INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

International Institute Of Information Technology - Hyderabad

ladaiLadai

Aryan Gupta, Aarush jain, Keyur Chaudhari

ICPC Asia West Finals 2021

Om Shri Krishna Sharanam namah

Combinatorial (1)

1.1 Permutations

1.1.1 Derangements

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

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

1.1.2 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

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

1.2 Partitions and subsets

1.2.1 Partition function

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

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$
$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

1.3 General purpose numbers

1.3.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

1.3.2 Eulerian numbers

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

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

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

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

1.3.3 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,

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

1.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.
$S(n,k) = S(n-1,k-1) + kS(n-1,k)$
$S(n,1) = S(n,n) = 1$

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

1.3.5 Labeled unrooted trees

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

1.3.6 Catalan numbers

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

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

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

- 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.

1.3.7 Pick Theorem

S = area of polygon, I = Number of integer points inside, B = Number of integer points on boundary
$I + B/2 - 1 = S$

1.4 DP Optimizations

Quadrangle Inequality. f satisfies it if $\forall a \le b \le c \le d, f(a,d) - f(a,c) \ge f(b,d) - f(b,c)$.

PointsOnBoundary.h

Description: given a polygon A in order, returns count of points(integer) on boundary Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

ad93fa, 9 lines

```
int boundary(vector<pair<int, int>>& A) {
    int ats = A.size();
    for (int i = 0; i < A.size(); i++) {
        int dx = (A[i].first - A[(i + 1) % A
            .size()].first);
        int dy = (A[i].second - A[(i + 1) %
            A.size()].second);
        ats += abs(__gcd(dx, dy)) - 1;
    }
    return ats;
}
```

1D-1D.h

Description: Applicable if $dp_i = min_{j>i}(dp_j + cost(i,j)$ s.t. $opt_i \le opt_j$ when $i \le j$ (which holds if quadrangle)
Time: $\mathcal{O}(n \log n)$

d76777, 26 lines

```
#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 && 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);}}
```

Dynamic-CHT.h

Description: Add lines $y = ax + b$ and query for min at given x
Time: $\mathcal{O}(logn)$ per update/query.

b518fa, 27 lines

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

Divide-and-Conquer.h

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

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);
}
```

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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \ge f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

2556d3, 18 lines

```
const int N = 1001;int dp[N][N];int opt[N][N
    ];
int rec() {
    vector<int> pref(m);
    for (int i = 0; i < m; i++) {
        pref[i] = A[i];
        if (i) pref[i] += pref[i - 1];}
    for (int i = 0; i < m; i++) {
        opt[i][i] = i; dp[i][i] = 0;}
    for (int i = m - 2; i >= 0; i--) {
        for (int j = i + 1; j < m; j++) {
            int mn = mod * 1000;
            int cost = ;  // COST [i,j]
            for (int k = opt[i][j - 1]; k <=
                 min(j - 1, opt[i + 1][j]);
                k++) {
                 if (mn >= dp[i][k] + dp[k +
                     1][j] + cost) {
                     opt[i][j] = k;
                     mn = dp[i][k] + dp[k +
                         1][j] + cost;}}
            dp[i][j] = mn;}}
    return dp[0][m - 1];}
```

Numerical (2)

2.1 Polynomials

PolyInterpolate.h

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

a4f803, 10 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;}
```

2.2 Matrices

IntDeterminant.h

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

669167, 14 lines

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

SolveLinear.h

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

d6dca7, 30 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
int solveLinear(vector<vd>& A, vd& b, vd& x)
    {
  int n = SZ(A), m = SZ(x), rank = 0, br, bc
     ;
  if (n) assert(SZ(A[0]) == m);
  VI col(m); iota(ALL(col), 0);
  REP(i,0,n) {
    double v, bv = 0;
    REP(r,i,n) REP(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
    REP(j,i,n) if (fabs(b[j]) > eps) return
       -1;
    break;
    }
    swap(A[i], A[br]); swap(b[i], b[br]);
    swap(col[i], col[bc]);
    REP(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    REP(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      REP(k,i+1,m) A[j][k] -= fac*A[i][k];}
    rank++;}
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    REP(j,0,i) b[j] -= A[j][i] * b[i];}
  return rank;} // (multiple solutions if
     rank < m)
```

SolveLinear2.h

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

"SolveLinear.h" 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: $\mathcal{O}(n^2m)$

c9c00b, 25 lines

