Содержание

1 Common

1 Setup

1. Terminal: font Monospace 12
2. Gedit: Oblivion, font Monospace 12, auto indent, display line numbers, tab 4, highlight matching brackets, highlight current line, F9 (side panel)
3. /.bashrc: export CXXFLAGS='-Wall -Wshadow -Wextra -Wconversion -Wno-unused-result -Wno-deprecated-declarations -O2 -std=gnu++11 -g -DLOCAL'
4. `for i in {A..K}; do mkdir $i; cp main.cpp $i/$i.cpp; done`

2 Template

main.cpp:

```
#define FNAME ""

#undef __STRICT_ANSI__

#ifdef LOCAL
  #define _GLIBCXX_DEBUG
#endif

#include <bits/stdc++.h>

using namespace std;

#define pb push_back
#define mp make_pair
#define fs first
#define sc second
#define fst first
#define snd second
#define sz(x) (int)((x).size())
#define forn(i,n) for (int i = 0; (i) < (n); ++i)
#define fornr(i,n) for (int i = (int)(n) - 1; (i) >= 0; --i)
#define forab(i,a,b) for (int i = (a); (i) < (b); ++i)
#define forba(i,a,b) for (int i = (int)(b) - 1; (i) >= (a);
 ↪  --i)
#define forit(it,c) for(__typeof((c).begin()) it =
 ↪  (c).begin(); it != (c).end(); ++it)
#define all(c) (c).begin(),(c).end()

#ifdef LOCAL
  #define eprintf(...) fprintf(stderr, __VA_ARGS__),
   ↪  fflush(stderr)
#else
  #define eprintf(...) (void) 0;
#endif


typedef long long LL;
typedef unsigned long long ULL;
typedef double dbl;
typedef long double LD;
typedef unsigned int uint;
typedef vector <int> vi;
typedef pair <int, int> pii;


int main() {



  return 0;
}
```

3 Stress

stress.sh:

```
!/bin/bash

for ((i = 0;; i++)); do
  ./gen $i >in || exit
  ./main <in >out1 || exit
  ./stupid <in >out2 || exit
  diff out2 out2 || exit
  echo $i OK
done
```

4 Java

Java template:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.*;

public class Main {
  FastScanner in;
  PrintWriter out;

  void solve() {
    int a = in.nextInt();
    int b = in.nextInt();
    out.print(a + b);
  }

  void run() {
    try {
      in = new FastScanner("input.txt");
      out = new PrintWriter("output.txt");
      solve();
      out.flush();
      out.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      System.exit(1);
    }
  }

  class FastScanner {
    BufferedReader br;
    StringTokenizer st;

    public FastScanner() {
      br = new BufferedReader(new
       ↪  InputStreamReader(System.in));
    }

    public FastScanner(String s) {
      try {
        br = new BufferedReader(new FileReader(s));
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }
    }

    String nextToken() {
      while (st == null || !st.hasMoreElements()) {
        try {
          st = new StringTokenizer(br.readLine());
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      return st.nextToken();
    }

    int nextInt() {
      return Integer.parseInt(nextToken());
    }

    long nextLong() {
      return Long.parseLong(nextToken());
    }

    double nextDouble() {
      return Double.parseDouble(nextToken());
    }

    char nextChar() {
      try {
        return (char) (br.read());
      } catch (IOException e) {
        e.printStackTrace();
      }
      return 0;
    }

    String nextLine() {
      try {
        return br.readLine();
      } catch (IOException e) {
        e.printStackTrace();
      }
      return "";
    }
  }

  public static void main(String[] args) {
    new Main().run();
  }
}
```

2 Big numbers

5 Big uint

```
#include <cstdio>
#include <cassert>
#include <cstring>
#include <cmath>
#include <algorithm>

#define forn(i, n) for (int i = 0; i < (int)(n); i++)

typedef long long ll;

const int BASE_LEN = 9;
const int NUM_LEN = 50000 / BASE_LEN + 2; // LEN <= NUM_LEN *
 ↪  BASE_LEN
const int BASE = pow(10, BASE_LEN);
const ll INF = 8e18, ADD = INF / BASE;

struct num {
  ll a[NUM_LEN];
  int len; // always > 0

  inline const ll& operator [] ( int i ) const { return a[i];
   ↪  }
  inline ll& operator [] ( int i ) { return a[i]; }

  num& operator = ( const num &x ) { // copy
    len = x.len;
    memcpy(a, x.a, sizeof(a[0]) * len);
    return *this;
  }

  num( const num &x ) { *this = x; } // copy

  num() { len = 1, a[0] = 0; } // 0
  num( ll x ) { // x
    len = 0;
    while (!len || x) {
      assert(len < NUM_LEN); // to catch overflow
      a[len++] = x % BASE, x /= BASE;
    }
  }

  num& cor() {
    while (a[len - 1] >= BASE) {
      assert(len < NUM_LEN); // to catch overflow
      if (a[len - 1] >= 2 * BASE)
        a[len] = a[len - 1] / BASE, a[len - 1] %= BASE;
      else
        a[len] = 1, a[len - 1] -= BASE;
      len++;
    }
    while (len > 1 && !a[len - 1])
      len--;
    return *this;
  }

  int length() {
    if (!len)
      return 0;
    int x = a[len - 1], res = 0;
    assert(x);
    while (x || !res)
      x /= 10, res++;
    return res + (len - 1) * BASE_LEN;
  }

  void out() const {
    int i = len - 1;
    printf("%d", (int)a[i--]);
    while (i >= 0)
      printf("%0*d", BASE_LEN, (int)a[i--]);
    puts("");
  }

  void init( const char *s ) {
    int sn = strlen(s);
    while (sn && s[sn - 1] <= 32)
      sn--;
    len = (sn + BASE_LEN - 1) / BASE_LEN;
    memset(a, 0, sizeof(a[0]) * len);
    forn(i, sn) {
      ll &r = a[(sn - i - 1) / BASE_LEN];
      r = r * 10 + (s[i] - ’0’);
    }
  }

  bool read() {
    static const int L = NUM_LEN * BASE_LEN + 1;
    static char s[L];
    if (!fgets(s, L, stdin))
      return 0;
    assert(!s[L - 2]);
    init(s);
    return 1;
  }

  void mul2() {
```

```
  forn(i, len)
    a[i] <<= 1;
  forn(i, len - 1)
    if (a[i] >= BASE)
      a[i + 1]++, a[i] -= BASE;
  cor();
}

void div2() {
  for (int i = len - 1; i >= 0; i--) {
    if (i && (a[i] & 1))
      a[i - 1] += BASE;
    a[i] >>= 1;
  }
  cor();
}

static ll cmp( ll *a, ll *b, int n ) {
  while (n--)
    if (a[n] != b[n])
      return a[n] - b[n];
  return 0;
}

int cmp( const num &b ) const {
  if (len != b.len)
    return len - b.len;
  for (int i = len - 1; i >= 0; i--)
    if (a[i] != b[i])
      return a[i] - b[i];
  return 0;
}

bool zero() {
  return len == 1 && !a[0];
}

/** c = this/b, this %= b */
num &div( num b, num &c ) {
  c.len = len - b.len;
  for (int i = c.len; i >= 0; i--) {
    int k = (1.0L * a[len - 1] * BASE + (len >= 2 ? a[len -
    ↪  2] : 0)) / (1.0L * b[b.len - 1] * BASE + (b.len >=
    ↪  2 ? b[b.len - 2] + 1 : 0));
    c[i] = 0;
    if (k > 0) {
      c[i] += k;
      forn(j, b.len)
        a[i + j] -= (ll)b[j] * k;
      forn(j, b.len)
        if (a[i + j] < 0) {
          ll k = (-a[i + j] + BASE - 1) / BASE;
          a[i + j] += k * BASE, a[i + j + 1] -= k;
        }
    }
    if (i)
      len--, a[len - 1] += a[len] * BASE, a[len] = 0;
    else if (cmp(a, b.a, b.len) >= 0) {
      c[0]++;
      forn(j, b.len)
        if ((a[j] -= b[j]) < 0)
          a[j] += BASE, a[j + 1]--;
    }
  }
  if (c.len < 0)
    c[c.len = 0] = 0;
  forn(i, c.len)
    if (c[i] >= BASE)
      c[i + 1] += c[i] / BASE, c[i] %= BASE;
  c.len += (!c.len || c[c.len]);
  return cor();
```

```
  }
};

num& operator += ( num &a, const num &b ) {
  while (a.len < b.len)
    a[a.len++] = 0;
  forn(i, b.len)
    a[i] += b[i];
  forn(i, a.len - 1)
    if (a[i] >= BASE)
      a[i] -= BASE, a[i + 1]++;
  return a.cor();
}

num& operator -= ( num &a, const num &b ) {
  while (a.len < b.len)
    a[a.len++] = 0;
  forn(i, b.len)
    a[i] -= b[i];
  forn(i, a.len - 1)
    if (a[i] < 0)
      a[i] += BASE, a[i + 1]--;
  assert(a[a.len - 1] >= 0); // a >= b
  return a.cor();
}

num& operator *= ( num &a, int k ) {
  if (k == 1)
    return a;
  if (k == 0) {
    a.len = 0;
    return a;
  }
  forn(i, a.len)
    a[i] *= k;
  forn(i, a.len - 1)
    if (a[i] >= BASE)
      a[i + 1] += a[i] / BASE, a[i] %= BASE;
  return a.cor();
}

num& operator /= ( num &a, int k ) {
  if (k == 1)
    return a;
  assert(k != 0);
  for (int i = a.len - 1; i > 0; i--)
    a[i - 1] += (ll)(a[i] % k) * BASE, a[i] /= k;
  a[0] /= k;
  return a.cor();
}

num& mul( const num &a, const num &b, num &x ) {
  assert(a.len + b.len - 1 <= NUM_LEN);
  memset(x.a, 0, sizeof(x[0]) * (a.len + b.len - 1));
  forn(i, a.len)
    forn(j, b.len)
      if ((x[i + j] += a[i] * b[j]) >= INF)
        x[i + j + 1] += ADD, x[i + j] -= INF;
  x.len = a.len + b.len - 1;
  forn(i, x.len - 1)
    if (x[i] >= BASE)
      x[i + 1] += x[i] / BASE, x[i] %= BASE;
  return x.cor();
}

bool operator == ( const num &a, const num &b ) { return
↪  a.cmp(b) == 0; }
bool operator != ( const num &a, const num &b ) { return
↪  a.cmp(b) != 0; }
bool operator < ( const num &a, const num &b )  { return
↪  a.cmp(b) < 0; }
```

```
bool operator > ( const num &a, const num &b )  { return
 ↪  a.cmp(b) > 0; }
bool operator <= ( const num &a, const num &b ) { return
 ↪  a.cmp(b) <= 0; }
bool operator >= ( const num &a, const num &b ) { return
 ↪  a.cmp(b) >= 0; }

num& add( const num &a, const num &b, num &c ) { c = a; c +=
 ↪  b; return c; }
num& sub( const num &a, const num &b, num &c ) { c = a; c -=
 ↪  b; return c; }
num& mul( const num &a, int k, num &c )        { c = a; c *=
 ↪  k; return c; }
num& div( const num &a, int k, num &c )        { c = a; c /=
 ↪  k; return c; }

num& operator *= ( num &a, const num &b ) {
  static num tmp;
  mul(a, b, tmp);
  return a = tmp;
}

num operator ^ ( const num &a, int k ) {
  num res(1);
  forn(i, k)
    res *= a;
  return res;
}

num& gcd_binary( num &a, num &b ) {
  int cnt = 0;
  while (!a.zero() && !b.zero()) {
    while (!(b[0] & 1) && !(a[0] & 1))
      cnt++, a.div2(), b.div2();
    while (!(b[0] & 1))
      b.div2();
    while (!(a[0] & 1))
      a.div2();
    if (a.cmp(b) < 0)
      b -= a;
    else
      a -= b;
  }
  if (a.zero())
    std::swap(a, b);
  while (cnt)
    a.mul2(), cnt--;
  return a;
}

num& gcd( num &a, num &b ) {
  static num tmp;
  return b.zero() ? a : gcd(b, a.div(b, tmp));
}
```

6 FFT

