



International Institute Of Information Technology - Hyderabad

# tesla\_protocol

Anurudh Peduri, Arjun P, Devansh Gautam

44th ICPC World Finals 2020

October 5, 2021

1	Contest	1
2	Combinatorial	1
2.1	Permutations	1
2.2	Partitions and subsets	1
2.3	General purpose numbers	2
2.4	DP Optimizations	2
3	Numerical	3
3.1	Polynomials and recurrences	3
3.2	Optimization	3
3.3	Matrices	4
3.4	Fourier transforms	5
4	Number theory	6
4.1	Modular arithmetic	6
4.2	Primality	7
4.3	Divisibility	7
4.4	Fractions	8
4.5	Pythagorean Triples	8
4.6	Primes	8
4.7	Estimates	9
4.8	Mobius Function	9
5	Data structures	10
5.1	Set and Map like	10
5.2	Matrix	11
5.3	Range DS	11
6	Strings	12
6.1	String Matching	12
6.2	Palindromes	13
6.3	Suffix DS	14
7	Graph	15
7.1	Network flow	15
7.2	Matching	15
7.3	DFS algorithms	15
7.4	Coloring	15
7.5	Heuristics	15
7.6	Trees	17
7.7	Math	17
8	Geometry	17
8.1	Geometric primitives	17
8.2	Circles	18
8.3	Polygons	19
8.4	Misc. Point Set Problems	20
8.5	3D	21
9	Mathematics	21

9.1	Equations	21
9.2	Recurrences	21
9.3	Trigonometry	22
9.4	Geometry	22
9.5	Derivatives/Integrals	22
9.6	Sums	22
9.7	Series	22
9.8	Probability theory	22
9.9	Markov chains	23
10	Miscellaneous	23
10.1	RNG, Intervals, Ternary Search	23
10.2	Debugging tricks	23
10.3	Optimization tricks	24

## Contest (1)

```
template.cpp27 lines
#include <bits/stdc++.h>
using namespace std;

#define REP(i, a, b) for(int i = (a); i < (b); ++i)
#define ALL(x) begin(x), end(x)
#define SZ(x) ((int)(x).size())
#define SET(a,v) memset((a), (v), sizeof(a))
#define PB push_back
#define EB emplace_back
#define MP make_pair
#define fst first
#define snd second
using ll = long long;
using db = double;
using PII = pair<int, int>;
using VI = vector<int>;

int main() {
#ifdef LOCAL_EXEC
    // freopen("sample.in", "r", stdin);
    // freopen("sample.out", "w", stdout);
#else
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
#endif
    return 0;
}
```

```
CMakeLists.txt18 lines
cmake_minimum_required(VERSION 3.15)
project(contest)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS, "-std=c++17 -Wall -pedantic -Wconversion \
-Wshadow -Wfloat-equal")
if (DEBUG EXEC) # set in 'Debug'
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O0 -g -fsanitize=
address,undefined")
else () # in 'Run' and 'OJ'
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2")
endif()
if (LOCAL_EXEC) # set in 'Debug' and 'Run'
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DLOCAL_EXEC")
endif()
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ..) # folder with source
```

set(srcs a b c d e f g h i j k l m) # adjust names as necessary  
foreach(F IN LISTS srcs)  
 add\_executable(\${F}.out \${F}.cpp)  
endforeach()

CLion setup:

- File → New Project → C++ executable
- File → Settings → Keymap → Sublime Text
- Settings → Build... → CMake
- Add profiles: Debug, Run, OJ; use default toolchain.

Sublime Keybindings

- Add Selection for Next Occurrence: Ctrl-D
- Move Line Up: Ctrl-Shift-UP
- Move Line Down: Ctrl-Shift-DOWN
- Delete Line: Ctrl-Shift-K
- Duplicate Line or Selection: Ctrl-Shift-D
- Start New Line: Ctrl+Enter
- Start New Line Before Current: Ctrl+Shift+Enter
- Move Caret to Matching Brace: Ctrl-M
- Terminal: Ctrl-T

.vimrc6 lines  
set cin aw ai is ts=2 sw=2 tm=50 nu noeb bg=dark ru cul  
sy on | im jk <esc>  
" Select region and then type :Hash to hash your selection.  
" Useful for verifying that there aren't mistypes.  
ca Hash w !cpp -dD -P -fpreprocessed \ | tr -d '[:space:]' \  
 \ | md5sum \ | cut -c-6

hash.sh3 lines  
# Hashes a file, ignoring all whitespace and comments. Use for  
# verifying that code was correctly typed.  
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6

Combinatorial (2)

2.1 Permutations

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

```
int permToInt(VI& v) {
    int use = 0, i = 0, r = 0;
    for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
        use |= 1 << x;
    return r;
}
```

261010, 6 lines

### 2.1.2 Cycles

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

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

### 2.1.3 Derangements

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

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

### 2.1.4 Burnside’s lemma

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

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

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

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

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

## 2.2 Partitions and subsets

### 2.2.1 Partition function

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

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

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

$n$	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2\text{e}5$	$\sim 2\text{e}8$

### 2.2.2 Lucas’ Theorem

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

### 2.2.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1+\dots+k_n}{k_1,k_2,\dots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\dots k_n!}$ .

```
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;
}
```

037a49, 6 lines

## 2.3 General purpose numbers

### 2.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t-1}$  (FFT-able).

$$B[0,\dots] = [1, -\tfrac{1}{2}, \tfrac{1}{6}, 0, -\tfrac{1}{30}, 0, \tfrac{1}{42}, \dots]$$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

### 2.3.2 Stirling numbers of the first kind

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

$$\begin{aligned} c(n,k) &= c(n-1,k-1) + (n-1)c(n-1,k), \; c(0,0) = 1 \\ \sum_{k=0}^n c(n,k) x^k &= x(x+1) \dots (x+n-1) \end{aligned}$$

$$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$$

$$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$$

### 2.3.3 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.  $\pi(j) > j$ .

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

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

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

### 2.3.4 Stirling numbers of the second kind

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

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

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

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

### 2.3.5 Bell numbers

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

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

### 2.3.6 Labeled unrooted trees

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

### 2.3.7 Catalan numbers

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

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

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

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

## 2.4 DP Optimizations

1D-1D.h

**Description:** Applicable if  $dp_i = \min_{j>i} (dp_j + cost(i,j))$  s.t.  $opt_i \leq opt_j$  when  $i \leq j$  (which holds if quadrangle)  
**Time:** FIXME

```
#define until first
#define opt second
ll dp[100000];
ll cost(int i, int j) {
    return dp[j] /* + cost to jump from i to j */;
}
void solve(int n) {
    dp[n] = 0;
    vector<PII> v;
    v.EB(n-1, n);
    for (int i = n-1, ipos = 0; i >= 1; i--) {
        while (ipos+1 < SZ(v) && i <= v[ipos+1].until) ipos++;
        dp[i] = cost(i, v[ipos].opt);
        while (v.back().until < i
```

d76777, 27 lines

```
&& cost(v.back().until, i)
    <= cost(v.back().until, v.back().opt))
{ v.pop_back(); }
int l = 1, r = min(i - 1, v.back().until);
while (l <= r) {
    int mid = (l + r)/2;
    if (cost(mid, i) <= cost(mid, v.back().opt)) {
        l = mid + 1;
    } else { r = mid - 1; }
}
if (l - 1 >= 1) v.EB(l - 1, i);
}
}
```

