공군정보체계관리단

Team Note

778기, 788기, 804기

Based on KACTL
https://github.com/kth-competitive-programming/kactl

2022-07-22

Contest (1)

template.cpp 14 lines

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

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

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

Mathematics (2)

2.1 Geometry

2.1.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

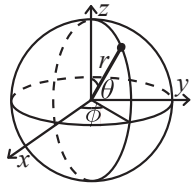
$$s_a = \sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

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

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

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

2.1.2 Spherical coordinates



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

2.2 Sums

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

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

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

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

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

2.3 Series

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

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

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

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

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

2.4 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where 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,

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

2.4.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

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

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

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

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 $\text{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

2.4.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Data structures (3)

OrderStatisticTree.h

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

782797, 16 lines

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

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

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

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).
Usage: `hash_map<int, int> table({}, {}, {}, {}, {1 << 16});`

1d6e64, 21 lines

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

struct splitmix64_hash {
  // http://xorshift.di.unimi.it/splitmix64.c
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

  size_t operator()(uint64_t x) const {
    return splitmix64(x + 0x2425260000242526);
  }
};

template <typename K, typename V>
using hash_map = __gnu_pbds::gp_hash_table<K, V,
    splitmix64_hash>;

template <typename K>
using hash_set = hash_map<K, __gnu_pbds::null_type>;
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`vector<int> vec = {1,2,3};`
`vec = (A^N) * vec;`

c43c7d, 26 lines

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

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$

8ec1c7, 30 lines

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

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

ConvexHullTrick.h

Description: Store min/max line data for using convex hull trick

aleebe, 57 lines

```
struct CHT {
    struct Line {
        ll a, b; // y = ax + b
    };

    struct R {
        ll u, d;
        bool operator<=(const R& rhs) const {
            return u * rhs.d <= rhs.u * d;
        }
    };

    vector<Line> d;
    int pos;

    void clear() {
        d.clear();
        pos = 0;
    }

    R getx(const Line& a, const Line& b) {
        ll u = b.b - a.b;
        ll d = a.a - b.a;
        ll g = gcd(u, d);
        return { u / g, d / g };
    }

    void insert(ll a, ll b) {
        Line cur = { a, b };
        while(d.size() > 1) {
            Line& b1 = d[d.size() - 1];
            Line& b2 = d[d.size() - 2];
            if(getx(cur, b1) <= getx(b1, b2)) d.pop_back();
            else break;
        }
        d.push_back(cur);
    }

    ll get(ll x) {
        R rx = { x, 1 };
        int l = 0, r = d.size() - 1;
        while(l < r) {
            int m = (l + r) / 2;
            if(getx(d[m], d[m + 1]) <= rx) l = m + 1;
            else r = m;
        }
        return d[l].a * x + d[l].b;
    }

    ll get_mono(ll x) {
        while(pos < d.size() - 1 && (d[pos].b - d[pos + 1].b) <
              x * (d[pos + 1].a - d[pos].a)) pos++;
        return d[pos].a * x + d[pos].b;
    }
```

```
};

// min value -> a decrease
// max value -> a increase
```

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

806f55, 47 lines

```
template <typename T>
class binary_indexed_tree {
  const size_t n;
  vector<T> tree;

public:
  binary_indexed_tree(size_t n) : n(n), tree(n + 1) {}
  // a[i] += val
  void update(size_t i, T val) {
    assert(0 <= i and i <= n);
    for (++i; i <= n; i += i & -i)
      tree[i] += val;
  }
  // return the sum of the range [0, i)
  T query(size_t i) const {
    assert(0 <= i and i <= n);
    T ret = 0;
    for (; i; i &= i - 1)
      ret += tree[i];
    return ret;
  }
  // return the sum of the range [l, r)
  T query(size_t l, size_t r) const {
    return query(r) - query(l);
  }
  // return a[i]
  T get(size_t i) const {
    assert(0 <= i and i < n);
    return i & 1 ? query(i, i + 1) : tree[i + 1];
  }
  // return minimum i s.t. sum[0...i] >= k
  size_t lower_bound(T k) const {
    size_t x = 0;
    for (size_t pw = 1 << 25; pw; pw >>= 1)
      if ((x | pw) <= n && tree[x | pw] < k)
        k -= tree[x |= pw];
    return x;
  }
  // return minimum i s.t. sum[0...i] > k
  size_t upper_bound(T k) const {
    size_t x = 0;
    for (size_t pw = 1 << 25; pw; pw >>= 1)
      if ((x | pw) <= n && tree[x | pw] <= k)
        k -= tree[x |= pw];
    return x;
  }
};
```

BITRange.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos], and updates elements in [l, r].
Usage: FenwickRange fw(n);
fw.update(l, r, 10);
fw.query(5);
Time: $\mathcal{O}(\log N)$.

96a6a5, 34 lines

```
struct FenwickRange
{
  int n;
  vector<ll> tmul, tadd;
  FenwickRange(int _n) : n(_n + 1), tmul(_n + 1, 0), tadd(_n +
      1, 0) {}

  void udtImpl(int pos, ll mul, ll add)
  {
    for(; pos < n; pos += (pos & -pos)) {
      tmul[pos] += mul;
      tadd[pos] += add;
    }
  }

  void update(int l, int r, ll v)
  {
    l++;
    r++;
    udtImpl(l, v, -v * (l - 1));
    udtImpl(r, -v, v * r);
  }

  ll query(int pos)
  {
    pos++;
    ll mul = 0, add = 0;
    int st = pos;
    for(; pos > 0; pos -= (pos & -pos)) {
      mul += tmul[pos];
      add += tadd[pos];
    }
    return mul * st + add;
  }
};
```

MoQueries.h

Description: Mo's query
Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

c843b2, 46 lines

```
//convert (x,y) to d
ll hilbertCurve(int x, int y, ll n) {
  ll rx, ry, s = 1, d = 0;
  while(s < n) s *= 2;
  n = s;
  for(s = n / 2; s > 0; s /= 2) {
    rx = (x & s) > 0;
    ry = (y & s) > 0;
    d += s * s * ((3 * rx) ^ ry);
    // rotate
    if(ry == 0) {
      if(rx == 1) {
        x = n - 1 - x;
        y = n - 1 - y;
      }
      swap(x, y);
    }
  }
  return d;
}

struct Query {
  int l, r, idx, h;
};

int main() {
  vector<Query> ql(m);
  for(auto& q : ql) q.h = hilbertCurve(q.l, q.r, n);
  sort(ql.begin(), ql.end(), [](const auto& l, const auto& r) {
    return l.h < r.h;
  });

  auto add = [&](int idx) {};
  auto del = [&](int idx) {};
  auto calc = [&]() -> int {};

  vector<int> res(m);
  int cl = ql[0].l, cr = ql[0].l - 1;
  for(auto [l, r, idx, _] : ql) {
    while(l < cl) add(--cl);
    while(cr < r) add(++cr);
    while(cl < l) del(cl++);
    while(r < cr) del(cr--);
    res[idx] = calc();
  }
}
```

