - lo)/2;
      l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
      l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

Numerical (4)

4.1 Polynomials and recurrences

PolyRoots.h

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

e897c3, 40 lines

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

PolyInterpolate.h

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

08bf48, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: `berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}`
Time: $\mathcal{O}\left(N^2\right)$

0323c8, 27 lines

```
const ll mod = 1000000007; // faster if const

ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}

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

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

f4e444, 22 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };
  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}\left(\log((b-a)/\epsilon)\right)$

31d45b, 14 lines

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

4.3 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}\left(N^3\right)$

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
```

```
  return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$ 3313dc, 18 lines

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

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2m)$ 44c9ab, 36 lines

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

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:
08e495, 7 lines

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}(n^2m)$ fa2d7a, 33 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$ ebfff6, 32 lines

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

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$ 8f9fa8, 26 lines

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

```
  return b;
}
```

4.4 Misc.

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

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

FFT.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

f982eb, 63 lines

```
using C=complex<double>; const ll mod=998244353;
void fft(vector<C>&a){
  int n=sz(a),L=31-__builtin_clz(n);
  static vector<complex<long double>>R(2,1);
  static vector<C> rt(2,1);
  for(static int k=2;k<n;k*=2){
    R.resize(n);rt.resize(n);
    auto x=polar(1.0L,acos(-1.0L)/k);
    for(int i=k;i<2*k;i++)
      rt[i]=R[i]=i&1?R[i/2]*x:R[i/2];
  }
  vector<int> rev(n);
  for(int i=0;i<n;i++)
    rev[i]=(rev[i/2]|(i&1)<<L)/2;
  for(int i=0;i<n;i++)
    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)
      for(int j=0;j<k;j++){
        auto x=(double*)&rt[j+k],y=(double*)&a[i+j+k];
        C z(x[0]*y[0]-x[1]*y[1],x[0]*y[1]+x[1]*y[0]);
        a[i+j+k]=a[i+j]-z;
        a[i+j]+=z;}}}
template<ll M>
vector<ll> convMod(const vector<ll>&a, const vector<ll>&b){
  if(a.empty() || b.empty()) return {};
  vector<ll> 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);
  for(int i=0;i<sz(a);i++)
    L[i]=C((int)a[i]/cut,(int)a[i]%cut);
  for(int i=0;i<sz(b);i++)
    R[i]=C((int)b[i]/cut,(int)b[i]%cut);
  fft(L),fft(R);
  for(int i=0;i<n;i++){
    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);
  for(int i=0;i<sz(res);i++){
    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;}
vlli shift(vector<ll> &a, ll v){
  ll n=sz(a)-1;
  vlli f(n+1), g(n+1), i_fact(n+1);
  f[0]=a[0];
  g[n]=1;
  i_fact[0]=1;
  ll fact=1,potk=1;
  for(int i=1;i<n+1;i++){
    fact=fact*i%mod;
    f[i]=fact*a[i]%mod;
    potk=(potk*v%mod+mod)%mod;
    g[n-i]=((potk*inv(fact))%mod+mod)%mod;
    i_fact[i]=inv(fact);}
  auto p = convMod<mod>(f,g);
  vlli res(n+1);
  for(int i=0;i<n+1;i++)
    res[i]=(p[i+n]*i_fact[i]%mod+mod)%mod;
  return res;}
```

LinearBaseXor.py

Description: Linearbase : prend une liste de nombres a et renvoie une base linéaire (ensemble minimal d'el indépendants en termes de bits). Crée une représentation réduite où chaque nombre est une combinaison linéaire unique des nombres dans cette base. Baseinsert : Prend une base b et un nombre x, et essaie insertion (sans redondance = inséré = renvoie True).
Time: X

0bd6da, 21 lines

```
mx_bit=20
def linearbase(a):
    res=[0]*mx_bit
    for x in a:
        for i in range(mx_bit-1,-1,-1):
            if x>>i&1:
                if res[i]:
                    x^=res[i]
                else:
                    res[i]=x
                    break
    return res
def baseinsert(b,x):
    for i in range(mx_bit-1,-1,-1):
        if x>>i&1:
            if b[i]:
                x^=b[i]
            else:
                b[i]=x
                return True
    return False
```

Number theory (5)

5.1 Modular arithmetic

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;
}
```

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" 724c2e, 19 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;
  }
    bool operator==(Mod b) const { return x == b.x; }
};
```

5.2 Divisibility

CRT.py

Description: a, n = values, modulos For non-coprime moduli : crtNonCoprime

3e205a, 27 lines

```
def crt(a, n):
    x = 0
    p = 1
    for ni in n: p *= ni
    for ai, ni in zip(a, n):
        xi = p // ni
        try: inv = pow(xi, -1, ni)
        except ValueError: return None
        x += ai * xi * inv
    return x % p

def crtNonCoprime(a, n):
    prime = {}
    for ai, ni in zip(a, n):
        for k, v in getPrimeFactors(ni).items():
            m = pow(k, v)
            aj, nj = prime.get(k, (ai % m, m))
            if aj != (ai % m) % nj:
                return None
            if nj > m:
                continue
            prime[k] = (ai % m, m)
    newa, newn = [], []
    for k, (ai, ni) in prime.items():
```

```
        newa.append(ai)
        newn.append(ni)
    return crt(newa, newn)
```

5.2.1 Bézout's identity

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

$$ax + by = d$$

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

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

Bezout.py

Description: bezout(a, b) calcule une solution à l'équation ax + by = pgcd(a, b) solve(a, b, n) calcule (x, y) tq ax + by = n

1086d6, 24 lines

```
def bezout(a, b):
    px, py = 1, 0
    x, y = 0, 1
    while b != 0:
        a, (q, b) = b, divmod(a, b)
        px, x = x, px - q * x
        py, y = y, py - q * y
# pgcd, x, y
    return a, px, py

def solve(c1, a, c2, b, n):
    g, x, y = bezout(a, b)
    if n%g: return -1
    x *= n//g; y *= n//g
    a //= g; b //= g
    lo = -(x // b)
    hi = y // a
    if lo > hi: return -1
# minimize c1 * x
    res1 = c1 * (x + b * lo) + c2 * (y - a * lo)
# minimize c2 * y
    res2 = c1 * (x + b * hi) + c2 * (y - a * hi)
    if res1 < res2: return x + b * lo, y - a * lo
    return x + b * hi, y - a * hi
```

5.3 Primality

Sieve.py

Description: Sieve of eratosthenes

235da5, 32 lines

```
maxn = 1000000
divisors = [[] for _ in range(maxn + 1)]
for i in range(1, maxn + 1):
    for j in range(i, maxn + 1, i):
        divisors[j].append(i)

spf = [-1] * (maxn + 2)
for i in range(2, maxn+1):
    if spf[i] == -1:
        for j in range(i+i, maxn+1, i):
            spf[j] = i

mobius = [0] * (maxn + 2)
mobius[1] = 1
for i in range(2, maxn + 1):
    if spf[i // spf[i]] == spf[i]:
        mobius[i] = 0
    else:
        mobius[i] = -1*mobius[i//spf[i]]

def getnpf(x):
    pfac = defaultdict(int)
    while spf[x] != -1:
        pfac[spf[x]] += 1
        x //= spf[x]
    if x != 1:
        pfac[x] += 1
    tot, pf = 1, 0
    for k, v in pfac.items():
        tot *= (v + 1)
        pf += 1
    return tot - pf
```

PrimeFactorsAndDivisors.py

Description: Prime factors of N

Time: $\mathcal{O}\left(\sqrt{N}\right)$.

ae0593, 28 lines

```
def getPrimeFactors(n):
    res = {}
    if n % 2 == 0:
        res[2] = 0
        while n % 2 == 0:
            res[2] += 1
            n //= 2
    for i in range(3,int(n**.5)+1,2):
        if n % i == 0:
            res[i] = 0
            while n % i == 0:
                res[i] += 1
                n //= i
    if n > 1: res[n] = 1
    return res

def getDivisors(n):
    primeFactors = getPrimeFactors(n)
    res, pw = [], 1
    for factor in primeFactors:
        pw *= primeFactors[factor] + 1
    for i in range(pw):
        divisor = 1
        for factor in primeFactors:
            divisor *= factor ** (i % (primeFactors[factor] +
                1))
            i //= primeFactors[factor] + 1
        res.append(divisor)
    return res
```

5.4 Misc

Totient.py

Description: Counts the positive integers up to n that are relatively prime to n

0922a5, 8 lines

```
def totient(n):
    res = 1
    for p, a in pfac(n).items():
        res *= pow(p, a - 1) * (p - 1)
    return res

def sum_coprimes(n):
    return (totient(n) * n) // 2
```

MultipleCount.py

Description: Number integers within the interval 1,2,. . . ,n that are divisible by at least one of the prime numbers.

1af7dd, 12 lines

```
nb = 0
for mask in range(1, 1<<len(primes)):
    num = 1
    for i in range(len(primes)):
        if mask >> i & 1:
            num *= primes[i]
        if num > n:
            break
    if mask.bit_count() % 2:
        nb += n // num
    else:
        nb -= n // num
```

5.5 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.

5.6 Primes

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

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

5.7 Estimates

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

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

5.8 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)

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: $\mathcal{O}(n)$

044568, 6 lines

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

6.1.2 Derangements

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

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

6.1.3 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

Partitions.py

Description: Génère partitions de n

7090ac, 16 lines

```
def generateur_partitions(n):
    def partition_suivante(a, k, N):
        b = a[:k] + [a[k]-1]
        q, r = divmod((N+1), b[k])
        b = b + q*[b[k]]
        if r!=0: b.append(r)
        while b[k]!=1:
            k+=1
            if k==len(b): break
        return (b, k-1, len(b) - k)
    p = [n]
    yield p
    k, N = (-1, 1) if n==1 else (0, 0)
    while k >= 0:
        p, k, N = partition_suivante(p, k, N)
        yield p
```

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

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 $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$

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;
}
```

6.3 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:

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

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.

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

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

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

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.