### Divide-and-Conquer.h

**Description:** Works when  $dp_{k,i} = \min_{j < i}(dp_{k-1,j} + cost(j,i))$  and  $opt_k(i) \leq opt_k(i+1)$ . (This holds when quadrangle)  
**Usage:** find dp[1], then: for(i = 2 to n) solve(i, 1, n, 1, n)  
**Time:** FIXME

1db0cf, 15 lines

```
ll dp[100][100]; // set correctly
ll cost(int i, int j); // cost to go from i to j, 1-indexed.
void solve(int i, int l, int r, int optl, int optr) {
    const ll inf = 1e18; // set correctly
    if (l > r || optl > optr) return;
    int mid = (l + r)/2; pair<ll, int> best = {inf, -1};
    for (int j = optl; j <= min(mid, optr); j++) {
        pair<ll,int> cand(dp[i - 1][j] + cost(j, mid), j);
        if (best.second == -1) best = cand;
        else best = min(best, cand);
    }
    dp[i][mid] = best.first;
    solve(i, l, mid - 1, optl, best.second);
    solve(i, mid + 1, r, best.second, optr);
}
```

### Li-Chao.h

**Description:** FIXME  
**Time:** FIXME

b35345, 19 lines

```
int n, tree[1 << 16]; // set limits correctly
int eval(int f, int x); // evaluate f(x)
// Add func f to set which crosses every other func atmost once
void update(int f, int root = 1, int s = 1, int e = n) {
    int mid = (s + e)/2;
    bool lless = eval(f, s) < eval(tree[root], s);
    bool mless = eval(f, mid) < eval(tree[root], mid);
    if (mless) swap(f, tree[root]);
    if (s == e) return;
    if (lless != mless) update(f, 2*root, s, mid);
    else update(f, 2*root + 1, mid + 1, e);
}
// Evaluate mininum at x over all functions.
ll query(int x, int root = 1, int s = 1, int e = n) {
    int mid = (s + e)/2; ll ans = eval(tree[root], x);
    if (s == e) return ans;
    if (x <= mid) return min(query(x, 2*root, s, mid), ans);
    return min(query(x, 2*root + 1, mid + 1, e), ans);
}
```

### Dynamic-CHT.h

**Description:** FIXME  
**Time:** FIXME

f03adc, 55 lines

```
const dbl INF = 1e16;
struct HullDynamic {
#define CLREF const Line&
    struct Line {
        int a; ll b, val=0; dbl xLeft = -INF; bool type=0;
```

```
Line(int a = 0, ll b = 0): a(a), b(b) {}
ll eval(int x) const{ return a * 1ll * x + b; }
bool operator< (CLREF l2) const {
    return l2.type ? (xLeft > l2.val) : (a < l2.a);
}
};
using Iter = set<Line>::iterator;
bool parallel(CLREF l1, CLREF l2) { return l1.a == l2.a; }
dbl meetX(CLREF l1, CLREF l2) {
    return parallel(l1, l2) ? INF :
        (l2.b-l1.b) / (dbl)(l1.a-l2.a);
}
set<Line> hull;
bool hasPrev(Iter it) { return it != hull.begin(); }
bool hasNext(Iter it) {
    return it != hull.end() && next(it) != hull.end();
}
bool bad(CLREF l1, CLREF l2, CLREF l3){
    return meetX(l1,l3) <= meetX(l1,l2);
}
bool bad(Iter it) {
    return hasPrev(it) && hasNext(it)
        && (bad(*next(it), *it, *prev(it)));
}
Iter upd_left_border(Iter it) {
    if(!hasNext(it)) return it;
    dbl val = meetX(*it, *next(it));
    Line buf(*it); it = hull.erase(it);
    buf.xLeft = val; return hull.insert(it, buf);
}
void insert_line(int a, ll b) {
    Line l3 = Line(a, b); auto it = hull.lower_bound(l3);
    if (it != hull.end() && parallel(*it , l3)) {
        if (it->b <= b) return;
        it = hull.erase(it);
    }
    it = hull.insert(it, l3);
    if (bad(it)) { hull.erase(it); return; }
    while (hasPrev(it) && bad(prev(it))) hull.erase(prev(it));
    while (hasNext(it) && bad(next(it))) hull.erase(next(it));
    it = upd_left_border(it);
    if (hasPrev(it)) upd_left_border(prev(it));
    if (hasNext(it)) upd_left_border(next(it));
}
ll eval(int x) {
    Line q; q.val = x; q.type = 1;
    auto best = hull.lower_bound(q);
    return (best == hull.end()) ? INF : best->eval(x);
}
};
```

### KnuthDP.h

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

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

## 3.1 Polynomials and recurrences

### Polynomial.h

2ad8d0, 17 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();
    }
};
```

### PolyRoots.h

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

7118fe, 23 lines

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

### PolyInterpolate.h

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

a4f803, 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;
}
```

## Numerical (3)

```
}

// ...
}

```

### BerlekampMassey.h

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

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

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

"/>

```
vector<ll> berlekampMassey(vector<ll> s) {
    int n = SZ(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    REP(i,0,n) { ++m;
        ll d = s[i] % mod;
        REP(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        REP(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }
}
```

```
C.resize(L + 1); C.erase(C.begin());
for (ll& x : C) x = (mod - x) % mod;
return C;
}
```

### LinearRecurrence.h

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

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

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

```
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;
}
```

## 3.2 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. The maximum error in the result is  $\epsilon$ . Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

**Usage:** `double func(double x) { return 4+x+.3*x*x; }`

`double xmin = gss(-1000,1000,func);`

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

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

### HillClimbing.h

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

**typedef** array<double, 2> P;

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

### Integrate.h

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

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

### IntegrateAdaptive.h

**Description:** Fast integration using an adaptive Simpson's rule.

**Usage:** `double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x*x + y*y + z*z < 1; }}});});`

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
```

```
template <class F>
```

```
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

### Simplex.h

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

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

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

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

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

**typedef** double T; // long double, Rational, double + mod<P>...

**typedef** vector<T> vd;

**typedef** vector<vd> vvd;

**const** T eps = 1e-8, inf = 1/.0;

**#define** MP make\_pair

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

**struct** LPSolver {

int m, n;

VI N, B;

vvd D;

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

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

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

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

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

### 3.3 Matrices

#### Determinant.h

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

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

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

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

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

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

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

#### SolveLinear2.h

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

```
"SolveLinear.h"
5cad07, 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:**  $\mathcal{O}(n^2m)$

```
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  $A \bmod p$ , and  $k$  is doubled in each step.  
**Time:**  $\mathcal{O}(n^3)$

```
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

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

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

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

Fails if the solution is not unique.

If  $|d_i| > |p_i| + |q_{i-1}|$  for all  $i$ , or  $|d_i| > |p_{i-1}| + |q_i|$ , or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.

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

a74eda, 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;
}
```

3.4 Fourier transforms

FastFourierTransform.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  $N = 2^{22}$ )

a333b0, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = SZ(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L)) / k;
```

```
        REP(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    }
    VI rev(n);
    REP(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    REP(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) REP(j,0,k) {
            C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(SZ(a) + SZ(b) - 1);
    int L = 32 - __builtin_clz(SZ(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(ALL(a), begin(in));
    REP(i,0,SZ(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    REP(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    REP(i,0,SZ(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod})$ .