PST.h

Description: Persistent SegTree

6f402e, 89 lines

```
struct PST
{
    using Type = ll;
    Type merge(Type l, Type r) {
        return l + r;
    }
    const Type empty = 0;

    struct Node
    {
        int l = -1, r = -1;
        Type v = empty;
    };

    vector<Node> t;
    int stLeaf;
    vector<int> root;

    PST(int n) {
        root.push_back(1);

        stLeaf = 1;
        while(stLeaf < n) stLeaf *= 2;
        t.resize(stLeaf * 2);
    }

    void initv(int idx, Type v) {
        t[stLeaf + idx].v = v;
    }

    void build() {
        for(int i = stLeaf - 1; i > 0; --i) {
            t[i].v = merge(t[i * 2].v, t[i * 2 + 1].v);
            t[i].l = i * 2;
            t[i].r = i * 2 + 1;
        }
    }

    Type queryImpl(int cl, int cr, int l, int r, int node) {
        if(l <= cl && cr <= r) return t[node].v;
        else if(cr < l || r < cl) return empty;
        int m = (cl + cr) / 2;
        return merge(queryImpl(cl, m, l, r, t[node].l),
            queryImpl(m + 1, cr, l, r, t[node].r));
    }

    Type query(int l, int r, int version) {
        return queryImpl(0, stLeaf - 1, l, r, root[version]);
    }
```

```
    void update(int idx, Type v) {
        int cl = 0, cr = stLeaf - 1;
        int node = root.back();

        int newnode = t.size();
        root.push_back(newnode);
        t.push_back(t[node]);

        while(cl != cr) {
            int m = (cl + cr) / 2;
            if(idx <= m) {
                cr = m;
                t[newnode].l = newnode + 1;
                newnode++;

                node = t[node].l;
                t.push_back(t[node]);
            } else {
                cl = m + 1;
                t[newnode].r = newnode + 1;
                newnode++;

                node = t[node].r;
                t.push_back(t[node]);
            }
        }
        t[newnode].v = v;
        newnode--;
        while(newnode >= root.back()) {
            t[newnode].v = merge(t[t[newnode].l].v, t[t[newnode
                ].r].v);
            newnode--;
        }
    }

    void remove(int numrt) {
        int rmrt = root[root.size() - numrt];
        t.erase(t.begin() + rmrt, t.end());
        root.erase(root.end() - numrt, root.end());
    }
};
```

Bulldozer.h

Description: Find maximum subarray sweepped by a line in any angle. 787c00, 48 lines

```
using point = pair<int, int>;
#define x first
#define y second
point operator-(point a, point b) { return point(a.x - b.x, a.y
    - b.y); }
long long det(point a, point b) { return 1LL * a.x * b.y - 1LL
    * a.y * b.x; }
signed main()
{
  int N; cin >> N;
  vector<pair<point, long long>> P(N);
  for(auto& it : P) cin >> it.first.x >> it.first.y >> it.
      second;
  sort(P.begin(), P.end(), [](auto& a, auto& b) {
    if(a.first.y == b.first.y) return a.first.x < b.first.x;
    return a.first.y < b.first.y;
  });
  vector<pair<int, int>> swp;
  for(int i = 0; i < N; ++i)
    for(int j = i + 1; j < N; ++j)
      swp.push_back({ i, j });
  auto comp2 = [&](pair<int, int>& a, pair<int, int>& b) ->
      bool {
    long long d = det(P[a.second].first - P[a.first].first, P[b
        .second].first - P[b.first].first);
    if(d) return d > 0;
    else
    {
      if(P[a.first].first != P[b.first].first) return P[a.first
          ].first < P[b.first].first;
      return P[a.second].first < P[b.second].first;
    }
  };
  sort(swp.begin(), swp.end(), comp2);
  vector<pair<point, long long>> srt = P;
  vector<int> pos(N); iota(pos.begin(), pos.end(), 0);
  auto comp3 = [&](pair<int, int>& a, pair<int, int>& b) ->
      long long {
    return det(P[a.second].first - P[a.first].first, P[b.second
        ].first - P[b.first].first);
  };
  for(int i = 0; i < (int)swp.size(); )
  {
    int j = i;
    while(j < (int)swp.size() && comp3(swp[i], swp[j]) == 0)
    {
      int x = swp[j].first, y = swp[j].second;
      int& px = pos[x], & py = pos[y];
      swap(srt[px], srt[py]);
      swap(px, py);
      ++j;
    }
    i = j;
  }
  return 0;
}
```

SegmentTreeBeats.h

Description: Lazy segment tree with has min/max query cfc14a, 100 lines

```
template <int N>
struct SegTree
{
    struct Node {
        int mx1, mx2, mx1cnt;
        ll sum;
    };
    Node t[N * 3];
    int stLeaf, n;

    void init(int n) {
        stLeaf = 1;
        while(stLeaf < n) stLeaf *= 2;
    }
    void initv(int idx, int v) {
        t[stLeaf + idx] = { v, -1, 1, v };
    }
    void build() {
        for(int i = stLeaf - 1; i > 0; --i) merge(i);
    }

    void merge(int node) {
        if(node >= stLeaf) return;

        Node l = t[node * 2];
        Node r = t[node * 2 + 1];
        Node& cur = t[node];

        if(l.mx1 == r.mx1) {
            cur.mx1 = l.mx1;
            cur.mx1cnt = l.mx1cnt + r.mx1cnt;
            cur.mx2 = max(l.mx2, r.mx2);
        } else {
            if(l.mx1 < r.mx1) swap(l, r);
            cur.mx1 = l.mx1;
            cur.mx1cnt = l.mx1cnt;
            cur.mx2 = max(l.mx2, r.mx1);
        }
        cur.sum = l.sum + r.sum;
    }

    void propagate(int l, int r, int node) {
        if(node >= stLeaf) return;

        Node& cur = t[node];
        for(int i = 0; i < 2; ++i) {
            int cnode = node * 2 + i;
            Node& c = t[cnode];
            if(cur.mx1 >= c.mx1) continue;

            c.sum -= c.mx1cnt * (ll)(c.mx1 - cur.mx1);
            c.mx1 = cur.mx1;
        }
    }

    void updateImpl(int cl, int cr, int l, int r, int node, int
        v) {
        propagate(cl, cr, node);
        if(cr < l || r < cl || t[node].mx1 <= v) return;
        if(l <= cl && cr <= r && t[node].mx1 > v && v > t[node
            ].mx2) {
            t[node].sum -= t[node].mx1cnt * (ll)(t[node].mx1 -
                v);
            t[node].mx1 = v;

            propagate(cl, cr, node);
            return;
        }
        int m = (cl + cr) / 2;
        updateImpl(cl, m, l, r, node * 2, v);
        updateImpl(m + 1, cr, l, r, node * 2 + 1, v);
        merge(node);
    }
    void minUpdate(int l, int r, int v) {
        updateImpl(0, stLeaf - 1, l, r, 1, v);
    }

    ll getmaxImpl(int cl, int cr, int l, int r, int node) {
        propagate(cl, cr, node);
        if(cr < l || r < cl) return 0;
        if(l <= cl && cr <= r) return t[node].mx1;
        int m = (cl + cr) / 2;
        ll res = getmaxImpl(cl, m, l, r, node * 2);
        res = max(res, getmaxImpl(m + 1, cr, l, r, node * 2 +
            1));
        return res;
    }
    ll getmax(int l, int r) {
        return getmaxImpl(0, stLeaf - 1, l, r, 1);
    }

    ll getsumImpl(int cl, int cr, int l, int r, int node) {
        propagate(cl, cr, node);
        if(cr < l || r < cl) return 0;
        if(l <= cl && cr <= r) return t[node].sum;
        int m = (cl + cr) / 2;
        ll res = getsumImpl(cl, m, l, r, node * 2);
        res += getsumImpl(m + 1, cr, l, r, node * 2 + 1);
        return res;
    }
    ll getsum(int l, int r) {
```

```
        return getsumImpl(0, stLeaf - 1, l, r, 1);
    }
};
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

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

BerlekampMassey.h

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

a6463b, 62 lines

```
constexpr ll mod = 1000000007;
ll modpow(ll b, ll e) {
  ll ans = 1;
  for(; e; b = b * b % mod, e /= 2) if(e & 1) ans = ans * b %
      mod;
  return ans;
}