```
typedef bitset<1000> bs;
int solveLinear(vector<bs>& A, VI& b, bs& x,
    int m) {
  int n = SZ(A), rank = 0, br; assert(m <=
      SZ(x));
  VI col(m); iota(ALL(col), 0);
  REP(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any())
        break;
    if (br == n) {
      REP(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]); swap(b[i], b[br]);
    swap(col[i], col[bc]);
    REP(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);}
    REP(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];}
    rank++;}
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    REP(j,0,i) b[j] ^= A[j][i];}
  return rank;} // (multiple solutions if
      rank < m)
```

MatrixInverse.h

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

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

ff56f5, 27 lines

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

2.3 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 $N = |A| + |B|$ (~$1s$ 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 $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i] b[x - i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).
Time: $\mathcal{O}(N \log N)$

"../number-theory/ModPow.h" 14d0bb, 30 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: $\mathcal{O}(N \log N)$

795695, 13 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, 7 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);}
```

Number theory (3)

3.1 Modular arithmetic

ModPow.h

fe17f7, 7 lines

```
int power(long long x, unsigned int y, int p
    ){
  int res = 1; x = x % p;
  if (x == 0) return 0;
  while (y > 0){
    if (y & 1) res = (res * x) % p;
    y = y >> 1; x = (x * x) % p;}
  return res;}
```

NCR.h

Description: Calculates ncr for large N and prime Mod

8f060e, 12 lines

```
#define MAX_N_FACT (int)3e5 + 5
vector<long long> factorial(MAX_N_FACT, 1),
    inverse_factorial(MAX_N_FACT, 1), inv(
    MAX_N_FACT, 1);
void prec_factorials() {
    for (int i = 2; i < MAX_N_FACT; i++) {
        factorial[i] = factorial[i - 1] * i
            % mod;
        inv[i] = (mod - (mod / i) * inv[mod
            % i] % mod);
        inverse_factorial[i] = (inv[i] *
            inverse_factorial[i - 1]) % mod;
    }
}
long long ncr(int n, int k) {
    return factorial[n] * inverse_factorial[
        k] % mod * inverse_factorial[n - k]
        % mod;
}
```

ModLog.h

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

0ff368, 10 lines

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

3.2 Primality

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 $a^b \mod c$.

"ModMulLL.h" 60dcd1, 10 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n |
      1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775,
      9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing
      zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n &&
        i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;}
  return 1;}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}(n^{1/4})$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" 65f857, 15 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;}
```

3.3 Divisibility

euclid.h

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

33ba8f, 4 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 $0 \le x < \text{lcm}(m,n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$

"euclid.h" 04d93a, 6 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;}
```

phiFunction.h

Description: *Euler's* ϕ 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 $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$.

e4742a, 6 lines

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

3.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

3.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

3.5 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)

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

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

Mobius.h

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

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 (4)

Trie.h

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

1cba01, 19 lines

```
const int NX = int(1e6) + int(5e5);int arr[
    NX][26];
int root; int lastocc;
void Trie() {
    root = 0, lastocc = 0;
    memset(arr, 0, sizeof(int) * NX * 26);}
void insert(const string &x) {
    int curptr = root;
    for (auto ch : x) {
        if (arr[curptr][ch - 'a'] == 0)
            arr[curptr][ch - 'a'] = ++
                lastocc;
        curptr = arr[curptr][ch - 'a'];}}
int search(const string &x) {
    int curptr = root;
    for (auto ch : x) {
        if (arr[curptr][ch - 'a'] == 0)
            return 0;
        else
            curptr = arr[curptr][ch - 'a'];}
    return 1;}
```