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$ 044568, 6 lines

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

6.4 Computation

6.4.1 Catalan Balanced Sequences

Function: `catalan_bal`
Counts bracket sequences of length n with balance ≥ 0.
Formula:

$$\text{catalan\_bal}(n,s,e) = \begin{cases} 0, & (n+s+e) \mod 2 \neq 0 \text{ or } s<0 \text{ or } e<0, \\ \binom{n}{\frac{n+e-s}{2}} - \binom{n}{\frac{n-e-s-2}{2}}, & \text{otherwise.} \end{cases}$$

6.4.2 Grid Path Calculations

Grid Path: `path`
Paths from $(0,0)$ to (x,y):

$$\text{path}(x,y) = \binom{x+y}{x}.$$

Avoiding Low Barrier: `path_low`
Avoiding $y = x+b$, $b<0$:

$$\text{path\_low}(x,y,b) = \begin{cases} 0, & b \geq 0, \\ \text{path}(x,y) - \text{path}(y-b, x+b), & b<0. \end{cases}$$

Avoiding Upper Barrier: `path_up`
Avoiding $y = x+b$, $b>0$:

$$\text{path\_up}(x,y,b) = \begin{cases} 0, & b \leq 0, \\ \text{path}(x,y) - \text{path}(y-b, x+b), & b>0. \end{cases}$$

Alternating Constraints: `calc_LUL`
Paths from $(0,0)$ to (x,y) alternating constraints $y = x+b_1$ and $y = x+b_2$:

$$\text{calc\_LUL}(x,y,b_1,b_2) = \begin{cases} 0, & x < b_1 \text{ or } y < -b_1, \\ \text{path}(x-b_1, y+b_1) - \text{calc\_LUL}(y+b_1, x-b_1, -b_2, -b_1), & \text{otherwise.} \end{cases}$$

Two Barriers: `path_2`
Avoiding $y = x+b_1$ and $y = x+b_2$:

$$\text{path\_2}(x,y,b_1,b_2) = \text{path}(x,y) - \text{calc\_LUL}(x,y,b_1,b_2) - \text{calc\_LUL}(y,x,-b_2,-b_1).$$

6.4.3 Gambler's Ruin Problem

Right Boundary: `gambler_ruin_right`
Probability of reaching R starting at L:

$$\text{gambler\_ruin\_right}(L,R,p_{\text{right}}) = \begin{cases} \frac{L}{L+R}, & p_{\text{right}} = 0.5, \\ 1, & p_{\text{right}} = 1, \\ 0, & p_{\text{right}} = 0, \\ \frac{1-v^L}{1-v^{L+R}}, & \text{otherwise}, \ v = \frac{1-p_{\text{right}}}{p_{\text{right}}}. \end{cases}$$

Left Boundary: `gambler_ruin_left`
Probability of reaching 0:

$$\text{gambler\_ruin\_left}(L,R,p_{\text{left}}) = 1 - \text{gambler\_ruin\_right}(L,R,1-p_{\text{left}}).$$

Dynamic programming (7)

TargetSum.py
Description: Equivalent to bitset<1000001> dp; dp.set(0); for(int j = 0; j < n; j++) dp —= dp << s[j];
Time: $\mathcal{O}(N \cdot MAX)$ 25de18, 6 lines

```
MAX = 1000001
dp = [0] * MAX
dp[0] = 1
for x in a:
    for j in range(MAX-1, x-1, -1):
        dp[j] |= dp[j-x]
```

BitmaskDP.py
Description: GospersHack(k,n) génère toutes les combinaisons possibles de k bits parmi n bits
Time: X f969ed, 18 lines

```
def GospersHack(k,n):
    if k==0: yield 0
    cur=(1<<k)-1
    while 0<cur<1<<n:
        yield cur
        lb=cur&-cur
        r=cur+lb
        cur=(r^cur)>>lb.bit_length()+1|r
dp = [1<<59] * (1<<N)
dp[0] = 0
for mask in range(1, 1<<N):
    for col1 in range(N):
        if mask>>col1&1 == 0: continue
        c = 0
        for col2 in range(N):
            if mask>>col2&1: continue
            c += cost[col1][col2]
        dp[mask] = min(dp[mask], dp[mask^(1<<col1)]+c)
```

LIS.py
Description: Compute LIS
Time: $\mathcal{O}(N \log N)$ 1e1871, 15 lines

```
def longest_increasing_subsequence(lst):
    idxs, nums = [], []
    for n in lst:
        idx = bisect.bisect_left(nums, n)
        if idx == len(nums):nums.append(n)
        else:nums[idx] = n
        idxs.append(idx)
    ct = len(nums) - 1
    ret, ret_idx = [], []
    for i in range(len(lst)-1,-1,-1):
        if idxs[i] == ct:
            ret.append(lst[i])
            ret_idx.append(i); ct-=1
    ret.reverse(); ret_idx.reverse()
    return [ret, ret_idx]
```

TSP.py
Description: Also retrieve path
Time: $\mathcal{O}\left(N^2 \cdot 2^N\right)$ 327cdc, 25 lines

```
def tsp_dp(n, dist):
    dp = [[-1] * (1<<n) for _ in range(n)]
    _next = [[-1] * (1<<n) for _ in range(n)]
    def F(i, mask):
        mask ^= (1<<i)
        if mask == 0:
            return dist[i][0]
        if dp[i][mask] != -1:
            return dp[i][mask]
        dp[i][mask] = 1<<59
        for bit in range(n):
            if mask & (1<<bit):
                new = F(bit, mask) + dist[i][bit]
                if new < dp[i][mask]:
                    dp[i][mask] = new
                    _next[i][mask] = bit
        return dp[i][mask]
    F(0, (1 << n) - 1)
    path, node, mask = [], 0, (1 << n) - 1
    while node != -1:
        mask ^= (1 << node)
        path.append(node)
        node = _next[node][mask]
    path.append(0)
    return path
```

StockSpan.py
Description: Max Reactangle Area in Histogram
Time: $\mathcal{O}(N)$ 62ca62, 16 lines

```
def compute_extension(a):
    extension, ms = [0], [0]
    for i in range(1, len(a)):
        while ms and a[i] <= a[ms[-1]]: ms.pop()
        if ms: extension.append(i - ms[-1] - 1)
        else: extension.append(i)
        ms.append(i)
    return extension
n = int(input())
a = list(map(int,input().split()))
left_extension = compute_extension(a)
right_extension = compute_extension(a[::-1])[::-1]
cover = [a[i] * (r+l+1) for i, (l,r) in
    enumerate(zip(left_extension, right_extension))]
print(*left_extension); print(*right_extension)
print(*cover); print(max(cover))
```

CYK.py
Description: Cocke–Younger–Kasami
Time: $\mathcal{O}\left(n^3 \cdot |G|\right)$, n length of parsed string, $|G|$ size of the CNF grammar. 30cdf9, 19 lines

```
s = input(); n = len(s)
dp = [[[1<<59] * k for _ in range(n)] for _ in range(n)]
for i in range(n):
```

```
    dp[i][i][chars.find(s[i])] = 0
for span in range(1, n):
    for l in range(n - span):
        # dp[l][l+span][z] = min| dp[l][l+d][x] + dp[l+d+1][l+
            span][y] + cost(x,y) if x+y -> z
        for d in range(span):
            for x in range(k):
                for y in range(k):
                    c, z = rule[x][y]
                    dp[l][l+span][z] = min(dp[l][l+span][z],
                        dp[l][l+d][x] + dp[l+d+1][l+span][y] +
                            c)
bestcost = float('inf')
best = None
for char in chars:
    if bestcost > dp[0][n-1][chars.find(char)]:
        bestcost = dp[0][n-1][chars.find(char)]
        best = char
```

EditDist.py

Description: Edit Distance: suppression, ajout, match ou remplace
Time: $\mathcal{O}\left(N^2\right)$

19ff89, 12 lines

```
def edit_dist(a, b):
    if len(a) > len(b): a, b = b, a
    dp = [[1<<20]*(len(a)+1)for _ in range(len(b)+1)]
    dp[0][0] = 0
    for i in range(1, len(b)+1):
        if i < len(a)+1: dp[0][i] = i
        dp[i][0] = i
    for i in range(1, len(b)+1):
        for j in range(1, len(a)+1):
            dp[i][j] = min(dp[i-1][j]+1,dp[i][j-1]+1,
                dp[i-1][j-1]+int(a[j-1]!=b[i-1]))
    return dp[len(b)][len(a)]
```

Interleaving.py

Description: Test if string C is an interleaving of strings a and b
Time: $\mathcal{O}\left(N^2\right)$

897224, 11 lines

```
if len(a) + len(b) != len(c): print('NO') & exit()
dp = [[0] * (len(b)+1) for _ in range(len(a)+1)]
for i in range(len(a)+1):
    for j in range(len(b)+1):
        if i == 0 and j == 0: dp[i][j] = 1
        elif i == 0 and b[j-1] == c[j-1]: dp[i][j]=dp[i][j-1]
        elif j == 0 and a[i-1] == c[i-1]: dp[i][j]=dp[i-1][j]
        else:
            if a[i-1] == c[i+j-1]: dp[i][j] = dp[i-1][j]
            if b[j-1] == c[i+j-1]: dp[i][j] |= dp[i][j-1]
print('YES' if dp[len(a)][len(b)] else 'NO')
```

LCSubsequence.py

Description: Longest Common Subsequence of two strings. Reconstruct it from the dp table
Time: $\mathcal{O}\left(N^2\right)$

815122, 6 lines

```
dp = [[0] * (len(a)+1) for _ in range(len(b)+1)]
for i in range(1, len(b)+1):
    for j in range(1, len(a)+1):
        if b[i-1] == a[j-1]: dp[i][j] = dp[i-1][j-1]+1
        else: dp[i][j] = max(dp[i-1][j], dp[i][j-1])
print(dp[len(b)][len(a)])
```

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}\left(N \max(w_i)\right)$

b20ccc, 16 lines

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

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \leq f(a,d)$ and $f(a,c) + f(b,d) \leq f(a,d) + f(b,c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}\left(N^2\right)$

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i)\leq k<hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}\left((N + (hi - lo)) \log N\right)$

d38d2b, 17 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

Graph (8)

8.1 Fundamentals

Bellman-Ford

$d_k[v]$ = shortest length from source to v using at most k edges

$$d(v) = \min_u d(u) + w_{uv}$$

Floyd-Warshall

$d_k[u][v]$ = shortest length between u and v using only nodes $< k$

$$d(u,v) = \min_w d(u,w) + d(w,v)$$

System of weighted contraints

$A - B \leq w$: add edge $B \to A$ with weight w. Add edge $(0, \text{vtx}, 0)$ for each vertex. Negative cycle: no solution. Otherwise, solution: $-\min(D)$.

