Universitas Indonesia

YouR Lovely JatengPRiDe

Thanks to our handsome members:

- Ahmad Haulian Yoga Pratama (Yoga Segtree),
- Muhammad Salman Al-Farisi (God Salman),
- Muhammad Yusuf Sholeh (Ucup Imba).

ACM-ICPC Regional Jakarta 2018

Nov 11, 2018

Contest (1)

Contents.txt 10 lines

```
1. Contest
2. Mathematics
3. Data structures
4. Numerical
5. Number Theory
6. Combinatorial
7. Graph
8. Geometry
9. Strings
10. Various
```

template.cpp 15 lines

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

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

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

compile.bat 1 lines

```
g++ -std=c++11 -Wall -O2 -o %1 %1.cpp -Wl,--stack,268435456
```

Mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

$$\sin(v+w)=\sin v\cos w+\cos v\sin w$$
$$cos(v+w)=\cos v\cos w-\sin v\sin w$$

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a,b,c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

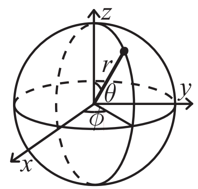
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

$$(n+1)^{k+1} - 1 = \sum_{m=1}^{n} ((m+1)^{k+1} - m^{k+1})$$

$$\sum_{m=1}^{n} ((m+1)^{k+1} - m^{k+1}) = \sum_{p=0}^{k} \binom{k+1}{p} (1^p + 2^p + ... + n^p)$$

$$\sum_{k=0}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{k=0}^{n} k^5 = \frac{2n^6+6n^5+5n^4-n^2}{12}$$

$$\sum_{k=0}^{n} kx^k = \frac{x-(n+1)x^{n+1}+nx^{n+2}}{(x-1)^2}$$

$\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$ $\quad$ $\sum_{k=0}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$

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

$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$ $\quad$ $\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k} = F(n+1)$

$\sum_{j=0}^{m} \binom{m}{j}^2 = \binom{2m}{m}$ $\quad$ $\sum_{i=0}^{n} i\binom{n}{i}^2 = \frac{n}{2}\binom{2n}{n}$

$\sum_{i=0}^{n} i^2\binom{n}{i}^2 = n^2\binom{2n-2}{n-1}$ $\quad$ $\sum_{k=q}^{n} \binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$

$\sum_{k=-a}^{a} (-1)^k\binom{2a}{k+a}^3 = \frac{(3a)!}{a!^3}$

$$\sum_{k=-a}^{a} (-1)^k \binom{a+b}{a+k}\binom{b+c}{b+k}\binom{c+a}{c+k} = \frac{(a+b+c)!}{a!b!c!}$$

2.7 Series

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

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

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

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

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

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

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

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

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

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

Normal distribution

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

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

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

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

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

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>, <ext/rope> 24 lines

```
using namespace __gnu_pbds;
using namespace __gnu_cxx;

typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
ordered_set S;
// S.find_by_order(x) -> return pointer to the x-th element
// (int)S.order_of_key(x) -> return the position of lower_bound
    (x)

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);
  cout<<*X.find_by_order(1)<<endl;          // array index ke-1
  cout<<(end(X)==X.find_by_order(6))<<endl; // end(X) = pointer
  cout<<X.order_of_key(400)<<endl;          // idx lower_bound
     400
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

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;

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.
Usage: For minimum: change m,c to negative.
Then, the result of the query change to -result.
Time: $\mathcal{O}(\log N)$

35 lines

```
const ll is_query = -(1LL<<62);
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct HullDynamic : public multiset<Line> { // will maintain
    upper hull for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
        if (z == end()) return 0;
        return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m -
        x->m);// beware overflow!
  }
  void insert_line(ll m, ll b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }
  ll eval(ll x) { // becareful when there is no line returned.
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
};
```

Treap.h

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

55 lines

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

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

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

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

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

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

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

ImplicitAVL.h

Description: Implicit AVL. Delete array at index id1, and insert at indices id2. Preorder to restore the array.
Time: $\mathcal{O}(Q \log N)$

145 lines

```
int n, x, id1, id2;
struct node{
  int key, height, size;
  struct node* left;
  struct node* right;
};

inline int getH(struct node *cur){
  if (cur == NULL) return 0;
  return cur -> height;
}

inline int getB(struct node* cur) {
  if (cur == NULL) return 0;
  return getH(cur->left) - getH(cur->right);
}

inline int getS(struct node* cur) {
  if (cur == NULL) return 0;
  return cur -> size;
}

inline void updateH(struct node* cur) {
  int a = getH(cur->left);
  int b = getH(cur->right);
  cur->height = ((a > b) ? a : b) +1;
}
```

```
inline void updateS(struct node* cur) {
  cur->size = 1 + getS(cur->left) + getS(cur->right);
}

inline struct node* newnode(int key) {
  struct node* cur = new node();
  cur -> key = key;
  cur -> left = NULL;
  cur -> right = NULL;
  cur -> height = 1;
  cur -> size = 1;
  return cur;
}

inline struct node* RotateRight(struct node* y) {
  struct node* x = y->left;
  struct node* xR = x->right;
  x->right = y;
  y->left = xR;
  updateH(y); updateH(x);
  updateS(y); updateS(x);
  return x;
}

inline struct node* RotateLeft(struct node* y) {
  struct node* x = y->right;
  struct node* xL = x->left;
  x->left = y;
  y->right = xL;
  updateH(y); updateH(x);
  updateS(y); updateS(x);
  return x;
}

struct node* minval(struct node* cur) {
  struct node* proc = cur;
  while(proc->left != NULL) proc=proc->left;
  return proc;
}

int value;

struct node* refresh(struct node* cur) {
  if (cur == NULL) return cur;
  updateH(cur); updateS(cur);
  int balance = getB(cur);
  if (balance > 1 && getB(cur->left) >= 0) return RotateRight(
      cur);
  if (balance > 1 && getB(cur->left) <  0) {cur->left =
      RotateLeft(cur->left); return RotateRight(cur);}
  if (balance <-1 && getB(cur->right) <= 0) return RotateLeft(
      cur);
  if (balance <-1 && getB(cur->right) > 0) {cur->right =
      RotateRight(cur->right); return RotateLeft(cur);}
  return cur;
}
struct node* DKTHS(struct node* cur, int K, bool left) {
  if (left) {
    if (cur -> left != NULL) cur -> left = DKTHS(cur -> left, K
        , left);
    else {
      struct node* temp = cur -> left? cur->left : cur -> right
          ;
      if (temp == NULL) {
        temp = cur;
        cur = NULL;
      } else *cur = *temp;
      free(temp);
    }
  } else if (cur == NULL) return cur;
  else if (K <= getS(cur->left)) cur -> left = DKTHS(cur->left,
      K, 0);
  else if (K == getS(cur->left) + 1) {
    value = cur->key;
    if ((cur->left==NULL)||(cur->right==NULL)) {
      struct node* temp = cur -> left? cur->left : cur -> right
          ;
      if (temp == NULL) {
        temp = cur;
        cur = NULL;
      } else *cur = *temp;
      free(temp);
    } else {
      node* temp = minval(cur->right);
      cur->key = temp->key;
      cur->right = DKTHS(cur->right, cur->key, 1);
    }
  } else cur -> right = DKTHS(cur->right, K - getS(cur->left) -
      1, 0);
  return refresh(cur);
}
struct node* KTHS(struct node* cur, int K, bool left) {
  if (cur == NULL) return newnode(value);
  if (left) {
    if (cur == NULL) return newnode(value);
    else cur -> left = KTHS(cur->left, K, 1);
  }
  else if (K <= getS(cur->left)) cur -> left = KTHS(cur->left,
      K, 0);
  else if (K == getS(cur->left) + 1) {
    if (cur -> right == NULL) cur->right = KTHS(cur->right,K,0)
        ;
    else cur->right = KTHS(cur->right, K, 1);
  } else cur -> right = KTHS(cur->right, K - getS(cur->left) -
      1, 0);
  return refresh(cur);
}
int cnt(0);
void preOrder(struct node* cur) {
  if (cur == NULL) return;
  preOrder(cur->left);
  A[++cnt] = cur->key;
  // printf("%d ",cur->key);
  preOrder(cur->right);
}

int main() {
  scanf("%d %d",&n, &x);
  struct node* root = NULL;
  for(int i = 1; i <= n; i++) {
    value = i;
    root = KTHS(root, i, 0);
  }
  for(int i = 1; i <= x; i++) {
    scanf("%d %d",&id1, &id2);
    root = DKTHS(root, id1, 0);
    root = KTHS(root, id2 - 1, 0);
  }
}
```

AVL.h

Description: Not Implicit. insert, delete, count, kth

42 lines