```
//typedef complex<dbl> Num;
struct Num{
  dbl x, y;
  Num(){}
  Num(dbl _x, dbl _y):x(_x),y(_y){}

  inline dbl real() const{ return x; }
  inline dbl imag() const{ return y; }

  inline Num operator +(const Num &B) const{
    return Num(x+B.x, y+B.y);
  }
  inline Num operator -(const Num &B) const{
    return Num(x-B.x, y-B.y);
  }
  inline Num operator *(dbl k) const{
    return Num(x*k, y*k);
  }
  inline Num operator *(const Num &B) const{
    return Num(x*B.x - y*B.y, x*B.y + y*B.x);
  }

  inline void operator +=(const Num &B){
    x+=B.x, y+=B.y;
  }
  inline void operator /=(dbl k){
    x/=k, y/=k;
  }
  inline void operator *=(const Num &B){
    *this = *this * B;
  }
};

inline Num sqr(const Num &x){ return x * x; }
inline Num conj(const Num &x){ return Num(x.real(),
 ↪   -x.imag()); }

inline int getN(int n){
  int k = 1;
  while(k < n)
    k <<= 1;
  return k;
}

const int LOG = 18;
const int MAX_N = 1 << LOG;

Num rt[MAX_N];
int rev[MAX_N];

void fft(Num *a, int n){
  assert(rev[1]); // don't forget to init
  int q = MAX_N / n;
  forn(i, n)
    if(i < rev[i] / q)
      swap(a[i], a[rev[i] / q]);
  for(int k = 1; k < n; k <<= 1)
    for(int i = 0; i < n; i += 2 * k)
      forn(j, k){
        const Num z = a[i + j + k] * rt[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
      }
}

void fft_inv(Num *a, int n){
  fft(a, n);
  reverse(a + 1, a + n);
  forn(i, n)
    a[i] /= n;
}

void double_fft(Num *a, Num *fa, Num *fb, int n){ // only if
 ↪   you need it
  fft(a, n);
  const int n1 = n - 1;
  forn(i, n){
    const Num &z0 = a[i], &z1 = a[(n - i) & n1];
    fa[i] = Num(z0.real() + z1.real(), z0.imag() - z1.imag())
     ↪   * 0.5;
    fb[i] = Num(z0.imag() + z1.imag(), z1.real() - z0.real())
     ↪   * 0.5;
  }
}
```

```
Num tmp[MAX_N];
template<class T>
void mult(T *a, T *b, T *r, int n){ // n = 2^k
  forn(i, n)
    tmp[i] = Num((dbl)a[i], (dbl)b[i]);
  fft(tmp, n);
  const int n1 = n - 1;
  const Num c = Num(0, -0.25 / n);
  fornr(i, n / 2 + 1){
    const int j = (n - i) & n1;
    const Num z0 = sqr(tmp[i]), z1 = sqr(tmp[j]);
    tmp[i] = (z1 - conj(z0)) * c;
    tmp[j] = (z0 - conj(z1)) * c;
  }
  fft(tmp, n);
  forn(i, n)
    r[i] = (T)round(tmp[i].real());
}

void init(){ // don't forget to init
  forn(i, MAX_N)
    rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (LOG - 1));

  rt[1] = Num(1, 0);
  for(int k = 1, p = 2; k < LOG; ++k, p *= 2){
    const Num x(cos(M_PI / p), sin(M_PI / p));
    for(int i = p / 2; i < p; ++i)
      rt[2 * i] = rt[i], rt[2 * i + 1] = rt[i] * x;
  }
}
```

7 FFT by mod and FFT with digits up to 10^6

```
Num ta[MAX_N], tb[MAX_N], tf[MAX_N], tg[MAX_N];

const int HALF = 15;

void mult(int *a, int *b, int *r, int N, int mod){
  int tw = (1 << HALF) - 1;
  forn(i, N){
    int x = int(a[i] % mod);
    ta[i] = Num(x & tw, x >> HALF);
  }
  forn(i, N){
    int x = int(b[i] % mod);
    tb[i] = Num(x & tw, x >> HALF);
  }
  fft(ta, N);
  fft(tb, N);

  forn(i, N){
    int j = (N - i) & (N - 1);
    Num a1 = (ta[i] + conj(ta[j])) * Num(0.5, 0);
    Num a2 = (ta[i] - conj(ta[j])) * Num(0, -0.5);
    Num b1 = (tb[i] + conj(tb[j])) * Num(0.5 / N, 0);
    Num b2 = (tb[i] - conj(tb[j])) * Num(0, -0.5 / N);
    tf[j] = a1 * b1 + a2 * b2 * Num(0, 1);
    tg[j] = a1 * b2 + a2 * b1;
  }

  fft(tf, N);
  fft(tg, N);

  forn(i, N){
    LL aa = LL(tf[i].x + 0.5);
    LL bb = LL(tg[i].x + 0.5);
    LL cc = LL(tf[i].y + 0.5);
    r[i] = int((aa + ((bb % mod) << 15) + ((cc % mod) << 30))
     ↪  % mod);
  }
}

int tc[MAX_N], td[MAX_N];

const int MOD1 = 1.5e9, MOD2 = MOD1 + 1;
void multLL(int *a, int *b, LL *r, int N){
  mult(a, b, tc, N, MOD1);
  mult(a, b, td, N, MOD2);

  forn(i, N)
    r[i] = tc[i] + (td[i] - tc[i] + (LL)MOD2) * MOD1 % MOD2 *
     ↪  MOD1;
}
```

3 Data Structures

8 Centroid Decomposition

```
vector <int> g[MAX_N];
int d[MAX_N], par[MAX_N], centroid;
//d и par - в дереве центроидов

int find(int v, int p, int total) {
  int size = 1, ok = 1;
  for (int to : g[v])
    if (d[to] == -1 && to != p) {
      int s = find(to, v, total);
      if (s > total / 2) ok = 0;
      size += s;
    }
  if (ok && size > total / 2)
    centroid = v;
  return size;
}

void calc_in_component(int v, int p, int level) {
  // do something
  for (int to : g[v])
    if (d[to] == -1 && to != p)
      calc_in_component(to, v, level);
}

//decompose(0, -1, 0)
void decompose(int root, int parent, int level) {
  find(root, -1, find(root, -1, INF));
  int c = centroid;
  par[c] = parent;
  d[c] = level;
  calc_in_component(centroid, -1, level);
  for(int to : g[c])
    if(d[to] == -1)
      decompose(to, c, level + 1);
}
```

9 Convex Hull Trick

```
struct Line {
  int k, b;
  Line() {}
  Line(int kk, int bb): k(kk), b(bb) {}
  LL get(int x) {
    return b + k * 1ll * x;
  }
  bool operator <(const Line &l) const {
    return k < l.k; //знак на > если в другую сторону
  }
};
//проверяет, что пересечение (a,b) левее (a,c)
inline bool check(Line a, Line b, Line c) {
  return (a.b - b.b) * 1ll * (c.k - a.k) < (a.b - c.b) * 1ll *
   ↪  (b.k - a.k);
}

vector <Line> st;

inline void add(Line l) {
  while(sz(st) >= 2 && !check(st[sz(st) - 2], st[sz(st) - 1],
   ↪   l))
    st.pop_back();
  st.pb(l);
}

int get(int x) {
  int l = 0, r = sz(st);
  while (r - l > 1) {
    int m = (l + r) / 2; //знак на > если в другую сторону
    if ((st[m - 1].b - st[m].b) < x * 1ll * (st[m].k - st[m -
     ↪   1].k))
      l = m;
    else
      r = m;
  }
  return l;
}

void buildConvexHull(vector <Line> lines) {
  sort(all(lines));
  for(Line l : lines)
    add(l);
}
```

10 Fenwick Tree

```
namespace FenwickTree {
  int t[MAX_N];
  int n;

  int get(int ind) {
     int res = 0;
     for (; ind >= 0; ind &= (ind + 1), ind--)
       res += t[ind];
     return res;
  }

  void add(int ind, int x) {
     for (; ind < n; ind |= (ind + 1))
       t[ind] += x;
  }

  int sum(int l, int r) { //[l, r)
    return get(r - 1) - get(l - 1);
  }
}
```

11 Hash Table

```
namespace HashTable {
  typedef long long H;

  const int HT_SIZE = 1<<20, HT_AND = HT_SIZE - 1, HT_SIZE_ADD
   ↪   = HT_SIZE / 100;
  H ht[HT_SIZE + HT_SIZE_ADD];
  int data[HT_SIZE + HT_SIZE_ADD];

  int get(const H &hash){
    ll k = ((ll) hash) & HT_AND;
    while(ht[k] && ht[k] != hash)
      ++k;
    return k;
  }

  void insert(const H &hash, int x){
    int k = get(hash);
    if(!ht[k])
      ht[k] = hash, data[k] = x;
  }

  bool count(const H &hash, int x){
    int k = get(hash);
    return ht[k] != 0;
  }
}
```

12 Heavy Light Decomposition

```
namespace HeavyLightDecomposition {
  vi g[MAX_N];
  int size[MAX_N], comp[MAX_N], num[MAX_N], top[MAX_N],
   ↪   pr[MAX_N], tin[MAX_N], tout[MAX_N];
  vi t[MAX_N], toPush[MAX_N], lst[MAX_N];
  int curPath = 0, curTime = 0;

  void pushST(int path, int v, int vl, int vr) {
    if (toPush[path][v] != -1) {
      if (vl != vr - 1)
        forn (j, 2)
          toPush[path][2 * v + j] = toPush[path][v];
      else
        t[path][v] = toPush[path][v];
      toPush[path][v] = -1;
    }
  }

  int getST(int path, int v, int vl, int vr, int ind) {
    pushST(path, v, vl, vr);
    if (vl == vr - 1)
      return t[path][v];
    int vm = (vl + vr) / 2;
    if (ind >= vm)
      return getST(path, 2 * v + 1, vm, vr, ind);
    return getST(path, 2 * v, vl, vm, ind);
  }

  void setST(int path, int v, int vl, int vr, int l, int r,
   ↪   int val) {
    if (vl >= l && vr <= r) {
      toPush[path][v] = val;
      pushST(path, v, vl, vr);
      return;
    }
    pushST(path, v, vl, vr);
    if (vl >= r || l >= vr)
      return;
    int vm = (vl + vr) / 2;
    setST(path, 2 * v, vl, vm, l, r, val);
    setST(path, 2 * v + 1, vm, vr, l, r, val);
    t[path][v] = min(t[path][2 * v], t[path][2 * v + 1]);
  }

  bool isUpper(int v, int u) {
     return tin[v] <= tin[u] && tout[v] >= tout[u];
  }

  int getHLD(int v) {
    return getST(comp[v], 1, 0, sz(t[comp[v]]) / 2, num[v]);
  }

  int setHLD(int v, int u, int val) {
    int ans = 0, w = 0;
    forn (i, 2) {
       while (!isUpper(w = top[comp[v]], u))
         setST(comp[v], 1, 0, sz(t[comp[v]]) / 2, 0, num[v] +
          ↪   1, val), v = pr[w];
      swap(v, u);
    }
    setST(comp[v], 1, 0, sz(t[comp[v]]) / 2, min(num[v],
    ↪  num[u]), max(num[v], num[u]) + 1, val);
    return ans;
  }

  void dfs(int v, int p) {
    tin[v] = curTime++;
    size[v] = 1;
    pr[v] = p;
    for (int u : g[v])
      if (u != p) {
        dfs(u, v);
        size[v] += size[u];
      }
    tout[v] = curTime++;
  }

  void build(int v) {
    if (v == 0 || size[v] * 2 < size[pr[v]]) {
      top[curPath] = v;
      comp[v] = curPath;
      num[v] = 0;
      curPath++;
    }
    else {
      comp[v] = comp[pr[v]];
      num[v] = num[pr[v]] + 1;
    }
    lst[comp[v]].pb(v);
    for (int u : g[v])
         if (u != pr[v])
        build(u);
  }

  void initHLD() {
    dfs(0, 0);
    build(0);

    forn (i, curPath) {
      int curSize = 1;
      while (curSize < sz(lst[i]))
        curSize *= 2;
      t[i].resize(curSize * 2);
      toPush[i] = vi(curSize * 2, -1);
      //initialize t[i]
    }
  }
}
```

13 Next Greater in Segment Tree