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

998917, 68 lines

```
mt19937 gen(time(0));
uniform_int_distribution<int> rng;
typedef struct node{
    int prior,size,val,sum,lazy;
    //value in array,info of segtree,lazy
        update
    struct node *l,*r;
}node;typedef node* pnode;
int sz(pnode t){
    return t?t->size:0;}
void upd_sz(pnode t){
    if(t)t->size=sz(t->l)+1+sz(t->r);}
void lazy(pnode t){
    if(!t || !t->lazy)return;
    t->val+=t->lazy;//operation of lazy
    t->sum+=t->lazy*sz(t);
    if(t->l)t->l->lazy+=t->lazy;//propagate
        lazy
    if(t->r)t->r->lazy+=t->lazy;
    t->lazy=0;}
void reset(pnode t){
    if(t)t->sum = t->val;}//lazy already
        propagated}
void combine(pnode& t,pnode l,pnode r){//
    combine segtree ranges
    if(!l || !r)return void(t = l?l:r);
    t->sum = l->sum + r->sum;}
void operation(pnode t){//operation of
    segtree
    if(!t)return;
    reset(t);//node represents single
        element of array
    lazy(t->l);lazy(t->r);//imp:propagate
        lazy before combining l,r
    combine(t,t->l,t);combine(t,t,t->r);}
void split(pnode t,pnode &l,pnode &r,int pos
    ,int add=0){
    if(!t)return void(l=r=NULL);
    lazy(t);int curr_pos = add + sz(t->l);
    if(curr_pos<=pos)//element at pos goes
        to "l"
        split(t->r,t->r,r,pos,curr_pos+1),l=
            t;
    else    split(t->l,l,t->l,pos,add),r=t;
    upd_sz(t);operation(t);}
void merge(pnode &t,pnode l,pnode r){//
    result/left/right array
    lazy(l);lazy(r);
    if(!l || !r) t = l?l:r;
    else if(l->prior>r->prior)merge(l->r,l->
        r,r),t=l;
    else    merge(r->l,l,r->l),t=r;
    upd_sz(t);operation(t);}
pnode init(int val){
    pnode ret = (pnode)malloc(sizeof(node));
    ret->prior=rng(gen);ret->size=1;
    ret->val=val;ret->sum=val;ret->lazy=0;
    return ret;}
int range_query(pnode t,int l,int r){//[l,r]
    pnode L,mid,R;
    split(t,L,mid,l-1);split(mid,t,R,r-l);//
        note: r-l!!
    int ans = t->sum;
    merge(mid,L,t);merge(t,mid,R);
    return ans;}
void range_update(pnode t,int l,int r,int
    val){//[l,r]
    pnode L,mid,R;
    split(t,L,mid,l-1);split(mid,t,R,r-l);//
        note: r-l!!
    t->lazy+=val; //lazy_update
    merge(mid,L,t);merge(t,mid,R);}
void reverse(pnode t, int l, int r) {
    pnode t1 = NULL, t2 = NULL, t3 = NULL;
    split(t, t1, t2, l - 1);
    split(t2, t2, t3, r - l);
    assert(t2); t2->rev ^= true;
    merge(t, t1, t2);merge(t, t, t3);}
void output(pnode t) {
    if (!t) return;
    output(t->l);cout << t->val << " ";
    output(t->r);}
pnode Treap = NULL;
```

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

f9c99d, 22 lines

```
struct node{int val,prior,size;node *l,*r;};
typedef node* pnode;int sz(pnode t){return t
    ?t->size:0;}
void upd_sz(pnode t){if(t)t->size = sz(t->l)
    +1+sz(t->r);}
void split(pnode t,pnode &l,pnode &r,int key
    ){if(!t)l=r=NULL;
else if(t->val<=key)split(t->r,t->r,r,key),l
    =t;//key in l
else split(t->l,l,t->l,key),r=t;upd_sz(t);
}void merge(pnode &t,pnode l,pnode r){if(!l
    || !r)t=l?l:r;
else if(l->prior> r->prior)merge(l->r,l->r,r
    ),t=l;
else merge(r->l,l,r->l),t=r;upd_sz(t);
}void insert(pnode &t,pnode it){if(!t) t=it;
else if(it->prior>t->prior)split(t,it->l,it
    ->r,it->val),t=it;
else insert(t->val<it->val?t->r:t->l,it);
    upd_sz(t);
}void erase(pnode &t,int key){if(!t)return;
else if(t->val==key){pnode x=t;merge(t,t->l,
    t->r);free(x);}
else erase(t->val<key?t->r:t->l,key);upd_sz(
    t);
}void unite (pnode &t,pnode l, pnode r){
if(!l||!r)return void(t=l?l:r);pnode lt,rt;
if(l->prior<r->prior)swap(l,r);split(r,lt,rt
    ,l->val);
unite(l->l,l->l,lt);unite(l->r,l->r,rt);t=l;
    upd_sz(t);
}pnode init(int val){pnode ret = (pnode)
    malloc(sizeof(node));
ret->val=val;ret->size=1;ret->prior=rand();
    ret->l=ret->r=NULL;
return ret;}insert(init(x),head);
```

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, 18 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;}};
```

4.1 Range DS

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
Time: $\mathcal{O}(\log N)$

710629, 19 lines

```
template <typename T>
struct segTree {
  T unit;
  T (*f) (T obj1, T obj2);
  vector<T> s;
  int n;
  segTree(int n, T (*c)(T obj1, T obj2), T
      def) : s(2 * n, def), n(n), f(c), unit
      (def) {}
  void update(int pos, T val) {
    for (s[pos += n] = val; pos /= 2;)
      s[pos] = f(s[pos * 2], s[pos * 2 + 1])
          ;
  }
  T query(int b, int e) {  // query [b, e]
    e++;
    T ra = unit, rb = unit;
    for (b += n, e += n; b < e; b /= 2, e /=
        2) {
      if (b % 2) ra = f(ra, s[b++]);
      if (e % 2) rb = f(s[--e], rb);
    }
    return f(ra, rb);}};
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: Node* tr = new Node(v, 0, SZ(v));
Time: $\mathcal{O}(\log N)$.