Dijkstra.py

Description: Dijkstra for dense graphs.
Time: $\mathcal{O}\left(V^2\right)$

16fbc9, 14 lines

```
def dijkstra(s, t, adjlist, adjmat):
    n = len(adjlist); d = [float('inf')] * n; d[s] = 0
    vis = [False] * n
    for _ in range(n):
        v = -1
        for j in range(n):
            if not vis[j] and (v == -1 or d[j] < d[v]): v=j
        if d[v] == float('inf'): break
        vis[v] = True
        for u in adjlist[v]:
            if (v==s and u==t) or (v==t and u==s):continue
            if d[v] + adjmat[v][u] < d[u]:
                d[u] = d[v] + adjmat[v][u]
    return d[t] if d[t] != float('inf') else -1
```

BellmanFord.py

Description: Bellman-Ford
Usage: edges 1-indexed
Time: $\mathcal{O}\left(VE\right)$

5d1684, 15 lines

```
def bellmanford(n, edges):
    dist = [float('inf')] * (n + 1)
    prec = [-1] * (n + 1)
    for _ in range(n):
        x = -1
        for u, v, w in edges:
            if dist[u] + w < dist[v]:
                prec[v] = u; dist[v] = dist[u] + w; x = v
    if x == -1: return []
    for _ in range(n): x = prec[x]
    end, path = x, [x]
    while prec[x] != end:
        x = prec[x]
        path.append(x)
    return (path+[end])[::-1]
```

Dials.h

Description: Dial's algorithm, Faster than dijkstra for graphs with weights <= 10
Time: $\mathcal{O}\left(V \cdot lim + E\right)$.

77655f, 15 lines

```
void dials(int st, vector<vpii> adj, int lim=10){
    int n = sz(adj);
    vector<int> dist(n, -1);
    vector<vector<int>> Qs(lim + 1);
    dist[st] = 0; Qs[0].push_back(st);
    for (int d = 0, mx = 0; d <= mx; d++) {
        for (auto& Q = Qs[d % (lim + 1)]; Q.size();) {
            int cur = Q.back(); Q.pop_back();
            if (dist[cur] != d) continue;
            for (const auto& [nxt, cost] : adj[cur]) {
                if (dist[nxt] == -1 || dist[nxt] > d + cost) {
                    dist[nxt] = d + cost;
                    Qs[dist[nxt] % (lim + 1)].push_back(nxt);
                    mx = max(mx, dist[nxt]);}}}}
}
```

Number of Spanning Trees

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

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 $k = 1 \ldots n$,

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

8.2 Network flow

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.

f6e693, 38 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vector<int> 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, int((adj[b]).size()), c, c});
    adj[b].push_back({a, int((adj[a]).size()) - 1, rcap, rcap})
        ;}
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < int((adj[v]).size()); 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;
        for(int L=0; L<31; L++){ // 'int L=30' maybe faster for
            random data
            do {
                lvl = ptr = vector<int>(int((q).size()));
                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; }
};
```

dinic.py

Description: Dinic Maximum flow, fonctionne si arcs bidirectionnels à capacité partagée
Time: $\mathcal{O}(V^2 * E)$

1dadab, 31 lines

```
def dinic(source, target, graph):
    def _dinic_step(lev, u, t, limit):
        if limit <= 0: return 0
        if u == t: return limit
        val = 0
        for v in graph[u]:
            residuel = graph[u][v] - flow[u][v]
            if lev[v] == lev[u] + 1 and residuel > 0:
                z = min(limit, residuel)
                aug = _dinic_step(lev, v, t, z)
                flow[u][v] += aug; flow[v][u] -= aug
                val += aug; limit -= aug
        if val == 0: lev[u] = None
        return val
    n, Q, total = len(graph), deque(), 0
    flow = [[0] * n for _ in range(n)]
    while True:
        Q.appendleft(source)
        lev = [None] * n; lev[source] = 0
        while Q:
            u = Q.pop()
            for v in graph[u]:
                if lev[v] is None and graph[u][v] > flow[u][v]:
                    lev[v] = lev[u] + 1; Q.appendleft(v)
        if lev[target] is None: break
        UB = sum(graph[source][v] for v in graph[source]) -
            total
        total += _dinic_step(lev, source, target, UB)
    return flow, total
def add_edge(u, v, c):
    if v not in graph[u]: graph[u][v] = 0; graph[v][u] = 0
    graph[u][v] += c
```

GlobalMinCut.h

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

8b0e19, 21 lines

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

GomoryHu.h

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

e2b333, 13 lines

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

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.

3ce322, 69 lines

```
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,(int)(ed[to]).size(),cap,
        cost,0 });
    ed[to].push_back(edge{ to,from,(int)(ed[from]).size()-1,0,-
        cost,0 });
  }
  void path(int s) {
    fill(seen.begin(), seen.end(), 0);
    fill(dist.begin(), dist.end(), 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 });
        }
      }
    }
    for(int i=0; i<N; i++) 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;
    }
  }
  for(int i=0; i<N; i++) 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(pi.begin(), pi.end(), INF); pi[s] = 0;
  int it = N, ch = 1; ll v;
  while (ch-- && it--)
    for(int i=0; i<N; i++) 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
}
};
```

CirculationDemands.py

Description: Circulation with lowerbounds d(e) <= f(e) <= c(e)
Time: X

965054, 18 lines

```
graph = [{} for _ in range(N+2)]
isum, osum, edges = [0]*N, [0]*N, []
for _ in range(M):
    a, b, d, c = map(int,input().split())
    add_edge(graph, a, b, c-d)
    isum[b]+=d; osum[a]+=d
s, t = N, N+1
indemands = outdemands = 0
for v in range(N):
    need = isum[v] - osum[v]
    if need < 0:
        add_edge(graph, v, t, -need); outdemands -= need
    if need > 0:
        add_edge(graph, s, v, need); indemands += need
if indemands != outdemands: print("NON")
else:
    _, mf = dinic(s, t, graph)
    print("OUI" if mf == indemands else "NON")
```

8.3 $s-t$ coupe minimum pour graphe planaire

Dans le graphe *dual*, chaque cellule est un sommet, et il existe deux sommets supplémentaires s' et t'. Le sommet s' représente le bord gauche et haut de la grille et le sommet t' le bord droit et bas. Dans ce nouveau graphe, deux sommets sont reliés s'ils sont séparés par une arête du graphe primal. Le poids des arêtes est le même. Tout $s'-t'$ chemin dans le graphe dual de longueur w correspond à une $s-t$ coupe dans le graphe primal de même valeur w et vice-versa.

8.4 Matching

MCBF.py

Description: Max BM
Time: Very fast in practice

0ff26f, 35 lines

```
class BipartiteMatching:
    def __init__(self, n, m):
        self._n, self._m = n, m
        self._to = [[] for _ in range(n)]
    def add_edge(self, a, b):
        self._to[a].append(b)
    def solve(self):
        n, m, to = self._n, self._m, self._to
        prev, root, p, q = [-1]*n, [-1]*n, [-1]*n, [-1]*m
        updated = True
        while updated:
            updated = False
            s, s_front = [], 0
            for i in range(n):
                if p[i] == -1:
                    root[i] = i; s.append(i)
            while s_front < len(s):
                v = s[s_front]
                s_front += 1
                if p[root[v]] != -1: continue
                for u in to[v]:
                    if q[u] == -1:
                        while u != -1:
                            q[u] = v; p[v], u = u, p[v]
                            v = prev[v]
                        updated = True; break
                    u = q[u]
                    if prev[u] != -1: continue
                    prev[u] = v; root[u] = root[v]
                    s.append(u)
            if updated:
                for i in range(n):
                    prev[i] = -1; root[i] = -1
        return [(v, p[v])
                for v in range(n) if p[v] != -1]
```

VertexCover.py

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

16a9bb, 30 lines

```
bm = BipartiteMatching(n, m)
matching = bm.solve()
graph = [{} for _ in range(n + m)]
unmatched_left = [True] * n
# left side : 0 to n - 1, right side : n to n + m - 1
for e, o in matching:
    # matched edges start from the right side of the graph to
        the left side
    unmatched_left[e] = False
    graph[o+n][e] = 1; graph[e][o+n] = -1
# free edges start from the left side of the graph to the right
    side
for e, o in edges:
    if o+n not in graph[e]:
        graph[e][o+n] = 1; graph[o+n][e] = -1
# Run DFS from unmatched nodes of the left side,
# in this traversal some nodes will become visited, others will
    stay unvisited.
vis = [False] * (n + m)
for e in range(n):
    if unmatched_left[e]:
        Q = [e]
        while Q:
            v = Q.pop()
            vis[v] = True
            for u, w in graph[v].items():
                if w==1 and not vis[u]: Q.append(u)
# The MVC nodes are the visited nodes from the right side, and
    unvisited nodes from the left side.
MVC = []
for e in range(n):
    if not vis[e]: MVC.append(even[e])
for o in range(n, n + m):
    if vis[o]: MVC.append(odd[o - n])
```

Dilworth.py

Description: Dilworth's theorem states that in a directed acyclic graph, the size of a minimum general path cover equals the size of a maximum antichain. Largeur d'un ordre partiel = taille de la plus grande anti-chaîne.
Usage: Produire un graphe biparti H(V-, V+, E) avec V-, V+ étant des copies de V, et (u-, v+) appartient à E ssi (u, v) appartient à A. Retourne partition en chaînes
Time: X

ace890, 15 lines