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

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

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

int get_nth(const vector<ll>& rec, const vector<ll>& dp, ll n)
    {
  int m = rec.size();
  vector<ll> s(m), t(m);
  s[0] = 1;
  if(m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](const vector<ll>& v, const vector<ll>& w) {
    int m = v.size();
    vector<ll> t(2 * m);
    for(int j = 0; j < m; j++) {
      for(int k = 0; k < m; k++) {
        t[j + k] += 1ll * v[j] * w[k] % mod;
        if(t[j + k] >= mod) t[j + k] -= mod;
      }
    }
    for(int j = 2 * m - 1; j >= m; j--) {
      for(int k = 1; k <= m; k++) {
        t[j - k] += 1ll * t[j] * rec[k - 1] % mod;
        if(t[j - k] >= mod) t[j - k] -= mod;
      }
    }
    t.resize(m);
    return t;
  };
  while(n) {
    if(n & 1) s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  ll ret = 0;
  for(int i = 0; i < m; i++) ret += 1ll * s[i] * dp[i] % mod;
  return ret % mod;
}
```

4.2 Matrices

Determinant.h

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

bd5cec, 15 lines

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

IntDeterminant.h

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

3313dc, 18 lines

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

SolveLinear.h

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

44c9ab, 38 lines

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

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

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

```
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

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

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

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

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

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

MatrixInverse.h

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

ebfff6, 35 lines

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

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

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

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

MatrixInverse-mod.h

Description: Invert matrix A modulo a prime. Returns rank; result is stored in A unless singular (rank < n). 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: $\mathcal{O}(n^3)$

"../number-theory/ModPow.h" a6f68f, 36 lines

```
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(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 (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
found:
    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]);
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }

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

  rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod
        : 0);
  return n;
}
```

4.3 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 $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

00ced6, 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" b82773, 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"` ced03d, 33 lines

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

FastSubsetTransform.h

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

464cf3, 16 lines

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

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

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

`"euclid.h"` 35bfea, 18 lines

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

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

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

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

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: $\mathcal{O}\left(\sqrt{m}\right)$

c040b8, 11 lines

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

ModSum.h

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

5c5bc5, 16 lines

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

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

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

ModMulLL.h

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

bbbd8f, 11 lines

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

ModSqrt.h

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

`"ModPow.h"` 19a793, 24 lines

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

```
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than n.
Time: $\mathcal{O}(N)$

e6f974, 32 lines

```
vector<int> minFactor, mobius, primes, phi;

void initSieve(int n)
{
  minFactor.resize(n + 1, 0);
  mobius.resize(n + 1, 0);
  mobius[1] = 1;
  for(int i = 2; i <= n; i++) {
    if(minFactor[i] == 0) {
      minFactor[i] = i;
      primes.push_back(i);
    }
    for(int p : primes) {
      if(i * p > n) break;
      minFactor[i * p] = p;
      mobius[i * p] = (i % p != 0) * (-mobius[i]);
      if(i % p == 0) break;
    }
  }

  // euler phi
  phi.resize(n + 1, 0);
  iota(phi.begin(), phi.end(), 0);
  for(int i = 2; i <= n; ++i) {
    if(minFactor[i] != i) continue;
    for(int j = 1; j * i <= n; ++j) {
      phi[i * j] = (i - 1) * (phi[i * j] / i);
    }
  }
}

// phi[i] = (p^a - p^(a-1)) ...
```

PrimalityTest.h

Description: Miller-Rabin and Pollard's rho
Time: isprime(n) : $\mathcal{O}(\log n)$, factorize(n) : $\mathcal{O}\left(n^{1/4}\right)$

5bdb20, 61 lines

```
class primality_test {
  using num = unsigned long long;
  const vector<num> base_small = {2, 7, 61},
            base_large = {2, 325, 9375, 28178, 450775, 9780504,
                 1795265022};

public:
  bool is_prime(num n) const {
    if (n < 2) return false;
    if (n == 2 || n == 3) return true;
    if (n % 6 != 1 && n % 6 != 5) return false;
    const auto& base = n < 4759123141ULL ? base_small :
         base_large;
    const int s = __builtin_ctzll(n - 1);
    const num d = n >> s;
    for (const auto& b : base) {
      if (b >= n) break;
      if (check_composite(n, b, d, s)) return false;
    }
    return true;
  }
  vector<num> factorize(num n) const {
    if (n == 1) return {};
    if (is_prime(n)) return {n};
    const num x = pollard(n);
    auto l = factorize(x), r = factorize(n / x);
    decltype(l) ret(l.size() + r.size());
    merge(l.begin(), l.end(), r.begin(), r.end(), ret.begin());
    return ret;
  }

private:
  num pow_mod(num a, num p, num m) const {
    num ret = 1;
    for (; p; p >>= 1) {
      if (p & 1) ret = mul_mod(ret, a, m);
      a = mul_mod(a, a, m);
    }
    return ret;
  }
  num mul_mod(num a, num b, num m) const {
    int64_t ret = a * b - m * num(1.L / m * a * b);
    return ret + m * (ret < 0) - m * (ret >= int64_t(m));
  }
  bool check_composite(num n, num x, num d, int s) const {
    x = pow_mod(x, d, n);
    if (x == 1 || x == n - 1) return false;
    while (--s) {
      x = mul_mod(x, x, n);
      if (x == n - 1) return false;
    }
    return true;
  };
  num pollard(num n) const {
    auto f = [&](num x) { return mul_mod(x, x, n) + 1; };
    num x = 0, y = 0, prd = 2, i = 1, q;
    for (int t = 30; t++ % 40 || gcd(prd, n) == 1; x = f(x), y
        = f(f(y))) {
      if (x == y) x = ++i, y = f(x);
      if ((q = mul_mod(prd, x > y ? x - y : y - x, n))) prd = q
          ;
    }
    return gcd(prd, n);
  }
};
```

5.3 Divisibility

euclid.h

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

2ddb01, 17 lines

```
struct ENode {
  ll s, t, g;
};

ENode eeuc(ll a, ll b) {
  if(b == 0) return { 1, 0, a };

  ll r = a % b;
  auto res = eeuc(b, r);
  ll x = res.t;
  ll y = res.s - a / b * res.t;
  return { x, y, res.g };
}
```

```
// Ax+By=C, D=gcd(A, B)=g
// x0 = s * C/D          y0 = t * C/D
// s = x0 + k * B/D      t = y0 - k * A/D
```

CRT.h

Description: Chinese Remainder Theorem.

"euclid.h" 2b833d, 19 lines

```
// x = v (mod m)
struct Con {
  ll v, m;
};

Con crt(Con c1, Con c2) {
  if(c1.m < c2.m) swap(c1, c2);
  ENode en = eeuc(c1.m, c2.m);
  if((c1.v - c2.v) % en.g != 0) return { -1, -1 };

  ll c = (c2.v - c1.v) % c2.m;
  if(c < 0) c += c2.m;

  ll resm = c1.m * c2.m / en.g;
  ll resv = (en.s * c) % c2.m / en.g * c1.m + c1.v;
  resv %= resm;
  if(resv < 0) resv += resm;
  return { resv, resm };
}
```

5.4 Primes

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

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

5.5 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)

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

6.1.2 Derangements

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Binomials

BinomialCoefficient.h

Description: Finds binomial coefficient. MOD must be prime 0432e9, 40 lines

```
template <int mod>
struct BiCoeff
{
  vector<ll> fact, invfact;

  BiCoeff(int n) : fact(n + 1), invfact(n + 1) {
    fact[0] = 1;
    for(int i = 1; i <= n; ++i) fact[i] = fact[i - 1] * i % mod
        ;
    invfact[n] = pw(fact[n], mod - 2);
    for(int i = n - 1; i >= 0; --i) invfact[i] = invfact[i + 1]
         * (i + 1) % mod;
  }

  ll pw(ll a, ll b) {
    ll res = 1;
    while(b > 0) {
      if(b & 1) res = res * a % mod;
      a = a * a % mod;
      b >>= 1;
    }
    return res;
  }

  ll get(int n, int r) {
    return fact[n] * (invfact[r] * invfact[n - r] % mod) % mod;
  }

  ll get_lucas(ll n, ll r) {
    ll res = 1;
    while(n > 0 || r > 0) {
      ll a = n % mod;
      ll b = r % mod;
      res *= ncr(a, b);
      res %= mod;

      n /= mod;
      r /= mod;
    }
    return res;
  }
};
```

multinomial.h

Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$. a0a312, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

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

6.3.7 Catalan numbers

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

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

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

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

6.3.8 Catalan Convolution

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

- count of balanced parentheses sequences consisting of $n+k$ pairs of parentheses where the first k symbols are open brackets.
- ex) n=3, k=2 ((AAABBBBB

Graph (7)

7.1 Fundamentals

SPFA.h

Description: Calculates shortest paths from st in a graph that might have negative edge weights. Return empty vector if the graph has a negative cycle.

c72159, 24 lines