```
//на позиции строго большей чем pos число строго большее x
int nextGreaterX(int v, int l, int r, int pos, int x) {
  if (r <= pos + 1 || tree[v] <= x)
    return INF;
  if (v >= tSize)
    return v - tSize;
  int ans = nextGreaterX(2 * v, l, (l + r) / 2, pos, x);
  if (ans == INF)
    ans = nextGreaterX(2 * v + 1, (l + r) / 2, r, pos, x);
  return ans;
}
```

14 Treap (Rope)

```
#include<bits/stdc++.h>
const int INF=1e9;
using namespace std;

struct Node;
typedef Node *pNode;

struct Node{
    static pNode null;
    pNode l, r;
    int y, val, size, m;

    Node(): l(this), r(this), y(-1), val(INF), size(0),
    ↪  m(INF){}
    Node(int v): l(null), r(null), y(rand()), val(v), size(1),
    ↪  m(v){}

    void calc(){
        size = 1 + l->size + r->size;
        m = min(val, min(l->m, r->m));
    }
};

pNode Node::null = new Node();

void merge(pNode &t, pNode l, pNode r){
    if(l == Node::null)
        t = r;
    else if(r == Node::null)
        t = l;
    else if(l->y < r->y)
        merge(l->r, l->r, r), (t = l)->calc();
    else
        merge(r->l, l, r->l), (t = r)->calc();
}

void split(pNode t, pNode &l, pNode &r, int k){
    if(t == Node::null)
        l = r = t;
    else if(t->l->size >= k)
        split(t->l, l, t->l, k), (r = t)->calc();
    else
        split(t->r, t->r, r, k - t->l->size - 1), (l =
    ↪  t)->calc();
}

void insert(pNode &root, int k, int x){
  pNode r, n = new Node(x);
  split(root, root, r, k);
  merge(root, root, n);
  merge(root, root, r);
}

void erase(pNode &root, int k){
  pNode l, n;
  split(root, l, root, k);
  split(root, n, root, 1);
  merge(root, l, root);
}

pNode build(int k){
    if(k == 1)
        return new Node(0);
    pNode root;
    merge(root, build(k / 2), build((k + 1) / 2));
    return root;
}

void print(pNode t, bool root = 1){
    if(t != Node::null){
        print(t->l, 0);
        printf("%d ", t->val);
        print(t->r ,0);
    }
    if(root)
    puts("");
}

int main(){
  pNode r=Node::null; // r is an empty tree
  // work with r
}
```

15 Segtree 2D

```
const int ST_SZ=1<<10, ST_SZ2=2*ST_SZ;

struct Node_1d{
  Node_1d *l,*r;
  LL val,need;
  Node_1d():l(NULL),r(NULL),val(0),need(0){}
  inline void norm(){
    if(l==NULL)
      l=new Node_1d();
    if(r==NULL)
      r=new Node_1d();
  }
  LL Get(int L0,int R0,int L=0,int R=ST_SZ){
    if(L0>=R || L>=R0)
      return 0;
    if(L0<=L && R<=R0)
      return val;
    int a=max(L0,L), b=min(R0,R), M=(L+R)>>1;
    norm();
    return l->Get(L0,R0,L,M)+r->Get(L0,R0,M,R)+need*LL(b-a);
  }
  void Add(int L0,int R0,int x,int L=0,int R=ST_SZ){
    if(L0>=R || L>=R0)
      return;
    if(L0<=L && R<=R0){
      need+=x;
      val+=x*LL(R-L);
      return;
    }
    int M=(L+R)>>1;
    norm();
    l->Add(L0,R0,x,L,M), r->Add(L0,R0,x,M,R);
    val=l->val+r->val+need*(R-L);
  }
};

struct Node_2d{
  Node_2d *l,*r;
  Node_1d *val,*need;
  Node_2d():l(NULL),r(NULL),val(new Node_1d()),need(new
   ↪  Node_1d()){}
  inline void norm(){
    if(l==NULL)
      l=new Node_2d();
    if(r==NULL)
      r=new Node_2d();
  }
  LL Get(int L0,int R0,int L1,int R1,int L=0,int R=ST_SZ){
    if(L0>=R || L>=R0)
      return 0;
    if(L0<=L && R<=R0)
      return val->Get(L1,R1);
    int a=max(L0,L), b=min(R0,R), M=(L+R)>>1;
    norm();
    return
     ↪  l->Get(L0,R0,L1,R1,L,M)+r->Get(L0,R0,L1,R1,M,R)+need->Get(L1,R1)*LL(b-a);
  }
  void Add(int L0,int R0,int L1,int R1,int x,int L=0,int
   ↪  R=ST_SZ){
    if(L0>=R || L>=R0)
      return;
    if(L0<=L && R<=R0){
      need->Add(L1,R1,x);
      val->Add(L1,R1,x*LL(R-L));
      return;
    }
    int a=max(L0,L), b=min(R0,R), M=(L+R)>>1;
    norm();
    l->Add(L0,R0,L1,R1,x,L,M), r->Add(L0,R0,L1,R1,x,M,R);
    val->Add(L1,R1,x*LL(b-a));
  }
};


int main(){
  Node_2d *Z=new Node_2d();

  int x,y,Q;
  scanf("%d%d%d",&x,&y,&Q);
  forn(i,Q){
    int type,a,b,c,d;
    scanf("%d%d%d%d%d",&type,&a,&b,&c,&d);
    --a, --b;
    if(type==1){
      int w;
      scanf("%d",&w);
      Z->Add(a,c,b,d,w);
    }else
      printf(lld"\n",Z->Get(a,c,b,d));
  }
}
```

16 Segtree 2D — Fenwick

```
const int MAX_N=1002;
LL F[4][MAX_N][MAX_N];
int N,M;

inline int Z(int a){
  return a&~(a-1);
}

inline void add(int k,int x,int y,LL a){
  for(;x<=N;x+=Z(x))
    for(int j=y;j<=M;j+=Z(j))
      F[k][x][j]+=a;
}
inline LL get(int k,int x,int y){
  LL s=0;
  for(;x>0;x-=Z(x))
    for(int j=y;j>0;j-=Z(j))
      s+=F[k][x][j];
  return s;
}

inline LL Get(int a,int b){
  return LL(a+1)*(b+1)*get(0,a,b)-(b+1)*get(1,a,b)
      -(a+1)*get(2,a,b)+get(3,a,b);
}

inline void Add(int a,int b,LL w){
  add(0,a,b,w);
  add(1,a,b,w*a);
  add(2,a,b,w*b);
  add(3,a,b,w*a*b);
}
inline LL Get(int a,int b,int c,int d){
  return Get(c,d)-Get(a-1,d)-Get(c,b-1)+Get(a-1,b-1);
}

inline void Add(int a,int b,int c,int d,LL w){
  Add(a,b,w);
  if(d<M)
    Add(a,d+1,-w);
  if(c<N)
    Add(c+1,b,-w);
  if(c<N && d<M)
    Add(c+1,d+1,w);
}

int main(){
  int Q;
  scanf("%d%d%d",&N,&M,&Q);
  forn(i,Q){
    int type,a,b,c,d;
    scanf("%d%d%d%d%d",&type,&a,&b,&c,&d);
    if(type==1){
      int w;
      scanf("%d",&w);
      Add(a,b,c,d,w);
    }else
      printf(lld"\n",Get(a,b,c,d));
  }
}
```

4 Flows

17 Utilities

```
//for directed unweighted graph
struct Edge {
  int v, u, c, f;
  Edge() {}
  Edge(int v, int u, int c): v(v), u(u), c(c), f(0) {}
};

vector <Edge> edges;

inline void addFlow(int e, int flow) {
   edges[e].f += flow;
   edges[e ^ 1].f -= flow;
}

inline void addEdge(int v, int u, int c) {
  g[v].pb(sz(edges));
  edges.pb(Edge(v, u, c));
  g[u].pb(sz(edges));
  edges.pb(Edge(u, v, 0)); //for undirected 0 should be c
}

void read(int m) {
   forn (i, m) {
      int v, u, c;
      scanf("%d%d%d", &v, &u, &c);
      addEdge(v - 1, u - 1, c);
   }
}
```

18 Ford-Fulkerson

```
namespace FordFulkerson {
  int used[MAX_N], pr[MAX_N];
  vi g[MAX_N];
  int curTime = 1;

  #include "Utilities.cpp"

  int dfs(int v, int can, int toPush, int t) {
    if (v == t)
      return can;
     used[v] = curTime;
     for (int edge : g[v]) {
       auto &e = edges[edge];
       if (used[e.u] != curTime && e.c - e.f >= toPush) {
          int flow = dfs(e.u, min(can, e.c - e.f), toPush, t);
          if (flow > 0) {
            addFlow(edge, flow);
            pr[e.u] = edge;
            return flow;
         }
       }
     }
     return 0;
  }

  int fordFulkerson(int n, int m, int s, int t) {
    read(m);
    int ansFlow = 0, flow = 0;
    //Without scaling
    while ((flow = dfs(s, INF, 1, t)) > 0) {
      ansFlow += flow;
      curTime++;
    }
    //With scaling
    /*
    fornr (i, INF_LOG)
      for (curTime++; (flow = dfs(s, INF, (1 << i), t)) > 0; curTime++)
        ansFlow += flow;
    */
    return ansFlow;
  }
}
```

19 Dinic

```
namespace Dinic {
  int pr[MAX_N], d[MAX_N], q[MAX_N], first[MAX_N];
  vector <int> g[MAX_N];