```
def dilworth(graph):
    n = len(graph)
    match = [None] * n
    for a, b in bm.solve():
        match[b] = a
    part = [None] * n
    nb_chains = 0
    for v in range(n - 1, -1, -1):
        if part[v] is None:
            u = v
            while u is not None:
                part[u] = nb_chains
                u = match[u]
            nb_chains += 1
    return part
```

KuhnMunkres.py

Description: Couplage parfait de profit maximal - si minimal, changer signe des poids - si abs(U) > abs(V), ajouter sommets à V reliés à tous les sommets de U par un poids 0 - si graph non complet, le compléter avec arêtes de poids float('-inf')
Time: $\mathcal{O}\left(\text{abs}(U)^2 \times \text{abs}(V)\right)$

3a9cfe, 36 lines

```
def kuhn_munkres(G, TOLERANCE=1e-6):
    assert len(G) <= len(G[0])
    nU, nV = len(G), len(G[0])
    U, V = range(nU), range(nV)
    mu, mv = [None] * nU, [None] * nV
    lu, lv = [max(row) for row in G], [0] * nV
    for root in U:
        au = [False] * nU; au[root] = True
        Av = [None] * nV
        slack = [(lu[root]+lv[v]-G[root][v],root)for v in V]
        while True:
            (delta, u), v = min(
                (slack[v], v)for v in V if Av[v] is None)
            assert au[u]
            if delta > TOLERANCE:
                for u0 in U:
                    if au[u0]: lu[u0] -= delta
                for v0 in V:
                    if Av[v0] is not None: lv[v0] += delta
                    else:
                        (val, arg) = slack[v0]
                        slack[v0] = (val - delta, arg)
            assert abs(lu[u] + lv[v] - G[u][v]) <= TOLERANCE
            Av[v] = u
            if mv[v] is None: break
            u1 = mv[v]
            assert not au[u1]
            au[u1] = True
            for v1 in V:
```

```
            if Av[v1] is None:
                alt = (lu[u1] + lv[v1] - G[u1][v1], u1)
                if slack[v1] > alt: slack[v1] = alt
        while v is not None:
            u = Av[v]; prec = mu[u]
            mv[v], mu[u] = u, v; v = prec
    return (mu, sum(lu) + sum(lv))
```

GaleShapley.py

Description: Mariages stables.
Usage: men et women numérotés de 0 à n - 1. Le tableau men contient pour chaque homme la liste des femmes par préférence décroissante. Même chose pour women
Time: $\mathcal{O}(V^2)$

cc837e, 18 lines

```
def gale_shapley(men, women):
    n = len(men)
    assert n == len(women)
    suiv, mari = [0] * n, [None] * n
    rank = [[0] * n for j in range(n)]  # build rank
    for j in range(n):
        for r in range(n): rank[j][women[j][r]] = r
    singles = deque(range(n))  # all men are single
    while singles:             # and get in the queue
        i = singles.popleft()
        j = men[i][suiv[i]]; suiv[i] += 1
        if mari[j] is None: mari[j] = i
        elif rank[j][mari[j]] < rank[j][i]:
            singles.append(i)
        else:
            singles.put(mari[j])  # divorce mari[j]
            mari[j] = i
    return mari
```

8.5 DFS algorithms

SCC.py

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

a890c0, 22 lines

```
def tarjan(G):
    n = len(G)
    SCC, S, P = [], [], []
    Q, st = list(range(n)), [0] * n
    while Q:
        node = Q.pop()
        if node < 0:
            d = st[~node] - 1
            if P[-1] > d:
                SCC.append(S[d:])
                del S[d:]; P.pop()
                for v in SCC[-1]:
                    st[v] = -1
        elif st[node] > 0:
            while P[-1] > st[node]:P.pop()
        elif st[node] == 0:
            S.append(node)
            P.append(len(S))
            st[node] = len(S)
            Q.append(~node)
            Q.extend(G[node])
    return SCC
```

BiconnectedComponents.py

Description: Finds all biconnected components in an undirected graph. 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 (not part of any cycle).
Time: $\mathcal{O}(E + V)$

6d7289, 72 lines

```
class BiconnectedComponents:
    def __init__(self, graph):
        self.n = n = len(graph)
        self.m = sum(len(dsts) for dsts in graph) >> 1
        self.nbcs, self.bcvp, self.graph = 0, [], graph
        if n == 0: self.nbcs = 0; return
        used, parent, order = bytearray(n),[0]*n,[0]*n
        def dfs(start, idx):
            Q, parent[start] = [start], -1
            while Q:
                cur = Q.pop()
                if used[cur]: continue
                used[cur], order[idx] = 1, cur
                idx += 1
                for dst in graph[cur]:
                    if not used[dst]:
                        parent[dst] = cur; Q.append(dst)
            return idx
        idx = 0
        for s in range(n):
            if not used[s]: idx = dfs(s, idx)
        v2dfs = [0] * n
        for i in range(n): v2dfs[order[i]] = i
        low = v2dfs[:]
        for p in range(n):
            for e in graph[p]:
                low[p] = min(low[p], v2dfs[e])
        for i in reversed(range(n)):
            p = order[i]
            pp = parent[p]
            if pp >= 0: low[pp] = min(low[pp], low[p])
        nbcs = 0
        for p in order:
            if parent[p] < 0: continue
            pp = parent[p]
            #if low[p] >= v2dfs[pp]: cut_nodes.add(pp)
            #if low[p] > v2dfs[pp]: cut_edges.append((pp, p))
            if low[p] < v2dfs[pp]:
                low[p] = low[pp];self.bcvp.append((low[p], p))
            else:
                low[p] = nbcs; nbcs += 1
                self.bcvp.append((low[p], pp));self.bcvp.append
                    ((low[p], p))
        for s in range(n):
            if not graph[s]:
                self.bcvp.append((nbcs, s)); nbcs += 1
        self.nbcs = nbcs
    def __len__(self): return self.nbcs
    def bcc(self):
        bcc_ = [[] for _ in range(self.nbcs)]
        for idx, v in self.bcvp: bcc_[idx].append(v)
        return bcc_
    def merged_bcc(self):
        N, bcc_, repr_ = 0, [], [-1] * self.n
        for vlst in self.bcc():
            if len(vlst) <= 2: continue
            to_merge = []
            for v in vlst:
                if repr_[v] == -1: repr_[v] = N
                else: to_merge.append(repr_[v])
            if to_merge == []:
                bcc_.append(vlst); N += 1
            else:
                main = min(to_merge)
                for v in vlst: repr_[v] = main
                for i in to_merge:
                    for v in bcc_[i]: repr_[v] = main
        BCCs = [[] for _ in range(N)]
        for v in range(self.n):
            if repr_[v] == -1:
                repr_[v] = N; BCCs.append([v]); N += 1
            else: BCCs[repr_[v]].append(v)
        return BCCs, repr_
```

2sat.py

Description: Solve 2SAT
Time: $\mathcal{O}(N)$

90402d, 21 lines

```
class TwoSat:
    def __init__(self, n):
        self.n = n
        self.graph = [[] for _ in range(2*n)]
    def _imply(self, x, y):
        self.graph[x].append(y if y >= 0 else 2*self.n+y)
    def either(self, x, y):
        self._imply(~x, y)
        self._imply(~y, x)
    def set(self, x):
        self._imply(~x, x)
    def implies(self, x, y):
        self.either(~x, y)
    def solve(self):
        SCC = tarjan(self.graph)
        order = [0] * (2 * self.n)
        for i, comp in enumerate(SCC):
            for x in comp: order[x] = i
        for i in range(self.n):
            if order[i] == order[~i]: return False, None
        return True,[+(order[i] > order[~i])for i in range(self
            .n)]
```

EulerTour.py

Description: Check connectivity - Undirected graph : connected and all nodes have even deg, or only 2 have odd degree - Directed graph : At most one node has out_i - in_i = 1 and at most one node has in_i - out_i = 1 - Multigraph : Mind case with only one node
Time: $\mathcal{O}(N)$

f294be, 29 lines

```
def eulerian_tour(m, s, graph):
    # also works with a multigraph if directed
    P, Q = [], [s]
    while Q:
        u = Q[-1]
        if len(graph[u]):
            v = graph[u].pop()
            # graph[v].discard(u) # if undir
            Q.append(v)
        else:
            P.append(Q.pop())
    return P[::-1] if len(P)==m+1 else -1

def eulerian_tour_undir_multigraph(m, s, graph, rem):
    P, Q = [], [s]
    while Q:
        u = Q[-1]
        while len(graph[u]):
            if rem[u][len(graph[u])-1]:
                graph[u].pop()
            else: break
        if len(graph[u]):
            rem[u][len(graph[u])-1] = True
            v, j = graph[u].pop()
            rem[v][j] = True
            Q.append(v)
        else:
            P.append(Q.pop())
    return P[::-1] if len(P)==m+1 else -1
```

TopoSort.py

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

bf5396, 13 lines

```
revgraph, outdeg = [[] for _ in range(n)], [0] * n
for _ in range(m):
    a, b = map(int,input().split())
    revgraph[b].append(a); outdeg[a] += 1
Q = [-node for node in range(n) if outdeg[node] == 0]
heapify(Q); order = []
while Q:
    node = -heappop(Q)
    order.append(node + 1)
    for neigh in revgraph[node]:
        outdeg[neigh] -= 1
        if outdeg[neigh] == 0: heappush(Q, -neigh)
print(*order[::-1])
```

DfsTree.py

Description: Compute DFS-Tree
Time: $\mathcal{O}(N)$

063486, 19 lines

```
if cut_nodes_edges(g): print(0) & exit()
ans = [set() for _ in range(n)]
seen = [False]*n; seen[0]=True; d = [-1]*n
@bootstrap
def dfs(p, v):
    d[v] = d[p]+1
    for u in g[v]:
        if u == p: continue
        if seen[u]:
            # back edge
            if d[v] < d[u]: ans[u].add(v)
            else: ans[v].add(u)
        else:
            seen[u]=True; ans[v].add(u)  # dfs tree edge
            (yield dfs(v, u))
    yield
dfs(0, 0)
for v in range(n):
    for u in ans[v]: print(v+1, u+1)
```

8.6 General

ChromaticNumber.py

Description: Minimum Clique Cover <=> Coloring of Complement Graph
Constraints : N <= 20, No multiples or self edge
Time: $\mathcal{O}\left(2^N\right)$

d7362d, 18 lines