```
node* insert(node* cur, int key) {
  if (cur == NULL) return newnode(key);
  if (key == cur -> key) return cur;
  if (key < cur -> key) cur -> left = insert(cur->left, key);
  else cur -> right = insert(cur->right, key);
  return balance(cur);
}

node* Delete(node* cur, int key) {
  if (cur == NULL) return cur;
  if (key < cur -> key) cur->left = Delete(cur->left, key);
  else if (key > cur -> key) cur->right = Delete(cur->right,
      key);
  else {
    if ((cur->left==NULL)||(cur->right==NULL)) {
      node* temp = cur -> left? cur->left : cur -> right;
      if (temp == NULL) {
        temp = cur;
        cur = NULL;
      } else *cur = *temp;
      free(temp);
    } else {
      node* temp = minval(cur->right);
      cur -> key = temp -> key;
      cur -> right = Delete(cur->right, temp->key);
    }
  }
  if (cur == NULL) return cur;
  updateH(cur); updateS(cur);
  return balance(cur);
}

int KTHS(node* cur, int K) {
  if (K <= getS(cur->left)) return KTHS(cur->left, K);
  else if (K == getS(cur->left) + 1) return cur->key;
  else return KTHS(cur->right, K - getS(cur->left) - 1);
}

int COUNT(node* cur, int K) { // how many <= K
  if (cur == NULL) return 0;
  if (K < cur->key) return COUNT(cur->left, K);
  else return getS(cur->left) + 1 + COUNT(cur->right, K);
}
```

RMQ.h

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

Usage: RMQ rmq(values);
rmq.query(inclusive, exclusive);

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

17 lines

```
template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) {
    int N = sz(V), on = 1, depth = 1;
    while (on < sz(V)) on *= 2, depth++;
    jmp.assign(depth, V);
    rep(i,0,depth-1) rep(j,0,N)
      jmp[i+1][j] = min(jmp[i][j],
      jmp[i][min(N - 1, j + (1 << i))]);
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

Numerical (4)

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`

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

14 lines

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

Polynomial.h

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: `poly_roots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`

Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

"Polynomial.h" 23 lines

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

PolyInterpolate.h

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

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

13 lines

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

BerlekampMassey.h

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

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

"../number-theory/ModPow.h" 20 lines

```
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());
  trav(x, C) x = (mod - x) % mod;
  return C;
}
```

LinearRecurrence.h

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

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

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

26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(S);

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

Integrate.h

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

8 lines

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

IntegrateAdaptive.h

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

Usage: `double z, y;`
`double h(double x) { return x*x + y*y + z*z <= 1; }`
`double g(double y) { ::y = y; return quad(h, -1, 1); }`
`double f(double z) { ::z = z; return quad(g, -1, 1); }`
`double sphereVol = quad(f, -1, 1), pi = sphereVol*3/4;`

16 lines

```
typedef double d;
d simpson(d (*f)(d), d a, d b) {
  d c = (a+b) / 2;
  return (f(a) + 4*f(c) + f(b)) * (b-a) / 6;
}
d rec(d (*f)(d), d a, d b, d eps, d S) {
  d c = (a+b) / 2;
  d S1 = simpson(f, a, c);
  d S2 = simpson(f, c, b), T = S1 + S2;
  if (abs (T - S) <= 15*eps || b-a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps/2, S1) + rec(f, c, b, eps/2, S2);
}
d quad(d (*f)(d), d a, d b, d eps = 1e-8) {
  return rec(f, a, b, eps, simpson(f, a, b));
}
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.

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

15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
```

```
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

IntDeterminant.h

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

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case. 68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

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

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

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

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

SolveLinear.h

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

SolveLinear2.h

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

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}(n^2 m)$ 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 $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$ 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;
}
```

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

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

4.1 Fourier transforms

FastFourierTransform.h

Description: Computes $\hat{f}(k) = \sum_x f(x) \exp(-2\pi ikx/N)$ for all k. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. For convolutions of integers, consider using a number-theoretic transform instead, to avoid rounding issues. This works safely when the cofficient result is under 10^{14}.
Time: $\mathcal{O}(N \log N)$ 43 lines

```
const double PI = acos(-1);
typedef complex<double> base;
void fft (vector<base> & a, bool invert) {
  int n = (int) a.size();
  for (int i=1, j=0; i<n; ++i) {
    int bit = n >> 1;
    for (; j>=bit; bit>>=1)
      j -= bit;
    j += bit;
    if (i < j)
      swap (a[i], a[j]);
  }
  for (int len=2; len<=n; len<<=1) {
    double ang = 2*PI/len * (invert ? -1 : 1);
    base wlen (cos(ang), sin(ang));
    for (int i=0; i<n; i+=len) {
      base w (1);
      for (int j=0; j<len/2; ++j) {
        base u = a[i+j],  v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert)
    for (int i=0; i<n; ++i)
      a[i] /= n;
}
void multiply (const vector<int> &a, const vector<int> &b,
    vector<int> &res){
  vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end()
      );
  size_t n = 1;
  while (n < max (a.size(), b.size()))  n <<= 1;
  n <<= 1;
  fa.resize (n),  fb.resize (n);
  fft (fa, false),  fft (fb, false);
  for (size_t i=0; i<n; ++i)
    fa[i] *= fb[i];
  fft (fa, true);
  res.resize (n);
  for (size_t i=0; i<n; ++i)
    res[i] = int (fa[i].real() + 0.5); // <- beware if res[i]
         negative, floor function retard
}
```

NumberTheoreticTransform.h

Description: Can be used for convolutions modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$ 63 lines

```
// just check if g^((p-1)/r) % p != 1 for every possible r (r =
      prime factor of p-1)
// instead of find g such that g^1, g^2, g^3, ... g^p-1 are the
      permutation of {1, 2, 3, ..., p-1}.
bool check_generator_version_langrage(ll g) {
  for(int i = 1; i <= top; i++) {
    ll value = power(g, (p-1) / prime_factors[i]);
    if (value == 1) return false;
  }
  return true;
}
const int mod = 7340033; // c * 2^k + 1
const ll root = 5; // root = g ^ c % mod
const ll root_1 = 4404020; // root_l = (root)^-1 % mod
const ll root_pw = 1<<20; // root_pw = (1 << k)
int rev[7340033];
ll getmod(ll a, ll tmod) {return ((a%tmod)+tmod)%tmod;}
void fft (vector<ll> & a, bool invert) {
  int n = (int) a.size();
  for (int i=1, j=0; i<n; ++i) {
    int bit = n >> 1;
    for (; j>=bit; bit>>=1)
      j -= bit;
    j += bit;
    if (i < j)
      swap (a[i], a[j]);
  }
  for (int len=2; len<=n; len<<=1) {
    ll wlen = invert ? root_1 : root;
    for (int i=len; i<root_pw; i<<=1)
      wlen = ll (wlen * 1ll * wlen % mod);
    for (int i=0; i<n; i+=len) {
      ll w = 1;
      for (int j=0; j<len/2; ++j) {
        ll u = a[i+j],  v = ll (a[i+j+len/2] * 1ll * w % mod);
        a[i+j] = getmod(u+v,mod);
        a[i+j+len/2] = getmod(u-v,mod);
        w = ll (w * 1ll * wlen % mod);
      }
    }
```

```
  }
  if (invert) {
    for (int i=0; i<n; ++i)
      a[i] = int (a[i] * 1ll * rev[n] % mod);
  }
}
void precalc(){ // calculate inverse of MOD in O(MOD)
  rev[1] = 1;
  for (int i=2; i<mod; i++)
    rev[i] = (mod - (mod/i) * rev[mod%i] % mod) % mod;
}
void multiply (const vector<ll> & a, const vector<ll> & b,
    vector<ll> & res) {
  vector <ll> fa (a.begin(), a.end()),  fb (b.begin(), b.end())
      ;
  size_t n = 1;
  while (n < max (a.size(), b.size()))  n <<= 1;
  n <<= 1;
  fa.resize (n),  fb.resize (n);
  fft (fa, false),  fft (fb, false);
  forn(i,n)
    fa[i] *= fb[i];
  fft (fa, true);
  res.resize (n);
  forn(i,n) // for(i=0;i<n;)
    res[i] = fa[i] % mod;
}
```