  #include "Utilities.cpp"

  int dfs(int v, int can, int toPush, int t) {
    if (v == t)
      return can;
    int sum = 0;
    for (; first[v] < (int) g[v].size(); first[v]++) {
      auto &e = edges[g[v][first[v]]];
      if (d[e.u] != d[v] + 1 || e.c - e.f < toPush)
        continue;
      int flow = dfs(e.u, min(can, e.c - e.f), toPush, t);
      addFlow(g[v][first[v]], flow);
      can -= flow, sum += flow;
      if (!can)
        return sum;
    }
    return sum;
  }

  bool bfs(int n, int s, int t, int curPush) {
    for (int i = 0; i < n; i++)
      d[i] = INF, first[i] = 0;
    int head = 0, tail = 0;
    q[tail++] = s;
    d[s] = 0;
    while (tail - head > 0) {
      int v = q[head++];
      for (int edge : g[v]) {
        auto &e = edges[edge];
        if (d[e.u] > d[v] + 1 && e.c - e.f >= curPush) {
          d[e.u] = d[v] + 1;
          q[tail++] = e.u;
        }
      }
    }
    return d[t] != INF;
  }

  int dinic(int n, int m, int s, int t) {
    read(m);
    int ansFlow = 0;
    //Without scaling
    while(bfs(n, s, t, 1))
      ansFlow += dfs(s, INF, 1, t);
    //With scaling
    /*
    fornr (j, INF_LOG)
      while (bfs(n, s, t, 1 << j))
        ansFlow += dfs(s, INF, 1 << j, t);
    */
    return ansFlow;
  }
}
```

20 Hungarian

```
const int INF = 1e9;
int a[MAX_N][MAX_N];

// min = sum of a[pa[i],i]
// you may optimize speed by about 15%, just change all
 ↪  vectors to static arrays
vi Hungrian(int n) {
  vi pa(n + 1, -1), row(n + 1, 0), col(n + 1, 0), la(n + 1);
  forn(k, n) {
    vi u(n + 1, 0), d(n + 1, INF);
    pa[n] = k;
    int l = n, x;
    while ((x = pa[l]) != -1) {
      u[l] = 1;
      int minn = INF, tmp, l0 = l;
      forn(j, n)
        if (!u[j]) {
          if ((tmp = a[x][j] + row[x] + col[j]) < d[j])
            d[j] = tmp, la[j] = l0;
          if (d[j] < minn)
            minn = d[j], l = j;
        }
      forn(j, n + 1)
        if (u[j])
          col[j] += minn, row[pa[j]] -= minn;
        else
          d[j] -= minn;
    }
    while (l != n)
      pa[l] = pa[la[l]], l = la[l];
  }
  return pa;
}
```

21 Min Cost Max Flow

```
namespace MinCostMaxFlow {
  const int MAX_M = 1e4;
  int pr[MAX_N], in[MAX_N], q[MAX_N * MAX_M], used[MAX_N],
   ↪  d[MAX_N], pot[MAX_N];
  vi g[MAX_N];

  struct Edge {
    int v, u, c, f, w;
    Edge() {}
    Edge(int v, int u, int c, int w): v(v), u(u), c(c), f(0),
     ↪  w(w) {}
  };

  vector <Edge> edges;

  inline void addFlow(int e, int flow) {
     edges[e].f += flow;
     edges[e ^ 1].f -= flow;
  }

  inline void addEdge(int v, int u, int c, int w) {
    g[v].pb(sz(edges));
    edges.pb(Edge(v, u, c, w));
    g[u].pb(sz(edges));
    edges.pb(Edge(u, v, 0, -w));
  }

  void read(int m) {
     forn (i, m) {
        int v, u, c, w;
        scanf("%d%d%d%d", &v, &u, &c, &w);
        addEdge(v - 1, u - 1, c, w);
     }
  }

  int dijkstra(int n, int s, int t) {
    forn (i, n)
      used[i] = 0, d[i] = INF;
    d[s] = 0;
    while (1) {
       int v = -1;
       forn (i, n)
         if (!used[i] && (v == -1 || d[v] > d[i]))
           v = i;
       if (v == -1 || d[v] == INF)
         break;
         used[v] = 1;
       for (int edge : g[v]) {
          auto &e = edges[edge];
          ll w = e.w + pot[v] - pot[e.u];
          if (e.c > e.f && d[e.u] > d[v] + w)
            d[e.u] = d[v] + w, pr[e.u] = edge;
       }
    }
    if (d[t] == INF)
      return d[t];
    forn (i, n)
      pot[i] += d[i];
    return pot[t];
  }

  int fordBellman(int n, int s, int t) {
    forn (i, n)
      d[i] = INF;
    int head = 0, tail = 0;
    d[s] = 0;
    q[tail++] = s;
    in[s] = 1;
    while (tail - head > 0) {
      int v = q[head++];
      in[v] = 0;
      for (int edge : g[v]) {
        auto &e = edges[edge];
        if (e.c > e.f && d[e.u] > d[v] + e.w) {
          d[e.u] = d[v] + e.w;
          pr[e.u] = edge;
          if (!in[e.u])
            in[e.u] = 1, q[tail++] = e.u;
        }
      }
    }
    return d[t];
  }
  int minCostMaxFlow(int n, int m, int s, int t) {
    read(m);
    int ansFlow = 0, ansCost = 0, dist;
    while((dist = dijkstra(n, s, t)) != INF) {
      int curFlow = INF;
      for (int cur = t; cur != s; cur = edges[pr[cur]].v)
        curFlow = min(curFlow, edges[pr[cur]].c -
         ↪  edges[pr[cur]].f);
      for (int cur = t; cur != s; cur = edges[pr[cur]].v)
        addFlow(pr[cur], curFlow);
      ansFlow += curFlow;
      ansCost += curFlow * dist;
    }
    return ansCost;
  }
}
```

5 Geometry

22 ClosestPoints (SweepLine)

```
#include "header.h"

const int N = 2e5;

struct Pnt {
  int x, y, i;
  bool operator <(const Pnt &p) const{
    return mp(y, i) < mp(p.y, p.i);
  }
};

LL d2 = 8e18, d = (LL)sqrt(d2) + 1;
Pnt p[N];

inline LL sqr(int x){
  return (LL)x * x;
}

inline void relax(const Pnt &a, const Pnt &b){
  LL tmp = sqr(a.x - b.x) + sqr(a.y - b.y);
  if (tmp < d2)
    d2 = tmp, d = (LL)(sqrt(d2) + 1 - 1e-9); // round up
}

inline bool xless(const Pnt &a, const Pnt &b){
  return a.x < b.x;
}

int main() {
  int n;
  scanf("%d", &n);
  forn(i, n)
    scanf("%d%d", &p[i].x, &p[i].y), p[i].i = i;
  sort(p, p + n, xless);

  set <Pnt> s;
  int l = 0;
  forn(r, n){
    set<Pnt>::iterator it_r = s.lower_bound(p[r]), it_l =
     ↪  it_r;
    for (; it_r != s.end() && it_r->y - p[r].y < d; ++it_r)
      relax(*it_r, p[r]);
    while (it_l != s.begin() && p[r].y - (--it_l)->y < d)
      relax(*it_l, p[r]);
    s.insert(p[r]);
    while (l <= r && p[r].x - p[l].x >= d)
      s.erase(p[l++]);
  }
  printf("%.9f\n", sqrt(d2));
  return 0;
}
```

23 ConvexHull

```
typedef vector<Pnt> vpnt;

inline bool byAngle(const Pnt &a, const Pnt &b){
  dbl x = a % b;
  return eq(x, 0) ? a.len2() < b.len2() : x < 0;
}

vpnt convexHull(vpnt p){
  int n = sz(p);
  assert(n > 0);
  swap(p[0], *min_element(all(p)));
  forab(i, 1, n)
  p[i] = p[i] - p[0];
  sort(p.begin() + 1, p.end(), byAngle);

/*  Если надо оставить развёрнутые углы, нужно (1) и (2)
  (1):
  int k = p.size() - 1;
  while(k > 0 && eq((p[k - 1] - p.back()) % p.back(), 0))
    --k;
  reverse(pi.begin() + k, pi.end());*/

  int rn = 0;
  vpnt r(n);
  r[rn++] = p[0];
  forab(i, 1, n){
    Pnt q = p[i] + p[0];
    while(rn >= 2 && geq((r[rn - 1] - r[rn - 2]) % (q - r[rn -
     ↪  2]), 0)) // (2) ge
      --rn;
    r[rn++] = q;
  }
  r.resize(rn);
  return r;
}
```

24 GeometryBase

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

typedef long long LL;
typedef long double LD;
typedef double dbl;

const dbl EPS = 1e-9;
const int PREC = 20;
inline bool eq(dbl a, dbl b){ return abs(a-b)<=EPS; }
inline bool gr(dbl a, dbl b){ return a>b+EPS; }
inline bool geq(dbl a, dbl b){ return a>=b-EPS; }
inline bool ls(dbl a, dbl b){ return a<b-EPS; }
inline bool leq(dbl a, dbl b){ return a<=b+EPS; }

struct Pnt{
    dbl x,y;
    Pnt():x(0),y(0){}
    Pnt(dbl _x,dbl _y):x(_x),y(_y){}

    inline Pnt operator +(const Pnt &B) const{ return
     ↪  Pnt(x+B.x, y+B.y); }
    inline Pnt operator -(const Pnt &B) const{ return
     ↪  Pnt(x-B.x, y-B.y); }
    inline dbl operator *(const Pnt &B) const{ return x*B.x +
     ↪  y*B.y; } // LL
    inline dbl operator %(const Pnt &B) const{ return x*B.y -
    ↪  y*B.x; } // LL

    inline Pnt operator *(dbl k) const{ return Pnt(x*k, y*k);
    ↪  }
    inline Pnt operator /(dbl k) const{ return Pnt(x/k, y/k);
    ↪  }
    inline Pnt operator -() const{ return Pnt(-x, -y); }

    inline void operator +=(const Pnt &B){ x+=B.x, y+=B.y; }
    inline void operator -=(const Pnt &B){ x-=B.x, y-=B.y; }
    inline void operator *=(dbl k){ x*=k, y*=k; }

    inline bool operator ==(const Pnt &B){ return
    ↪  abs(x-B.x)<=EPS && abs(y-B.y)<=EPS; }
    inline bool operator !=(const Pnt &B){ return
    ↪  abs(x-B.x)>EPS || abs(y-B.y)>EPS; }
    inline bool operator <(const Pnt &B){ return
    ↪  abs(x-B.x)<=EPS ? y<B.y-EPS : x<B.x; }

    inline dbl angle() const{ return atan2(y, x); } // LD
    inline dbl len2() const{ return x*x+y*y; } // LL
    inline dbl len() const{ return sqrt(x*x+y*y); } // LL, LD
    inline Pnt getNorm() const{
        auto l = len();
        return Pnt(x/l, y/l);
    }
    inline void normalize(){
        auto l = len();
        x/=l, y/=l;
    }

    inline Pnt getRot90() const{ //counter-clockwise
        return Pnt(-y, x);
    }
    inline Pnt getRot(dbl a) const{ // LD
        dbl si = sin(a), co = cos(a);
        return Pnt(x*co - y*si, x*si + y*co);
    }

    inline void read(){
        int _x,_y;
        scanf("%d%d",&_x,&_y);
        x=_x, y=_y;
    }
    inline void write() const{
        printf("%.*f %.*f ", PREC, (double)x, PREC,
        ↪  (double)y);
    }
};

struct Line{
    dbl a, b, c;
    Line():a(0),b(0),c(0){}
    Line(dbl _a, dbl _b, dbl _c):a(_a),b(_b),c(_c){}

    Line(const Pnt &A, const Pnt &B){ // it normalizes (a,b),
    ↪  important in d(), normalToP()
        Pnt n = (B-A).getRot90().getNorm();
        a = n.x, b = n.y, c = -(a*A.x + b*A.y);
    }

    inline dbl d(const Pnt &p) const{ return a*p.x + b*p.y +
    ↪  c; }
    inline Pnt no() const {return Pnt(a, b);}
    inline Pnt normalToP(const Pnt &p) const{ return Pnt(a,b)
    ↪  * (a*p.x + b*p.y + c); }

    inline void write() const{
        printf("%.*f %.*f %.*f ", PREC, (double)a, PREC,
        ↪  (double)b, PREC, (double)c);
    }
};
```