```
def chromatic_number(n:int, edges:list[tuple[int,int]])->int:
    edge = [0] * n
    for uv in edges:
        u, v = uv; edge[u]|=1<<v; edge[v]|=1<<u
    dp = [0] * (1 << n); dp[0] = 1; cur = [0] * (1 << n)
    for bit in range(1, 1 << n):
        v = (~bit & (bit-1)).bit_count()
        dp[bit] = dp[bit^(1<<v)]+dp[(bit^(1<<v))&(~edge[v])]
    for bit in range(1 << n):
        if (n - bit.bit_count()) & 1: cur[bit] = -1
        else: cur[bit] = 1
    for k in range(1, n):
        tmp = 0
        for bit in range(1 << n):
            cur[bit] *= dp[bit]; tmp += cur[bit]
        if tmp != 0: res = k; break
    else: res = n
    return res
```

MIS.py

Description: Complement of Minimum Vertex Cover. Max clique of the complement graph
Time: Works for N < 40

4e62e9, 31 lines

```
def maximum_independent_set(n: int, edges: list[tuple[int, int
    ]])\
    -> tuple[int, list[int]]:
    adj = [0] * n
    for u, v in edges:
        if u > v: u, v = v, u
        adj[u] |= 1 << v
    dp = {0: 0}
    for i in range(n):
        nex = dict()
        for S, val in dp.items():
            if S >> i & 1 == 0:
                S1 = S | adj[i]
                nv = (val + (1 << n)) | (1 << i)
                if S1 in nex:
                    if nex[S1] < nv:
                        nex[S1] = nv
                else:
                    nex[S1] = nv
            S2 = S & ~(1 << i)
            if S2 in nex:
                if nex[S2] < val:
                    nex[S2] = val
            else:
                nex[S2] = val
        dp = nex
    size, bitset = divmod(dp[0], 1 << n)
    res = []
    for i in range(n):
        if bitset >> i & 1:
            res.append(i)
    return size, res
```

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;
}
```

MaximalCliques.h

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

b0d5b1, 12 lines

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

Edmonds.h

Description: maximum matching (graph not necessarily bipartite)
Time: $\mathcal{O}\left(V^4\right)$

e1160a, 63 lines

```
struct EdmondsBlossom{
  int N;
  vector<int> match,vis;
  EdmondsBlossom(int N): N(N){}
  void couple (int, int m) { match[n]=m; match[m]=n; }
  //returns true if something interesting has been found,
  // thus an augmenting path or a blossom
  bool dfs(int n, vector<vector<bool>> &conn, vector<int>&
      blossom) {
    vis[n]=0;
    for(int i=0;i<N;i++) if(conn[n][i]) {
      if(vis[i]==-1) {
        vis[i]=1;
        if(match[i]==-1 || dfs(match[i], conn, blossom)) {
          couple(n,i); return true; }
      }
      if(vis[i]==0 || blossom.size()) { // found flower
        blossom.push_back(i); blossom.push_back(n);
        if(n==blossom[0]) {match[n]=-1; return true; }
        return false;
      }
    }
    return false;
  }
  // search for an augmenting path
  bool augment(vector<vector<bool>> &conn) {
    for(int m=0;m<N;m++) if(match[m]==-1) {
      vector<int> blossom;
      vis=vector<int>(N,-1);
      if(!dfs(m,conn,blossom)) continue;
      if(blossom.size()==0) return true; // augmenting path
          found
      // blossom is found so build shrunken graph
      int base=blossom[0], S=blossom.size();
      vector<vector<int>> newconn=conn;
      for(int i=1;i<S-1;i++)
        for(int j=0;j<N;j++)
          newconn[base][j]=newconn[j][base]|=conn[blossom[i]][j
              ];
```

```
      for(int i=1;i<S-1;i++)
        for(int j=0;j<N;j++)
          newconn[blossom[i]][j]=newconn[j][blossom[i]]=0;
      newconn[base][base]=0; // is now the new graph
      if(!augment(newconn)) return false;
      int n=match[base];
      // if n!=-1 the augmenting path ended on this blossom
      if(n!=-1) for(int i=0;i<S;i++) if(conn[blossom[i]][n]) {
        couple(blossom[i],n);
        if(i&1) for(int j=i+1; j<S; j+=2) couple(blossom[j],
            blossom[j+1]);
        else for(int j=0;j<i;j+=2) couple(blossom[j],blossom[j
            +1]);
        break;
      }
      return true;
    }
    return false;
  }
  // conn is a NxN adjacency matrix
  // returns size of max matching
  // matching can be found in match vector
  int compute(vector<vector<int>> &conn) {
    int res=0;
    match=vector<int>(N,-1);
    while(augment(conn)) res++;
    return res;
  }
}
```

PeoChordal.h

Description: Simple undirected unweighted graph Graph chordal = no induced cycle of length four or more. PEO = ordering of the vertices such that for every vertex v, v and the neighbors of v that appear after it in the ordering form a clique. Graph is chordal <=> it has a PEO. Find PEO or induced cycle of length four or more. To color the graph greedily according to the PEO: Starting from the last vertex in the PEO, assign colors to each vertex in order, choosing the smallest available color that hasn't been assigned to its already-colored neighbors. Because the graph is chordal, each vertex's neighbors form a clique among vertices appearing later in the PEO.
Time: $\mathcal{O}(N)$

6fc884, 68 lines

```
struct Set {list<int> L; int last; Set() { last = 0; }};
struct PEO {
    int N;
    vector<vector<int> > g;
    vector<int> vis, res; list<Set> L;
    vector<list<Set>::iterator> ptr;
    vector<list<int>::iterator> ptr2;
    PEO(int n, vector<vector<int> > _g) {
        N = n; g = _g;
        for (int i = 1; i <= N; i++) sort(g[i].begin(), g[i].
            end());
        vis.resize(N + 1); ptr.resize(N + 1); ptr2.resize(N +
            1);
        L.push_back(Set());
        for (int i = 1; i <= N; i++) {
            L.back().L.push_back(i);
            ptr[i] = L.begin(); ptr2[i] = prev(L.back().L.end()
                );
        }
    }
    pair<bool, vector<int>> Run() {
        int time = 0;
        while (!L.empty()) {
            if (L.front().L.empty()) { L.pop_front(); continue;
                }
            auto it = L.begin();
            int n = it->L.front(); it->L.pop_front();
            vis[n] = ++time;
            res.push_back(n);
            for (int next : g[n]) {
                if (vis[next]) continue;
                if (ptr[next]->last != time) {
                    L.insert(ptr[next], Set()); ptr[next]->last
                        = time;
                }
                ptr[next]->L.erase(ptr2[next]); ptr[next]--;
                ptr[next]->L.push_back(next);
                ptr2[next] = prev(ptr[next]->L.end());
            }
        }
        // PEO existence check
        for (int n = 1; n <= N; n++) {
            int mx = 0;
            for (int next : g[n]) if (vis[n] > vis[next]) mx =
                max(mx, vis[next]);
            if (mx == 0) continue;
            int w = res[mx - 1];
            for (int next : g[n]) {
                if (vis[w] > vis[next] && !binary_search(g[w].
                    begin(), g[w].end(), next)){
                    vector<int> chk(N+1), par(N+1, -1); // If w
                        and next are not connected, the graph
                        is not chordal
                    deque<int> dq{next}; chk[next] = 1;
                    while (!dq.empty()) {
                        int x = dq.front(); dq.pop_front();
                        for (auto y : g[x]) {
                            if (chk[y] || y == n || y != w &&
                                binary_search(g[n].begin(), g[
                                n].end(), y)) continue;
                            dq.push_back(y); chk[y] = 1; par[y]
                                = x;
                        }
                    }
                    vector<int> cycle{next, n};
                    for (int x=w; x!=next; x=par[x]) cycle.
                        push_back(x);
                    return {false, cycle};
                }
            }
        }
        reverse(res.begin(), res.end());
        return {true, res};
    }
};
vector<vector<int>> G(N+1);
for(int i=1,s,e; i<=M; i++)
    cin >> s >> e, G[s+1].push_back(e+1), G[e+1].push_back(s+1)
        ;
auto [flag,vec] = PEO(N, G).Run();
if(flag){for(auto i : vec) cout << i - 1 << " ";} //YES
else{for(auto i : vec) cout << i - 1 << " ";} //NO
```

Trees (9)

BinaryLifting.py

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

9e41a8, 23 lines

```
LOG = n.bit_length()
up = [[0] * LOG for _ in range(n)]
depth = [0] * n
for node in range(1, n):
    up[node][0] = prev[node]
    depth[node] = depth[prev[node]] + 1
for k in range(1, LOG):
    for node in range(1, n):
        up[node][k] = up[up[node][k-1]][k-1]
def get_lca(a, b):
    if depth[a] < depth[b]: a, b = b, a
    k = depth[a] - depth[b]
    for j in range(LOG):
        if k & (1<<j): a = up[a][j]
    if a == b: return a
    for j in range(LOG - 1, -1, -1):
        if up[a][j] != up[b][j]: a,b = up[a][j],up[b][j]
    return up[a][0]
def get_kth_ancestor(a, k):
    if depth[a] < k: return -1
    for j in range(LOG):
        if k >> j & 1: node = up[node][j]
    return node
```

SubtreeQueries.py

Description: For subtree queries with Fenwick
Usage: `update(index[u], x - val[index[u]]); query(index[u], index[u] + sz[u])`
Time: $\mathcal{O}(N)$

ce7874, 28 lines

```
prev = [None] * n
sz, ts = [1] * n, [-1] * n
ts[0] = 0
Q, order = [0], []
while Q:
    v = Q[-1]
    if ts[v] == len(adj[v]):
        for u in adj[v]:
            if u == prev[v]: continue
            sz[v] += sz[u]
        order.append(Q.pop())
    else:
        u = adj[v][ts[v]]
        if u == prev[v]:
            ts[v] += 1
            if ts[v] == len(adj[v]):continue
            u = adj[v][ts[v]]
        ts[v] += 1; ts[u] = 0; prev[u] = v
        Q.append(u)
order.reverse()
index = [None] * n
for i, e in enumerate(order):
    index[e] = i
val = [val[i] for i in order]
T = list(val)
for i, v in enumerate(val):
    j = i | (i+1)
    if j < n: T[j] += T[i]
```