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

"FastFourierTransform.h" 860295, 22 lines

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

NumberTheoreticTransform.h

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

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

"../number-theory/ModPow.h" 14d0bb, 33 lines

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
```

```
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = SZ(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        REP(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    VI rev(n);
    REP(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    REP(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) REP(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = SZ(a) + SZ(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    REP(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

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

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

b1dc68, 16 lines

```
void FST(VI& a, bool inv) {
    for (int n = SZ(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) REP(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? PII(v - u, u) : PII(v, u + v); // AND
                inv ? PII(v, u - v) : PII(u + v, u); // OR
                PII(u + v, u - v); // XOR
        }
    }
    if (inv) for (int& x : a) x /= SZ(a); // XOR only
}
VI conv(VI a, VI b) {
    FST(a, 0); FST(b, 0);
    REP(i,0,SZ(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

WalshHadamard.h

**Description:**  $C_k = \sum_{i \otimes j = k} A_i B_j$

**Usage:** Apply the transform, point multiply and invert

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

905e71, 11 lines

```
void WalshHadamard(poly &P, bool invert) {
    for (int len = 1; 2 * len <= SZ(P); len <= 1) {
        for (int i = 0; i < SZ(P); i += 2 * len) {
            REP(j, 0, len) {
                auto u = P[i + j], v = P[i + len + j];
                P[i + j] = u + v, P[i + len + j] = u - v; // XOR
```

```
    }
    }
}
if (invert) for (auto &x : P) x /= SZ(P);
}
```

OnlineFFT.h  
Description: Given  $B_1, \dots B_m$ , compute  $A_i = \sum_{j=1}^{i-1} A_j * B_{i-j}$   
Usage: 1-indexed, pad B[i] = 0 for i > m  
Time:  $\mathcal{O}(N \log^2 N)$

```
void online(const Poly &B, CD a1, int n, Poly &A) {
    const int m = SZ(B) - 1;
    A.assign(n + 1, 0); A[1] = a1;
    auto bst = B.begin(), ast = A.begin();
    REP(i, 1, n) {
        A[i + 1] += A[i] * B[1];
        if (i + 2 <= n) A[i + 2] += A[i] * B[2];
        for (int pw = 2; i % pw == 0 && pw + 1 <= m; pw <= 1) {
            Poly blockA(ast + i - pw, ast + i);
            Poly blockB(bst + pw + 1, bst + min(pw * 2, m) + 1);
            Poly prod = conv(blockA, blockB);
            REP(j, 0, SZ(prod)) {
                if (i + 1 + j <= n)
                    A[i + 1 + j] += prod[j];
            }
        }
    }
}
```

## Number theory (4)

### 4.1 Modular arithmetic

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

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

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

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
REP(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

```
ModPow.h
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;
}
```

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

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

ModSum.h  
Description: Sums of mod'ed arithmetic progressions.  
 $\text{modsum}(\text{to}, c, k, m) = \sum_{i=0}^{\text{to}-1} (ki + c) \% m$ . divsum is similar but for floored division.  
Time:  $\log(m)$ , with a large constant.

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

ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
```

```
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h  
Description: Calculate  $a \cdot b \pmod c$  (or  $a^b \pmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .  
Time:  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow

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

ModSqrt.h  
Description: Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 = a \pmod p$  ( $-x$  gives the other solution).  
Time:  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$   
"ModPow.h"

```
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = modpow(a, (s + 1) / 2, p);
    ll b = modpow(a, s, p), g = modpow(n, s, p);
    for (; r = m) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
        if (m == 0) return x;
        ll gs = modpow(g, 1LL << (r - m - 1), p);
        g = gs * gs % p;
        x = x * gs % p;
        b = b * g % p;
    }
}
```

### 4.2 Primality

FastEratosthenes.h  
Description: Prime sieve for generating all primes smaller than LIM.  
Time: LIM=1e9 ≈ 1.5s

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

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

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



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

Factor.h

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

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

```
"ModMulLL.h", "MillerRabin.h"
65f857, 18 lines

ull pollard(ull n) {
    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), ALL(r));
    return l;
}
```

4.3 Divisibility

```
euclid.h
Description: Finds two integers x and y, such that ax + by = gcd(a,b). If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of a (mod 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;
}
```

CRT.h

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

```
"euclid.h"
04d93a, 7 lines

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

4.3.1 Bézout’s identity

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

$ax + by = d$

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

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

phiFunction.h

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

```
const int LIM = 5000000;
int phi[LIM];
void calculatePhi() {
    REP(i,0,LIM) phi[i] = i&1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

4.4 Fractions

```
ContinuedFractions.h
Description: Given N and a real number x ≥ 0, finds the closest rational approximation p/q with p,q ≤ N. It will obey |p/q − x| ≤ 1/qN.
For consecutive convergents, pk+1qk − qk+1pk = (−1)k. (pk/qk alternates between > x and < x.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a’s eventually become cyclic.
Time: O(log N)
dd6c5e, 21 lines

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

FracBinarySearch.h

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

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
```

```
if (f(lo)) return lo;
assert(f(hi));
while (A || B) {
    ll adv = 0, step = 1; // move hi if dir, else lo
    for (int si = 0; step; (step *= 2) >= si) {
        adv += step;
        Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
        if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
            adv -= step; si = 2;
        }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !adv;
}
return dir ? hi : lo;
}
```

4.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

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

4.7 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  $n < 1e19$ .

4.8 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)

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

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

Mobius.h

Description: Dirichlet -  $H(n) = \sum_{x,y=n} a_x b_y, 1 \leq n \leq N$

306cf1, 21 lines

```
VI mobius(int N) {
    VI mu(N + 1, 1);
    vector<bool> ispr(N + 1, 1);
    for (int i = 2; i <= N; ++i) {
        if (!ispr[i]) continue;
        for (int j = i; j <= N; j += i) {
            ispr[j] = 0;
            mu[j] *= -1;
        }
        if (i * 1ll * i > N) continue;
        for (int j = i * i, ii = i * i; j <= N; j += ii)
            mu[j] = 0;
    }
    return mu;
}

VI DirichletConvolution(const VI &a, const VI &b, int N) {
    VI h(N + 1, 0);
    REP(i, 1, N + 1)
        for (int j = i; j <= N; j += i) h[j] += a[i] * b[j / i];
    return h;
}
```

## Data structures (5)

### 5.1 Set and Map like

OrderStatisticTree.h

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

782797, 16 lines

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

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

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

HashMap.h

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

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 1ll(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({}, {}, {}, {}, {1<16});
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data. Time:  $\mathcal{O}(\log N)$

9556fc, 55 lines

```
struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }
}

Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

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

Sec1c7, 30 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;
    }
};
```

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

f79afc, 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;
    }
};
```

### 5.2 Matrix

SubMatrix.h

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

fe231e, 13 lines

```
template<class T>
struct SubMatrix {
    vector<vector<T>> p;
    SubMatrix(vector<vector<T>>& v) {
        int R = SZ(v), C = SZ(v[0]);
        p.assign(R+1, vector<T>(C+1));
        REP(r,0,R) REP(c,0,C)
            p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
    }
};
```

```
    }
    T sum(int u, int l, int d, int r) {
        return p[d][r] - p[d][l] - p[u][r] + p[u][l];
    }
};
```

Matrix.h

**Description:** Basic operations on square matrices.

**Usage:** Matrix<int, 3> A;

A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};

vector<int> vec = {1,2,3};

vec = (A^N) \* vec;

d8e33f, 26 lines

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

### 5.3 Range DS

FenwickTree.h

**Description:** Computes partial sums  $a[0] + a[1] + \dots + a[pos - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.