```
auto spfa = [&](int st) -> vector<ll> {
  vector<ll> dst(n, LNF), cycle(n, 0);
  vector<char> inq(n, false);

  queue<int> q;
  q.push(st);
  dst[st] = 0;
  while(!q.empty()) {
    int cur = q.front(); q.pop(); inq[cur] = false;

    for(auto [nxt, cost] : g[cur]) {
      if(dst[nxt] > dst[cur] + cost) {
        dst[nxt] = dst[cur] + cost;
        if(!inq[nxt]) {
          q.push(nxt);
          inq[nxt] = true;
        }
        cycle[nxt]++;
        if(cycle[nxt] > n) return {};
      }
    }
  }
  return dst;
};
```

7.2 Network flow

Dinic.h

Description: Dinic algorithm

4ac623, 58 lines

```
struct Dinic {
  struct Edge {
    ll par, nxt, cap, flow;
    Edge(ll par, ll nxt, ll cap) :par(par), nxt(nxt), cap(cap),
          flow(0) {}
  };
  ll n, src, sink;
  vector<ll> st, cur, lv;
  vector<Edge> edge;
  Dinic(ll n) :n(n), src(n + 1), sink(n + 2), st(n + 5, -1) {}
  void addEdge(ll a, ll b, ll cap) {
    edge.emplace_back(st[a], b, cap);
    st[a] = edge.size() - 1;
    edge.emplace_back(st[b], a, 0);
    st[b] = edge.size() - 1;
  }
  bool bfs() {
    lv = vector<ll>(n + 5);
    lv[src] = 1;
    queue<ll> q;
    q.push(src);
    while(!q.empty()) {
      ll idx = q.front();
      q.pop();
      for(ll i = st[idx]; i != -1; i = edge[i].par) {
        Edge& e = edge[i];
        if(!lv[e.nxt] && e.flow < e.cap) {
          lv[e.nxt] = lv[idx] + 1;
          q.push(e.nxt);
        }
      }
    }
    return lv[sink];
  }
  ll dfs(ll idx, ll flow) {
    if(idx == sink) return flow;
    for(ll& i = cur[idx]; i != -1; i = edge[i].par) {
      Edge& e = edge[i];
      if(e.flow < e.cap && lv[e.nxt] == lv[idx] + 1) {
        ll tmp = dfs(e.nxt, min(flow, e.cap - e.flow));
        if(tmp) {
          e.flow += tmp;
          edge[i ^ 1].flow -= tmp;
          return tmp;
        }
      }
    }
    return 0;
  }
  ll solve() {
    ll ret = 0;
    while(bfs()) {
      cur = st;
      ll tmp;
      while(tmp = dfs(src, INF)) ret += tmp;
    }
    return ret;
  }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow.

0ba5f2, 99 lines

```
struct MCMF
{
  struct Edge {
    int dst;
    ll c;
    ll f;
    ll cost;
    int revIdx;
  };

  vector<vector<Edge>> g;
  vector<int> pre;
  vector<Edge*> path;
  vector<char> inQ;
  vector<ll> dis;
  int n;

  MCMF(int _n) {
    n = _n;
    g.resize(n);
    pre.resize(n);
    path.resize(n);
    inQ.resize(n);
    dis.resize(n);
  }

  void addEdge(int s, int e, ll c, ll cost) {
    Edge e1 = { e, c, 0, cost, -1 };
    Edge e2 = { s, 0, 0, -cost, -1 };
    e1.revIdx = g[e].size();
    e2.revIdx = g[s].size();
    g[s].push_back(e1);
    g[e].push_back(e2);
  }

  void addFlow(Edge& e, ll f) {
    e.f += f;
    g[e.dst][e.revIdx].f -= f;
  }

  pair<ll, ll> flow(int st, int ed) {
    for(int i = 0; i < n; ++i) {
      pre[i] = -1;
      inQ[i] = false;
      dis[i] = INF;
    }
    queue<int> q;
    q.push(st);
    inQ[st] = true;
    dis[st] = 0;
    while(q.empty() == false) {
      int cur = q.front();
      q.pop();
      inQ[cur] = false;
      for(auto& nx : g[cur]) {
        int nxt = nx.dst;
        ll c = nx.c;
        ll f = nx.f;
        ll cost = nx.cost;
        if(c > f && dis[nxt] > dis[cur] + cost) {
          dis[nxt] = dis[cur] + cost;
          pre[nxt] = cur;
          path[nxt] = &nx;
          if(inQ[nxt] == false) {
            q.push(nxt);
            inQ[nxt] = true;
          }
        }
      }
    }
    if(pre[ed] == -1) return { 0, 0 };
    ll flow = LNF;
    int idx = ed;
    while(idx != st) {
      flow = min(flow, path[idx]->c - path[idx]->f);
      idx = pre[idx];
    }
    idx = ed;
    ll cost = 0;
    while(idx != st) {
      addFlow(*path[idx], flow);
      cost += path[idx]->cost * flow;
      idx = pre[idx];
    }
    return { flow, cost };
  }

  pair<ll, ll> mcmf(int st, int ed) {
    pair<ll, ll> res = { 0, 0 };
    while(1) {
      pair<ll, ll> f = flow(st, ed);
      if(f.first == 0) break;
      res.first += f.first;
      res.second += f.second;
    }

    return res;
  }
};
```

MCMF-Dinic.h

Description: Dinic-style Min-cost max-flow.

7a842c, 66 lines

```
struct MCMF {
  struct Edge {
    int nxt, cap, cost, ridx;
  };
```

```
  int src, snk;
  vector<int> dist, work;
  vector<bool> vis;
  vector<vector<Edge>> adj;
  MCMF(int n, int src, int snk) : src(src), snk(snk), dist(n +
      5), work(n + 5), vis(n + 5), adj(n + 5)
  {}
  void addedge(int st, int en, int cap, int cost) {
    adj[st].push_back({ en, cap, cost, sz(adj[en]) });
    adj[en].push_back({ st, 0, -cost, sz(adj[st]) - 1 });
  }
  bool spfa() {
    fill(all(dist), inf);
    fill(all(vis), false);
    queue<int> que;
    dist[src] = 0; que.push(src);
    while(sz(que)) {
      int cur = que.front(); que.pop();
      vis[cur] = false;
      for(auto& [nxt, cap, cost, _] : adj[cur]) {
        if(cap > 0 && dist[nxt] > dist[cur] + cost) {
          dist[nxt] = dist[cur] + cost;
          if(vis[nxt] == false) {
            que.push(nxt);
            vis[nxt] = true;
          }
        }
      }
    }
    return dist[snk] != inf;
  }
  int dfs(int cur, int flow) {
    if(cur == snk) {
      return flow;
    }
    vis[cur] = true;
    for(int& i = work[cur]; i < sz(adj[cur]); i++) {
      auto& [nxt, cap, cost, ridx] = adj[cur][i];
      if(vis[nxt] || cap == 0 || dist[nxt] != dist[cur] + cost)
            {
        continue;
      }
      int& rcap = adj[nxt][ridx].cap;
      int ret = dfs(nxt, min(flow, cap));
      if(ret) {
        cap -= ret;
        rcap += ret;
        return ret;
      }
    }
    return 0;
  }
  pi solve() {
    pi ret = pi(0, 0);
    while(spfa()) {
      fill(all(work), 0);
      for(int flow = dfs(src, inf); flow; flow = dfs(src, inf))
            {
        ret.first += flow;
        ret.second += flow * dist[snk];
      }
    }
    return ret;
  }
};
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

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

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

8b0e19, 21 lines

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

7.2.1 Circulation

정점에 수요, 공급 개념이 추가된다. 공급은 -(수요) 라고 생각하면 된다.
일단 양수 수요의 절대값 합과 음수 수요의 절댓값 합이 같아야 한다. 그 합을 D라 하자.
소스와 싱크를 만들고 소스에서 수요가 양수인 정점으로 간선을 잇고, 수요가 음수인 정점에서 싱크로 간선을 잇는다. 이제 최대 유량이 D이면 서큘레이션이 존재하는 것이다.

7.2.2 LR Flow

간선 (u, v)에 용량 R-L을 부여하고, u의 수요에 +L, v의 수요에 -L을 한다. 즉 간선에 이미 L이 흐르고 있다고 모델링을 해주는 것이다.
소스와 싱크가 있는 그래프에서 LR flow 구하기 새로운 소스와 싱크를 만든다. 기존 싱크에서 기존 소스로 용량이 무한대인 간선을 잇는다. 위 내용처럼 모델링을 해주고, 새로운 소스에서 새로운 싱크로의 최대 유량이 하한 L의 합과 같은지 보면 된다. 최대 유량을 구하고 싶다면 이렇게 구성한 유량 그래프에서 원래 소스에서 원래 싱크로 가는 유량을 찾을 수 있을 때까지 계속 찾으면 된다.

7.2.3 empty

7.3 Matching

hopcroftKarp.h

Description: Hopcroft-Karp algorithm for bipartite matching.

Time: $\mathcal{O}\left(\sqrt{V}E\right)$

f9fb28, 39 lines