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: $\mathcal{O}(N \log N)$ 48 lines

```
void FWHT(vi &P,int bits ,bool inverse = false) { //FWHT xor on
      vector P
  int x = 1<<bits ,u,v;
    for (int len = 1; 2 * len <= x ; len <<= 1)
        for (int i = 0; i < x; i += 2 * len)
            for (int j = 0; j < len; j++){
                u = P[i + j] ;
                v = P[i + len + j] ;
                P[i + j] = (u + v)%mod ;
                P[i + len + j] = (u - v + mod)%mod ;
            }
    if (inverse){
  int xinv = pmod(x,mod-2) ;
  for (int i = 0; i < x ; i++) P[i] = ((ll)P[i]*(ll)xinv)%mod ;
    }
}
FWHT(P,16);
  FN(i,1<<16) P[i] = pmod(P[i],N) ; //pmod(x,y)=x^y
FWHT(P,16,true);
void to_transform(ll dim, ll *data) { // and transform
    ll len, i, j, u, v;
    for (len = 1; 2 * len <= dim; len <<= 1) {
        for (i = 0; i < dim; i += 2 * len) {
            for (j = 0; j < len; j++) {
                u = data[i + j];
                v = data[i + len + j];
                data[i + j] = v;
                data[i + len + j] = (u + v);
                moddo(data[i + len + j]);
            }
        }
    }
}
void inv_transform(ll dim, ll *data) {
    ll len, i, j, u, v;
    for (len = 1; 2 * len <= dim; len <<= 1) {
        for (i = 0; i < dim; i += 2 * len) {
            for (j = 0; j < len; j++) {
                u = data[i + j];
                v = data[i + len + j];
                data[i + j] = mod - u + v;
                data[i + len + j] = u;
                moddo(data[i + j]);
            }
        }
    }
}
And matrices : {0 1; 1 1;} Or matrices : {1 1; 1 0}
inv And : {-1 1; 1 0}  inv Or : {0 1; 1 -1}
```

Number theory (5)

5.1 Modular arithmetic

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. 19 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) {
    ull to2 = (to * k + c) / m;
    res += to * to2;
    res -= divsum(to2, m-1 - c, m, k) + to2;
  }
  return res;
}

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);
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$
"ModPow.h" 30 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);
  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;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.2 Primality

MillerRabinAndPollardRho.h

Description: Miller-Rabin primality probabilistic test. if $n <$ 3,825,123,056,546,413,051, it is enough to test a = 2, 3, 5, 7, 11, 13, 17, 19, and 23. This pollard rho and miller rabin both works for $n < 10^{18}$
Time: 15 times the complexity of $a^b \mod c$. 62 lines

```
vector<long long> A({2, 3, 5, 7, 11, 13, 17, 19, 23});

long long largemul(long long a, long long b, long long n) {
  // assert(0 <= a && a < n && 0 <= b && b < n);
  long long r = 0;
  for (; b; b >>= 1, a <<= 1) {
    if (a >= n) a -= n;
    if (b & 1) {
      r += a;
      if (r >= n) r -= n;
    }
  }
  return r;
}

long long fastexp(long long a, long long b, long long n) {
  // assert(0 <= a && a < n && b >= 0);
  long long ret = 1;
  for (; b; b >>= 1, a = largemul(a, a, n))
    if (b & 1) ret = largemul(ret, a, n);
  return ret;
}

bool mrtest(long long n) {
  if (n == 1) return false;
  long long d = n-1;
  int s = 0;
  while ((d & 1) == 0) {
    s++;
    d >>= 1;
  }
  s--;
  if (s < 0) s = 0;
  for (int j = 0; j < (int)A.size(); j++) {
    if (A[j] >= n) continue;
    long long ad = fastexp(A[j], d, n);
    if (ad == 1) continue;
    bool notcomp = false;
    long long a2rd = ad;
    for (int r = 0; r <= s; r++) {
      if(a2rd == n-1) {notcomp = true; break;}
      a2rd = largemul(a2rd, a2rd, n);
    }
    if (!notcomp) {
      return false;
```

```
    }
  }
  return true;
}
long long gcd(long long a, long long b) { return a ? gcd(b % a,
      a) : b; }
long long pollard_rho(long long n) {
  int i = 0, k = 2;
  long long x = 3, y = 3; // random seed = 3, other values
       possible
  while (1) {
    i++;
    x = largemul(x, x, n)-1; // generating function
    if (x < 0) x += n;
    long long d = gcd(llabs(y - x), n); // the key insight
    if (d != 1 && d != n) return d;
    if (i == k) y = x, k <<= 1;
  }
}
```

5.3 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

8 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

// beware overflow when get the actual result (x * c/gcd). Use
     bezout identity to make the result small. (by using mod b/
     gcd(a,b))
ll euclid(ll a, ll b, ll &x, ll &y) {
  if (b) { ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d; }
  return x = 1, y = 0, a;
}
```

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

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

10 lines

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

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

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$

21 lines

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

FracBinarySearch.h

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

24 lines

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

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

5.5 Chinese remainder theorem

chinese.h

Description: Chinese Remainder Theorem.
Time: $\log(m + n)$

"euclid.h" 22 lines

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y)
     .
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}
// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
  PII ret = make_pair(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, x[i
          ], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}
```

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.7 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. 664 579 primes under 10^7, 5 761 455 primes under 10^7, 50.847.534 primes under 10^9.

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

5.8 Estimates

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

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

5.9 Mobius Inversion

5.9.1 Definition

$$\mu(n) = \begin{cases} 0 & n\, isn't\, square\, free \\ 1 & if\, n\, is\, equal\, to\, 1 \\ (-1)^k & n\, has\, k\, distinct\, prime \end{cases}$$

If f ang d are arithmetic function satisfying

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

Then

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

5.9.2 Example

Find:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} gcd(i,j)$$

Can be transformed into:

$$G = \sum_{g=1}^{n} h(g) * cnt(g)$$

In this example, $h(g) = g$. Present $h(g)$ as mobius with:

$$rep(i,1,n) rep(j,i,n,i) f[j] = h[i] * \mu(\frac{j}{i})$$

Then:

$$G = \sum_{g=1}^{n} (\sum_{d|g} f(d)) * cnt(g)$$

$$G = \sum_{d=1}^{n} f(d) * cnt2(d)$$

here $cnt2(d) =$ how many d such that $gcd(i,j)$ is multiple of d. This is easy to count:

$$G = \sum_{d=1}^{n} f(d) \binom{\frac{n}{d}}{2}$$

$f(d)$ is happened to be totient here. Note that we can calculate f using 2 for loops above.

5.9.3 Example 2

Find $triplet(i,j,k)$ such that $gcd(a[i],a[j],a[k]) = 1$.
Same as step above, but $h(g) = 1$ only if $g = 1$ else $h(g) = 0$.
After some calculation, $cnt2(d) = gcd(a[i],a[j],a[k])$ is multiple of d.
If $dp[x]$ is the number of i such that $x|a[i]$, then $cnt2[d] = C(dp[x],3)$

5.9.4 Example 3

Find out sum of $lcm(x,y)$ for each pair (x,y) in range $(1,n)$
Answer:

$$g(l) = \sum_{d|l} \mu(d) l d$$

$$f(n) = \sum_{l=1}^{n} \binom{\frac{n}{l}+1}{2}^2 g(l)$$

5.9.5 Helper $O(2\sqrt{n})$

Loop through different values of n/d.
for(int $i = 1; la; i <= n; i = la + 1$) la = n / (n / i);
n/x yields the same value for $i <= x <= la$

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.)
Time: $\mathcal{O}(n)$ 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Binomials

binomialModPrime.h

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.
Time: $\mathcal{O}(\log_p n)$ 9 lines

```
// in case MOD is small yet a and b can be larger than it
int comb (int a,int b) {
  int res = 1;
  while(a > 0 && b > 0) { // beware of a%mod < b%mod, should
      return 0
    res = (res * C[a % MOD][b % MOD]) % MOD;
    a /= MOD; b /= MOD;
  }
  return res;
}
```

6.3 General purpose numbers

6.3.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.2 Eulerian numbers

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

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

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

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

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.4 Bell numbers

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

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

6.3.5 Catalan numbers

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

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

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

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

Graph (7)

7.1 Fundamentals

bellmanFord.h

Description: Calculates shortest path in a graph that might have negative edge distances. Propagates negative infinity distances (sets dist = -inf), and returns true if there is some negative cycle. Unreachable nodes get dist = inf.
Time: $\mathcal{O}(EV)$ 27 lines

```
typedef ll T; // or whatever
struct Edge { int src, dest; T weight; };
struct Node { T dist; int prev; };
struct Graph { vector<Node> nodes; vector<Edge> edges; };

const T inf = numeric_limits<T>::max();
bool bellmanFord2(Graph& g, int start_node) {
  trav(n, g.nodes) { n.dist = inf; n.prev = -1; }
  g.nodes[start_node].dist = 0;

  rep(i,0,sz(g.nodes)) trav(e, g.edges) {
    Node& cur = g.nodes[e.src];
    Node& dest = g.nodes[e.dest];
    if (cur.dist == inf) continue;
    T ndist = cur.dist + (cur.dist == -inf ? 0 : e.weight);
    if (ndist < dest.dist) {
      dest.prev = e.src;
      dest.dist = (i >= sz(g.nodes)-1 ? -inf : ndist);
    }
  }
  bool ret = 0;
  rep(i,0,sz(g.nodes)) trav(e, g.edges) {
    if (g.nodes[e.src].dist == -inf)
      g.nodes[e.dest].dist = -inf, ret = 1;
  }
  return ret;
}
```