5d1ef3, 31 lines

```
int ST[4*N],lazy[4*N],A[N];
#define lc (x<<1)
#define rc (x<<1)|1
void push(int x,int l,int r){
ST[x]+=lazy[x]; //Operation of lazy
if(l==r-1)lazy[x]=0;
if(!lazy[x])return;
lazy[lc]+=lazy[x];
lazy[rc]+=lazy[x];lazy[x]=0;//Propagate Lazy
}void up(int x){//Operation of Segtree
ST[x] = min(ST[lc],ST[rc]);
}void build(int l=0,int r=N,int x=1){
lazy[x]=0;//clear lazy
if(l==r-1)return void(ST[x]=A[l]);
int m = (l+r)/2;
build(l,m,lc);build(m,r,rc);up(x);
}void update(int L,int R,int add,int l=0,int
    r=N,int x=1){
push(x,l,r);int m = (l+r)/2;
if(l>=R || r<=L)return;
if(l>=L && r<=R){
lazy[x]+=add;//operation of lazy update
return push(x,l,r);
}update(L,R,add,l,m,lc);
update(L,R,add,m,r,rc);up(x);
}int query(int L,int R,int l=0,int r=N,int x
   =1){
push(x,l,r);int m = (l+r)/2;
if(l>=R||r<=L)return INF;//nothing here
if(l>=L && r<=R)return ST[x];
int la = query(L,R,l,m,lc);
int ra = query(L,R,m,r,rc);
return min(la,ra);//operation of segtree}
```

PersistentSegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: Node* tr = new Node(v, 0, SZ(v));
Time: $\mathcal{O}(\log N)$.

6aa2ac, 42 lines

```
struct Vertex {
    Vertex *l, *r;
    int sum;

    Vertex(int val) : l(nullptr), r(nullptr)
        , sum(val) {}
    Vertex(Vertex* l, Vertex* r) : l(l), r(r
        ), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(int a[], int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm),
        build(a, tm + 1, tr));
}

int get_sum(Vertex* v, int tl, int tr, int l
    , int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
```

```
    return get_sum(v->l, tl, tm, l, min(r,
       tm)) + get_sum(v->r, tm + 1, tr, max
       (l, tm + 1), r);
}

Vertex* update(Vertex* v, int tl, int tr,
   int pos, int new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl,
           tm, pos, new_val), v->r);
    else
        return new Vertex(v->l, update(v->r,
            tm + 1, tr, pos, new_val));
}
int tl = 0, tr = MAX_VALUE + 1;
std::vector<Vertex*> roots;
roots.push_back(build(tl, tr));
for (int i = 0; i < a.size(); i++) {
    roots.push_back(update(roots.back(), tl,
        tr, i, a[i]));
}
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

367d01, 8 lines

```
struct FT {vector<ll> s;
    FT() = default;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;}};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V|\log|V| + Q)$

9a1bbf, 11 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: $\mathcal{O}\left(N\sqrt{Q}\right)$

3015c5, 43 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: $\mathcal{O}\left(n^{5/3}\right)$ when block = $n^{2/3}$

303d07, 14 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();}}
```

Strings (5)

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

e21281, 12 lines

```
VI pi(const string& s) {
  VI p(SZ(s));
  REP(i,1,SZ(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);}
  return p;}
VI match(const string& s, const string& pat)
    {
  VI p = pi(pat + '\0' + s), res;
  REP(i,SZ(p)-SZ(s),SZ(p)){
  if (p[i] == SZ(pat)) res.push_back(i - 2 *
      SZ(pat));}
  return res;}
```

Zfunc.h

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

a706e2, 10 lines

```
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
Time: construction $\mathcal{O}(26N)$,where N =sum patterns