25 GeometryInterTangent

```
void buildTangent(Pnt p1, dbl r1, Pnt p2, dbl r2, Line &l) {
↪  // r1, r2 = radius with sign
    Pnt p = p2 - p1;
    l.c = r1;
    dbl c2 = p.len2(), c1 = sqrt(c2 - sqr(r2));
    l.a = (-p.x * (r1 - r2) + p.y * c1) / c2;
    l.b = (-p.y * (r1 - r2) - p.x * c1) / c2;
    l.c -= l.no() * p1;
    assert(eq(l.d(p1), r1));
    assert(eq(l.d(p2), r2));
}


struct Circle {
    Pnt p;
    dbl r;
};

vector<Pnt> v; // to store intersection

// Intersection of two lines
int line_line(const Line &l, const Line &m){
    dbl z = m.a * l.b - l.a * m.b,
            x = m.c * l.b - l.c * m.b,
            y = m.c * l.a - l.c * m.a;
    if(fabs(z) > EPS){
        v.pb(Pnt(-x/z, y/z));
        return 1;
    }else if(fabs(x) > EPS || fabs(y) > EPS)
        return 0; // parallel lines
    else
        return 2; // same lines
}

// Intersection of Circle and line
void circle_line(const Circle &c, const Line &l){
    dbl d = l.d(c.p);
    if(fabs(d) > c.r + EPS)
        return;
    if(fabs(fabs(d) / c.r - 1) < EPS)
        v.pb(c.p - l.no() * d);
    else{
        dbl s = sqrt(fabs(sqr(c.r) - sqr(d)));
        v.pb(c.p - l.no() * d + l.no().getRot90() * s);
        v.pb(c.p - l.no() * d - l.no().getRot90() * s);
    }
}

// Intersection of two circles
void circle_circle(const Circle &a, const Circle &b){
    circle_line(a, Line((b.p - a.p) * 2, a.p.len2() -
    ↪  b.p.len2() + sqr(b.r) - sqr(a.r)));
}

// Squared distance between point p and segment [a..b]
dbl dist2(Pnt p, Pnt a, Pnt b){
    if ((p - a) * (b - a) < 0) return (p - a).len2();
    if ((p - b) * (a - b) < 0) return (p - b).len2();
    return fabs((p - a) % (b - a)) / (b - a).len2();
}
```

26 GeometrySimple

```
int sign(dbl a){ return (a > EPS) - (a < -EPS); }

// Checks, if point is inside the segment
```

```
inline bool inSeg(const Pnt &p, const Pnt &a, const Pnt &b) {
    return eq((p - a) % (p - b), 0) && leq((p - a) * (p - b), 0);
}

// Checks, if two intervals (segments without ends) intersect AND do not lie on the same line
inline bool subIntr(const Pnt &a, const Pnt &b, const Pnt &c, const Pnt &d){
    return
            sign((b - a) % (c - a)) * sign((b - a) % (d - a)) == -1 &&
            sign((d - c) % (a - c)) * sign((d - c) % (b - c)) == -1;
}

// Checks, if two segments (ends are included) has an intersection
inline bool checkSegInter(const Pnt &a, const Pnt &b, const Pnt &c, const Pnt &d){
    return inSeg(c, a, b) || inSeg(d, a, b) || inSeg(a, c, d) || inSeg(b, c, d) || subIntr(a, b, c, d);
}

inline dbl area(vector<Pnt> p){
    dbl s = 0;
    int n = p.size();
    p.pb(p[0]);
    forn(i, n)
        s += p[i + 1] % p[i];
    p.pop_back();
    return abs(s) / 2;
}

// Check if point p is inside polygon <n, q[]>
int contains_slow(Pnt p, Pnt *z, int n){
    int cnt = 0;
    forn(j, n){
        Pnt a = z[j], b = z[(j + 1) % n];
        if (inSeg(p, a, b))
            return -1; // border
        if (min(a.y, b.y) - EPS <= p.y && p.y < max(a.y, b.y) - EPS)
            cnt += (p.x < a.x + (p.y - a.y) * (b.x - a.x) / (b.y - a.y));
    }
    return cnt & 1; // 0 = outside, 1 = inside
}


//for convex polygon
//assume polygon is counterclockwise-ordered
bool contains_fast(Pnt p, Pnt *z, int n) {
    Pnt o = z[0];
    if(gr((p - o) % (z[1] - o), 0) || ls((p - o) % (z[n - 1] - o), 0))
        return 0;
    int l = 0, r = n - 1;
    while(r - l > 1){
        int m = (l + r) / 2;
        if(gr((p - o) % (z[m] - o), 0))
            r = m;
        else
            l = m;
    }
    return leq((p - z[l]) % (z[r] - z[l]), 0);
}

// Checks, if point "i" is in the triangle "abc" IFF triangle in CCW order
inline int isInTr(int i, int a, int b, int c){
    return
            gr((p[b] - p[a]) % (p[i] - p[a]), 0) &&
            gr((p[c] - p[b]) % (p[i] - p[b]), 0) &&
            gr((p[a] - p[c]) % (p[i] - p[c]), 0);
}
```

27 Halfplanes Intersection

```
const int maxn = (int)4e5 + 9;
const dbl eps = 1e-12;

dbl sqr( dbl x ) { return x * x; }

struct pnt{
  LL operator * ( pnt p ) { return (LL)x * p.y - (LL)y * p.x; }
  LL operator ^ ( pnt p ) { return (LL)x * p.x + (LL)y * p.y; }
  pnt ort() { return pnt(-y, x); }
  dbl ang() { return atan2(y, x); }
  LL d2() { return x * x + y * y; }
};

pnt st, v, p[maxn];
int n, sp, ss[maxn], ind[maxn], no[maxn], cnt[maxn], k = 0, a[maxn], b[maxn];
dbl ang[maxn];

pnt Norm( int k ){  return (p[a[k]] - p[b[k]]).ort();}

void AddPlane( int i, int j ){
  a[k] = i, b[k] = j, ind[k] = k;
  ang[k] = Norm(k).ang();
  k++;
}

bool angLess( int i, int j ){ return ang[i] < ang[j];}

void Unique()
{
  int i = 0, k2 = 0;
  while (i < k)
  {
    int ma = ind[i], st = i;
    pnt no = Norm(ma);

    for (i++; i < k && fabs(ang[ind[st]] - ang[ind[i]]) < eps; i++)
      if ((no ^ p[a[ma]]) < (no ^ p[a[ind[i]]]))
        ma = ind[i];
    ind[k2++] = ma;
  }
  k = k2;
}

dbl xx, yy, tmp;

#define BUILD(a1, b1, c1, i) \
  dbl a1 = Norm(i).x; \
  dbl b1 = Norm(i).y; \
  tmp = sqrt(a1 * a1 + b1 * b1); \
  a1 /= tmp, b1 /= tmp; \
  dbl c1 = -(a1 * p[a[i]].x + b1 * p[a[i]].y);

void FindPoint( int i, int j, dbl step = 0.0 ){
  BUILD(a1, b1, c1, i);
  BUILD(a2, b2, c2, j);

  xx = -(c1 * b2 - c2 * b1) / (a1 * b2 - a2 * b1);
  yy = (c1 * a2 - c2 * a1) / (a1 * b2 - a2 * b1);
  dbl no = sqrt(sqr(a1 + a2) + sqr(b1 + b2));
  xx += (a1 + a2) * step / no;
  yy += (b1 + b2) * step / no;
}

void TryShiftPoint( int i, int j, dbl step )
{
  FindPoint(i, j, step);

  forn(i, k){
    BUILD(a1, b1, c1, ind[i]);
    if (a1 * xx + b1 * yy + c1 < eps)
      return;
  }

  puts("Possible");
  printf("%.20lf %.20lf\n", (double)xx, (double)yy);
  exit(0);
}

void PushPlaneIntoStack( int i )
{
  while (sp >= 2 && ang[i] - ang[ss[sp - 2]] + eps < M_PI){
    FindPoint(i, ss[sp - 2]);

    BUILD(a1, b1, c1, ss[sp - 1]);
    if ((a1 * xx + b1 * yy + c1) < -eps)
      break;

    sp--;
  }
  ss[sp++] = i;
}

int main()
{
  scanf("%d", &n);
  forn(i, n)
    scanf("%d%d", &p[i].x, &p[i].y);
  p[n] = p[0];

  // Find set of planes
  forn(i, sp)
    AddPlane(max(ss[i], ss[i + 1]), min(ss[i], ss[i + 1]));
  forn(i, n - 1)
    AddPlane(i + 1, i);
  sort(ind, ind + k, angLess);

  int oldK = k;
  Unique();

  forn(i, oldK)
    no[i] = i;
  forn(i, k){
    int j = oldK + i, x = ind[i];
    ang[j] = ang[x] + 2 * M_PI;
    a[j] = a[x];
    b[j] = b[x];
    ind[i + k] = j, no[j] = x;
  }

  sp = 0;
  forn(i, 2 * k)
    PushPlaneIntoStack(ind[i]);
  forn(t, sp)
    if (++cnt[no[ss[t]]] > 1){
      TryShiftPoint(ss[t], ss[t - 1], 1e-5);
      break;
    }
  return 0;
}
```

6 Graphs

28 2-SAT

```
//MAX_N - 2 * vars
vector <int> g[MAX_N], rg[MAX_N], tsort;
vector <bool> values;
int used[MAX_N], comp[MAX_N];

void dfs(int v) {
  used[v] = 1;
  for(int to : g[v])
    if (!used[to])
      dfs(to);
  tsort.pb(v);
}

void rdfs(int v, int num) {
  used[v] = 1;
  comp[v] = num;
  for(int to : rg[v])
    if (!used[to])
      rdfs(to, num);
}

void addEdge(int a, int b) {
  g[a ^ 1].pb(b);
  g[b ^ 1].pb(a);
  rg[b].pb(a ^ 1);
  rg[a].pb(b ^ 1);
}
//n - удвоенное
bool sat2(const vector <pii> &v, int n) {
  forn(i, sz(v)) {
    addEdge(v[i].fst, v[i].snd);
  }
  memset(used, 0, sizeof(used));
  forn(i, n)
    if (!used[i])
      dfs(i);
  memset(used, 0, sizeof(used));
  int num = 0;
  fornr(i, n) {
    int u = tsort[i];
    if (!used[u])
      rdfs(u, num), num++;
  }
  values.resize(n);
  for(int i = 0; i < n; i += 2)
    if (comp[i] == comp[i ^ 1])
      return 0;
    else if (comp[i] > comp[i ^ 1])
      values[i] = 1, values[i ^ 1] = 0;
    else
      values[i] = 0, values[i ^ 1] = 1;
  return 1;
}
```