**Time:** Both operations are  $\mathcal{O}(\log N)$ .

b06af0, 22 lines

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

FenwickTree2d.h

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

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

"FenwickTree.h" d5c1b7, 22 lines

```
struct FT2 {
    vector<VI> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < SZ(ys); x |= x + 1) ys[x].push_back(y);
    }
    void init() {
        for (VI& v : ys) sort(ALL(v)), ft.emplace_back(SZ(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(ALL(ys[x]), y) - ys[x].begin()); }
    void update(int x, int y, ll dif) {
        for (; x < SZ(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }
};
```

RMQ.h

**Description:** Range Minimum Queries on an array. Returns  $\min(V[a], V[a + 1], \dots, V[b - 1])$  in constant time.

**Usage:** RMQ rmq(values);

`rmq.query(inclusive, exclusive);`

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

9a1bbf, 16 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= SZ(V); pw *= 2, ++k) {
            jmp.emplace_back(SZ(V) - pw * 2 + 1);
            REP(j,0,SZ(jmp[k]))
                jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
```

MoQueries.h

**Description:** Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge  $(a, c)$  and remove the initial `add` call (but keep `in`).

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

3015c5, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
```

```
VI mo(vector<PII> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
    VI s(SZ(Q)), res = s;
    #define K(x) PII(x.first/blk, x.second ^ -(x.first/blk & 1))
    iota(ALL(s), 0);
```

```
sort(ALL(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
for (int qi : s) {
    PII q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
}
return res;
}
```

```
VI moTree(vector<array<int, 2>> Q, vector<VI>& ed, int root=0){
    int N = SZ(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
    VI s(SZ(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
    add(0, 0), in[0] = 1;
    auto dfs = [&](int x, int p, int dep, auto& f) -> void {
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
        if (!dep) I[x] = N++;
        R[x] = N;
    };
    dfs(root, -1, 0, dfs);
    #define K(x) PII(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
    iota(ALL(s), 0);
    sort(ALL(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) REP(end,0,2) {
        int &a = pos[end], b = Q[qi][end], i = 0;
    #define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                    else { add(c, end); in[c] = 1; } a = c; }
        while (!L[b] <= L[a] && R[a] <= R[b])
            I[i++] = b, b = par[b];
        while (a != b) step(par[a]);
        while (i--) step(I[i]);
        if (end) res[qi] = calc();
    }
    return res;
}
```

MoWithUpdates.h

**Description:** Supports point updates at position

**Time:**  $\mathcal{O}(n^{5/3})$  when  $\text{block} = n^{2/3}$

303d07, 17 lines

```
struct Query { int l, r, id, t; }
struct Update { int pos, pre, now; };
void MoWithUpdates(vector<Query> qs, vector<Update> upd) {
    int BLK; // set block size
    sort(qs.begin(), qs.end(), [&](Query a, Query b) {
        return {a.l/BLK, a.r/BLK, a.t} < {b.l/BLK, b.r/BLK, b.t};
    });
    for (auto q : qs) {
        while (t < q.t) ++t, apply(upd[t].pos, upd[t].now);
        while (t > q.t) apply(upd[t].pos, upd[t].pre), --t;
        while (l > q.l) add(--l);
        while (l < q.l) remove(l++);
        while (r < q.r) add(++r);
        while (r > q.r) remove(r--);
        ans[q.id] = get();
    }
}
```

SegmentTreeBeats.h

**Description:** example below - range sum query and two range updates:

$a_i \leftarrow a_i \bmod x$  and  $a_i \leftarrow x$

d3ae07, 13 lines

```
bool break_condition() // when can we break
// eg. (l > rr || r < ll || max_val[node] < x)
```

```
bool tag_condition(); // when can we put tag for lazy update
// eg. (l >= ll && r <= rr && max_val[node] == min_val[node])
void modify(int node, int l, int r, int ll, int rr) {
    if (break_condition()) return;
    if (tag_condition()) { puttag(node); return; }
    pushdown(node);
    int mid = (l + r) >> 1;
    modify(node * 2, l, mid, ll, rr);
    modify(node * 2 + 1, mid + 1, r, ll, rr);
    update();
}
```

## Strings (6)

### 6.1 String Matching

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

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

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

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

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

**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  $\mathcal{O}(NM)$ .
f7553c, 66 lines

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;
}
};
```

```
Hashing.h
Description: Self-explanatory methods for string hashing.
6bb778, 44 lines

// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
    typedef uint64_t ull;
    ull x; H(ull x=0) : x(x) {}
#define OP(0,A,B) H operator 0(H o) { ull r = x; asm \
    (A "addq %rdx, %0\n adcq $0,%0" : "+a"(r) : B); return r; }
    OP(+,,"d"(o.x)) OP(*,"mul %l\n", "r"(o.x) : "rdx")
    H operator-(H o) { return *this + ~o.x; }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (1l)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(SZ(str)+1), pw(ha) {
        pw[0] = 1;
        REP(i,0,SZ(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (SZ(str) < length) return {};
    H h = 0, pw = 1;
    REP(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    REP(i,length,SZ(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

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

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

### 6.2 Palindromes

**Manacher.h**  
**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).  
**Time:**  $\mathcal{O}(N)$

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

## PalindromicTree.h

Description: FIXME

Time: FIXME c514c5, 47 lines

```
const int SIGMA = 256;
struct node { // use suff link, smart link after copying
    int suf_lnk, len, next[SIGMA], smt_lnk[SIGMA];
    node() : suf_lnk(0), len(0) { SET(next, -1), SET(smt_lnk, 0); }
};
struct eertree {
    int rto, rte, n, last;
    VI s; vector<node> tree;
    eertree() : rto(0), rte(1), n(0), last(1), s(1, -1), tree(2) {
        tree[rto].suf_lnk = tree[rte].suf_lnk = rto;
        tree[rto].len = -1;
        tree[rte].len = 0;
    }
    int add(int c) {
        s.PB(c);
        n++;
        if (s[n - tree[last].len - 1] != c) {
            last = tree[last].smt_lnk[c];
        }
        int flag = (tree[last].next[c] == -1);
        if (flag) {
            int idx = SZ(tree);
            tree.PB(node());
            tree[idx].len = tree[last].len + 2;
            if (tree[idx].len == 1) {
                tree[idx].suf_lnk = rte;
            } else {
                tree[idx].suf_lnk = tree[tree[last].smt_lnk[c]].next[c];
            }
        }
        if (tree[idx].len == 1) {
            REP(cc, 0, SIGMA) { tree[idx].smt_lnk[cc] = 0; }
            tree[idx].smt_lnk[c] = 1;
        } else {
            REP(cc, 0, SIGMA) {
                int x = tree[idx].suf_lnk;
                if (s[n - tree[x].len] == cc)
                    tree[idx].smt_lnk[cc] = x;
                else
                    tree[idx].smt_lnk[cc] = tree[x].smt_lnk[cc];
            }
        }
        tree[last].next[c] = idx;
    }
    last = tree[last].next[c];
    return tree[last].len;
}
```