f02538, 68 lines

```
struct ahocorasick {
    vector<vector<int>> next;
    vector<int> fail, out, finish, cnt;
    vector<string> words;
    // if there are repeated words and it is
        necessary to show them
    // 'finish' has to be vector<vector<int
        >>
    // fail stores the suffix links
    // finish stores index of word which
        ends at that index(if  any)
    // cnt stores the number of words ending
        at i
    // out stores the index in trie of that
        suffix link where some word ends
    ahocorasick() {
        next.push_back(vector<int>(26));
        finish.push_back(0);
        cnt.push_back(0);
    }
    void insert(string &s) {
        int u = 0;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (!next[u][c]) {
                next[u][c] = next.size();
                next.push_back(vector<int
                    >(26));
                finish.push_back(-1);
                cnt.push_back(0);}
            u = next[u][c];}
        finish[u] = words.size(); ++cnt[u];
        words.push_back(s);}
    int get_fail(int pfail, int c) {
        while (!next[pfail][c] && pfail !=
            0)
            pfail = fail[pfail];
        return next[pfail][c];}
    void update_out(int u) {
        out[u] = fail[u];
        while (finish[out[u]] == -1)
            out[u] = fail[out[u]];}
    void buildf() {
        queue<int> q;fail.assign(next.size()
            , 0);
        out.assign(next.size(), 0);
        for (int i = 0; i < 26; ++i)
            if (next[0][i])
                q.push(next[0][i]);
        while (q.size()) {
            int u = q.front(); q.pop();
            for (int i = 0; i < 26; ++i) {
                int v = next[u][i];
                if (v) {
                    fail[v] = get_fail(fail[
                        u], i);
                    cnt[v] += cnt[fail[v]];
                    q.push(v);
// update_out is similar to while loop in
    match functions
// since it goes through all strings while
    end at that node
// so comment if unnecessary
                    update_out(v);}}}}
    int match(string &s) {
        int cur = 0, matches = 0;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (next[cur][c])
                cur = next[cur][c];
```

```
        else
            cur = get_fail(fail[cur], c)
                ;
        matches += cnt[cur];
        int t = cur;
        while (t != 0) {
            if (finish[t] != -1) {
                cout << words[finish[t]]
                    << endl;}
            t = out[t];}}
    return matches;}};
```

Hashing.h

Description: Self-explanatory methods for string hashing.

cc1766, 26 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(O,A,B) H operator O(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 %1\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 = (ll)1e11+3; // (order ~ 3
   e9; 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) and 0 indexed
    return ha[b] - ha[a] * pw[b - a];}};
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`

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

43e521, 6 lines

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

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

1deebd, 10 lines

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

5.3 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, 21 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++);}};
```

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 occurences = 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, 32 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 (6)

BellmanFord.h

Description: Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

11ca03, 17 lines

```
const ll inf = LLONG_MAX;
struct Ed {int a, b, w, s() {return a < b ?
    a : -a;}};
struct Node { ll dist = inf; int prev = -1;
    };
void bellmanFord(vector<Node>& nodes, vector
    <Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(ALL(eds),[](Ed a, Ed b){return a.s()<
      b.s();});
  int lim = SZ(nodes) / 2 + 2; // /3+100
      with shuffled vertices
  REP(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed
        .b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);}}
  REP(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;}}
```

FloydWarshall.h

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

0ff4bf, 10 lines

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

6.1 Network flow

Dinic.h

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

abfd54, 34 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 $\mathcal{O}\left(V^2E^2\right)$

19b593, 43 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] != 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 $\mathcal{O}\left(E^2\right)$

12d3dc, 57 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;VI seen;VL dist,
      pi;
  vector<VL> cap, flow, cost;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: $\mathcal{O}\left(V^3\right)$

1d69cc, 17 lines

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

6.2 Matching

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

8d4fc6, 31 lines

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

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

c6a3ff, 15 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, 23 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]) {
      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: Valid Assignemnt

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.

fdd092, 56 lines

```
struct two_sat {
    int n;
    vector<vector<int>> g, gr;
                                // gr is the
       reversed graph
    vector<int> comp, topological_order,
       answer;  // comp[v]: ID of the SCC
       containing node v
    vector<bool> vis;
    two_sat() {}
    two_sat(int _n) { init(_n); }
    void init(int _n) {
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        comp.resize(2 * n);
        vis.resize(2 * n);
        answer.resize(2 * n);}
    void add_edge(int u, int v) {
        g[u].push_back(v);
        gr[v].push_back(u);}
    // For the following three functions
    // int x, bool val: if 'val' is true, we
         take the variable to be x.
       Otherwise we take it to be x's
       complement.
    // At least one of them is true
    void add_clause_or(int i, bool f, int j,
         bool g) {
        add_edge(i + (f ? n : 0), j + (g ? 0
            : n));
        add_edge(j + (g ? n : 0), i + (f ? 0
            : n));}
    // Only one of them is true
    void add_clause_xor(int i, bool f, int j
       , bool g) {
        add_clause_or(i, f, j, g);
        add_clause_or(i, !f, j, !g);}
    // Both of them have the same value
    void add_clause_and(int i, bool f, int j
       , bool g) {
        add_clause_xor(i, !f, j, g);}
    // Topological sort
    void dfs(int u) {
        vis[u] = true;
        for (const auto &v : g[u])
            if (!vis[v]) dfs(v);
        topological_order.push_back(u);}
    // Extracting strongly connected
       components
    void scc(int u, int id) {
        vis[u] = true;comp[u] = id;
        for (const auto &v : gr[u])
            if (!vis[v]) scc(v, id);}
    // Returns true if the given proposition
         is satisfiable and constructs a
       valid assignment
    bool satisfiable() {
        fill(vis.begin(), vis.end(), false);
        for (int i = 0; i < 2 * n; i++)
            if (!vis[i]) dfs(i);
        fill(vis.begin(), vis.end(), false);
        reverse(topological_order.begin(),
           topological_order.end());
        int id = 0;
        for (const auto &v :
           topological_order)
            if (!vis[v]) scc(v, id++);
        // Constructing the answer
        for (int i = 0; i < n; i++) {
            if (comp[i] == comp[i + n])
                return false;
            answer[i] = (comp[i] > comp[i +
               n] ? 1 : 0);}
        return true;}};
```

6.4 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;
}
```