29 Bridges

```
struct Edge {
  int to, id;
  Edge(int aa, int bb) : to(aa), id(bb) {}
};

int up[MAX_N], tin[MAX_N], timer;
vector <Edge> g[MAX_N];
vector <vector <int>> comp;
vector <int> st;

void newComp(int size = 0) {
  comp.emplace_back(); // новая пустая
  while (sz(st) > size) {
    comp.back().pb(st.back());
    st.pop_back();
  }
}

void find_bridges(int v, int parentEdge = -1) {
  if (up[v]) // уже были
    return;
  up[v] = tin[v] = ++timer;
  st.pb(v); // st - stack
  for (Edge e : g[v]) {
    if (e.id == parentEdge)
      continue;
    int u = e.to;
    if (!tin[u]) {
      int size = sz(st);
      find_bridges(u, e.id) ;
      if (up[u] > tin[v])
        newComp(size);
    }
    up[v] = min(up[v], up[u]);
  }
}
//после вызова find_bridges newComp() для корня
void run(int n) {
  forn(i, n) {
    if (!up[i]) {
      find_bridges(i);
      newComp();
    }
  }
}
```

30 Cactuses

```
namespace Cactus {
  int used[MAX_N], inCycle[MAX_N], dp[MAX_N],
   ↪  inProcess[MAX_N];
  vi g[MAX_N], sons[MAX_N], st, cycle;
  set<pii> forbidden;
  vector<vi> cycles;
  int curCycle = 0;

  void getCycles(int v, int p) {
    used[v] = 1;
    st.pb(v);
    for (int u : g[v])
      if (u != p && used[u] == 1) {
        cycle.clear();
        fornr (i, sz(st)) {
          cycle.pb(st[i]);
          inCycle[st[i]] = curCycle;
          if (st[i] == u)
            break;
        }
        curCycle++;
        reverse(all(cycle));
        cycles.pb(cycle);
      }
      else if (u != p && !used[u])
        getCycles(u, v);
    st.pop_back();
    used[v] = 2;
  }

  bool isForbidden(int v, int u) {
     return forbidden.count(mp(v, u)) || forbidden.count(mp(u,
      ↪  v));
  }

  void dfs(int v, int p);

  void calcTree(int v, int p) {
    used[v] = 1;
    for (int u : g[v])
      if (u != p && !isForbidden(v, u)) {
        dfs(u, v);
        //calc dp
      }
  }

  void calcCycle(int v, int p) {
    int c = inCycle[v];
    for (int u : cycles[c])
      inProcess[u] = 1;
    for (int u : cycles[c])
      for (int w : g[u])
        if (w != p && inCycle[w] != c)
          dfs(w, u), sons[u].pb(w);
    //calc dp on cactus
    for (int u : cycles[c])
      inProcess[u] = 0, used[u] = 1;
  }

  void dfs(int v, int p) {
    if (used[v])
      return;
    if (!inProcess[v] && inCycle[v] != -1)
      calcCycle(v, p);
    else
      calcTree(v, p);
  }

  int init(int n) {
    forn (i, n)
      inCycle[i] = -1;
    getCycles(0, -1);
    forn (i, n)
      used[i] = 0;
    dfs(0, -1);
    return dp[0];
  }
}
```

31 Cut Points

```
struct Edge {
  int to, id;
  Edge(int aa, int bb) : to(aa), id(bb) {}
};

vector<Edge> g[MAX_N]; // (to, id)
vector<int> st; // stack
bool used[MAX_M];
int tin[MAX_N], timer, is_cut[MAX_N], color[MAX_M], compCnt;

int dfs(int v, int parent = -1) {
  tin[v] = ++timer;
  int up = tin[v], x = 0, y = (parent != -1);
  for (Edge p : g[v]){
    int u = p.to, id = p.id;
    if (id != parent) {
      int t, size = sz(st);
      if (!used[id]){
        st.push_back(id);
        used[id] = 1;
      }
      if (!tin[u]) { // not visited yet
        t = dfs(u, id);
        if (t >= tin[v]){
          ++x, ++compCnt;
          while(sz(st) != size){
            color[st.back()] = compCnt;
            st.pop_back();
          }
        }
      } else
        t = tin[u];
      up = min(up, t);
    }
  }
  if(x + y >= 2)
    is_cut[v] = 1; // v is cut vertex
  return up;
}
```

32 DP tree

```
int dfs(int v) {
  forn(i, n + 1)
    dp[v][i] = -INF;
  dp[v][1] = num[v];
  int mxsz = 1;
  for (int to : g[v]) {
    int size = dfs(to);
    forba(i, 1, mxsz + 1)
      fornr(j, size + 1)
        dp[v][i + j] = max(dp[v][i + j], dp[v][i] +
          ↪  dp[to][j]);
    mxsz += size;
  }
  return mxsz;
}
```

33 Eulerian Cycle

```
struct Edge {
  int to, used;
  Edge(): to(-1), used(0) {}
  Edge(int v): to(v), used(0) {}
};

vector <Edge> edges[MAX_M];
vector <int>  g[MAX_N], res, ptr;
//не забывать чистить ptr

void dfs(int v) {
  for(; ptr[v] < sz(g[v]);) {
    int id = g[v][ptr[v]++];
    if (!edges[id].used) {
      edges[id].used = edges[id ^ 1].used = 1;
      dfs(edges[id].to);
//      res.pb(id); // для ребер
    }
  }
  res.pb(v); // в res вершины
}
```

34 Kuhn's algorithm

```
//первая доля - n вершин вторая доля - m вершин
//нумерация сквозная
const int MAX_N = 1e5 + 100;

int n, m, paired[2 * MAX_N], used[2 * MAX_N];
vector <int> g[MAX_N];

bool dfs(int v) {
  if (used[v])
    return false;
  used[v] = 1;
  for(int to : g[v])
    if (paired[to] == -1 || dfs(paired[to])) {
      paired[to] = v;
      paired[v] = to;
      return true;
    }
  return false;
}

int kuhn() {
  int ans = 0;
  forn(i, n + m)
    paired[i] = -1;
  for (int run = 1; run;) {
    run = 0;
    memset(used, 0, sizeof(used));
    forn(i, n)
      if (!used[i] && paired[i] == -1 && dfs(i)) {
        ans++;
        run = 1;
      }
  }
  return ans;
}
//Стартуем из вершин без пары из первой доли, ходим из первой
 ↪  доли, из второй - только по парсочу.
//Max независимое - A+, B-
//Min покрытие    - A-, B+

vector <int> minCover, maxIndependent;

void dfsCoverIndependent(int v) {
  if (used[v])
    return;
  used[v] = 1;
  for(int to : g[v])
    if (!used[to])
      used[to] = 1, dfsCoverIndependent(paired[to]);
}

//Сперва Куна
void findCoverIndependent() {
  memset(used, 0, sizeof(used));
  forn(i, n)
    if (paired[i] == -1)
      dfsCoverIndependent(i);
  forn(i, n)
    if(used[i])
      maxIndependent.pb(i);
    else
      minCover.pb(i);
  forab(i, n, n + m)
    if (used[i])
      minCover.pb(i);
    else
      maxIndependent.pb(i);
}
```

7 Math

35 CRT (KTO)

```
namespace Math {
  vi crt(vi a, vi mod) {
    int n = sz(a);
    vi x(n);
    forn (i, n) {
      x[i] = a[i];
      forn (j, i) {
         x[i] = inverse(mod[j], mod[i]) * (x[i] - x[j]) %
          ↪  mod[i];
         if (x[i] < 0)
```

```
        x[i] += mod[i];
      }
    }
    return x;
  }
}
```

36 Discrete Logariphm

```
namespace Math {
  int discreteLogariphm(int a, int b, int mod) { //returns x:
   ↪  a^x = b (mod mod) or -1, if no such x exists
    int sq = sqrt(mod);
    int sq2 = mod / sq + (mod % sq ? 1 : 0);
    vector<pii> powers(sq2);
    forn (i, sq2)
      powers[i] = mp(power(a, (i + 1) * sq, mod), i + 1);
    sort(all(powers));
    forn (i, sq + 1) {
       int cur = power(a, i, mod);
       cur = (cur * 1ll * b) % mod;
       auto it = lower_bound(all(powers), mp(cur, 0));
       if (it != powers.end() && it->fs == cur)
         return it->sc * sq - i;
    }
    return -1;
  }
}
```

37 Discrete Root

```
namespace Math {
  //returns x: x^k = a mod mod, mod is prime
  int discreteRoot(int a, int k, int mod) {
    if (a == 0)
      return 0;
    int g = primitiveRoot(mod);
    int y = discreteLogariphm(power(g, k, mod), a, mod);
    return power(g, y, mod);
  }
}
```

38 Eratosthenes

```
namespace Math {
  vi eratosthenes(int n) {
    vi minDiv(n + 1, 0);
    minDiv[1] = 1;
    for (int i = 2; i <= n; i++)
      if (minDiv[i] == 0)
        for (int j = i; j <= n; j += i)
          if (minDiv[j] == 0)
            minDiv[j] = i;
    return minDiv;
  }

  vi eratosthenesFast(int n) {
    vi minDiv(n + 1, 0);
    vi primes;
    minDiv[1] = 1;
    for (int i = 2; i <= n; i++) {
      if (minDiv[i] == 0) {
        minDiv[i] = i;
        primes.pb(i);
      }
      for (int j = 0; j < sz(primes) && primes[j] <= minDiv[i]
       ↪  && i * primes[j] <= n; j++)
        minDiv[i * primes[j]] = primes[j];
    }
    return minDiv;
```

```
  }
}
```

39 Factorial

```
namespace Math {
  //returns pair (rem, deg), where rem = n! % mod,
  //deg = k: mod^k | n!, mod is prime, O(mod log mod)
  pii fact(int n, int mod) {
    int rem = 1, deg = 0;
    int n2 = n;
    while (n2)
      n2 /= mod, deg += n2;
    while (n > 1) {
      rem = (rem * ((n / mod) % 2 ? -1 : 1) + mod) % mod;
      for (int i = 2; i <= n % mod; i++)
        rem = (rem * 1ll * i) % mod;
      n /= mod;
    }
    return mp(rem % p, deg);
  }
}
```

40 Gauss

```
namespace Math {
  const double EPS = 1e-9;

  int gauss(double **a, int n, int m) { //n is number of
   ↪  equations, m is number of variables
    int row = 0, col = 0;
    vi par(m, -1);
    vector<double> ans(m, 0);
    for (col = 0; col < m && row < n; col++) {
      int best = row;
      for (int i = row; i < n; i++)
        if (abs(a[i][col]) > abs(a[best][col]))
          best = i;
      if (abs(a[best][col]) < EPS)
        continue;
      par[col] = row;
      for (int i = 0; i <= m; i++)
        swap(a[row][i], a[best][i]);
      for (int i = 0; i < n; i++)
        if (i != row) {
          long double k = a[i][col] / a[row][col];
          for (int j = col; j <= m; j++)
            a[i][j] -= k * a[row][j];
        }
      row++;
    }
    int single = 1;
    for (int i = 0; i < m; i++)
      if (par[i] != -1)
        ans[i] = a[par[i]][m] / a[par[i]][i];
      else
        single = 0;
    for (int i = 0; i < n; i++) {
      long double cur = 0;
      for (int j = 0; j < m; j++)
        cur += ans[j] * a[i][j];
      if (abs(cur - a[i][m]) > EPS)
        return 0;
    }
    if (!single)
      return 2;
    return 1;
  }
}
```