7.2 Euler walk

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm.
Usage: For eulerian path, should pass cur with odd degree to eulerian().
Time: $\mathcal{O}(E)$ where E is the number of edges. 21 lines

```
void eulerian(int cur){
  stack<int> st;
  vector<int> ans;
  st.push(cur);
  //V is multiset
  while(!st.empty()){
    int cur = st.top();
    if(V[cur].size()){
      auto it = V[cur].begin();
      st.push(*it);
      V[cur].erase(it);
      //use this for bidirectional graph
      //if(V[*it].count(cur)){
      //  V[*it].erase(V[*it].find(cur));
      //}
    }else{
      ans.pb(cur);
      st.pop();
    }
  }
}
```

7.3 Network flow

MinCostMaxFlow.h

Description: Min-cost max-flow. Not handling negative cycle. When there is negative cycle, there is no answer for MCMF.
Time: Approximately $\mathcal{O}(E^2)$ 70 lines

```
struct edge{
  int to, rev;
  int flow, cap;
  int cost;
};
vector<edge> G[500];
inline void add(int s, int t, int capa, int costs) {
  edge a = {t, G[t].size(), 0, capa,  costs};
  edge b = {s, G[s].size(), 0,    0, -costs};
  G[s].push_back(a);
  G[t].push_back(b);
}
inline bool SPFA() {
  for(int i = 0; i <= sink; i++) dist[i] = INF, flag[i] = false
      , bt[i] = -1, idx[i] = -1;
  dist[source] = 0;
  queue<int> q;
```

```
  q.push(source);
  flag[source] = true;
  while(!q.empty()) {
    int now = q.front();
    q.pop();
    flag[now] = false;
    int size = G[now].size();
    for(int i = 0; i < size; i++) {
      int to = G[now][i].to;
      int cost = G[now][i].cost;
      int capa = G[now][i].cap;
      if (capa > 0 && dist[to] > dist[now] + cost) {
        dist[to] = dist[now] + cost;
        bt[to] = now;
        idx[to] = i;
        if (!flag[to]) {
          flag[to] = true;
          q.push(to);
        }
      }
    }
  }
  return bt[sink] != -1;
}

pair<int,int> MCMF() {
  pair<int,int> res; res.first = 0, res.second = 0;
  while(true) {
    if (!SPFA()) break;
    int mins = INF;
    int ptr = sink;
    int total = 0;
    while(ptr != source) {
      int from = bt[ptr];
      int id = idx[ptr];
      if (G[from][id].cap < mins)
        mins = G[from][id].cap;
      total += G[from][id].cost;
      ptr = from;
    }
    res.first += mins;
    res.second += total * mins;
    ptr = sink;
    while(ptr != source) {
      int from = bt[ptr];
      int id = idx[ptr];
      int rev = G[from][id].rev;
      G[from][id].cap -= mins;
      G[ptr][rev].cap += mins;
      ptr = from;
    }
  }
  return res;
}
```

Dinic.h

Description: Flow algorithm with guaranteed complexity $O(V^2E)$. 59 lines

```
struct edge{
  int to, rev;
  int flow, cap;
};
vector<edge> G[MAXE];
inline void add(int s, int t, int cap) {
  edge a = {t, G[t].size(), 0, cap};
  edge b = {s, G[s].size(), 0, 0};
  G[s].push_back(a);
  G[t].push_back(b);
}
inline bool search() {
  for(int i = 0; i <= n + 1; i++) dist[i] = -1;
  dist[source] = 0;
  int tail = 0;
  q[tail] = source;
  for(int head = 0; head <= tail; head++) {
    int u = q[head];
    int sz = G[u].size();
    for(int i = 0; i < sz; i++) {
      int v = G[u][i].to;
      if (dist[v] < 0 && G[u][i].flow < G[u][i].cap) {
        dist[v] = dist[u] + 1;
        q[++tail] = v;
      }
    }
  }
  return dist[sink] >= 0;
}
int dinic(int now, int flo) {
  if (now == sink)
    return flo;
  int size = G[now].size();
  for(int &i = work[now]; i < size; i++) {
    int to = G[now][i].to, flow = G[now][i].flow, cap = G[now][
       i].cap, rev = G[now][i].rev;
    if (flow >= cap) continue;
    if (dist[to] == dist[now] + 1) {
      int fflow = dinic(to, min(flo, cap - flow));
      if (fflow) {
        G[now][i].flow += fflow;
        G[to][rev].flow -= fflow;
        return fflow;
      }
    }
  }
  return 0;
}
inline int maxflow() {
  int ans = 0;
  while(search()) {
    for(int i = 0; i <= n + 1; i++) work[i] = 0;
    while(true) {
      int res = dinic(source, INF);
      if (res == 0) break;
      ans += res;
    }
  }
  return ans;
}
```

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: Stoer-Wagner. Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$ 31 lines

```
pair<int, vi> GetMinCut(vector<vi>& weights) {
  int N = sz(weights);
  vi used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    vi w = weights[0], added = used;
    int prev, k = 0;
    rep(i,0,phase){
      prev = k;
      k = -1;
      rep(j,1,N)
        if (!added[j] && (k == -1 || w[j] > w[k])) k = j;
      if (i == phase-1) {
        rep(j,0,N) weights[prev][j] += weights[k][j];
        rep(j,0,N) weights[j][prev] = weights[prev][j];
        used[k] = true;
        cut.push_back(k);
        if (best_weight == -1 || w[k] < best_weight) {
          best_cut = cut;
          best_weight = w[k];
        }
      } else {
        rep(j,0,N)
          w[j] += weights[k][j];
        added[k] = true;
      }
    }
  }
  return {best_weight, best_cut};
}
```

7.4 Matching

hopcroftKarp.h

Description: Find a maximum matching in a bipartite graph.
Usage: node from 1..n || 0 is NIL
left side 1..n || right side n+1..n+m
G = {0} U {1..n} U {n+1..n+m}
Time: $\mathcal{O}\left(\sqrt{V}E\right)$ 47 lines

```
bool bfs() {
  queue<int> q;
  for(int i = 1 ; i <= n ; i++)
    if(match[i] == 0) {
      dist[i] = 0;
      q.push(i);
    }
    else
      dist[i] = INF;
  dist[0] = INF;

  while(!q.empty()) {
    int cur = q.front();
    q.pop();
    if(cur) {
      for(int nex : adj[cur]) {
        if(dist[match[nex]] == INF) {
          dist[match[nex]] = dist[cur] + 1;
          q.push(match[nex]);
        }
      }
    }
  }
  return dist[0] != INF;
}
int dfs(int now) {
  if(now == 0) return 1; // found 1 augmenting path
  for(int nex : adj[now]) {
    if(dist[match[nex]] == dist[now] + 1 && dfs(match[nex])) {
      match[nex] = now;
      match[now] = nex;
      return 1;
    }
```

```
  }
  dist[now] = INF;
  return 0;
}
int hopcroftKarp() {
  int ret = 0;
  memset(match, 0, sizeof match);
  while(bfs()) {
    for(int i = 1 ; i <= n ; i++)
      if(match[i] == 0)
        ret += dfs(i);
  }
  return ret;
}
```

DFSMatching.h

Description: This is a simple matching algorithm but should be just fine in most cases. Graph g should be a list of neighbours of the left partition. n is the size of the left partition and m is the size of the right partition. If you want to get the matched pairs, $match[i]$ contains match for vertex i on the right side or -1 if it's not matched.
Time: $\mathcal{O}(EV)$ where E is the number of edges and V is the number of vertices.

24 lines

```
vi match;
vector<bool> seen;
bool find(int j, const vector<vi>& g) {
  if (match[j] == -1) return 1;
  seen[j] = 1; int di = match[j];
  trav(e, g[di])
    if (!seen[e] && find(e, g)) {
      match[e] = di;
      return 1;
    }
  return 0;
}
int dfs_matching(const vector<vi>& g, int n, int m) {
  match.assign(m, -1);
  rep(i,0,n) {
    seen.assign(m, 0);
    trav(j,g[i])
      if (find(j, g)) {
        match[j] = i;
        break;
      }
  }
  return m - (int)count(all(match), -1);
}
```

Hungarian.h

Description: Min cost bipartite matching. Negate costs for max cost. Becareful of INF.
Usage: `k = max(N,M) where N is left and M is right.`
Time: $\mathcal{O}(N^3)$

41 lines

```
const int INF = 1e9;
int a[N][N],u[N],v[N],ans[N],minv[N],p[N],way[N];
bool used[N];
int Assign(){
  for(int i = 1 ; i <= k ; i++){
    p[0] = i;
    int j0 = 0;
    for(int j = 1 ; j <= k ; j++)
      minv[j] = INF,used[j] = 0;
    do{
      used[j0] = 1;
      int i0 = p[j0],delta = INF, j1;
      for(int j = 1 ; j <= k ; j++)
        if(!used[j]){
          int cur = val[i0][j] - u[i0] - v[j];
          if(cur < minv[j])
            minv[j] = cur,way[j] = j0;
          if(minv[j] < delta)
            delta = minv[j], j1 = j;
        }
      for(int j = 0 ; j <= k ; j++)
        if(used[j])
          u[p[j]] += delta,  v[j] -= delta;
        else
          minv[j] -= delta;
      j0 = j1;
    }while(p[j0] != 0);

    do{
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    }while(j0);
  }
  for(int i = 1 ; i <= k ; i++)
    ans[p[i]] = i;
  int ret = 0;
  for(int i = 1 ; i <= k ; i++)
    ret += val[i][ans[i]]; // i is matched with job ans[i]
  return ret;
}
```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}(N^3)$