MinimumBipartiteEdgeColoring.h

Description: color edges of a bipartite graph with minimum colors
Time: $\mathcal{O}(NM)$

8c04bb, 42 lines

```
// change max-values of m and n according to
    problem
// v is vector of edges and cv is the color
   of corresponding edge
// solve returns the maximum number of
   colors used
// call the solve function to get the colors
    from 1 to d in O(n*m)
// Space occupied by deg is 2*(max nodes
   possible on 1 side)
// Space occupied by has is 2*2*(max nodes
   possible on 1 side)*(max colors possible
   )
// Space occupied by deg is (max edges
   possible)
struct edge_color {
    int deg[2][MAXN];II has[2][MAXN][MAXN];
    int color[MAXM];int c[2];
    void clear(int n) {
        for (int t = 0; t < 2; t++) {
            for (int i = 0; i <= n; i++) {
                deg[t][i] = 0;
                for (int j = 0; j <= n; j++)
                    {
                    has[t][i][j] = II(0, 0);
                        }}}}
    void dfs(int x, int p) {
        auto i = has[p][x][c[!p]];
        if (has[!p][i.first][c[p]].second)
            dfs(i.first, !p);
        else
            has[!p][i.first][c[!p]] = II(0,
                0);
        has[p][x][c[p]] = i;
        has[!p][i.first][c[p]] = II(x, i.
            second);
        color[i.second] = c[p];}
    int solve(vector<II> v, vector<int> &cv)
        {
        int m = SZ(v);int ans = 0;
        for (int i = 1; i <= m; i++) {
            int x[2];x[0] = v[i - 1].first;
            x[1] = v[i - 1].second;
            for (int d = 0; d < 2; d++) {
                deg[d][x[d]] += 1;
                ans = max(ans, deg[d][x[d]])
                    ;
                for (c[d] = 1; has[d][x[d]][
                    c[d]].second; c[d]++)
                    ;}
            if (c[0] != c[1]) dfs(x[1], 1);
            for (int d = 0; d < 2; d++) has[
                d][x[d]][c[0]] = II(x[!d], i
                );
            color[i] = c[0];}
        cv.resize(m);
        for (int i = 1; i <= m; i++) {
            cv[i - 1] = color[i];color[i] =
                0;}
        return ans;}};
```

6.5 Trees

BinaryLifting.h

Description: Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

c4e44c, 19 lines

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

```
  REP(i,0,SZ(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;}
int lca(vector<VI>& tbl, VI& depth, int a,
   int b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(tbl, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = SZ(tbl); i--;) {
    int c = tbl[i][a], d = tbl[i][b];
    if (c != d) a = c, b = d;}
  return tbl[0][a];}
```

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$

"../data-structures/RMQ.h" cbd116, 15 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){ret depth[a]+depth[b]-2*depth[
   lca(a,b)];}
```

CompressTree.h

Description: Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

0292cd, 13 lines

```
bool cmp(int u,int v){return arr[u]<arr[v];}
int create_tree(){//return root of tree
set<int> S;//get distinct nodesFord
REP(i,k)S.insert(Q[i]);k=0;for(auto it : S)Q
   [k++]=it;
sort(Q,Q+k,cmp);int kk = k;//distinct
   initial nodes
//add lca of adjacent pairs
for(int i=0;i<kk-1;i++){int x = lca(Q[i],Q[i
   +1]);
if(S.count(x))continue;Q[k++]=x;S.insert(x);
}sort(Q,Q+k,cmp);stack<int> s;s.push(Q[0]);
for(int i=1;i<k;i++){
while(!anc(s.top(),Q[i]))s.pop();
tree[s.top()].PB(Q[i]);tree[Q[i]].PB(s.top()
   );
s.push(Q[i]);}return Q[0];}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.
Time: $\mathcal{O}\left((\log N)^2\right)$