SmallToLarge.py

Description: Small-To-Large Merging. Query the number of distinct colors in each subtree.
Usage: `When done with u :`
`ans[u] = _sum[u]; merge(prev[u], u)`
Time: $\mathcal{O}(N \log N)$

307497, 17 lines

```
def merge(u, v):
    if len(colors[u]) < len(colors[v]):
        colors[v], colors[u] = colors[u], colors[v]
        _sum[v], _sum[u] = _sum[u], _sum[v]
        mx[v], mx[u] = mx[u], mx[v]
    for c, value in colors[v].items():
        if c not in colors[u]:
            colors[u][c] = 0
        colors[u][c] += value
```

```
        if colors[u][c] > mx[u]:
            mx[u] = colors[u][c]
            _sum[u] = c
        elif colors[u][c] == mx[u]:
            _sum[u] += c
_sum = list(map(int,input().split()))
mx = [1] * n
colors = [{c:1} for c in _sum]
```

HLD.h

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

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

03139d, 33 lines

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0; vector<vi> adj;
  vi par, siz, rt, pos; Node *tree;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
      rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
  void dfsSz(int v) {
    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for (int& u : adj[v]) {
      par[u] = v; dfsSz(u); siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);}}
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u); dfsHld(u);
    }}
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (pos[rt[u]] > pos[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v] + 1);
    }
    if (pos[u] > pos[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);}
  void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r, val); });
        }
  int queryPath(int u, int v) { // Modify depending on problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));});
    return res;}
  int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);}
};
```

CentroidDecomp.h

Description: Centroid Decomposition is a divide and conquer technique for trees. Works by repeated splitting the tree and each of the resulting subgraphs at the centroid, producing $\mathcal{O}(\log N)$ layers of subgraphs.

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

018bbd, 59 lines

```
const int INF = 1e9;
vector<vector<int>> adj; vector<int> subtree_size;
// min_dist[v] := the minimal distance between v and a red node
vector<int> min_dist; vector<bool> is_removed;
vector<vector<pair<int, int>>> ancestors;
int get_subtree_size(int node, int parent = -1) {
  subtree_size[node] = 1;
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    subtree_size[node] += get_subtree_size(child, node);}
  return subtree_size[node];
}
int get_centroid(int node, int tree_size, int parent = -1) {
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    if (subtree_size[child] * 2 > tree_size) {
      return get_centroid(child, tree_size, node);}}
  return node;
}
/* Calculate the distance between current `node` and the `
    centroid` it belongs
 * to. The distances between a node and all its centroid
    ancestors are stored
 * in the vector `ancestors`.
 * @param cur_dist the distance between `node` and `centroid`*/
void get_dists(int node, int centroid, int parent = -1, int
   cur_dist = 1) {
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    cur_dist++;
    get_dists(child, centroid, node, cur_dist);
    cur_dist--;
  }
  ancestors[node].push_back({centroid, cur_dist});}
void build_centroid_decomp(int node = 0) {
  int centroid = get_centroid(node, get_subtree_size(node));
  // For all nodes in the subtree rooted at `centroid`,
      calculate their distances to the centroid
  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    get_dists(child, centroid, centroid);}
  is_removed[centroid] = true;
  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    // build the centroid decomposition for all child
        components
    build_centroid_decomp(child);}
}
// Paint `node` red by updating all of its ancestors' minimal
    distances to a red node
void paint(int node) {
  for (auto &[ancestor, dist] : ancestors[node]) {
    min_dist[ancestor] = min(min_dist[ancestor], dist);}
  min_dist[node] = 0;}
// Print the minimal distance between `node` to a red node
void query(int node) {
  int ans = min_dist[node];
  for (auto &[ancestor, dist] : ancestors[node]) {
    if (!dist) { continue; }
    /* The distance between `node` and a red painted node is
        the sum of
     * the distance from `node` to one of its ancestors (`dist
        `) and the
     * distance from this ancestor to the nearest red node
     * (`min_dist[ancestor]`).*/
    ans = min(ans, dist + min_dist[ancestor]);}
  cout << ans << "\n";}
```

DirectedMST.h

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

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

"../data-structures/UnionFindRollback.h" 39e620, 49 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key; Node *l, *r; ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

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

Prim.py

Description: Min spanning tree for dense graphs

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

eedd2a, 14 lines

```
def prim(graph):
    visited = [False] * n
    distmin = [float('inf')] * n
    heap, total = [(0, 0)], 0
    while heap:
        d, i = heappop(heap)
        if visited[i]: continue
        visited[i] = True; total += d
        for j, w in graph[i]:
            if visited[j]: continue
            if d + w < distmin[j]:
                distmin[j] = d + w
                heappush(heap, (d + w, j))
    return total
```

Geometry (10)

10.1 Geometric primitives

Point.h

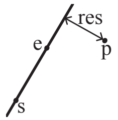
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

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

lineDistance.h

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

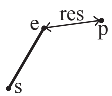


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



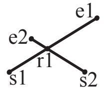
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

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



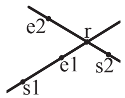
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;

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

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

lineIntersection.h

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



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

"Point.h" a01f81, 8 lines

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

sideOf.h

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

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

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

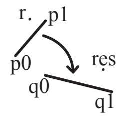
"Point.h" c597e8, 3 lines

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

linearTransformation.h

Description:

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



"Point.h" 03a306, 6 lines

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

LineProjectionReflection.h

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

"Point.h" b5562d, 5 lines

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

Angle.h

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

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

0f0602, 35 lines

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

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

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

10.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CircleLine.h

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

"Point.h" e0cfba, 9 lines

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

CirclePolygonIntersection.h

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

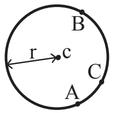
Time: $\mathcal{O}(n)$

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

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

circumcircle.h

Description:



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

"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

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

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

"circumcircle.h" 09dd0a, 17 lines

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

10.3 Polygons

InsidePolygon.h

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

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

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

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

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

PolygonArea.h

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

"Point.h" f12300, 6 lines

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

PolygonCenter.h

Description: Returns the center of mass for a polygon.

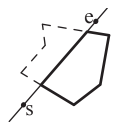
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

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

PolygonCut.h

Description:



Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

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

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

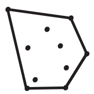
PolygonUnion.h

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

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

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

ConvexHull.h



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

"Point.h" 310954, 13 lines

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

HullDiameter.h

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

"Point.h" c571b8, 12 lines

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

PointInsideHull.h

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

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

```
typedef Point<ll> P;

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

LineHullIntersection.h

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

"Point.h" 7cf45b, 39 lines

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

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

10.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" ac41a6, 17 lines

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

ManhattanMST.h

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

"Point.h" df6f59, 23 lines

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

```
}
```

kdTree.h

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

"Point.h" bac5b0, 63 lines

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

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

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

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

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

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

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

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

10.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

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

3dHull.h

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

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

"Point3D.h" 5b45fc, 49 lines

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

611f07, 8 lines

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

10.6 Misc.

FacesPlanarGraph.py

Description: Compute faces delimited by segments, then sum of squared areas of faces

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

37cfea, 50 lines

```
from math import atan2, fsum
from collections import defaultdict, namedtuple
Point = namedtuple("p", "x y")
def shoelace(poly):
    fw = sum(poly[i - 1].x * poly[i].y for i in range(len(poly)
        ))
    poly = list(reversed(poly))
```

```
    bw = sum(poly[i - 1].x * poly[i].y for i in range(len(poly)
        ))
    return abs(fw - bw) / 2.0
def sort_ccw(p, points):
    return sorted(points, key=lambda point: atan2(point.y - p.y
        , point.x - p.x))
def find_face(neighbors, u, v):
    face, current, previous = [], v, u
    face.append(previous)
    while True:
        face.append(current)
        current_neighbors = neighbors[current]
        index = current_neighbors.index(previous)
        next_index = (index + 1) % len(current_neighbors)
        next_vertex = current_neighbors[next_index]
        if next_vertex == u: break
        previous, current = current, next_vertex
    face.append(u)
    return face
def find_outer_edge(coord):
    leftmost = min(coord, key=lambda p: (p.x, p.y))
    N_leftmost = coord[leftmost]
    sorted_neighbors = sorted(
        N_leftmost, key=lambda p: atan2(p.y-leftmost.y, p.x-
            leftmost.x))
    u = sorted_neighbors[0]
    return (leftmost, u)
n, coord = int(input()), defaultdict(list)
for l in range(n):
    x1, y1, x2, y2 = map(float, input().split())
    p1, p2 = Point(x1, y1), Point(x2, y2)
    coord[p1].append(p2); coord[p2].append(p1)
for p in coord:coord[p] = sort_ccw(p, coord[p])
seen = set()
p, q = find_outer_edge(coord)
outer = find_face(coord, p, q)
for i in range(len(outer)-1):seen.add((outer[i],outer[i+1%len(
    outer)]))
areas = []
for p in coord:
    for q in coord[p]:
        if (p, q) in seen: continue
        seen.add((p, q))
        face = find_face(coord, p, q)
        areas.append(shoelace(face) ** 2)
        for i in range(len(face) - 1):
            seen.add((face[i], face[i + 1 % len(face)]))
print(fsum(areas))
```