87 lines

```
int lca(vector<int>&match, vector<int>&base, vector<int>&p,int
     a,int b){
  vector<bool> used(SZ(match));
  while(true){
    a = base[a];
    used[a]=true;
    if(match[a]==-1)break;
    a = p[match[a]];
  }
  while(true){
    b = base[b];
    if(used[b])return b;
    b = p[match[b]];
  }
  return-1;
}
void markPath(vector<int>&match, vector<int>&base, vector<bool
     >&blossom, vector<int>&p,int v,int b,int children){
  for(; base[v]!= b; v = p[match[v]]){
    blossom[base[v]]= blossom[base[match[v]]]=true;
    p[v]= children;
    children = match[v];
  }
}
int findPath(vector<vector<int>>&graph, vector<int>&match,
     vector<int>&p,int root){
  int n = SZ(graph);
  vector<bool> used(n);
  FORIT(it, p)*it =-1;
  vector<int> base(n);
  for(int i =0; i < n;++i) base[i]= i;
  used[root]=true;
  int qh =0;
  int qt =0;
  vector<int> q(n);
  q[qt++]= root;
  while(qh < qt){
    int v = q[qh++];
    FORIT(it, graph[v]){
      int to =*it;
      if(base[v]== base[to]|| match[v]== to)continue;
      if(to == root || match[to]!=-1&& p[match[to]]!=-1){
        int curbase = lca(match, base, p, v, to);
        vector<bool> blossom(n);
        markPath(match, base, blossom, p, v, curbase, to);
        markPath(match, base, blossom, p, to, curbase, v);
        for(int i =0; i < n;++i){
          if(blossom[base[i]]){
            base[i]= curbase;
            if(!used[i]){
              used[i]=true;
              q[qt++]= i;
            }
          }
        }
      }elseif(p[to]==-1){
        p[to]= v;
        if(match[to]==-1)return to;
        to = match[to];
        used[to]=true;
        q[qt++]= to;
      }
    }
  }
  return-1;
}
int maxMatching(vector<vector<int>> graph){
  int n = SZ(graph);
  vector<int> match(n,-1);
  vector<int> p(n);
  for(int i =0; i < n;++i){
    if(match[i]==-1){
      int v = findPath(graph, match, p, i);
      while(v !=-1){
        int pv = p[v];
        int ppv = match[pv];
        match[v]= pv;
        match[pv]= v;
        v = ppv;
      }
    }
  }
  int matches = 0;
  for(int i = 0; i < n;++i){
    if(match[i]!=-1){
      ++matches;
    }
  }
  return matches/2;
}
```

MinimumVertexCover.h

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

"DFSMatching.h" 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  int res = dfs_matching(g, n, m);
  seen.assign(m, false);
  vector<bool> lfound(n, true);
  trav(it, match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
```

```
    lfound[i] = 1;
    trav(e, g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

7.5 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$

24 lines

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

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

TarjanAPandBridge.h

Description: Finds Articulation point and bridge
Time: $\mathcal{O}(E+V)$

35 lines

```
void tarjanAPB(int u){
  dlow[u] = dnum[u] = nxt++;
  for ( int i = 0; i < adlis[u].size(); i++ ){
      int v = adlis[u][i];
      if ( dnum[v] == -1 ) {
    dpar[v] = u;
    if ( u == dfs_root ) child_root++;
    tarjanAPB(v);
    if ( dlow[v] >= dnum[u] ) {
      isAP[u] = true;
    }
    if ( dlow[v] > dnum[u] ) {
      is_bridge[u][v] = true;
    }
    dlow[u] = min(dlow[u], dlow[v]);
      }
      else if ( v != dpar[u] ) {
    dlow[u] = min(dlow[u], dnum[v]);
      }
  }
}...
nxt=0;
RESET(dnum,-1);
RESET(dlow,-1);
RESET(dpar,-1);
RESET(isAP,0);
RESET(is_bridge,0);
for ( int i=0; i < nvert; i++ ){
  if ( dnum[i] == -1 ){
      dfs_root = i;
      child_root = 0;
      tarjanAPB(i);
      is_AP[dfs_root] = (child_root > 1);
  }
}
```

BiconnectedComponents.h

Description: Ntar isinya comps itu vector of vector setiap vector jadi satu komponen, kalok dia AP maka dia jadi edge yang menghubungkan komponen yang mempunyai AP tersebut.
Time: $\mathcal{O}(E+V)$

52 lines

```
void dfs(int now,int par){
  sudah[now]=true;
  disc[now]=low[now]=++idx;
  int anak=0;
  stk.pb(now);
  for(int i:g[now]){
    if(i==par)continue;
    if(!sudah[i]){
      dfs(i,now);
      anak++;
      low[now]=min(low[now],low[i]);
      if(low[i]>=disc[now]){
        comps.pb({now});
        while(comps.back().back()!=i){
          comps.back().pb(stk.back());
          stk.pop_back();
        }
      }
      if(now==1 && anak>1)
        ap[now]=true;
      if(now!=1 && low[i]>=disc[now])
        ap[now]=true;
    }
    else low[now]=min(low[now],disc[i]);
  }
}

int main(){
  dfs(1,0);
  idx=0;
  for(auto i:comps){
    idx++;
    for(int j:i){
      if(ap[j]){
        ve[j].pb(idx);
      }
      else{
        di[j]=idx;
      }
    }
  }
  for(int i=1;i<=n;i++){
    if(ap[i]){
      di[i]=++idx;
      ya[idx]=true;
      for(int j:ve[i]){
        G[idx].pb(j);
        G[j].pb(idx);
      }
    }
  }
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`
`ts.set_value(2); // Var 2 is true`
`ts.at_most_one({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.

57 lines

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

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

  int add_var() { // (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^1].push_back(j);
    gr[j^1].push_back(f);
  }
  void set_value(int x) { either(x, x); }

  void at_most_one(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = add_var();
      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);
    trav(e, gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    ++time;
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = time;
      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;
  }
};
```

7.6 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Possible optimization: on the top-most recursion level, ignore 'cands', and go through nodes in order of increasing degree, where degrees go down as nodes are removed.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

12 lines

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

7.7 Trees

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected. Can also find the distance between two nodes.
Usage: `LCA lca(undirGraph);`
`lca.query(firstNode, secondNode);`
`lca.distance(firstNode, secondNode);`
Time: $\mathcal{O}(N \log N + Q)$

"../data-structures/RMQ.h" 37 lines

```
typedef vector<pii> vpi;
typedef vector<vpi> graph;

struct LCA {
  vi time;
  vector<ll> dist;
  RMQ<pii> rmq;

  LCA(graph& C) : time(sz(C), -99), dist(sz(C)), rmq(dfs(C)) {}

  vpi dfs(graph& C) {
    vector<tuple<int, int, int, ll>> q(1);
    vpi ret;
    int T = 0, v, p, d; ll di;
    while (!q.empty()) {
      tie(v, p, d, di) = q.back();
      q.pop_back();
      if (d) ret.emplace_back(d, p);
      time[v] = T++;
      dist[v] = di;
      trav(e, C[v]) if (e.first != p)
        q.emplace_back(e.first, v, d+1, di + e.second);
    }
    return ret;
  }

  int query(int a, int b) {
    if (a == b) return a;
    a = time[a], b = time[b];
    return rmq.query(min(a, b), max(a, b)).second;
  }
  ll distance(int a, int b) {
    int lca = query(a, b);
    return dist[a] + dist[b] - 2 * dist[lca];
  }
};
```

CompressTree.h

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

"LCA.h" 20 lines

```
vpi compressTree(LCA& lca, const vi& subset) {
  static vi rev; rev.resize(sz(lca.dist));
  vi li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(all(li), cmp);
  int m = sz(li)-1;
  rep(i,0,m) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.query(a, b));
  }
  sort(all(li), cmp);
  li.erase(unique(all(li)), li.end());
  rep(i,0,sz(li)) rev[li[i]] = i;
  vpi ret = {pii(0, li[0])};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.query(a, b)], b);
  }
  return ret;
}
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

101 lines

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

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

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    make_root(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    make_root(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void make_root(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    // destroy right child
    if (u->c[1]) { u->c[1]->p = 0; u->c[1]->pp = u; }
    u->c[1]=0;
    u->fix();
```