c4840c, 20 lines

```
VI sz, sc, hd, en, ex, par, dep;
seg_tree_lazy<node, update> st(1, {0, 0}, {
   0, 0});
int timer = -1;
void hld(int u, int p, int ch, int d) {
    hd[u] = ch;en[u] = ++timer; par[u] = p;
       dep[u] = d;
    if (sc[u] != -1) hld(sc[u], u, ch, d +
       1);
    for (auto e : g[u]) {
        int v = U[e] ^ V[e] ^ u;
        if (v == p || v == sc[u]) continue;
        hld(v, u, v, d + 1);}
    ex[u] = timer;}
int path(int x, int y) {
    int ma = (int)-1e9;
    while (hd[x] != hd[y]) {
        if (dep[hd[x]] < dep[hd[y]]) swap(x,
           y);
        ma=max(st.query(en[hd[x]],en[x]).sum
           ,ma);
        x = par[hd[x]]; } //hd[x] -> x upar
           wali line
    if (dep[x] < dep[y]) swap(x, y);
    ma = max(ma,st.query(en[y],en[x]).sum);
       //y -> x
    return ma;}
```

CentriodDecomposition.h

Description: Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

f8ca72, 14 lines

```
VI U, V, W, isDel;
int dp[n][log2(n) + 1];
// (avoid deleted edges) in all 3 DFS
void decompose(int root, int p) {
    dfs_sz(root, -1); // calc sizes of
       subtrees
    int c = get_centroid(root, -1, sz[root])
       ;  // if sz[v] * 2 > sz[root] return
        get_centroid(v) else return u
    if (p == -1) p = root;
    // Add edge btwn p and c here
    dfs(c);  // to compute functions
    for (auto e : g[root]) {
        if (isDel[e]) continue;
        isDel[e] = 1;
        int v = U[e] ^ V[e] ^ u;
        decompose(v, root);}}
```

6.6 Math

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

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

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

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.

Geometry (7)

7.1 Geometric primitives

Point.h

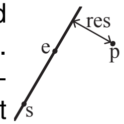
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

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

lineDistance.h

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

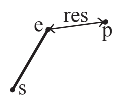


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



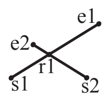
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

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



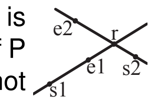
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (SZ(inter)==1)
cout << "segments intersect at " << inter[0] << endl;

"Point.h", "OnSegment.h" 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

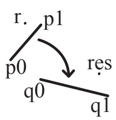
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"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();
}
```

LineProjectionReflection.h

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

"Point.h" b5562d, 5 lines

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

Angle.h

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

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y),
      t(t) {}
  Angle operator-(Angle b) const { return {x
      -b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
```

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

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

7.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CircleLine.h

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

"Point.h" eea4d5, 10 lines

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

CirclePolygonIntersection.h

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

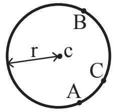
"../../content/geometry/Point.h" f5c096, 19 lines

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

```
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  REP(i,0,SZ(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % SZ(ps)] - c);
  return sum;
}
```

circumcircle.h

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



"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

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

"circumcircle.h" 69dd52, 13 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};}
```

CircleCircleArea.h

Description: Calculates the area of the intersection of 2 circles

8bf2b6, 12 lines

```
template<class P>
double circleCircleArea(P c, double cr, P d, double dr) {
  if (cr < dr) swap(c, d), swap(cr, dr);
  auto A = [&](double r, double h) {
    return r*r*acos(h/r)-h*sqrt(r*r-h*h);
  };
  auto l = (c - d).dist(), a = (l*l + cr*cr - dr*dr)/(2*l);
  if (l - cr - dr >= 0) return 0; // far away
  if (l - cr + dr <= 0) return M_PI*dr*dr;
  if (l - cr >= 0) return A(cr, a) + A(dr, l-a);
  else return A(cr, a) + M_PI*dr*dr - A(dr, a-l);
}
```

7.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`
Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 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

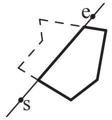
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"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" f50354, 18 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly,
    P s, P e) {
  if (SZ(poly) <= 2) return {};
  vector<P> res;
  REP(i,0,SZ(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] :
        poly.back();
    if (zero(s.cross(e, cur))) {
      res.push_back(cur);
      continue;
    }
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur,
          prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

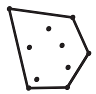
Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" a0db1d, 33 lines

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

ConvexHull.h

Description:



Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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

"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: $\mathcal{O}\left(\log N\right)$

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

"Point.h" 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 (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  REP(i,0,2) {
    int lo = endB, hi = endA, n = SZ(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n))
          / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + SZ(poly) + 1)
        % SZ(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" 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;
}
```

7.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(
     x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z)
       ; }
```

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

3dHull.h

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

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

"Point3D.h" 0754b0, 39 lines