41 Gauss mod 2

```
namespace Math {
  const int MAX = 1024;

  int gaussMod2(vector<bitset<MAX>> a, int n, int m) {
    int row = 0, col = 0;
    vi par(m, -1);
    for (col = 0; col < m && row < n; col++) {
      int best = row;
      for (int i = row; i < n; i++)
        if (a[i][col] > a[best][col])
          best = i;
      if (a[best][col] == 0)
        continue;
      par[col] = row;
      swap(a[row], a[best]);
      for (int i = 0; i < n; i++)
        if (i != row) {
          if (a[i][col])
            a[i] ^= a[row];
        }
      row++;
    }
    vi ans(m, 0);
    for (int i = 0; i < m; i++)
      if (par[i] != -1)
        ans[i] = a[par[i]][n] / a[par[i]][i];
    bool ok = 1;
    for (int i = 0; i < n; i++) {
      int cur = 0;
      for (int j = 0; j < m; j++)
        cur ^= (ans[j] & a[i][j]);
      if (cur != a[i][n])
        ok = 0;
    }
    return ok;
  }
}
```

42 Gcd

```
namespace Math {
  int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
  }

  int gcd(int a, int b, int &x, int &y) {
     if (b == 0) {
       x = 1, y = 0;
       return a;
     }
     else {
       int g = gcd(b, a % b, x, y);
      int newX = y;
      y = x - a / b * y;
      x = newX;
      return g;
    }
  }

  void diophant(int a, int b, int c, int &x, int &y) {
    int g = gcd(a, b, x, y);
    if (c % g != 0)
      return;
    x *= c / g, y *= c / g;
    //next solutions: x += b / g, y -= a / g
  }

  int inverse(int a, int mod) { //returns -1, if a and mod are
 ↪   not coprime
    int x, y;
    int g = gcd(a, mod, x, y);
    return g == 1 ? (x % mod + mod) % mod : -1;
  }

  vi inverseForAll(int mod) {
    vi r(mod, 0);
     r[1] = 1;
     for (int i = 2; i < mod; i++)
       r[i] = (mod - r[mod % i]) * (mod / i) % mod;
     return r;
  }
}
```

43 Gray

```
namespace Math {
  int gray(int n) {
    return n ^ (n >> 1);
  }

  int revGray(int n) {
     int k = 0;
     for (; n; n >>= 1)
       k ^= n;
     return k;
  }
}
```

44 Miller-Rabin Test

```
namespace Math {
  vector <int> primes = {2,3,5,7,11,13,17,19,23};

  bool isPrimeMillerRabin(ll n) {
    int k = 0;
    ll t = n - 1;
    while (t % 2 == 0)
      k++, t /= 2;
    for (auto p : primes) {
      ll g = __gcd(n, (ll) p);
      if (g > 1 && g < n)
        return 0;
      if (g == n)
        return 1;
      ll b = power(p, t, n);
      ll last = n - 1;
      bool was = 0;
      forn (i, k + 1) {
        if (b == 1 && last != n - 1)
          return 0;
        if (b == 1) {
          was = 1;
          break;
        }
        last = b;
        b = mul(b, b, n);
      }
      if (!was)
        return 0;
    }
    return 1;
  }
}
```

45 Phi

```
namespace Math {
  int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++)
      if (n % i == 0) {
        while (n % i == 0)
          n /= i;
        result -= result / i;
      }
    if (n > 1)
      result -= result / n;
    return result;
  }

  int inversePhi(int a, int mod) {
    return power(a, phi(mod) - 1, mod);
  }
}
```

46 Pollard

```
namespace Math {
  inline void pollardFoo(ll &x, ll mod) {
    x = (mul(x, x, mod) + 1) % mod;
  }

  vector <pair <ll, int> > factorize(ll n) {
    if (n == 1)
      return {};
    if (isPrimeMillerRabin(n))
      return {mp(n, 1)};
    if (n <= 100) {
      vector <pair <ll, int> > ans;
      for (int i = 2; i * i <= n; i++)
        if (n % i == 0) {
          int cnt = 0;
          while (n % i == 0)
            n /= i, cnt++;
          ans.pb(mp(i, cnt));
        }
      if (n != 1)
        ans.pb(mp(n, 1));
      sort(all(ans));
      return ans;
    }
    while (1) {
      ll a = rand() % n, b = a;
      while (1) {
        pollardFoo(a, n), pollardFoo(b, n), pollardFoo(b, n);
        ll g = __gcd(abs(a-b), n);
        if (g != 1) {
          if (g == n)
            break;
          else {
            auto ans1 = factorize(g);
            auto ans2 = factorize(n / g);
            vector <pair <ll, int> > ans;
            ans1.insert(ans1.end(), all(ans2));
            sort(all(ans1));
            for (auto np : ans1)
              if (sz(ans) == 0 || np.fs != ans.back().fs)
                ans.pb(np);
              else
                ans.back().sc += np.sc;
            return ans;
          }
        }
      }
    }
    assert(0);
  }
}
```

47 Primitive Root

```
namespace Math {
  int primitiveRoot(int mod) { //returns -1 if no primitive
   ↪   root exists
     vi fact;
     int ph = phi(mod);
     int n = mod;
     for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
          fact.pb(i);
           while (n % i == 0)
             n /= i;
        }
     }
     if (n > 1)
       fact.pb(n);
     forab (i, 2, mod + 1) {
        bool ok = 1;
        for (int j = 0; j < sz(fact) && ok; j++)
          ok &= power(i, ph / fact[j], mod) != 1;
        if (ok)
          return i;
     }
     return -1;
  }
}
```

48 Simpson

```
namespace Math {
  double f(double x) {
    return x;
  }

  double simpson(double a, double b, int iterNumber) {
    double res = 0, h = (b - a) / iterNumber;
    forn (i, iterNumber + 1)
      res += f(a + h * i) * ((i == 0) || (i == iterNumber) ? 1
       ↪   : ((i & 1) == 0) ? 2 : 4);
    return res * h / 3;
  }
}
```

8 Mix

49 Fast allocation (operator new)

```
#include <cassert>

/** Begin fast allocation */
const int MAX_MEM = 1e8;
int mpos = 0;
char mem[MAX_MEM];
inline void * operator new ( size_t n ) {
  char *res = mem + mpos;
  mpos += n;
  assert(mpos <= MAX_MEM);
  return (void *)res;
}
inline void operator delete ( void * ) { }
/** End fast allocation */

inline void * operator new [] ( size_t ) { assert(0); }
inline void operator delete [] ( void * ) { assert(0); }
```

50 Fast I/O (short)

```
inline int readChar();
inline int readInt();
template <class T> inline void writeInt( T x );
```

```
inline int readChar() {
  int c = getchar();
  while (c <= 32)
    c = getchar();
  return c;
}

inline int readInt() {
  int s = 0, c = getchar(), x = 0;
  if (c == ’-’)
    s = 1, c = getchar();
  while (’0’ <= c && c <= ’9’)
    x = x * 10 + c - ’0’, c = getchar();
  return s ? -x : x;
}

template <class T> inline void writeInt( T x ) {
  if (x < 0)
    putchar(’-’), x = -x;
  char s[24];
  int n = 0;
  while (x || !n)
    s[n++] = ’0’ + x % 10, x /= 10;
  while (n--)
    putchar(s[n]);
}
```

51 Masks tricks

```
forn(mask, 1 << d) {
  dp[mask][d] = 1;
  fornr(i, d) {
    dp[mask][i] = dp[mask][i + 1];
    if ((1 << i) & mask)
      dp[mask][i] += dp[mask ^ (1 << i)][i + 1];
  }
  cout << mask << " -> " << dp[mask][0] << ’\n’;
}

int num[64];

for (ULL i = 0; i < 64; ++i) {
  num[(1ull << i) % 67] = i;
}
```

52 Hash of pair

```
struct MyHash {
  size_t operator()(const pair<int,int> &t) const {
    return t.first * 239017 + t.second;
  }
};
```

9 Strings

53 Aho-Corasick

```
const int ALPHA = 26;
const int MAX_N = 1e5;

struct Node {
  int next[ALPHA], term; //Бор
  int go[ALPHA], suf, p, pch; //Автомат
  Node() {
    memset(next, -1, sizeof(next));
    term = 0;
    memset(go, -1, sizeof(go));
    suf = p = -1;
  }
};

Node g[MAX_N];
int last;

void add(const string &s) {
  int now = 0;
  for(char x : s) { // x -= ’a’
    if (g[now].next[x] == -1) {
      g[now].next[x] = ++last;
      g[last].p = now;
      g[last].pch = x;
    }
    now = g[now].next[x];
  }
  g[now].term = 1;
}

int go(int v, int c);

int get_link(int v) {
  if (g[v].suf == -1) {
    if (!v || !g[v].p)
      g[v].suf = 0;
    else
      g[v].suf = go(get_link(g[v].p), g[v].pch);
  }
  return g[v].suf;
}

int go(int v, int c) {
  if (g[v].go[c] == -1) {
    if (g[v].next[c] != -1)
      g[v].go[c] = g[v].next[c];
    else
      g[v].go[c] = !v ? 0 : go(get_link(v), c);
  }
  return g[v].go[c];
}
```

54 Prefix-function

```
// pr[len] - для префикса длины len
int k = 0;
pr[0] = pr[1] = 0;
for (int i = 2; i <= n; i++) {
  k = pr[i - 1];
  while (k && s[k] != s[i - 1])
    k = pr[k];
  if (s[k] == s[i - 1])
    k++;
  pr[i] = k;
}
```

55 Z-function

```
//z[i] - с позиции i сколько матчится
int l = -1, r = -1;
z[0] = 0;
for (int i = 1; i < n; i++) {
  int k = 0;
  if (r >= i)
    k = min(z[i - l], r - i);
  while (i + k < n && s[i + k] == s[k])
    k++;
  z[i] = k;
  if (i + z[i] > r)
    l = i, r = i + z[i];
}
```

56 Hash

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

typedef long long LL;

inline int byMod(int a, int m){
    return a >= m ? a - m : a;
}

const int MX = 1e9 + 9, MY = 1e9 + 7;

//typedef unsigned long long H;
struct H{
    int x, y;
    H(): x(0), y(0){}
    H(int _x): x(_x), y(_x){}
    H(int _x, int _y): x(_x), y(_y){}
    inline H operator +(const H &B) const{return H(byMod(x +
    ↪   B.x, MX), byMod(y + B.y, MY));}
    inline H operator -(const H &B) const{return H(byMod(x +
    ↪   MX - B.x, MX), byMod(y + MY - B.y, MY));}
    inline H operator *(LL k) const{return H(int((x * k) %
    ↪   MX), int((y * k) % MY));}
    inline H operator *(const H &B) const{return H(int((LL(x)
    ↪   * B.x) % MX), int((LL(y) * B.y) % MY));}
    inline bool operator ==(const H &B) const{return x == B.x
    ↪   && y == B.y;}
    inline bool operator !=(const H &B) const{return x != B.x
    ↪   || y != B.y;}
    inline bool operator <(const H &B) const{return x < B.x ||
    ↪   (x == B.x && y < B.y);}
    explicit inline operator LL() const{return (LL)x * MY + y
    ↪   + 1;} // > 0
};

const int P = 239017, MAX_N = 1e6 + 10;
H deg[MAX_N], h[MAX_N];
char s[MAX_N];

inline H Get(int a, int l){
    return h[a + l] - h[a] * deg[l];
}

int main(){
#ifdef LOCAL
    assert(freopen("test.in", "r", stdin));
    assert(freopen("test.out", "w", stdout));
#endif

    gets(s);
    int L = strlen(s);
    deg[0] = 1;
    for(int i = 0; i < L; ++i)
        h[i + 1] = h[i] * P + s[i], deg[i + 1] = deg[i] * P;

    return 0;
}
```

57 Manaker

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

using namespace std;

#define forn(i, n) for (int i = 0; i < (int)(n); i++)

void manaker( int n, char *s, int *z0, int *z1 ) {
  forn(t, 2) {
    int *z = t ? z1 : z0, l = -1, r = -1; // [l..r]
    forn(i, n - t) {
      int k = 0;
      if (r > i + t) {
        int j = l + (r - i - t);
        k = min(z[j], j - l);
      }
      while (i - k >= 0 && i + k + t < n && s[i - k] == s[i +
      ↪   k + t])
        k++;
      z[i] = k;
      if (k && i + k + t > r)
        l = i - k + 1, r = i + k + t - 1;
    }
  }
}

const int N = 1e5;

int n, r0[N], r1[N];
char s[N + 1];

int main() {
  assert(freopen("palindrome.in", "rt", stdin));
  assert(freopen("palindrome.out", "wt", stdout));

  gets(s);
  n = strlen(s);
  manaker(n, s, r0, r1);
  cout << accumulate(r0, r0 + n, 0LL) + accumulate(r1, r1 + n,
  ↪   0LL) - n << endl;
  return 0;
}
```