## 6.3 Suffix DS

### SuffixArray.h

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

**Time:**  $\mathcal{O}(n \log n)$  3e14f8, 23 lines

```
struct SuffixArray {
    VI sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = SZ(s) + 1, k = 0, a, b;
        VI x(ALL(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(ALL(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(ALL(y), n - j);
            REP(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(ALL(ws), 0);
            REP(i,0,n) ws[x[i]]++;
            REP(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            REP(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        REP(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i+]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};
```

### SuffixTree.h

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

**Time:**  $\mathcal{O}(26N)$  5e69b7, 50 lines

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

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
            q=r[v]-(q-r[m]); m+=2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r,r+N,SZ(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
    }
};
```

```
fill(t[1],t[1]+ALPHA,0);
s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
REP(i,0,SZ(a)) ukkadd(i, toi(a[i]));
}

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

SuffixAutomaton.h
Description: Each path in the automaton is a substring (if it ends in a terminal node, it is a suffix) And no. of occurrences = no. of ways to reach a terminal node. Or keep reverse edges of suffix links(all prefixes for that substring), then no. of ways to reach a root.
Time:  $\mathcal{O}(len)$  map accesses, map can be at most of size alphabet, can also use unordered_map 37fe84, 34 lines

struct SuffixAutomaton {
    vector<map<char, int>> edges;
    VI link, length; // length[i]: longest string in i-th class
    int last; // index of equivalence class of whole string
    SuffixAutomaton(string s) : edges{}, link{-1}, length{0}, last(0) {
        edges.emplace_back();
        REP(i, 0, SZ(s)) {
            edges.emplace_back();
            length.push_back(i + 1);
            link.push_back(0);
            int r = SZ(edges) - 1, p = last;
            while (p >= 0 && edges[p].find(s[i]) == edges[p].end()) {
                edges[p][s[i]] = r, p = link[p];
            }
            if (p != -1) {
                const int q = edges[p][s[i]];
                if (length[p] + 1 == length[q]) link[r] = q;
                else {
                    edges.push_back(edges[q]);
                    length.push_back(length[p] + 1);
                    link.push_back(link[q]);
                    const int qq = SZ(edges) - 1;
                    link[q] = link[r] = qq;
                    for (; p >= 0 && edges[p][s[i]] == q; p = link[p])
                        edges[p][s[i]] = qq;
                }
            }
            last = r;
        }
        VI terminals;
        for (int p = last; p > 0; p = link[p])
            terminals.push_back(p);
    }
};
```

## Graph (7)

### 7.1 Network flow

Dinic.h

**Description:** Flow algorithm with complexity  $O(VE \log U)$  where  $U = \max |cap|$ .  $O(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $O(\sqrt{VE})$  for bipartite matching.

abfd54, 42 lines

```
struct Dinic {
    struct Edge {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if you need flows
    };
    VI lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, SZ(adj[b]), c, c});
        adj[b].push_back({a, SZ(adj[a]) - 1, rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < SZ(adj[v]); i++) {
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0; q[0] = s;
        REP(L,0,31) do { // 'int L=30' maybe faster for random data
            lvl = ptr = VI(SZ(q));
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) {
                int v = q[qi++];
                for (Edge e : adj[v])
                    if (!lvl[e.to] && e.c >> (30 - L))
                        q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
            }
            while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
        } while (lvl[t]);
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MCMF-SPFA.h

**Description:** Multiedges and negative costs allowed.

**Time:** Approximately  $O(V^2E^2)$

19b593, 63 lines

```
template <typename FLOW, typename COST> struct MCMF {
    const COST INFC = 1e9, EPSC = 0;
    const FLOW INFF = 1e9, EPSF = 0;
    struct Edge {
        int from, to;
        FLOW flow, cap;
        COST cost;
    };
    int nodes, src, dest, m = 0;
    vector<vector<int>> adj;
    vector<Edge> edges;
    void add(int u, int v, FLOW cap, COST cost) {
        edges.EB(u, v, 0, cap, cost);
    }
};
```

```
adj[u].PB(m++);
edges.EB(v, u, 0, 0, -cost);
adj[v].PB(m++);
}
vector<COST> dis;
vector<bool> inQ;
VI par;
pair<FLOW, COST> SPFA() {
    fill(ALL(dis), INFC);
    fill(ALL(inQ), false);
    queue<int> Q;
    Q.push(src), dis[src] = 0, inQ[src] = true;
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        inQ[u] = false;
        for (int i : adj[u]) {
            auto &e = edges[i];
            if (e.cap - e.flow > EPSF
                && dis[e.to] - (dis[u] + e.cost) > EPSC) {
                dis[e.to] = dis[u] + e.cost;
                par[e.to] = i;
                if (!inQ[e.to]) { Q.push(e.to), inQ[e.to] = true; }
            }
        }
        if (dis[dest] + EPSC >= INFC) return {0, 0};
        FLOW aug = INFF;
        for (int u = dest; u != src; u = edges[par[u]].from) {
            aug = min(aug, edges[par[u]].cap - edges[par[u]].flow);
        }
        for (int u = dest; u != src; u = edges[par[u]].from) {
            edges[par[u]].flow += aug;
            edges[par[u] ^ 1].flow -= aug;
        }
        return {aug, aug * dis[dest]};
    }
}
MCMF(int n, int s, int t)
: nodes(n), src(s), dest(t), adj(n), dis(n), inQ(n), par(n) {}
pair<FLOW, COST> mincostmaxflow() {
    pair<FLOW, COST> ans(0, 0);
    while (true) {
        auto cur = SPFA();
        if (cur.first <= EPSF)
            break;
        ans.first += cur.first;
        ans.second += cur.second;
    }
    return ans;
};
```

MinCostMaxFlow.h

**Description:** Min-cost max-flow.  $cap[i][j] \neq cap[j][i]$  is allowed; double edges are not. 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:** Approximately  $O(E^2)$

92261c, 81 lines

#include <bits/extc++.h>

```
const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;
```

```
struct MCMF {
    int N;
    vector<VI> ed, red;
    vector<VL> cap, flow, cost;
    VI seen;
```

```
VL dist, pi;
vector<PII> par;
```

```
MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}
```

```
void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
}
```

```
void path(int s) {
    fill(ALL(seen), 0);
    fill(ALL(dist), INF);
    dist[s] = 0; ll di;
```

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

```
auto relax = [&](int i, ll cap, ll cost, int dir) {
    ll val = di - pi[i] + cost;
    if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i});
    }
};
```

```
while (!q.empty()) {
    s = q.top().second; q.pop();
    seen[s] = 1; di = dist[s] + pi[s];
    for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
    for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
}
REP(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}
```

```
pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
        totflow += fl;
        for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
            if (r) flow[p][x] += fl;
            else flow[x][p] -= fl;
    }
    REP(i,0,N) REP(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
}
```