```
// 0123456789012345678901234567890123456789
typedef Point3D<double> P3;
struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1);
      }
  int a, b;};
struct F { P3 q; int a, b, c; };
vector<F> hull3d(const vector<P3>& A) {
  assert(SZ(A) >= 4);
  vector<vector<PR>> E(SZ(A), vector<PR>(SZ(
      A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l)
      {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i])
        );
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins
        (i);
    FS.push_back(f);
  };
  REP(i,0,4) REP(j,i+1,4) REP(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);
  REP(i,4,SZ(A)) {
    REP(j,0,SZ(FS)) {F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();}}
    int nw = SZ(FS);
    REP(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf
    (f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);}}
  for (F& it : FS) if ((A[it.b] - A[it.a]).
      cross(
     A[it.c] - A[it.a]).dot(it.q) <= 0) swap(
         it.c, it.b);
  return FS;};
```

sphericalDistance.h

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

611f07, 8 lines

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

HalfPlane.h

Description: Computes the intersection of a set of half-planes. Input is given as a set of planes, facing left. Output is the convex polygon representing the intersection. The points may have duplicates and be collinear. Will not fail catastrophically if 'eps $>$ sqrt(2)(line intersection error)'. Likely to work for more ranges if 3 half planes are never guaranteed to intersect at the same point.

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

"Point.h", "sideOf.h", "lineIntersection.h" eda44b, 31 lines

```
typedef Point<double> P;
typedef array<P, 2> Line;
#define sp(a) a[0], a[1]
```

```
#define ang(a) (a[1] - a[0]).angle()

int angDiff(Line a, Line b) { return sgn(ang(a) - ang(b)); }
bool cmp(Line a, Line b) {
  int s = angDiff(a, b);
  return (s ? s : sideOf(sp(a), b[0])) < 0;
}
vector<P> halfPlaneIntersection(vector<Line> vs) {
  const double EPS = sqrt(2) * 1e-8;
  sort(all(vs), cmp);
  vector<Line> deq(sz(vs) + 5);
  vector<P> ans(sz(vs) + 5);
  deq[0] = vs[0];
  int ah = 0, at = 0, n = sz(vs);
  rep(i,1,n+1) {
    if (i == n) vs.push_back(deq[ah]);
    if (angDiff(vs[i], vs[i - 1]) == 0) continue;
    while (ah<at && sideOf(sp(vs[i]), ans[at-1], EPS) < 0)
      at--;
    while (i!=n && ah<at && sideOf(sp(vs[i]),ans[ah],EPS)<0)
      ah++;
    auto res = lineInter(sp(vs[i]), sp(deq[at]));
    if (res.first != 1) continue;
    ans[at++] = res.second, deq[at] = vs[i];
  }
  if (at - ah <= 2) return {};
  return {ans.begin() + ah, ans.begin() + at};
}
```

Mathematics (8)

8.1 Equations

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

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

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

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.

8.2 Recurrences

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

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

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

8.3 Trigonometry

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

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

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

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

8.4 Geometry

8.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

8.4.2 Quadrilaterals

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

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

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

8.5 Derivatives/Integrals

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

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

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

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

Integration by parts:

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

8.6 Sums

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

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

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

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

8.7 Series

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

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

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

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

8.8 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) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

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

For independent X and Y,

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

8.8.1 Discrete distributions

Binomial distribution

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 $\text{Fs}(p),\ 0 \leq p \leq 1$.

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

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

Poisson distribution

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

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

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

8.8.2 Continuous distributions

Uniform distribution

Exponential distribution

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

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

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

Normal distribution

Most real random values with mean μ and variance σ^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

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

Miscellaneous (9)

9.1 RNG, Intervals, T.S

TernarySearch.h

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

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

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

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;
}
```

RNGs.h

5 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,
```

DebuggingTricks.cpp

Description: Debug

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

26e792, 4 lines

```
1. 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).
2. feenableexcept(29);
kills the program on NaNs(1), 0-divs (4),
   infinities (8) and denormals (16).
```

Contest (10)

template.cpp

38 lines

```
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <bits/stdc++.h>
using namespace std;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
using o_set = tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
// order_of_key (val): no. of values less than val
// find_by_order (k): kth largest element .(0-based)
// t.join(t1) -> merges t1 with t in linear time
#define int long long
#define FOR(i, a, b) for (int i = (a); i < (b); ++i)
#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 F first
#define S second
using LL = long long;
using dbl = double;
using II = pair<int, int>;
using VI = vector<int>;
using VII = vector<II>;
using VVI = vector<VI>;
#define endl "\n"
const long long mod = 1e9 + 7;
signed main() {
  // freopen("sample.in", "r", stdin);
  // freopen("sample.out", "w", stdout);
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit); // RTE if out of bound
  return 0;
}
```