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

  // use this to aggregate:
  int aggregate(int a, int b) {
    make_root(&node[a]);
    return access(&node[b])->aggr;
  }
};
```

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--`. Remove the last row and column, and take the determinant.

Geometry (8)

8.1 Geometric primitives

Point.h

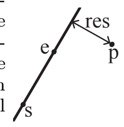
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

25 lines

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

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.



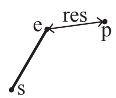
"Point.h" 11 lines

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

// return the distance and point at c, cannot 3D
double lineDist(const P& a, const P& b, const P& p, P& c) {
  double u = (p-a).dot(b-a) / (b-a).dist2();
  c = a + ((b - a) * u);
  return (c - p).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;`

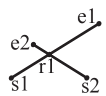
"Point.h" 20 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) { // beware overflow! better
     use the other one
  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;
}

// return the distance and point at c, cannot 3D:
double segDist(const P& a, const P& b, const P& p, P& c) {
  double u = (p-a).dot(b-a) / (b-a).dist2();
  if (u < 0.0) {
    c = a;
    return (p - a).dist();
  } else if (u > 1.0) {
    c = b;
    return (p - b).dist();
  }
  c = a + ((b - a) * u);
  return (c - p).dist();
}
```

SegmentIntersection.h

Description:
If a unique intersetion point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> 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. Use segmentIntersectionQ to get just a true/false answer.



Usage: `Point<double> intersection, dummy;`
`if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)`
`cout << "segments intersect at " << intersection << endl;`

"Point.h" 27 lines

```
template<class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*(a2/a); // beware overflow 3 point product!
  return 1;
}
```

SegmentIntersectionQ.h

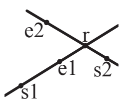
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 16 lines

```
template<class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
         b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
}
```

lineIntersection.h

Description:
If a unique intersetion point of the lines going through s1,e1 and s2,e2 exists r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If s1==e1 or s2==e2 -1 is returned. The wrong position will be returned if P is Point<int> 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: `point<double> intersection;`
`if (1 == LineIntersection(s1,e1,s2,e2,intersection))`
`cout << "intersection point at " << intersection << endl;`

"Point.h" 9 lines

```
template<class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
```

```
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
```

sideOf.h

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

"Point.h" 11 lines

```
template<class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
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

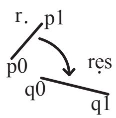
Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" 5 lines

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

RotationalSweeping.h

Description: Rotational sweeping based on origin. Remove the origin. Useful for many calculations such as total area of possible triangle.
Time: $\mathcal{O}(N \log N)$

65 lines

```
ll cross2(point p, point r) {
  return p.x * r.y - p.y * r.x;
}

// 2 points seen on the same angle will be sorted based on
     closest to origin.
bool cmp(point a, point b) {
  if (a.y == 0 && b.y == 0) {
    if (a.x * b.x >= 0) return a.x*a.x+a.y*a.y < b.x*b.x+b.y*b.
        y;
    return a.x > b.x; // x+ first
  }
  if (cross2(a,b) == 0) return a.x*a.x+a.y*a.y < b.x*b.x+b.y*b.
      y;
  return cross2(a, b) > 0;
}

// return a vector rotated ccw, from positive x-axis, and id
     being removed from vector.
vector<point> rotational_sweep(vector<point> &all, int id) {
  vector<point> up, low;
  for(int i = 0; i < sz(all); i++) if (i != id) {
    point p = all[i];
    p.x -= all[id].x, p.y -= all[id].y;
    if (p.y >= 0) up.push_back(p); // point on y==0 will
         treated on top
    else low.push_back(p);
  }
  sort(up.begin(), up.end(), cmp);
  sort(low.begin(), low.end(), cmp);
  vector<point> res = up;
  for(auto cur : low) res.push_back(cur);
  return res;
}

// apply this : last podal lie on the same line.
// note : true if segment a..b contain origin, or a.dist2() < b
     .dist2()
inline bool collinear_sweep(point a, point b) {
  if (cross2(a,b) != 0) return false;
  if ((a.x < 0 && b.x > 0) || (a.y < 0 && b.y > 0)) return true
      ;
  if ((a.x == 0 && a.y == 0) || (b.x == 0 && b.y == 0)) return
      true;
  return a.x*a.x+a.y*a.y < b.x*b.x+b.y*b.y;
}

void process(vector<point> &cur, int id) { // total area of
     possible triangle:
  vector<point> rot = rotational_sweep(cur, id);
  int size = sz(rot);

  ll sum_rx = rot[0].x, sum_ry = rot[0].y;

  // do rotate with i being the podal, r with the last podal
  // important: check whether there is collinear (:check the
       quadrant!)
  for(int i = 0, r = 0; i < size; i++) {
    while(cross2(rot[i], rot[(r+1)%size]) > 0 ||
      collinear_sweep(rot[i], rot[(r+1)%size])) { // toggle
           this

      r = (r + 1)%size;
      // do something
      sum_rx += rot[r].x, sum_ry += rot[r].y;
    }
    // process i..r:
    ans += rot[i].x * sum_ry - rot[i].y * sum_rx;
    // after process i..r, do cancel:
    sum_rx -= rot[i].x, sum_ry -= rot[i].y;
    if (i == r) {
      r = (i+1)%size;
      sum_rx = rot[r].x, sum_ry = rot[r].y;
    }
  }
}
```

8.2 Circles

CircleIntersection.h

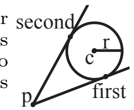
Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 14 lines

```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

circleTangents.h

Description:



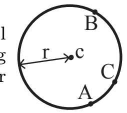
Returns a pair of the two points on the circle with radius r centered around c whos tangent lines intersect p. If p lies within the circle NaN-points are returned. P is intended to be Point<double>. The first point is the one to the right as seen from the p towards c.
Usage: `typedef Point<double> P;`
`pair<P,P> p = circleTangents(P(100,2),P(0,0),2);`

"Point.h" 6 lines

```
template<class P>
pair<P,P> circleTangents(const P &p, const P &c, double r) {
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
}
```

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" 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" 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  rep(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
```

```
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  rep(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
}
```

8.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).
Usage: `typedef Point<int> pi;`
`vector<pi> v; v.push_back(pi(4,4));`
`v.push_back(pi(1,2)); v.push_back(pi(2,1));`
`bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);`
Time: $\mathcal{O}(n)$

"Point.h", "onSegment.h", "SegmentDistance.h" 14 lines

```
template<class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
}
```

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" 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;
}
```

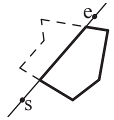
PolygonCenter.h

Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
}
```

PolygonCut.h

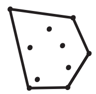


Description:
Returns a vector with the vertices of a polygon with everything to the left of the segment going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" 15 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.emplace_back();
      lineIntersection(s, e, cur, prev, res.back());
    }
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h



Description:
Returns a vector of indices of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Usage: `vector<P> ps, hull;`
`trav(i, convexHull(ps)) hull.push_back(ps[i]);`
Time: $\mathcal{O}(n \log n)$

"Point.h" 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].cross(\
  S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
}
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.

"ConvexHull.h" 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
  return ans.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "onSegment.h" 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(), p);
  else return insideHull2(hull, 1, sz(hull), p);
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. isct(a, b) 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.
Time: $\mathcal{O}(N + Q \log n)$

"Point.h" 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
```

```
  vector<P> p;
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps) {
    p.insert(p.end(), all(ps));
    int b = 0;
    rep(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
    rep(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
        hi = mid;
      else lo = mid;
    }
    return a[hi%N].second;
  }

  bool isign(P a, P b, int x, int y, int s) {
    return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
  }

  int bs2(int lo, int hi, P a, P b) {
    int L = lo;
    if (hi < lo) hi += N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (isign(a, b, mid, L, -1)) hi = mid;
      else lo = mid;
    }
    return lo;
  }

  pii isct(P a, P b) {
    int f = bs(a - b), j = bs(b - a);
    if (isign(a, b, f, j, 1)) return {-1, -1};
    int x = bs2(f, j, a, b)%N,
        y = bs2(j, f, a, b)%N;
    if (a.cross(p[x], b) == 0 &&
        a.cross(p[x+1], b) == 0) return {x, x};
    if (a.cross(p[y], b) == 0 &&
        a.cross(p[y+1], b) == 0) return {y, y};
    if (a.cross(p[f], b) == 0) return {f, -1};
    if (a.cross(p[j], b) == 0) return {j, -1};
    return {x, y};
  }
};
```

8.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(n \log n)$

"Point.h" 58 lines

```
template<class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template<class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y;}

template<class It, class IIt> /* IIt = vector<It>::iterator */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist()
        ;
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
}
```

kdTree.h

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

"Point.h" 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 the box is wider than high (not best
          heuristic...)
      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);
  }
};
```

VoronoiDiagram.h

Description: Voronoi Diagram using Half Plane Intersection. For each point, do process.
Time: $\mathcal{O}(n^2 \log n)$

<Point.h>, <RotationalSweep.h> 60 lines