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

```
    }
};
```

GlobalMinCut.h

**Description:** Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

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

```
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;
}
```

1d69cc, 21 lines

GomoryHu.h

**Description:** Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.

**Time:**  $\mathcal{O}(V)$  Flow Computations

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

b8cecf, 13 lines

7.2 Matching

MinimumVertexCover.h

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

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

57dfe3, 20 lines

```
        q.push_back(match[e]);
    }
}
REP(i,0,n) if (!lfound[i]) cover.push_back(i);
REP(i,0,m) if (seen[i]) cover.push_back(n+i);
assert(SZ(cover) == res);
return cover;
}
```

WeightedMatching.h

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

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

```
pair<int, VI> hungarian(const vector<VI> &a) {
    if (a.empty()) return {0, {}};
    int n = SZ(a) + 1, m = SZ(a[0]) + 1;
    VI u(n), v(m), p(m), ans(n - 1);
    REP(i,1,n) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        VI dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1, delta = INT_MAX;
            REP(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            REP(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    REP(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}
```

8d4fc6, 31 lines

GeneralMatching.h

**Description:** Matching for general graphs. Fails with probability  $N/mod$ .

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

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

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

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        REP(i,0,N) {
            mat[i].resize(M);
            REP(j,N,M) {
```

69fab6, 40 lines

```
                int r = rand() % mod;
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        } while (matInv(A = mat) != M);

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

7.3 DFS algorithms

SCC.h

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

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

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

```
c6a3ff, 24 lines
VI val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ? dfs(e, g, f));

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

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

BiconnectedComponents.h

**Description:** Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

```
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const VI& edgelist) {...});
Time:  $\mathcal{O}(E + V)$ 
```

69fe6f, 33 lines

```
VI num, st;
vector<vector<PII>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me;
    for (auto pa : ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
            if (num[y]) {
                top = min(top, num[y]);
                if (num[y] < me)
                    st.push_back(e);
            }
            else {
                int si = SZ(st);
                int up = dfs(y, e, f);
                top = min(top, up);
                if (up == me) {
                    st.push_back(e);
                    f(VI(st.begin() + si, st.end()));
                    st.resize(si);
                }
                else if (up < me) st.push_back(e);
                else { /* e is a bridge */ }
            }
        }
    }
    return top;
}

template<class F>
void bicomps(F f) {
    num.assign(SZ(ed), 0);
    REP(i,0,SZ(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h

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

def624, 56 lines

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

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

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

    void either(int f, int j) {
```

```
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }

    void setValue(int x) { either(x, x); }

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

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

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

8e52d2, 15 lines

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

7.4 Coloring

EdgeColoring.h

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

8618ee, 31 lines

```
VI edgeColoring(int N, vector<PII> eds) {
    VI cc(N + 1), ret(SZ(eds)), fan(N), free(N), loc;
    for (PII e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(ALL(cc)) + 1;
    vector<VI> adj(N, VI(ncols, -1));
    for (PII e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    REP(i,0,SZ(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
}
```

7.5 Heuristics

MaximalCliques.h

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

0c1a2a, 12 lines

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

MaximumClique.h

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

1f09ec, 49 lines

```
typedef vector<bitset<200>> vb;
```



```
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<VI> C;
    VI qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(ALL(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        REP(i,0,SZ(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (SZ(R)) {
            if (SZ(q) + R.back().d <= SZ(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (SZ(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1, mnk = max(SZ(qmax) - SZ(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(ALL(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                REP(k,mnk,mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (SZ(q) > SZ(qmax)) qmax = q;
            q.pop_back(), R.pop_back();
        }
    }
    VI maxClique() { init(V), expand(V); return qmax; }
    Maxclique(vb conn) : e(conn), C(SZ(e)+1), S(SZ(C)), old(S) {
        REP(i,0,SZ(e)) V.push_back({i});
    }
};
```

MaximumIndependentSet.h

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

## 7.6 Trees

### BinaryLifting.h

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

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

c4e44c, 25 lines

```
vector<VI> treeJump(VI& P) {
    int on = 1, d = 1;
    while(on < SZ(P)) on *= 2, d++;
    vector<VI> jmp(d, P);
    REP(i,1,d) REP(j,0,SZ(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];
    return jmp;
}
```

```
}

int jmp(vector<VI>& tbl, int nod, int steps) {
    REP(i,0,SZ(tbl))
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}

int lca(vector<VI>& tbl, VI& depth, int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = SZ(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];
        if (c != d) a = c, b = d;
    }
    return tbl[0][a];
}
```

### LCA.h

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

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

"/data-structures/LazySegmentTree.h"cbdl116, 21 lines

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

    LCA(vector<VI>& C) : time(SZ(C)), rmq((dfs(C,0,-1), ret)) {}
    void dfs(vector<VI>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }

    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

### CompressTree.h

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

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

"LCA.h"bcf310, 21 lines

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const VI& subset) {
    static VI rev; rev.resize(SZ(lca.time));
    VI li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(ALL(li), cmp);
    int m = SZ(li)-1;
    REP(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(ALL(li), cmp);
    li.erase(unique(ALL(li)), li.end());
    REP(i,0,SZ(li)) rev[li[i]] = i;
}
```

```
vpi ret = {PII(0, li[0])};
REP(i,0,SZ(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
}
return ret;
}
```

### HLD.h

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

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

"/data-structures/LazySegmentTree.h"e7978a, 46 lines

```
template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<VI> adj;
    VI par, siz, depth, rt, pos;
    Node *tree;
    HLD(vector<VI> adj_)
        : N(SZ(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
          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, depth[u] = depth[v] + 1;
            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 (depth[rt[u]] > depth[rt[v]]) swap(u, v);
            op(pos[rt[v]], pos[v] + 1);
        }
        if (depth[u] > depth[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]);
    }
};
```

## LinkCutTree.h

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

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

5909e2, 90 lines

```
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void pushFlip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            z->c[h ^ 1] = b ? x : this;
        }
        y->c[i ^ 1] = b ? this : x;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() {
        for (pushFlip(); p; ) {
            if (p->p) p->p->pushFlip();
            p->pushFlip(); pushFlip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() {
        pushFlip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
};
```