```
vector<int> a(n, -1), b(m, -1), level(n);
auto initLevel = [&]() {
  queue<int> q;
  for(int i = 0; i < n; ++i) {
    if(a[i] == -1) {
      q.push(i);
      level[i] = 0;
    } else level[i] = -1;
  }
  while(!q.empty()) {
    int cur = q.front(); q.pop();
    for(int nxt : g[cur]) {
      if(b[nxt] != -1 && level[b[nxt]] == -1) {
        level[b[nxt]] = level[cur] + 1;
        q.push(b[nxt]);
      }
    }
  }
};
auto dfs = [&](auto&& self, int cur) -> bool {
  for(int nxt : g[cur]) {
    if(b[nxt] == -1 || (level[b[nxt]] == level[cur] + 1 && self
        (self, b[nxt]))) {
      a[cur] = nxt;
      b[nxt] = cur;
      return true;
    }
  }
  return false;
};
int flow = 0;
while(1) {
  initLevel();
  int f = 0;
  for(int i = 0; i < n; ++i) {
    if(a[i] == -1 && dfs(dfs, i)) f++;
  }
  if(f == 0) break;
  flow += f;
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph.

"hopcroftKarp.h" 07ace1, 21 lines

```
auto cover = [&]() {
  vector<char> visitL(n, true), visitR(m, false);
  for(int i = 0; i < n; ++i) if(a[i] != -1) visitL[i] = false;
  queue<int> q;
  for(int i = 0; i < n; ++i) if(visitL[i]) q.push(i);
  while(!q.empty()) {
    int cur = q.front();
    q.pop();
    visitL[cur] = true;
    for(int nxt : g[cur]) {
      if(!visitR[nxt] && b[nxt] != -1) {
        visitR[nxt] = true;
        q.push(b[nxt]);
      }
    }
  }
  vector<int> res;
  for(int i = 0; i < n; ++i) if(!visitL[i]) res.push_back(i);
  for(int i = 0; i < m; ++i) if(visitR[i]) res.push_back(i + n)
      ;
  return res;
};
```

7.3.1 Bipartitie Graph

Minimum Vertex Cover = 모든 간선과 인접한 최소 정점 집합의 크기 = 최대 이분 매칭
Maximum Independent Set = 어떤 두 정점도 인접하지 않은 최대 정점 집합 = minimum vertex cover 의 여집합
Minimum Path Cover = 각 경로의 정점이 겹치지 않으면서 모든 정점을 커버하는 최소 경로 집합의 크기 = 전체 정점 수 - 각 노드를 in, out으로 분할한 최대 이분매칭(indegree ≤ 1, outdegree ≤ 1 임을 이용)(매칭 수는 간선 개수와 같고 경로 개수는 정점 수 - 간선 수와 같다)
Anti-Chain(반사슬) = 반사슬 내의 어떤 두 정점도 위상이 없음
Maximum Anti-Chain의 크기 = Minimum Path Cover = vertex cover 의 여집합

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost.
Time: $\mathcal{O}(N^2M)$ 1e0fe9, 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
}
```

FastGeneralMatching.h

Description: Matching for general graphs.
Usage: Blossom b(n);
b.connect(a, b);
b.solve();
Time: $\mathcal{O}(N^3)$ 27da9b, 67 lines

```
struct Blossom {
  int n, t;
  vector<vector<int>> adj;
  vector<int> orig, par, vis, match, aux;
  queue<int> Q;
  Blossom(int n) : n{ n }, t{ 0 }, adj(n + 1), orig(n + 1), par
      (n + 1),
    vis(n + 1), match(n + 1), aux(n + 1) {}
  void connect(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v], nv = match[pv];
      match[v] = pv, match[pv] = v;
      v = nv;
    } while(u != pv);
  }
  int lca(int v, int w) {
    ++t;
    while(1) {
      if(v) {
        if(aux[v] == t) return v;
        aux[v] = t, v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      par[v] = w, w = match[v];
      if(vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a;
      v = par[w];
    }
  }
  bool bfs(int u) {
    fill(vis.begin(), vis.end(), -1), iota(orig.begin(), orig.
        end(), 0);
    Q = queue<int>(); Q.push(u), vis[u] = 0;
    while(!Q.empty()) {
      int v = Q.front(); Q.pop();
      for(int x : adj[v]) {
        if(vis[x] == -1) {
          par[x] = v, vis[x] = 1;
          if(!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        } else if(vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a), blossom(v, x, a);
        }
      }
    }
    return false;
  }
  int solve() {
    int ans = 0;
    for(int x = 1; x <= n; x++) if(!match[x]) {
      for(int y : adj[x]) if(!match[y]) {
        match[x] = y, match[y] = x;
        ++ans; break;
      }
    }
    for(int i = 1; i <= n; i++) if(!match[i] && bfs(i)) ++ans;
    return ans;
  }
};
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: scc(g, n);
sccIdx[node] or sccs({0, 1, 3}, {2, 4}, ...)
Time: $\mathcal{O}(E+V)$ b39228, 27 lines

```
vector<vi> sccs;
vi d, st, sccIdx;
int dNum;
int dfs(vector<vi>& g, int cur) {
  d[cur] = dNum++;
  st.push_back(cur);
  int ret = d[cur];
  for(int nxt : g[cur]) {
    if(sccIdx[nxt] < 0) ret = min(ret, d[nxt] ? : dfs(g, nxt));
  }
  if(ret == d[cur]) {
    int top;
    sccs.push_back({});
    auto& scc = sccs.back();
    do {
      top = st.back(); st.pop_back();
      scc.push_back(top);
      sccIdx[top] = sccs.size();
    } while(top != cur);
  }
  return ret;
}
void scc(vector<vi>& g, int n)
{
  d.assign(n, 0); sccIdx.assign(n, -1); dNum = 1;
  rep(i,0,n) if (sccIdx[i] < 0) dfs(g, i);
}
```