HalfPlaneIntersec.h

Description: Compute half-plane intersection
Time: $\mathcal{O}(N \log N)$

a31d0a, 42 lines

```
typedef Point<double> P;
const long double eps = 1e-9, inf = 1e9;
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the
        direction vector of the line.
    P p, pq; long double angle;
    Halfplane() {}
    Halfplane(const P& a, const P& b) : p(a), pq(b - a) { angle
        = atan2l(pq.y, pq.x); }
  bool operator < (const Halfplane& e) const { return angle < e
      .angle; }
    // Check if point 'r' is outside this half-plane. Every
        half-plane allows the region to the LEFT of its line.
    bool out(const P& r) { return pq.cross(r - p) < -eps; }
    // Intersection point of the lines of two half-planes. It
        is assumed they're never parallel.
    friend P inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = (t.p - s.p).cross(t.pq) / s.pq.
            cross(t.pq);
        return s.p + (s.pq * alpha);}};
vector<P> hp_intersect(vector<Halfplane>& H) {
    P box[4] = {P(inf,inf), P(-inf,inf), P(-inf,-inf), P(inf,-
        inf)};
    for(int i = 0; i<4; i++) {
        Halfplane aux(box[i], box[(i+1) % 4]); H.push_back(aux)
            ;}
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))
            ) {
            dq.pop_back(); --len;}
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front(); --len;}
        if (len > 0 && fabsl(H[i].pq.cross(dq[len-1].pq)) < eps
            ) {
            if (H[i].pq.dot(dq[len-1].pq) < 0.0) return vector<
                P>();
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back(); --len;}
            else continue;
        }
        dq.push_back(H[i]); ++len;}
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back(); --len; }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front(); --len; }
    if (len < 3) return vector<P>();
    vector<P> ret(len);
    for(int i = 0; i+1 < len; i++) ret[i]=inter(dq[i], dq[i+1])
        ;
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;}
```

UnionRect.h

Description: Compute union of rectangles
Usage: `vector<pair<pair<int,int>,pair<int,int>>> a(n);`
`union_rectangles<int,double>(a);`
Time: $\mathcal{O}(N \log N)$

55b17b, 51 lines

```
constexpr int msb(int n){return n==0?-1:31-__builtin_clz(n);}
constexpr int lsb(int n){return n==0?-1:__builtin_ctz(n);}
template<typename T,typename T2=T>
T2 union_rectangles(const vector<pair<pair<T,T>,pair<T,T>>>&
    recs){
  if(recs.empty())return 0;
  struct E{
    T x; int l,r; int add;
    bool operator<(const E&rhs)const{return x<rhs.x;}
  };
  vector<E>query; vector<T>zy;
  query.reserve(recs.size()*2);
  zy.reserve(recs.size()*2);
  for(const auto&[a,b]:recs)zy.push_back(a.second),zy.push_back
      (b.second);
  sort(zy.begin(),zy.end()),zy.erase(unique(zy.begin(),zy.end()
      ),zy.end());
  for(const auto&[a,b]:recs){
    int l=lower_bound(zy.begin(),zy.end(),a.second)-zy.begin();
    int r=lower_bound(zy.begin(),zy.end(),b.second)-zy.begin();
    query.push_back({a.first,l,r,1});
    query.push_back({b.first,l,r,-1});
  }
  sort(query.begin(),query.end());
  int sz=zy.size()-1;
  int z=sz<=1?1:1<<(msb(sz-1)+1);
  vector<int>mn(z*2);
  vector<T>cnt(z*2);
  for(int i=0;i<zy.size()-1;i++)cnt[i+z]=zy[i+1]-zy[i];
  for(int i=z-1;i>=1;i--)cnt[i]=cnt[i*2]+cnt[i*2+1];
  auto upd=[&](int i){
    i>>=(lsb(i)+1);
    while(i){
      if(mn[i*2]==mn[i*2+1]){
        mn[i]+=mn[i*2]; mn[i*2]=mn[i*2+1]=0;
        cnt[i]=cnt[i*2]+cnt[i*2+1];}
      else if(mn[i*2]<mn[i*2+1]){
        mn[i]+=mn[i*2]; mn[i*2+1]-=mn[i*2]; mn[i*2]=0;
        cnt[i]=cnt[i*2];}
      else{
        mn[i]+=mn[i*2+1]; mn[i*2]-=mn[i*2+1]; mn[i*2+1]=0;
        cnt[i]=cnt[i*2+1];}
      i>>=1;}};
  T2 res=0; T prex=query[0].x; T aly=zy.back()-zy.front();
  for(auto&[x,l,r,add]:query){
    res+=T2(x-prex)*(aly-(mn[1]==0?cnt[1]:0));
    prex=x; l+=z,r+=z;
    int l2=l,r2=r;
    while(l<r){
      if(l&1)mn[l++]+=add;
      if(r&1)mn[--r]+=add;
      l>>=1,r>>=1;}
    upd(l2),upd(r2);}
  return res;}
```

Strings (11)

Trie.py

Description: size 10^6 for 10^5 words, $5*10^5$ seems to work too
Time: $\mathcal{O}(N)$

171a35, 16 lines

```
trie, j = [-1]*int(27*1e6), 27
#insert list of words
for w in words:
    i = 0
    for char in w:
        c = ord(char) - 97
        if trie[i+c]==-1: trie[i+c]=j; j+=27
        i = trie[i + c]
    trie[i + 26] = 1
#check if w in trie
i = 0
for char in w:
    c = ord(char) - 97
    i = trie[i + c]
    if i == -1:  break
    if trie[i + 26] == 1: pass# this is a valid word
```

Z.py

Description: Z[i] is length of the longest substr starting from str[i] which is also a prefix of str[0..n-1] (i + Z[i] == n) => prefix suffix Si recherche de motif, concaténer motif devant la string if i + Z[i] == n and n % (i+1) == 0 : period = max(period, i+1)
Time: $\mathcal{O}(N)$

f8780f, 5 lines

```
Z, l, r = [0] * n, 0, 0
for i in range(1, n):
    if i <= r: Z[i] = min(Z[i-l],r-i+1)
    while i+Z[i]<n and s[Z[i]]==s[i + Z[i]]: Z[i] += 1
    if i + Z[i] - 1 > r: l, r = i, i + Z[i] - 1
```

Manacher.py

Description: Preprocess the string to insert '#' between each character to handle even-length palindromes
Time: $\mathcal{O}(N)$

fd0df0, 14 lines

```
def longest_pal_substr(s):
    t = '#'.join('^{}$'.format(s))
    n = len(t); p = [0] * n
    center, right = 0, 0
    for i in range(1, n - 1):
        if i < right:
            mirror = 2 * center - i
            p[i] = min(right - i, p[mirror])
        while t[i+p[i]+1] == t[i-p[i]-1]: p[i] += 1
        if i + p[i] > right: center, right = i, i + p[i]
    max_len = max(p)
    center_idx = p.index(max_len)
    start_idx = (center_idx-max_len)>>1
    return s[start_idx:start_idx+max_len]
```

StringFactoring.py

Description: ababc -> 2(ab)c
Time: $\mathcal{O}(N^3)$

76994d, 18 lines

```
def prefix(s):
    n, pi = len(s), [0]*n
    for i in range(1,n):
        j = pi[i - 1]
        while j > 0 and s[i] != s[j]: j=pi[j-1]
        if s[i] == s[j]: j += 1
        pi[i] = j
    return pi[-1]
dp = [[701] * (n+1) for _ in range(n+1)]
for l in range(1, n + 1):
    for i in range(n - l + 1):
        j = i + l - 1
        if i == j: dp[i][j] = 1; continue
        for k in range(i, j):
            dp[i][j] = min(dp[i][j],dp[i][k]+dp[k+1][j])
        pref = prefix(s[i : i + l])
        if l % (l - pref) == 0:
            dp[i][j] = min(dp[i][j], dp[i][i+l-pref-1])
```

RollingHash.py

Description: Rolling hashes, use M=1e9+7, P1=31, P2=37
Time: $\mathcal{O}(N)$

d3e8e3, 9 lines

```
def create_string_hash(string, n, p, mod):
    hash_p, powers = [0] * (n + 1), [1] * (n + 1)
    for i in range(1, n + 1): powers[i] = (powers[i - 1] * p) %
        mod
    for i in range(n): hash_p[i+1]=(hash_p[i]*p+(ord(string[i])
        -ord("a")+1))%mod
    return hash_p, powers

def get_hash(hash_p, powers, mod, i, j):
    # Returns the hash of the substring s[i:j] (j excluded)
    return (hash_p[j] - hash_p[i] * powers[j-i]) % mod
```

hash.py

Description: Compute hash of a number
Time: $\mathcal{O}(1)$

788c93, 21 lines

```
MOD=1<<64
def hash(x):
    x=((x^(x>>30))*0xbf58476d1ce4e5b9)%MOD
    x=((x^(x>>27))*0x94d049bb133111eb)%MOD
    x=(x^(x>>31))%MOD
    return x
def unhash(x):
    x=((x^(x>>31)^(x>>62))*0x319642b2d24d8ec3)%MOD
    x=((x^(x>>27)^(x>>54))*0x96de1b173f119089)%MOD
    x=(x^(x>>30)^(x>>60))%MOD
    return x
def hash2(x):
    x=((x^(x>>33))*0xff51afd7ed558ccd)%MOD
    x=((x^(x>>33))*0xc4ceb9fe1a85ec53)%MOD
    x=(x^(x>>33))%MOD
    return x
def unhash2(x):
    x=((x^(x>>33))*0x9cb4b2f8129337db)%MOD
    x=((x^(x>>33))*0x4f74430c22a54005)%MOD
    x=(x^(x>>33))%MOD
    return x
```

SuffixArray.py

Description: Compute SuffixArray and LCP Tested on Kattis: Suffix Array and LCP (longest common prefix) : Suffix Sorting, Stammering Aliens LCS (longest common substr) : Life Forms, Longest Common Substring LRS (longest repeadted substr) : Dvaput Unique Substrings : Repeated Substrings
Time: $\mathcal{O}(N)$