```
struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

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

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

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

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

};
```

## 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"

719f12, 60 lines

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

    pair<ll, VI> dmst(int n, int r, vector<Edge>& g) {
        RollbackUF uf(n);
        vector<Node*> heap(n);
        for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
        ll res = 0;
        VI seen(n, -1), path(n), par(n);
        seen[r] = r;
        vector<Edge> Q(n), in(n, {-1, -1}), comp;
        deque<tuple<int, int, vector<Edge>>> cycs;
        REP(s, 0, n) {
            int u = s, qi = 0, w;
            while (seen[u] < 0) {
                if (!heap[u]) return {-1, {}};
```

```
Edge e = heap[u]->top();
heap[u]->delta -= e.w, pop(heap[u]);
Q[qi] = e, path[qi++] = u, seen[u] = s;
res += e.w, u = uf.find(e.a);
if (seen[u] == s) {
    Node* cyc = 0;
    int end = qi, time = uf.time();
    do cyc = merge(cyc, heap[w = path[--qi]]);
    while (uf.join(u, w));
    u = uf.find(u), heap[u] = cyc, seen[u] = -1;
    cycs.push_front({u, time, {&Q[qi], &Q[end]}});
}
}
REP(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
}

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

## 7.7 Math

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

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

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

**Mirsky's Theorem** Max length chain is equal to min partitioning into antichains. Max chain is height of poset.

**Dilworth's Theorem** Min partition into chains is equal to max length antichain. From poset create bipartite graph. Any edge from  $v_i - v_j$  implies  $LV_i - RV_j$ . Let  $A$  be the set of vertices such that neither  $LV_i$  nor  $RV_i$  are in vertex cover.  $A$  is an antichain of size  $n$ -max matching. To get min partition into chains, take a vertex from left side, keep taking vertices till a matching exist. Consider this as a chain. Its size is  $n$  - max matching.

**Matrix-tree Theorem** Let matrix  $T = [t_{ij}]$ , where  $t_{ij}$  is negative of the number of multiedges between  $i$  and  $j$ , for  $i \neq j$ , and  $t_{ii} = \deg_i$ . Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any  $k$ -th row and  $k$ -th column from  $T$ . If  $G$  is a multigraph and  $e$  is an edge of  $G$ , then the number  $\tau(G)$  of spanning trees of  $G$  satisfies recurrence  $\tau(G) = \tau(G - e) + \tau(G/e)$ , when  $G - e$  is the multigraph obtained by deleting  $e$ , and  $G/e$  is the contraction of  $G$  by  $e$  (multiple edges arising from the contraction are preserved.)

**Cycle Spaces** The (binary) cycle space of an undirected graph is the set of its Eulerian subgraphs. This set of subgraphs can be described algebraically as a vector space over the two-element finite field. One way of constructing a cycle basis is to form a spanning forest of the graph, and then for each edge  $e$  that does not belong to the forest, form a cycle  $C_e$  consisting of  $e$  together with the path in the forest connecting the endpoints of  $e$ . The set of cycles  $C_e$  formed in this way are linearly independent (each one contains an edge  $e$  that does not belong to any of the other cycles) and has the correct size  $m - n + c$  to be a basis, so it necessarily is a basis. This is fundamental cycle basis.

**Cut Spaces** The family of all cut sets of an undirected graph is known as the cut space of the graph. It forms a vector space over the two-element finite field of arithmetic modulo two, with the symmetric difference of two cut sets as the vector addition operation, and is the orthogonal complement of the cycle space. To compute the basis vector for the cut space, consider any spanning tree of the graph. For every edge  $e$  in the spanning tree, remove the edge and consider the cut formed. Thus dimension of the basis vector for cut space is  $n-1$ .

**Number of perfect matchings of a bipartite graph** is equal to the permanent of the adjacency matrix obtained. To check the parity of the number of perfect matchings, we can evaluate the permanent of the matrix in  $Z_2$  which can be done easily as  $\text{Permanent}(A) = \text{Determinant}(A)$ .

**Tutte Matrix.** For a simple undirected graph  $G$ , Let  $M$  be a matrix with entries  $A_{i,j} = 0$  if  $(i,j) \notin E$  and  $A_{i,j} = -A_{j,i} = X$  if  $(i,j) \in E$ .  $X$  could be any random value. If the determinants are non-zero, then a perfect matching exists, while other direction might not hold for very small probability.

**Kirchhoff's Theorem.** For a multigraph  $G$  with no loops, define Laplacian matrix as  $L = D - A$ .  $D$  is a diagonal matrix with  $D_{i,i} = \deg(i)$ , and  $A$  is an adjacency matrix. If you remove any row and column of  $L$ , the determinant gives a number of spanning trees.

**Brook's Theorem** If a graph is not a complete graph or an odd cycle then it can be coloured with max degree  $\neq$  of colours.

**Turan's Theorem** A graph without  $K_{r+1}$  and  $N$  vertices can have atmax  $\lfloor \frac{N^2}{2} * (1 - \frac{1}{r}) \rfloor$  edges.

Geometry (8)

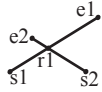
8.1 Geometric primitives

Point.h

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

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 << ")"; }
};
```



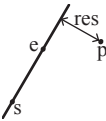
lineDistance.h

**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

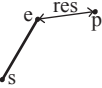
**Description:** Returns the shortest distance between point p and the line segment from point s to e.

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

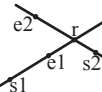
**Description:** If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common 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"

36c2d7, 13 lines

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



lineIntersection.h

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

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

"Point.h"

a01f81, 8 lines

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

sideOf.h

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

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

"Point.h"

3af81c, 9 lines

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

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

```
double l = (e-s).dist()*eps;
return (a > l) - (a < -l);
}
```

OnSegment.h

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

```
"Point.h" c597e8, 3 lines

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

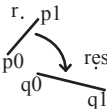
linearTransformation.h

**Description:**

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

```
"Point.h" 03a306, 6 lines

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



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
```

```
struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
    Angle t180() const { return {-x, -y, t + half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};

bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
        make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

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

Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}

Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 8.2 Circles

CircleIntersection.h

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

```
"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

```
"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

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

```
"Time: O(n)" f5c096, 19 lines

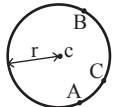
"../content/geometry/Point.h"

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

circumcircle.h

**Description:**

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



```
"Point.h" 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  $O(n)$

```
"circumcircle.h" 69dd52, 17 lines

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

## 8.3 Polygons

InsidePolygon.h

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

**Usage:** vector<P> v = {P{4,4}, P{1,2}, P{2,1}};

bool in = inPolygon(v, P{3, 3}, false);

**Time:**  $O(n)$

```
"Point.h", "OnSegment.h", "SegmentDistance.h" 2261c4, 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" e287fe, 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"7d84e0, 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;
}
```

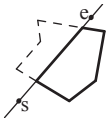
PolygonCut.h

**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"b09131, 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;
}
```



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"c5c490, 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).

"Point.h"261063, 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"efb6da, 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:  $\bullet(-1, -1)$  if no collision,  $\bullet(i, -1)$  if touching the corner  $i$ ,  $\bullet(i, i)$  if along side  $(i, i + 1)$ ,  $\bullet(i, j)$  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.