BCC.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each.
Time: $\mathcal{O}(E+V)$ 55f8a4, 43 lines

```
vector<vector<pair<int, int>>> bcc;
vector<int> d(n, 0), isCut(n, false);
vector<pair<int, int>> st;
int dNum;
auto dfs = [&](auto&& self, int cur, int pre) -> int {
  d[cur] = ++dNum;

  int ret = d[cur];
  for(int nxt : g[cur]) {
    if(nxt == pre) continue;

    if(d[nxt] == 0 || d[cur] > d[nxt]) {
      st.push_back({ cur, nxt });
    }

    if(d[nxt] == 0) {
      int t = self(self, nxt, cur);
      if(t >= d[cur]) {
        if(d[cur] != 0 || d[nxt] > 1) isCut[cur] = true;

        bcc.push_back({});
        vector<pair<int, int>>& cbcc = bcc.back();
        while(1) {
          auto top = st.back();
          st.pop_back();

          cbcc.push_back(top);
```

```
        if(top.first == cur) break;
      }
    }
    ret = min(ret, t);
  } else ret = min(ret, d[nxt]);
 }

 return ret;
};
for(int i = 0; i < n; ++i) {
 if(d[i] == 0) {
  dNum = 0;
  dfs(dfs, i, -1);
 }
}
// bridges: bcc[i].size() == 1
```

BCC-BridgeTree.h

Description: Create a tree that combine vertexs with same bcc based on bridges.

d2cec9, 30 lines

```
vector<int> par(n);
iota(par.begin(), par.end(), 0);
auto find = [&](int a) {
 vector<int> st;
 while(par[a] != a) {
  st.push_back(a);
  a = par[a];
 }
 for(int v : st) par[v] = a;
 return a;
};
auto uni = [&](int a, int b) {
 int ar = find(a);
 int br = find(b);
 if(ar == br) return;
 par[br] = ar;
};
for(auto& bc : bcc) {
 if(bc.size() == 1) continue;
 for(auto& p : bc) uni(p.first, p.second);
}
vector<vector<int>> g2(n);
for(auto& bc : bcc) {
 if(bc.size() != 1) continue;
 int a = find(bc[0].first);
 int b = find(bc[0].second);

 g2[a].push_back(b);
 g2[b].push_back(a);
}
```

BCC-BlockCutTree.h

Description: Create a tree that combine vertexs with same bcc based on articulation Point.

97c7e6, 40 lines

```
vector<int> idx(n);
vector<vector<int>> g2;
for(int i = 0; i < n; ++i) {
 if(isCut[i]) {
  g2.emplace_back();
  idx[i] = g2.size() - 1;
 }
}
unordered_map<ll, int> bridges;
vector<char> isUse(n, 0);
for(auto& b : bcc) {
 g2.emplace_back();
 int cur = g2.size() - 1;
 for(auto& p : b) {
  if(!isUse[p.first]) {
   if(isCut[p.first]) {
    g2[cur].emplace_back(idx[p.first]);
    g2[idx[p.first]].emplace_back(cur);
   } else idx[p.first] = cur;
   isUse[p.first] = true;
  }
  if(!isUse[p.second]) {
   if(isCut[p.second]) {
    g2[cur].emplace_back(idx[p.second]);
    g2[idx[p.second]].emplace_back(cur);
   } else idx[p.second] = cur;
   isUse[p.second] = true;
  }
 }
 if(b.size() == 1) {
  ll u = b[0].first;
  ll v = b[0].second;
  if(u > v) swap(u, v);
  bridges.insert({ u << 32 | v, cur });
 }
 for(auto& p : b) {
  isUse[p.first] = false;
  isUse[p.second] = false;
 }
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).

Usage: TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars

Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 62 lines

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

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

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

 void either(int f, int j) {
  f = max(2*f, -1-2*f);
  j = max(2*j, -1-2*j);
  gr[f].push_back(j^1);
  gr[j].push_back(f^1);
 }
 void setValue(int x) { either(x, x); }

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

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

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

// a^b == (~a|~b) & (a|b)
// a eq b == (~a|b) & (a|~b)
// a neq b == (a|b) & (~a|~b)
// a -> b == (~a|b)
// (a+b+c<=1) == (~a|~b) & (~a|~c) & (~b|~c)
```

EulerWalk.h

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

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

780b64, 15 lines

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

7.5 Coloring

7.6 Trees

HLD.h

d919c3, 40 lines

```
vector<int> dep(n), sz(n), top(n), par(n), idx(n);
vector<vector<int>> g2(n);
int num = 0;
```

```
dep[0] = 0;
sz[0] = 0;
auto dfs = [&](auto&& self, int cur, int p) -> void {
  for(int nxt : g[cur]) {
    if(nxt == p) continue;

    dep[nxt] = dep[cur] + 1;
    par[nxt] = cur;
    g2[cur].push_back(nxt);

    self(self, nxt, cur);
  }
};
dfs(dfs, 0, -1);
auto dfs2 = [&](auto&& self, int cur) -> int {
  sz[cur] = 1;
  for(int& nxt : g2[cur]) {
    sz[cur] += self(self, nxt);

    if(sz[g2[cur][0]] < sz[nxt]) swap(g2[cur][0], nxt);
  }
  return sz[cur];
};
dfs2(dfs2, 0);

top[0] = 0;
auto hld = [&](auto&& self, int cur) -> void {
  idx[cur] = num++;
  bool heavy = true;
  for(int nxt : g2[cur]) {
    top[nxt] = heavy ? top[cur] : nxt;
    self(self, nxt);
    heavy = false;
  }
};
hld(hld, 0);
```

Centroid.h

bd73f8, 40 lines

```
vector<int> sz(n);
vector<char> visit(n, false);
auto getsz = [&](auto&& self, int cur, int pre) -> int {
  sz[cur] = 1;
  for(int nxt : g[cur]) {
    if(nxt == pre || visit[nxt]) continue;
    sz[cur] += self(self, nxt, cur);
  }
  return sz[cur];
};
auto getcen = [&](auto&& self, int cur, int pre, int size) ->
    int {
  for(int nxt : g[cur]) {
    if(nxt == pre || visit[nxt]) continue;
    if(sz[nxt] > size / 2) return self(self, nxt, cur, size);
  }
  return cur;
};

auto search = [&](auto&& self, int cur, int pre) -> void {
  for(int nxt : g[cur]) {
    if(nxt == pre || visit[nxt]) continue;
    self(self, nxt, cur);
  }
};

vector<int> cenPar(n);
auto centroid = [&](auto&& self, int cur, int pre) -> void {
  int sz = getsz(getsz, cur, -1);
  int cen = getcen(getcen, cur, -1, sz);
  visit[cen] = true;
  cenPar[cen] = pre;
  search(search, cen, -1);

  for(int nxt : g[cen]) {
    if(nxt == pre || visit[nxt]) continue;
    self(self, nxt, cen);
  }
};
centroid(centroid, 0, -1);
```

ManhattanMST.h

Description: return candidate edges(w, u, v) of Manhattan MST (<= 4n)
Usage: T(distace type), U(point type)
run Kruskal's to get the 'true' ManhattanMST

69808c, 29 lines

```
template <typename T, typename U>
vector<tuple<T, int, int>> manhattan_MST(const vector<U>& a) {
  vector<int> id(a.size());
  iota(id.begin(), id.end(), 0);
  vector<tuple<T, int, int>> edges;
  edges.reserve(n << 2);
  for (int t = 0; t < 4; ++t) {
    sort(id.begin(), id.end(), [&](auto& lhs, auto& rhs) {
      return a[lhs].x - a[rhs].x < a[rhs].y - a[lhs].y;
    });
    map<T, int, greater<T>> sweep;
    for (const auto& i : id) {
      for (auto it = sweep.lower_bound(a[i].y); it != sweep.end
          (); it = sweep.erase(it)) {
        int j = it->y;
        T dx = a[i].x - a[j].x, dy = a[i].y - a[j].y;
        if (dy > dx) break;
        edges.emplace_back(dx + dy, i, j);
      }
      sweep[a[i].y] = i;
    }
    for (auto& [x, y] : a) {
      if (t & 1) {
        x = -x;
      } else {
        swap(x, y);
      }
    }
  }
}
```

7.7 Math

7.7.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--, mat[b][b]++` (and `mat[b][a]--, mat[a][a]++` if G is undirected). Remove the 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).