```
struct line{ LD m, c; P where; bool straight; };
line make_line(P p) {
  line ret; ret.where = p;
```

```
  if (fabs(p.y)<EPS) ret.straight = true, ret.c = p.x/2.0, ret.
      m = 1e9;
  else {
    ret.straight = false, ret.m = ((LD)-p.x)/p.y;
    ret.c = (-ret.m/2.0) * p.x + p.y/2.0;
  }
  return ret;
}
bool fine(line p, line q, line r) { // pot_p_q has different
     region with (0,0) seen from r
  LD px, py;
  if (p.where.cross(q.where)<=EPS||q.where.cross(r.where)<=EPS)
       return true;
  if (p.straight) px = p.c, py=q.m*p.c+q.c;
  else if (q.straight) px = q.c, py=p.m*q.c+p.c;
  else {px=(p.c-q.c)/(q.m-p.m); py=p.m*px+p.c;}
  if (r.straight && fabs(r.c-px) < EPS) return false;
  if (!r.straight && fabs(py-(r.m*px+r.c)) < EPS) return false;
  return r.straight? !((r.c+EPS>=0)^(r.c+EPS>=px)):!((0<=EPS+r.
      c)^(py-r.m*px<=EPS+r.c));
}
void process(vector<point> &po, int id) { // rotational up: x
     ==0+ to x>0-. dn: x==0- to x<0+
  vector<point> rot = rotational_sweep(po, id); // if collinear
     , put largest distance first
  deque<line> res;
  for(point curr: rot) {
    line cur = make_line(curr);
    P bef = sz(res)>=1?res[sz(res)-1].where:P(0,0), aft = cur.
        where;
    if (sz(res)>=1 && bef.cross(aft) < -EPS) { res.push_back(
        cur); continue; }
    if (sz(res)>=1 && fabs(bef.cross(aft)) <= EPS &&
       fabs(bef.dist()+aft.dist()-(aft-bef).dist()) > EPS)
        res.pop_back();
    while(sz(res) >= 2 && !fine(res[sz(res)-2], res[sz(res)-1],
        cur))
      res.pop_back();
    res.push_back(cur);
  }
  while(sz(res)>=3) {
    bool e = false;
    while(sz(res)>=3&&!fine(res[sz(res)-2],res[sz(res)-1],res
        [0])) res.pop_back(),e=true;
    while(sz(res)>=3&&!fine(res.back(), res[0], res[1])) res.
        pop_front(),e=true;
    if (!e) break;
  }
  vector<P> v; res.push_back(res[0]);
  for(int i = 0; i < sz(res)-1; i++) {
    line cur = res[i], nxt = res[i+1];
    if (cur.where.cross(nxt.where) <= EPS) {
      LD zz = max(fabs(cur.where.x), fabs(cur.where.y));
      LD zz2 = max(fabs(nxt.where.x), fabs(nxt.where.y));
      zz = (zz > MAX) ? 1.0 : ceil(MAX/zz);
      zz2 = (zz2 > MAX) ? 1.0 : ceil(MAX/zz2);
      P mid1 = (cur.where/2.0).perp() * zz + (cur.where/2.0);
      P mid2 = (nxt.where/2.0).perp().perp().perp() * zz2 + (
          nxt.where/2.0);
      v.push_back(mid1+po[id]), v.push_back(mid2+po[id]);
    } else {
      P pot; // assert(!cur.straight || !nxt.straight);
      if (cur.straight) pot.x = cur.c, pot.y = nxt.m*cur.c+nxt.
          c;
      else if (nxt.straight) pot.x = nxt.c, pot.y = cur.m*nxt.c
          +cur.c;
      else { pot.x = (nxt.c-cur.c)/(cur.m-nxt.m); pot.y = cur.m
          *pot.x+cur.c; }
      v.push_back(pot+po[id]);
    }
  }
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. 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" 90 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 {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return rot->r()->o->rot; }
};

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 q0 = new Quad{0,0,0,orig}, q1 = new Quad{0,0,0,arb},
    q2 = new Quad{0,0,0,dest}, q3 = new Quad{0,0,0,arb};
  q0->o = q0; q2->o = q2; // 0-0, 2-2
  q1->o = q3; q3->o = q1; // 1-3, 3-1
  q0->rot = q1; q1->rot = q2;
  q2->rot = q3; q3->rot = q0;
  return q0;
}
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) + 1) / 2;
  tie(ra, A) = rec({s.begin(), s.begin() + half});
  tie(B, rb) = rec({s.begin() + half, s.end()});
  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 = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

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

8.5 3D

PolyhedronVolume.h

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

6 lines

```
template<class V, class L>
double signed_poly_volume(const V& p, const L& trilist) {
  double v = 0;
  trav(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.

32 lines

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

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

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}\left(n^2\right)$

"Point3D.h" 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);
    }
  }
  trav(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. 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.

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: pi[x] computes the length of the longest prefix of t that ends at x (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

15 lines

```
int kmp(const string &T, const string &P) {
  if (P.empty()) return 0;
  vector<int> pi(P.size(), 0);
  for (int i = 1, k = 0; i < P.size(); ++i) {
    while (k && P[k] != P[i]) k = pi[k - 1];
    if (P[k] == P[i]) ++k;
    pi[i] = k;
  }
  for (int i = 0, k = 0; i < T.size(); ++i) {
    while (k && P[k] != T[i]) k = pi[k - 1];
    if (P[k] == T[i]) ++k;
    if (k == P.size()) return i - k + 1;
  }
  return -1;
}
```

Manacher.h

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

11 lines

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

MinRotation.h

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

8 lines

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

NextPermutation.h

Description: Finds the lexicographically smallest next permutation. Intuitively, find largest index i such that a[i] < a[i+1] then, find largest index j such that j >= i and a[j] > a[i] Swap(a[j], a[i-1]). Then reverse suffix start at i.
Usage: `return the array x and the permutation`
Time: $\mathcal{O}(N)$

12 lines

```
bool nextPermutation(int x[], int n) {
  int k = -1;
  for (int i = n - 2; k == -1 && i >= 0; --i)
    if (x[i] < x[i + 1]) k = i;
  if (k == -1) return false;
  int l = -1;
  for (int i = n - 1; l == -1 && i > k; --i)
    if (x[k] < x[i]) l = i;
  swap(x[k], x[l]);
  reverse(x + k + 1, x + n);
  return true;
}
```

SuffixArray.h

Description: Compute Suffix Array of Strings.
Usage: `compute_lcp(x,y) run in log N.`
`buildlcp() run in N and return lcp[i] = lcp SA[i] and SA[i-1].`
Time: $\mathcal{O}(|N| \log |N|)$

76 lines

```
class Element_suffix{
public:
  int rank_now, rank_pref, pos;
};

class Suffix{
private:
  inline bool same_rank(Element_suffix a, Element_suffix b) {
    return a.rank_now == b.rank_now && a.rank_pref == b.
         rank_pref;
  }
  inline void reset_freq(bool is_sort_now) {
    for(int i = 0; i <= end; i++) freq[i] = 0;
    for(int i = 0; i < n; i++) freq[ is_sort_now ? suf[i].
         rank_now+1 : suf[i].rank_pref+1 ]++;
    start[0] = 0;
    for(int i = 1; i <= end; i++) {
      start[i] = freq[i-1];
      freq[i] += freq[i-1];
    }
  }
public:
  int sorted[20][MAX], freq[MAX], start[MAX], SA[MAX], end, n;
  Element_suffix suf[MAX], tmp[MAX];
  void build_suffix() {
    n = strlen(s);
    if (n == 1) {
      SA[0] = 0;
      return;
    }
    end = max(n, 1 << 8);
```

```
    for(int i = 0; i < n; i++) sorted[0][i] = (int)s[i];
    int step = 1;
    for(int cnt = 1; cnt < n; step++, cnt *= 2) {
      for(int i = 0; i < n; i++) {
        suf[i].rank_pref = sorted[step-1][i];
        suf[i].rank_now = (i + cnt < n) ? sorted[step-1][i+cnt]
             : -1;
        suf[i].pos = i;
      }
      reset_freq(1);
      for(int i = 0; i < n; i++) tmp[start[suf[i].rank_now
           +1]++] = suf[i];
      reset_freq(0);
      for(int i = 0; i < n; i++) suf[start[tmp[i].rank_pref
           +1]++] = tmp[i];
      for(int i = 0; i < n; i++) {
        sorted[step][suf[i].pos] = (i && same_rank(suf[i], suf[
             i-1])) ? sorted[step][suf[i-1].pos] : i;
      }
    } step--;
    for(int i = 0; i < n; i++) SA[sorted[step][i]] = i;
  }
};
int compute_lcp(int x, int y) {
  int ans = 0;
  for(int k = 20; k >= 0; k--) {
    int s = (1 << k);
    if (x + s - 1 < n && y + s - 1 < n && sorted[k][x] ==
         sorted[k][y]) {
      ans += s;
      x += s;
      y += s;
    }
  }
  return ans;
}
void buildLCP(){
    phi[SA[0]] = -1;
    for(int i = 1 ; i < len ; i++)
        phi[SA[i]] = SA[i - 1];
    for(int i = 0, l = 0 ; i < len ; i++){
        if(phi[i] == -1)
            PLCP[i] = 0;
        else{
            while(s[i + l] == s[phi[i] + l]) l++;
            PLCP[i] = l;
            l = max(0,l - 1);
        }
    }
    for(int i = 0 ; i < len ; i++)
        LCP[i] = PLCP[SA[i]];
}
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

50 lines

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

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

  SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }

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

Hashing.h

Description: Various self-explanatory methods for string hashing.