"Point.h"331463, 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 (cmplL(endA) < 0 || cmpl(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    REP(i,0,2) {
        int lo = endB, hi = endA, n = SZ(poly);
        while ((lo + 1) % n != hi) {
```

```
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpl(m) == cmpl(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpl(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpl(res[0]) && !cmpl(res[1]))
        switch ((res[0] - res[1] + SZ(poly) + 1) % SZ(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h"ac393c, 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;
}
```

kdTree.h

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

"Point.h"269f22, 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);
    }
};
```

FastDelaunay.h

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

```
"Point.h" aaea8f, 88 lines

typedef Point<ll> P;
typedef struct Quad* Q;
typedef int128 t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    lll p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2;
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}

Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
    H = r->o; r->r()->r() = r;
    REP(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
    r->p = orig; r->F() = dest;
    return r;
}
```

FastDelaunay PolyhedronVolume Point3D 3dHull

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

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

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

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

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

8.5 3D

PolyhedronVolume.h

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

```
3058c3, 6 lines
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) 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:**  $\mathcal{O}(n^2)$

```
0754b0, 49 lines
"Point3D.h"
typedef Point3D<double> P3;

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

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

vector<F> hull3d(const vector<P3>& A) {
    assert(SZ(A) >= 4);
}
```

```
vector<vector<PR>> E(SZ(A), vector<PR>(SZ(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
vector<F> FS;
auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
        q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
};
REP(i,0,4) REP(j,i+1,4) REP(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

REP(i,4,SZ(A)) {
    REP(j,0,SZ(FS)) {
        F f = FS[j];
        if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
            E(a,b).rem(f.c);
            E(a,c).rem(f.b);
            E(b,c).rem(f.a);
            swap(FS[j--], FS.back());
            FS.pop_back();
        }
    }
    int nw = SZ(FS);
    REP(j,0,nw) {
        F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
        C(a, b, c); C(a, c, b); C(b, c, a);
    }
    for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
        A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
    return FS;
};
```

sphericalDistance.h  
**Description:** Returns the shortest distance on the sphere with radius *radius* between the points with azimuthal angles (longitude) *f1* ( $\phi_1$ ) and *f2* ( $\phi_2$ ) from *x* axis and zenith angles (latitude) *t1* ( $\theta_1$ ) and *t2* ( $\theta_2$ ) from *z* axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. *dx\*radius* is then the difference between the two points in the *x* direction and *d\*radius* is the total distance between the points.

```
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);
}
```

# Mathematics (9)

## 9.1 Equations

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

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

## sphericalDistance

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

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  $b$ .

## 9.2 Recurrences

If  $a_n = c_1a_{n-1} + \dots + c_ka_{n-k}$ , and  $r_1, \dots, r_k$  are distinct roots of  $x^k + c_1x^{k-1} + \dots + c_k$ , there are  $d_1, \dots, d_k$  s.t.

$$a_n = d_1r_1^n + \dots + d_kr_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.  
 $a_n = (d_1n + d_2)r^n$ .

## 9.3 Trigonometry

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

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

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

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

## 9.4 Geometry

### 9.4.1 Triangles

Side lengths:  $a, b, c$

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

### 9.4.2 Quadrilaterals

With side lengths  $a, b, c, d$ , diagonals  $e, f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F = b^2 + d^2 - a^2 - c^2$ , then  
 $4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2f^2 - F^2}$

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

### 9.4.3 Spherical coordinates

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

## 9.5 Derivatives/Integrals

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

Integration by parts:

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

## 9.6 Sums

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

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

9.7 Series

$$\begin{aligned}e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, \quad (-\infty < x < \infty) \\ \ln(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, \quad (-1 < x \leq 1) \\ \sqrt{1+x} &= 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, \quad (-1 \leq x \leq 1) \\ \sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, \quad (-\infty < x < \infty) \\ \cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, \quad (-\infty < x < \infty)\end{aligned}$$

9.8 Probability theory

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

Expectation is linear:

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

For independent  $X$  and  $Y$ ,

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

9.8.1 Discrete distributions

Binomial distribution

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

$$\begin{aligned}p(k) &= \binom{n}{k} p^k (1-p)^{n-k} \\ \mu &= np, \sigma^2 = np(1-p)\end{aligned}$$

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

First success distribution

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

$$\begin{aligned}p(k) &= p(1-p)^{k-1}, \quad k = 1, 2, \dots \\ \mu &= \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}\end{aligned}$$

Poisson distribution

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

$$\begin{aligned}p(k) &= e^{-\lambda} \frac{\lambda^k}{k!}, \quad k = 0, 1, 2, \dots \\ \mu &= \lambda, \sigma^2 = \lambda\end{aligned}$$

9.8.2 Continuous distributions

Uniform distribution

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

$$\begin{aligned}f(x) &= \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases} \\ \mu &= \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}\end{aligned}$$

Exponential distribution

The time between events in a Poisson process is  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$\begin{aligned}f(x) &= \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \\ \mu &= \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}\end{aligned}$$

Normal distribution

Most real random values with mean  $\mu$  and variance  $\sigma^2$  are well described by  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

9.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1, X_2, \dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P} = (p_{ij})$ , with  $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$ , and  $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)} = \Pr(X_n = i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi = \pi \mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i = \frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j/\pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

A Markov chain is an **A-chain** if the states can be partitioned into two sets **A** and **G**, such that all states in **A** are absorbing ( $p_{ii} = 1$ ), and all states in **G** leads to an absorbing state in **A**. The probability for absorption in state  $i \in \mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$ . The expected time until absorption, when the initial state is  $i$ , is  $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$ .

Miscellaneous (10)

10.1 RNG, Intervals, Ternary Search

RNGs.h6 lines

```
SEED = chrono::steady_clock::now().time_since_epoch().count();
// or use 'high_resolution_clock'
random_device rd; auto SEED = rd();
mt19937 rng(SEED);
uniform_int_distribution<> dis(MIN, MAX); // usage: dis(rng)
// others: uniform_real_distribution,
```

IntervalContainer.h564cdd, 23 lines

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



```
    R = max(R, it->second);
    is.erase(it);
}
return is.insert(before, {L,R});
}
```

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

IntervalCover.h

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

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

354a6a, 19 lines

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

ConstantIntervals.h

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

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

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

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  $\leq$ , and reverse the loop at (B). To minimize  $f$ , change it to  $>$ , also at (B).

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

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

a9cf52, 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;
}
```

## 10.2 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).

trace.h

10 lines

```
#define trace(...) { __f(#_VA_ARGS__, __VA_ARGS__); }
template<typename Arg> void __f(const char* name, Arg&& arg) {
    cerr << name << " = " << arg << endl;
}

template <typename Arg1, typename... Args>
void __f(const char* names, Arg1&& arg1, Args&&... args) {
    const char* comma = strchr(names + 1, ',');
    cerr.write(names, comma - names) << " = " << arg1<< " | ";
    __f(comma+1, args...);
}
}
```

## 10.3 Optimization tricks

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

### 10.3.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `REP(b,0,K) REP(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

### 10.3.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).
- `#pragma GCC optimize("unroll-loops")`
- `target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx")`

FastMod.h

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

751a02, 8 lines

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

FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.

**Usage:** ./a.out < input.txt

**Time:** About 5x as fast as cin/scanf.

7b3c70, 17 lines

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

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

BumpAllocator.h

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

2216bb, 9 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*) {}
void operator delete(void*, size_t) {}
```

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

"BumpAllocator.h"2dd6c9, 10 lines

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

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: vector<vector<int, small<int>>> ed(N);bb66d4, 14 lines

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

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

SIMD.h

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

721646, 43 lines

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

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

// High-level/specific methods:

```
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an imm
```

// Methods that work with most data types (append e.g. \_epi32):

```
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)
```

```
int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; REP(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
```

mi one() { return \_mm256\_set1\_epi32(-1); }

```
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }
```

```
ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
        mi vp = _mm256_madd_epi16(va, vb);
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; REP(i,0,4) r += u.v[i];
    for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
    return r;
}
```