7.7.2 Erdős–Gallai theorem

Source: https://en.wikipedia.org/wiki/ErdTest: stress-tests/graph/erdos-gallai.cpp A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

Geometry (8)

8.1 Geometric primitives

Point.h

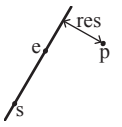
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

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

lineDistance.h

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

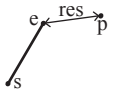


"Point.h"

f6bf6b, 4 lines

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

SegmentDistance.h



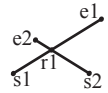
Description:
Returns the shortest distance between point p and the line segment from point s to e.
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

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



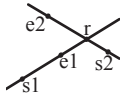
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

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

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

lineIntersection.h

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



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

"Point.h" a01f81, 8 lines

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

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: `bool left = sideOf(p1,p2,q)==1;`

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

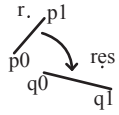
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

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



"Point.h" 03a306, 6 lines

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

Angle.h

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

0f0602, 35 lines

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

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

8.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

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

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

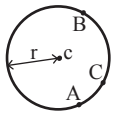
```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
```

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

circumcircle.h

Description:

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



"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

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

"circumcircle.h" 09dd0a, 17 lines

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

8.3 Polygons

InsidePolygon.h

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

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

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

PolygonArea.h

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

"Point.h" f12300, 6 lines

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

PolygonCenter.h

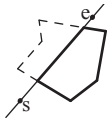
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

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

PolygonCut.h

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



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

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

ConvexHull.h

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

"Point.h" 310954, 13 lines

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

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

"Point.h" c571b8, 12 lines

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

PointInsideHull.h

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

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

```
typedef Point<ll> P;

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

LineHullIntersection.h

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

"Point.h" 7cf45b, 39 lines

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

```
  }
  return lo;
}

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

8.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" ac41a6, 17 lines

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

kdTree.h

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

"Point.h" bac5b0, 63 lines

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

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

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;
  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

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

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

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

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

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

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

FastDelaunay.h

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

"Point.h" eefdf5, 88 lines

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

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

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

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

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

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

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
```

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

8.4.1 픽의 정리

모든 꼭짓점이 격자점 위에 존재하는 단순 다각형의 넓이를 A, 격자 다각형 내부에 있는 격자점의 수를 i, 변 위에 있는 격자점의 수를 b라고 하면

$A = i + \frac{b}{2} - 1$

8.4.2 평면 그래프

꼭짓점의 수를 v, 변의 수를 e, 면의 수를 f라고 하면, 평면 그래프의 경우

$v - e + f = 2$

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

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

3dHull.h

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

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

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

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

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

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

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

sphericalDistance.h

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

611f07, 8 lines

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

Strings (9)

KMP.h

Description: KMP algorithm

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

78d1e2, 33 lines

```
struct KMP {
  vector<int> fail;

  void init_fail(const string& w) {
    int wn = w.size();
    fail.clear();
    fail.resize(wn, 0);

    for(int i = 1, j = 0; i < wn; ++i) {
      while(j > 0 && w[i] != w[j]) j = fail[j - 1];
      if(w[i] == w[j]) {
        fail[i] = j + 1;
        j++;
      }
    }
  }

  vector<int> get(const string& s, const string& w) {
    vector<int> res;
    init_fail(w);
    int sn = s.size(), wn = w.size();

    for(int i = 0, j = 0; i < sn; ++i) {
      while(j > 0 && s[i] != w[j]) j = fail[j - 1];
      if(s[i] == w[j]) j++;
      if(j == wn) {
        res.push_back(i - wn + 1);
        j = fail[j - 1];
      }
    }
    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: $\mathcal{O}(n)$

3ae526, 12 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;
}
```

Manacher.h

Description: For each position in a string, computes d1[i] = half length of longest even palindrome around pos i, d2[i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

efe46b, 31 lines

```
pair<vector<int>, vector<int>> manacher(const string& str) {
  int n = str.size();
  vector<int> d1(n), d2(n);
  int l = 0, r = -1;
  for(int i = 0; i < n; ++i) {
    int k = 1;
    if(i <= r) k = min(d1[l + r - i], r - i);
    while(0 <= i - k && i + k < n && str[i - k] == str[i + k])
        k++;
    d1[i] = --k;
    if(i + k > r) {
      r = i + k;
      l = i - k;
    }
  }

  l = 0;
  r = -1;
  for(int i = 0; i < n; ++i) {
    int k = 0;
    if(i <= r) k = min(d2[l + r - i + 1], r - i);
    while(0 <= i - k - 1 && i + k < n && str[i - k - 1] == str[
        i + k]) k++;
    d2[i] = k--;
    if(i + k > r) {
      r = i + k - 1;
      l = i - k;
    }
  }
  return { d1, d2 };
}
// d1: ab c ba -> 00200
// d2: ab b a -> 0020
```

MinRotation.h

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

d07a42, 8 lines

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

SuffixArray.h

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

d3fc67, 57 lines

```
struct SuffixArray {
  vi sa, lcp;

  vi ori, lg2;
  vector<vi> st;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim)), rank(n);
    x.push_back('\0');
    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] && a+j<n && b+j<n && 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];
        i+k<n-1 && j+k<n-1 && s[i + k] == s[j + k]; k++);

    // lcp RMQ build
    lg2.resize(n + 1);
    lg2[0] = lg2[1] = 0;
    rep(i,2,n+1) lg2[i] = lg2[i >> 1] + 1;

    ori.resize(n);
    int dep = lg2[n];
    st.resize(n);
    rep(i,0,n) {
      ori[sa[i]] = i;
      st[i].resize(dep + 1);
      st[i][0] = lcp[i];
    }
    rep(j,1,dep+1) {
      for(int i = 0; i + (1 << (j - 1)) < n; ++i) {
        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j -
            1]);
      }
    }
  }
  int get_lcp(int l, int r) {
    if(l == r) return sa.size() - l - 1;
    l = ori[l], r = ori[r];
    if(l > r) swap(l, r);
    int j = lg2[r - l];
    return min(st[l + 1][j], st[r - (1 << j) + 1][j]);
  }
};
// sa[0] = str.size(), sa.size() = str.size() + 1
// lcp[i] = lcp(sa[i - 1], sa[i]), lcp[0] = 0
```

Hashing.h

Description: Self-explanatory methods for string hashing.

3f02d8, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
  typedef uint64_t ull;
  ull x; H(ull x=0) : x(x) {}
#define OP(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 ~ 3e9; random also ok)

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

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

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

HashStr.h

Description: Get substring of hash.
Usage: `HashStr hs(str); v = hs.substr(0, 10);`
Time: $\mathcal{O}(n)$ *wheninit*, $\mathcal{O}(1)$ *toget*

ccc754, 43 lines

```
template <ll h1 = 3137, ll m1 = 998244353, ll h2 = 53, ll m2 =
    1610612741>
struct HashStr {
  vector<ll> hv, hpow;
  vector<ll> hv2, hpow2;

  HashStr(const string& str) {
    int n = str.size();
    hv.resize(n);
    hpow.resize(n);

    hv[0] = str[0];
    hpow[0] = 1;
    for(int i = 1; i < n; ++i) {
      hv[i] = (hv[i - 1] * h1 + str[i]) % m1;
      hpow[i] = (hpow[i - 1] * h1) % m1;
    }

    hv2.resize(n);
    hpow2.resize(n);

    hv2[0] = str[0];
    hpow2[0] = 1;
    for(int i = 1; i < n; ++i) {
```

```
      hv2[i] = (hv2[i - 1] * h2 + str[i]) % m2;
      hpow2[i] = (hpow2[i - 1] * h2) % m2;
    }
  }