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{}; trav(c,s) h=h*C+c; return h; }
```

AhoCorasick.h

Description: Aho-Corasick tree is used for dictionary matching. Initialize the tree like the example at main below.
Time: Function create is $\mathcal{O}(26N)$ where N is the sum of length of patterns. Becareful if the pattern allow duplicate. If not, the worst case is $N\sqrt{N}$.

<bits/stdc++.h> 113 lines

```
const int NALPHABET = 26;
struct Node {
  Node** children, go;
  bool leaf;
  char charToParent;
  Node* parent, suffLink, dictSuffLink;
  int count, value;

  Node(){
    children = new Node*[NALPHABET];
    go = new Node*[NALPHABET];
    for(int i = 0; i < NALPHABET;++i){
      children[i] = go[i] = NULL;
    }
    parent = suffLink = dictSuffLink = NULL;
    leaf = false;
    count = 0;
  }
};

Node* createRoot() {
  Node* node = new Node();
  node->suffLink = node;
  return node;
}

void addString(Node* node, const string& s, int value =-1) {
  for(int i = 0; i < s.length(); ++i){
    int c = s[i] - 'a';
    if(node->children[c] == NULL){
      Node* n = new Node();
      n->parent = node;
      n->charToParent = s[i];
      node->children[c] = n;
    }
    node = node->children[c];
  }
  node->leaf = true;
  node->count++;
```

```
  node->value = value;
}

Node* suffLink(Node* node);
Node* dictSuffLink(Node* node);
Node* go(Node* node, char ch);
int calc(Node* node);

Node* suffLink(Node* node) {
  if (node->suffLink == NULL){
    if (node->parent->parent == NULL){
      node->suffLink = node->parent;
    } else {
      node->suffLink = go(suffLink(node->parent),node->
           charToParent);
    }
  }
  return node->suffLink;
}

Node* dictSuffLink(Node* node) {
  if(node->dictSuffLink == NULL){
    Node* n = suffLink(node);
    if (node == n){
      node->dictSuffLink = node;
    } else {
      while (!n->leaf && n->parent != NULL){
        n = dictSuffLink(n);
      }
      node->dictSuffLink = n;
    }
  }
  return node->dictSuffLink;
}

Node* go(Node* node, char ch) {
  int c = ch -'a';
  if (node->go[c] == NULL){
    if (node->children[c] != NULL) {
      node->go[c]= node->children[c];
    } else {
      node->go[c]= node->parent == NULL? node : go(suffLink(
           node), ch);
    }
  }
  return node->go[c];
}

int calc(Node* node) {
  if (node->parent == NULL) {
    return 0;
  } else {
    return node->count + calc(dictSuffLink(node));
  }
}

int main() {
  Node* root = createRoot();
  addString(root,"a",0);
  addString(root,"aa",1);
  addString(root,"abc",2);

  string s("abcaadc");
  Node* node = root;
  for (int i = 0; i < s.length(); ++i){
    node = go(node, s[i]);
    Node* temp = node;
    while (temp != root) {
      if (temp->leaf) {
        printf("string (%d) occurs at position %d\n", temp->
             value, i);
      }
      temp = dictSuffLink(temp);
    }
  }
  return 0;
}
```

Various (10)

10.1 Known Problems

StableMarriage.h

Description: While there is a free man m: let w be the most preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

FlowShopScheduling.h

Description: Schedule N jobs on 2 machines to minimize completion time. i-th job takes ai and bi time to execute on 1st and 2nd machine, respectively. Each job must be first executed on the first machine, then on second. Both machines execute all jobs in the same order. solution -> sort jobs by key ai < bi ? ai : (oo-bi), i.e. first execute all jobs with ai < bi in order of increasing ai, then all other jobs in order of decreasing bi.

2sat.h

Description: Build an implication graph with 2 vertices for each variable (the variable itself and its inverse). For each clause x V y, add edges (x', y) and (y', x). The formula is satisfiable iff x and x' are in different SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (Kosaraju's last step). Assign true to all variables of the current SCC (if it hasn't been previously assigned false), and false to all inverses. There is a code above : 2sat.h

<2sat.h>

KonigTheorem.h

Description: Consider a bipartite graph where the vertices are partitioned into left (L) and right (R) sets. Suppose there is a maximum matching which partitions the edges into those used in the matching (Em) and those not (E0). Let T consist of all unmatched vertices from L, as well as all vertices reachable from those (starting from vertices of T) by going left-to-right along edges from E0 and right-to-left along edges from Em . This essentially means that for each unmatched vertex in L, we add into T all vertices that occur in a path alternating between edges from E0 and Em. minimum vertex cover : vertices in T are added if they are in R and subtracted if they are in L to obtain the minimum vertex cover. There is a code above.

<MinimumVertexCover.h>

MoserCircle.h

Description: Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution: g(n) = nC4 + nC2 + 1.

ChickenMcNugget.h

Description: Chicken McNugget Theorem states that for any two relatively prime positive integers m,n, the greatest integer that cannot be written in the form am+bn for nonnegative integers a,b is mn – m - n.

EulerFaceFormula.h

Description: V – E + F = 2 [V: vertices E: edges F: faces]

CayleyFormula.h

Description: There are n^{n-2} spanning trees of a complete graph with n labeled vertices. Spanning Tree of Complete Bipartite Graph is $N^{M-1} * M^{N-1}$.

PickTheorem.h

Description: Pick's Theorem: A = i + b/2 – 1. A is Area, I is internal points, and B is Border points .

JosephusProblem.h

Description: There are n person in a table waiting to be executed. Person 1 hold a knife. Each step whoever has the knife, kill the person next to him. Who's alive at the end?

3 lines

```
int x = 0;
for (int i = 2; i <= n; ++i)
  x = (x + i) % i;
```

ErdosGallai.h

Description: Given degree of n nodes. Is it possible to build the graph?

20 lines

```
sort(d+1, d+n+1, greater<int>);
for (i=1;i<=n;i++)
    x[i] = x[i-1] + d[i];
if (x[n]&1) {
    printf("Not possible\n");
    continue;
}
can = true;
for (k=1;k<=n;k++) {
    sum = x[k];
    tmp = k*(k-1);
    for (i=k+1;i<=n;i++)
        tmp += min(d[i], k);
    if (sum > tmp) {
        can = false;
        break;
    }
}
if (can) printf("Possible\n");
else printf("Not possible\n");
```

10.2 Desperate Optimization

FastRead.h

Description: Fast Read for Int/Long long
Usage: fastRead_int(x)

8 lines

```
inline void fastRead_int(int &x) {
    register int c = getchar_unlocked();
    x = 0;
    for(; ((c<48 || c>57) && c != '-'); c = getchar_unlocked())
        ;
    for(; c>47 && c<58 ; c = getchar_unlocked()) {
      x = (x<<1) + (x<<3) + c - 48;
    }
}
```

FastMod.h

Description: Fast MOD

Usage: `rem(a*b).  Dont forget to use manual C++ MOD at the end USE MM as (2^61) / MOD for safety`

3 lines

```
inline int rem(long long a) {
  return a-mod*((a>>29)*MM>>32);
}
```

ClockTime.h

Description: Elapsed time from the beginning of running program
Usage: `cek_time()`

5 lines

```
clock_t first_attempt = clock();
inline void cek_time(){
  clock_t cur = clock()- first_attempt;
  cerr<<"TIME : "<<(double) cur/CLOCKS_PER_SEC<<endl;
}
```

10.3 Misc. algorithms

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

13 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)
      a = mid;
    else
      b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

Karatsuba.h

Description: Faster-than-naive convolution of two sequences: $c[x] = \sum a[i]b[x-i]$. Uses the identity $(aX + b)(cX + d) = acX^2 + bd + ((a + c)(b + d) - ac - bd)X$. Doesn't handle sequences of very different length well. See also FFT, under the Numerical chapter.
Time: $\mathcal{O}\left(N^{1.6}\right)$

10.4 Dynamic programming

DivideAndConquerDP.h

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

18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

KnuthDP.h

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

10.5 Optimization tricks

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.