6181a8, 114 lines

```
def gen_sufarr_lcp(words):
    def radix_sort(p_, c_):
        count = [0 for _ in range(len(p_))]
        for x in p_: count[c_[x]] += 1
        new_p = p_.copy()
        pos = [0 for _ in range(len(p_))]
        for i in range(1, len(p_)): pos[i]=pos[i-1]+count[i-1]
        for x in p_: new_p[pos[c_[x]]]=x; pos[c_[x]]+=1
        return new_p
    s_, sep, sidx = [], 0, []
    for idx, word in enumerate(words):
        for char in word:
            s_.append(len(words)+ord(char)); sidx.append(idx)
        s_.append(sep); sidx.append(-1); sep += 1
    n = len(s_)
    a = [(s_[i], i) for i in range(n)]
    a.sort(key=lambda x: x[0])
    sufarr = [a[i][1] for i in range(n)]
    c = [0] * n; c[sufarr[0]] = 0
    for i in range(1, n):
        if a[i][0] == a[i-1][0]: c[sufarr[i]]=c[sufarr[i-1]]
        else: c[sufarr[i]] = c[sufarr[i-1]]+1
    k = 1
    while (1 << k) <= 2*n:
        sufarr = [(sufarr[i]-(1<<(k-1)))%n for i in range(n)]
        sufarr = radix_sort(sufarr, c)
        c_new = [0] * n
        prev = (c[sufarr[0]], c[(sufarr[0]+(1<<(k-1)))%n])
        for i in range(1, n):
            curr = (c[sufarr[i]], c[(sufarr[i]+(1<<(k-1)))%n])
            if prev == curr: c_new[sufarr[i]]=c_new[sufarr[i
                -1]]
            else: c_new[sufarr[i]]=c_new[sufarr[i-1]]+1
            prev = curr
        c = c_new; k += 1
        if c_new[sufarr[n-1]] == n - 1: break
    # sufarr is done generating
    lcp, k = [0] * (n-1), 0
    for i in range(n-1):
        pi = c[i]; j = sufarr[pi - 1]
        while s_[i+k] == s_[j+k]: k += 1
        lcp[pi-1] = k; k = max(k-1, 0)
    suf2s = [sidx[sufarr[i]] for i in range(len(words), n)]
    sufarr = sufarr[len(words):]; lcp = lcp[len(words)-1:]
    return sufarr, lcp, suf2s, s_

def longest_common_substring(S, T):
    L,A=len(S), S+"!"+T
    sa = suffix_array(A)
    lcp = lcp_array(A, sa)
    ans, best = [0]*4, 0
    for i in range(len(A)-1):
        if (sa[i]<L)==(sa[i+1]<L): continue
        if lcp[i]>best:
            best=lcp[i]
            if sa[i]<L:
                ans=[sa[i], sa[i]+best,
                    sa[i+1]-L-1, sa[i+1]-L-1+best]
            else:
                ans=[sa[i+1], sa[i+1]+best,
                    sa[i]-L-1, sa[i]-L-1+best]
    print(*ans)

def longest_common_substring(strings, k):
    # longest common substring such that k strings shares it
    n = len(strings)
    if n == 1:return len(strings[0]),[strings[0]]
    assert k > 1
    sufarr, lcp, suf2s, text = gen_sufarr_lcp(strings)
    st = SparseTable(lcp)
    l, sn = 0, [0] * n
    sn[suf2s[0]] += 1
    zeros, lcs, start_idx = n-1, 0, []
    for r in range(1, len(lcp)):
        if sn[suf2s[r]] == 0:zeros -= 1
        sn[suf2s[r]] += 1
        while (n - zeros) >= k:
            minimum = st.prod(l+1, r+1)
            if minimum > lcs:lcs = minimum; start_idx=[sufarr[r
                ]]
            elif minimum == lcs:start_idx.append(sufarr[r])
            sn[suf2s[l]] -= 1
            if sn[suf2s[l]] == 0:zeros += 1
            l += 1
    if lcs == 0: return 0, []
    substrings = sorted(''.join(chr(char - n) for char in text[
        st:st+lcs])
                    for st in start_idx)
    last, res = None, []
    for subs in substrings:
        if subs != last: res.append(subs)
        last = subs
    return lcs, res

def longest_repeated_substring(s):
    # overlap is OK : 'abrabra' -> 'abra'
    sufarr, lcp, _, _ = gen_sufarr_lcp([s])
    lrs = max(lcp)
    if lrs == 0: return 0, []
    substrings = []
    for i in range(len(lcp)):
        if lcp[i] == lrs:substrings.append(s[sufarr[i]:sufarr[i
            ]+lrs])
    substrings.sort()
    last, res = None, []
    for subs in substrings:
        if subs != last: res.append(subs)
        last = subs
    return lrs, res

def count_unique_substrings(s):
    _, lcp, _, _ = gen_sufarr_lcp([s])
    # return len(set(substrings of s))
    return (len(s)*(len(s)+1)>>1) - sum(lcp)
```

```python
def count_repeated_substrings(s):
    _, lcp, _, _ = gen_sufarr_lcp([s])
    return sum(max(lcp[i] - lcp[i-1], 0) for i in range(1, len(
         lcp)))
```

MinRotation.py

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

0beec2, 9 lines

```python
def minRotation(s):
    a,N=0,len(s)
    s+=s
    for b in range(N):
        for k in range(N):
            if a+k==b or s[a+k]<s[b+k]:
                b+=max(0,k-1);break
            if s[a+k]>s[b+k]:a=b;break
    return a
```

RK2d.h

Description: RabinKarp rolling hashes in 2d
Usage: RK2d$pattern, masterpiece, 31$
Time: $\mathcal{O}(N \cdot M)$

b5f2a2, 29 lines

```cpp
int RK2d(vector<string>&p, vector<string>&m, ll A) {
    int hp=sz(p),wp=sz(p[0]),hm=sz(m),wm=sz(m[0]);
    ll iA=Mod(A).invert(A).x;
    Mod rh=0;
    for(int i=0;i<hp;i++)
        for(int j=0;j<wp;j++)
            rh=rh+Mod(p[i][j])*(Mod(A)^(wp*i+j));
    vector<vector<Mod>>h1d(hm,vector<Mod>());
    for(int i=0;i<hm;i++) {
        Mod ch=0;
        for(int j=0;j<wp;j++)ch=ch+Mod(m[i][j])*(Mod(A)^j);
        h1d[i].push_back(ch);
        for(int j=wp;j<wm;j++) {
            ch=(ch-Mod(m[i][j-wp]))*Mod(iA)+\
            Mod(m[i][j])*(Mod(A)^(wp-1));
            h1d[i].push_back(ch);}}
    vector<vector<Mod>>h2d(hm-hp+1,vector<Mod>(wm-wp+1,Mod(0)))
         ;
    for(int j=wp;j<=wm;j++) {
        Mod ch=0;
        for(int i=0;i<hp;i++)ch=ch+h1d[i][j-wp]*(Mod(A)^(i*wp))
             ;
        h2d[0][j-wp]=ch;}
    for(int i=hp;i<hm;i++)for(int j=wp;j<=wm;j++)
        h2d[i-hp+1][j-wp]=(h2d[i-hp][j-wp]-h1d[i-hp][j-wp])*\
        (Mod(iA)^wp)+h1d[i][j-wp]*(Mod(A)^(wp*(hp-1)));
    int mt=0;
    for(int i=0;i<=hm-hp;i++)
        for(int j=0;j<=wm-wp;j++)if(h2d[i][j]==rh)mt++;
    return mt;
}
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll($-$, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

```cpp
struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

WPM.h

Description: Wildcard Pattern Matching Given strings S and T consisting of lowercase English letters and asterisks (*) wildcards, returns W where W[i] = 1 iff S[i:i+—T—] and T are matched, 0 otherwise S : abc*b*a***a, T : *b*a, W : 10111011
Time: approx. $\mathcal{O}(N)$

<tr2/dynamic_bitset> ed2d6f, 24 lines

```cpp
using namespace std::tr2;
void WPM(string s, string t){
    nt n = s.length(), m = t.length();
    vector<dynamic_bitset<>> b(26, dynamic_bitset<>(n));
    for(int i = 0; i < n; i ++){
        if(s[i] != '*')
            b[s[i] - 'a'][i] = true;
        else
            for(int c = 0; c < 26; c ++)
                b[c][i] = true;
    }
    vector<int> shift(26, 0);
    dynamic_bitset<> good(n);
    good.set();
    for(int i = 0; i < m; i ++){
        if(t[i] == '*') continue;
        b[t[i] - 'a'] >>= (i - shift[t[i] - 'a']);
        shift[t[i] - 'a'] = i;
        good &= (b[t[i] - 'a']);
    }
    for(int i = 0; i < n - m + 1; i ++)
        cout << good[i];
    cout << endl;
}
```

Various (12)

Grundy.py

Description: Computes the Minimum EXcluded for a given state. To convert to iterative version
Time: $\mathcal{O}(N \cdot M)$, N number of states, M average number of moves per state.

c4e4e9, 14 lines

```python
n = 1000
mex = [None] * (n+1)
mex[0] = 0; mex[1] = 1
def grundyValue(n):
    if mex[n] is not None:
        return mex[n]
    excluded = {grundyValue(n-2)}
    for i in range(2, n):
        excluded.add(grundyValue(i-2) ^ grundyValue(n-i-1))
    res = 0
    while res in excluded:
        res += 1
    mex[n] = res
    return res
```

Floyd.py

Description: Cycle detection
Time: $\mathcal{O}(N + M)$

fb7c43, 16 lines

```python
def floyd(x0, f):
    t = f(x0)
    h = f(f(x0))
    while t != h:
        t = f(t)
        h = f(f(h))
    start, t = 0, x0
    while t != h:
        t = f(t)
```

```
        h = f(h)
        start += 1
    period, h = 1, f(t)
    while t != h:
        h = f(h)
        period += 1
    return start, period
```

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

edce47, 22 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

9e9d8d, 19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

753a4c, 19 lines

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

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

9155b4, 11 lines

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

bootstrap.py

Description: Décorateur fonction récursive. Remplacer return par yield et appels rec par (yield ...). Yield None en fin de fonction.

9201e3, 17 lines

```
from types import GeneratorType
def bootstrap(f, stack=[]):
    def wrappedfunc(*args, **kwargs):
        if stack:
            return f(*args, **kwargs)
        else:
            to = f(*args, **kwargs)
            while True:
                if type(to) is GeneratorType:
                    stack.append(to)
                    to = next(to)
                else:
                    stack.pop()
                    if not stack: break
                    to = stack[-1].send(to)
            return to
    return wrappedfunc
```

12.1 Debugging tricks

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

12.2 Optimization tricks

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

12.2.1 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))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

12.2.2 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).

BumpAllocator.h

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

745db2, 8 lines

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