  // [l, r]
  ll substr(int l, int r) {
    ll res = hv[r];
    if(l > 0) {
      res -= hv[l - 1] * hpow[r - l + 1];
      res = ((res % m1) + m1) % m1;
    }
    ll res2 = hv2[r];
    if(l > 0) {
      res2 -= hv2[l - 1] * hpow2[r - l + 1];
      res2 = ((res2 % m2) + m2) % m2;
    }
    return res << 32 | res2;
  }
};
```

AhoCorasick.h

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

f35677, 66 lines

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

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

Various (10)

10.1 Intervals

IntervalContainer.h

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

edce47, 23 lines

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

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

IntervalCover.h

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

9e9d8d, 19 lines

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

10.2 Misc. algorithms

Random.h

6f1e10, 3 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
uniform_int_distribution<int> distribution(0, INT_MAX);
int getRandom() { return distribution(rng); }
```

TernarySearch.h

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

9155b4, 11 lines

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

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) {
  if (S.empty()) return {};
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
```

```
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

MaxQueryDeque.h

Description: Get longest segment that range max value - min value <= k.
Time: $\mathcal{O}(N)$

14142e, 25 lines

```
long long n, k;
vector<int> arr(n);

int l = 0;
int ans = 0;
deque<int> uq, bq;

auto insert = [&](int idx) {
  while (uq.size() && arr[uq.back()] <= arr[idx]) uq.pop_back()
      ;
  while (bq.size() && arr[bq.back()] >= arr[idx]) bq.pop_back()
      ;
  uq.push_back(idx), bq.push_back(idx);
};

auto del = [&](int idx) {
  if (uq.front() == idx) uq.pop_front();
  if (bq.front() == idx) bq.pop_front();
};

for(int i=0;i<n;++i) {
  insert(i);
  while (arr[uq.front()] - arr[bq.front()] > k) del(l++);
  ckmax(ans, i - l + 1);
}

// return ans;
```

10.3 Dynamic programming

KnuthDP.h

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

DnCOptimization.h

Description: Divide and Conquar Optimization DP example

b07a40, 25 lines

```
auto dnc = [&](auto&& self, int l, int r, int minj, int maxj) {
  if(l > r) return;

  int i = (l + r) / 2;

  int midj = minj;
  ll mn = INF;
  int ed = min(maxj, i);
  for(int j = minj; j <= ed; ++j) {
    ll v = dp[1][j] + (sum[i] - sum[j]) * (i - j);
    if(mn > v) {
      mn = v;
      midj = j;
    }
  }
  dp[0][i] = mn;
  self(self, l, i - 1, minj, midj);
  self(self, i + 1, r, midj, maxj);
};

for(int i = 0; i < g; ++i) {
  dnc(dnc, 1, n, 0, n);
  swap(dp[0], dp[1]);
  fill(dp[0].begin() + 1, dp[0].end(), INF);
}
```

AlienTrick.h

Description: Alien Trick.

5358f2, 41 lines

```
// 즉 이분탐색을 돌렸을 때 최종적으로 기울기 a를 얻지 못 할 수 있지만,
     기울기 a를 반드시 한 번거치게 된다. 그 때의 접선 mid*x + mx에 대하여
     x=K 일 때의 값 mid*K + mx가 답이 되고, 이것은다른 어떤 접선에서의
     x=K 일 때의 값보다 작거나 같으므로 answer = inf 로 놓고 answer
     =min(answer, mid*K + mx) 를 반복하면 결국 답을 구할 수 있게 되는
     것이다.// 반대로 볼록함수 + 최소값 구하기이면 연산할 때마다 mid를
     더한다고 생각하고 x=K 일 때 접선의값이 항상 크거나 같으므로 answer =
     -inf로 놓고 answer = max(answer, -mid*K + mn) 를 반복하면된다.

signed main()
{
  int N; int K; cin >> N >> K;
  vector<int> A(N);
  for(auto& it : A) cin >> it;
  vector<int> suf(N + 1);
  for(int i = N - 1; i >= 0; --i)
    suf[i] = suf[i + 1] + A[i];
  int lf = 0, rg = 1e16;
  int ans = inf;
  while(lf <= rg)
  {
    int mid = (lf + rg) / 2;
    vector<int> dp(N + 1);
    vector<int> cnt(N + 1);
    vector<pair<int, int>> memo(N + 1);
    pair<int, int> mx = make_pair(-mid, 0);
    for(int i = N - 1; i >= 0; --i)
    {
      dp[i] = suf[i] + mx.first;
      cnt[i] = mx.second + 1;
      memo[i] = max(memo[i + 1], make_pair(dp[i], cnt[i]));
      mx = max(mx, make_pair(memo[i].first - mid - suf[i], memo
          [i].second));
    }
    ans = min(ans, memo[0].first + mid * K);
    if(cnt[0] <= K) rg = mid - 1;
    else lf = mid + 1;
  }
  cout << ans << '\n';
  return 0;
}
```

10.4 Debugging tricks

10.5 Optimization tricks

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

10.5.1 Bit hacks

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

10.5.2 Pragmas

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

FastMod.h

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

751a02, 8 lines

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

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.

7b3c70, 17 lines

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

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

appendix (A)

A.1 Mobius Example

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=d]$$

$$\sum_{a=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{b=1}^{\lfloor\frac{m}{d}\rfloor}[gcd(a,b)=1] \qquad (i=ad, j=bd)$$

$$\sum_{a=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{b=1}^{\lfloor\frac{m}{d}\rfloor}\sum_{d|gcd(a,b)}\mu(d)$$

$$\sum_{a=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{b=1}^{\lfloor\frac{m}{d}\rfloor}\sum_{k=1}^{p}[k|gcd(a,b)]\mu(k) \qquad (p=min(\lfloor\frac{n}{d}\rfloor,\lfloor\frac{m}{d}\rfloor))$$

$$\sum_{a=1}^{\lfloor\frac{n}{d}\rfloor}\sum_{b=1}^{\lfloor\frac{m}{d}\rfloor}\sum_{k=1}^{p}[k|a][k|b]\mu(k)$$

$$\sum_{k=1}^{p}\mu(k)\sum_{a=1}^{\lfloor\frac{n}{d}\rfloor}[k|a]\sum_{b=1}^{\lfloor\frac{m}{d}\rfloor}[k|b]$$

$$\sum_{k=1}^{p}\mu(k)\lfloor\frac{n}{kd}\rfloor\lfloor\frac{m}{kd}\rfloor$$

A.2 Temp

FindCycle.h

Description: Description: simple cycle detection algorithm. implemented as non-reculsive way. return the vector of vertices on cycle. note that first and last vertex are repeated i.e. cycle.front() == cycle.back(). 4ec3e0, 39 lines

```
vector<int> find_cycle(vector<vector<int>>& adj) {
    vector<char> colour(adj.size());
    vector<int> cycle, eid;
    cycle.reserve(adj.size());
    eid.reserve(adj.size());
    auto dfs = [&](int u) -> void {
        colour[u] = 'g';
        cycle.emplace_back(u);
        eid.emplace_back(0);
        while (!cycle.empty()) {
            for (auto &u = cycle.back(), &i = eid.back();; ++i)
                {
                if (i == int(adj[u].size())) {
                    colour[u] = 'b';
                    cycle.pop_back();
                    eid.pop_back();
                    break;
                } else if (!colour[adj[u][i]]) {
                    colour[adj[u][i]] = 'g';
                    cycle.emplace_back(adj[u][i]);
                    eid.emplace_back(0);
                    break;
                } else if (colour[adj[u][i]] == 'g') {
                    cycle.emplace_back(adj[u][i]);
                    return;
                }
            }
        }
    };
    for (int i = 0; i < int(adj.size()); ++i) {
        if (!colour[i]) {
            dfs(i);
            if (!cycle.empty()) {
                cycle.erase(cycle.begin(), find(cycle.begin(),
                    cycle.end(), cycle.back()));
                return cycle;
            }
        }
    }
    return {};
};
```