58 Palindromic Tree

```
#define fill(a, x) memset(a, x, sizeof(a))

template<const int N>
struct PalindromeTree {
  struct Vertex {
    int suf, len, next[26];
  };

  int vn, v;
  Vertex t[N + 2];
  int n, s[N];

  int get( int i ) { return i < 0 ? -1 : s[i]; }

  void init() {
    fill(t, 0);
    t[0].len = -1, vn = 2, v = 0, n = 0;
  }

  void add( int ch ) {
    s[n++] = ch;
    while (v != 0 && ch != get(n - t[v].len - 2))
      v = t[v].suf;
    int &r = t[v].next[ch];
    if (!r) {
      t[vn].len = t[v].len + 2;
      if (!v)
        t[vn].suf = 1;
      else {
        v = t[v].suf;
        while (v != 0 && ch != get(n - t[v].len - 2))
          v = t[v].suf;
        t[vn].suf = t[v].next[ch];
      }
      r = vn++;
    }
    v = r;
  }
};

const int N = 1e5;

PalindromeTree<N> pt;

char s[N + 1];

int main() {
  gets(s);
  int n = strlen(s);
  pt.init();
  forn(i, n) {
    pt.add(s[i] - 'a');
    printf("%d ", pt.vn - 2);
  }
  return 0;
}
```

59 Suffix Array (+stable)

```
const int MAX_N = 250000;

int n, num[MAX_N + 1];
char s[MAX_N + 1];
int p[MAX_N], col[MAX_N], p2[MAX_N], len[MAX_N];

void BuildArray(){
  int ma = max(n, 256);
  forn(i, n)
    col[i] = s[i], p[i] = i;

  for (int k2 = 1; k2 / 2 < n; k2 *= 2){
    int k = k2 / 2;
    memset(num, 0, sizeof(num));
    forn(i, n)
      num[col[i] + 1]++;
    forn(i, ma)
      num[i + 1] += num[i];
    forn(i, n)
      p2[num[col[(p[i] - k + n) % n]]++] = (p[i] - k + n) % n;

    int cc = 0;
    forn(i, n){
      if (i && (col[p2[i]] != col[p2[i - 1]] ||
      col[(p2[i] + k) % n] != col[(p2[i - 1] + k) % n]))
        cc++;
      num[p2[i]] = cc;
    }
    forn(i, n)
      p[i] = p2[i], col[i] = num[i];
  }

  // make it stable
  memset(num, 0, sizeof(num));
  forn(i, n)
    num[col[i] + 1]++;
  forn(i, ma)
    num[i + 1] += num[i];
  forn(i, n)
    p2[num[col[i]]++] = i;
  forn(i, n)
    p[i] = p2[i];

  // calc inverse permutation
  forn(i, n)
    p2[p[i]] = i;
}

void BuildLCP(){
  int lcp = 0;
  forn(i, n){
    int j = p2[i];
    lcp = max(0, lcp - 1);
    if (j != n - 1)
      while (lcp < n && s[(p[j] + lcp) % n] == s[(p[j + 1] +
      ↪   lcp) % n])
        lcp++;
    len[j] = lcp;
    if (j != n - 1 && p[j + 1] == n - 1)
      lcp = 0;
  }
}

int main()
{
  scanf("%d%s", &n, s);

  BuildArray();
  BuildLCP();

  // res = sum of all LCP[i,i+1]
  LL res = 0;
  forn(i, n)
    res += len[i];
  printf("%.3f\n", (double)res / (n - 1));
  return 0;
}
```

60 Suffix Automaton

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

struct Vx{
    static const int AL = 26;
    int len, suf;
    int next[AL];
    Vx(){}
    Vx(int l, int s):len(l), suf(s){}
};

struct SA{
    static const int MAX_LEN = 1e5 + 100, MAX_V = 2 * MAX_LEN;
    int last, vcnt;
    Vx v[MAX_V];

    SA(){
        vcnt = 1;
        last = newV(0, 0); // root = vertex with number 1
    }
    int newV(int len, int suf){
        v[vcnt] = Vx(len, suf);
        return vcnt++;
    }

    int add(char ch){
        int p = last, c = ch - 'a';
        last = newV(v[last].len + 1, 0);
        while(!v[p].next[c])
            v[p].next[c] = last, p = v[p].suf;
        if(!p)
            v[last].suf = 1;
        else{
            int q = v[p].next[c];
            if (v[q].len == v[p].len + 1)
                v[last].suf = q;
            else{
                int r = newV(v[p].len + 1, v[q].suf);
                v[last].suf = v[q].suf = r;
                memcpy(v[r].next, v[q].next,
                 ↪   sizeof(v[r].next));
                while(p && v[p].next[c] == q)
```

```
                    v[p].next[c] = r, p = v[p].suf;
                }
            }
            return last;
        }
};
```

61 Suffix Tree

```
const int MAX_L=1e5+10;
char S[MAX_L];
int L;

struct Node;
struct Pos;
typedef Node *pNode;
typedef map<char,pNode> mapt;

struct Node{
  pNode P,link;
  int L,R;
  mapt next;

  Node():P(NULL),link(this),L(0),R(0){}
  Node(pNode P,int L,int R):P(P),link(NULL),L(L),R(R){}

  inline int elen() const{return R-L;}
  inline pNode add_edge(int L,int R){return next[S[L]]=new
  ↪    Node(this,L,R);}
};

struct Pos{
  pNode V;
  int up;
  Pos():V(NULL),up(0){}
  Pos(pNode V,int up):V(V),up(up){}

  pNode split_edge() const{
    if(!up)
      return V;
    int L=V->L, M=V->R-up;
    pNode P=V->P, n=new Node(P,L,M);
    P->next[S[L]]=n;
    n->next[S[M]]=V;
    V->P=n, V->L=M;
    return n;
  }
  Pos next_char(char c) const{
    if(up)
      return S[V->R-up]==c ? Pos(V,up-1) : Pos();
    else{
      mapt::iterator it=V->next.find(c);
      return it==V->next.end() ? Pos() :
       ↪    Pos(it->snd,it->snd->elen()-1);
    }
  }
};

Pos go_down(pNode V,int L,int R){
  if(L==R)
    return Pos(V,0);
  while(1){
    V=V->next[S[L]];
    L+=V->elen();
    if(L>=R)
      return Pos(V,L-R);
  }
}

inline pNode calc_link(pNode &V){
  if(!V->link)
    V->link=go_down(V->P->link,V->L+!V->P->P,V->R).split_edge();
  return V->link;
}

Pos add_char(Pos P,int k){
  while(1){
    Pos p=P.next_char(S[k]);
    if(p.V)
      return p;
    pNode n=P.split_edge();
    n->add_edge(k,MAX_L);
    if(!n->P)
      return Pos(n,0);
    P=Pos(calc_link(n),0);
  }
}

pNode Root;
void make_tree(){
  Root=new Node();
  Pos P(Root,0);
  forn(i,L)
    P=add_char(P,i);
}
```

10 C++ Tricks

62 Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
 ↪    tree_order_statistics_node_update> ordered_set;

void example() {
  ordered_set X;
  X.insert(1);
  cout << *X.find_by_order(1) << endl;
  cout << X.order_of_key(1) << endl;
}
```

11 Notes

63 Tree

Приемы для работы с деревьями:

1. Двоичные подъемы
2. Поддеревья как отрезки Эйлерова обхода
3. Вертикальные пути в Эйлеровом обходе (на ребрах вниз $+k$, на ребрах вверх $-k$).
4. Храним в вершине значение функции на пути от корня до нее, дальше LCA.
5. Спуск с DFS, поддерживаем ДО на пути до текущей вершины.
6. Heavy-light decomposition
7. Centroid decomposition

64 DP

Табличка с оптимизациями для динамики:

Name	Original recurrence	Sufficient Condition	From	To
Convex Hull Optimization 1	$dp[i] = \min\limits_{j<i} dp[j] + b[j] \cdot a[i]$	$b[j] \geqslant b[j+1] \vert\vert a[i] \leqslant a[i+1]$	$O(n^2)$	$O(n)$
Convex Hull Optimization 2	$dp[i][j] = \min\limits_{k<j} dp[i-1][k] + b[k] \cdot a[j]$	$b[k] \geqslant b[k+1] \vert\vert a[j] \leqslant a[j+1]$	$O(kn^2)$	$O(kn)$
D&C Optimization	$dp[i][j] = \min\limits_{k<j} dp[i-1][k] + c[k][j]$	$p[i,j] \leqslant p[i,j+1]$	$O(kn^2)$	$O(kn \log n)$
Knuth Optimization	$dp[i][j] = \min\limits_{i<k<j} dp[i][k] + dp[k][j] + c[i][j]$	$p[i,j-1] \leqslant p[i,j] \leqslant p[i+1,j]$	$O(n^3)$	$O(n^2)$

65 Combinatorics

Биномиальные коэффициенты:

Теорема Люка для биномиальных коэффициентов: Хотим посчитать C_n^k, разложим в p-ичной системе счисления, $n = (n_0, n_1, \ldots), k = (k_0, k_1, \ldots)$. $ans = C_{n_0}^{k_0} \cdot C_{n_1}^{k_1} \cdot \ldots$

Способы вычисления C_n^k:

1. $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$

   precalc: $O(n^2)$, query: $O(1)$.

2. $C_n^k = \frac{n!}{k!(n-k)!}$, предподсчитываем факториалы

   precalc: $O(n \log n)$, query: $O(\log n)$

3. Теорема Люка

   precalc: $O(p \log p)$, query: $O(logp)$.

4. $C_n^k = C_n^{k-1} \cdot \frac{n-k+1}{k}$

5. $C_n^k = \frac{n!}{k!(n-k)!}$, для каждого факториала считаем степень вхождения и остаток

   precalc: $O(p \log p)$, query: $O(logp)$.

$$C_n^{\frac{n}{2}} = \frac{2^n}{\sqrt{\frac{\pi n}{2}}}$$

66 Masks

Считаем динамику по маскам за $O(2^n \cdot n)$ $f[mask] = sum$ по $submask$ $g[submask]$. $dp[mask][i]$ — значение динамики для маски $mask$, если младшие i бит в ней зафиксированы (то есть мы не можем удалять оттуда). Ответ в $dp[mask][0]$. $dp[mask][len] = g[mask]$. Если i-ый бит 0, то $dp[mask][i] = dp[mask][i+1]$, иначе $dp[mask][i] = dp[mask][i + 1] + dp[mask^{(}1 << i)][i + 1]$.

Старший бит: предподсчет.

Младший бит: $x\& \sim (-x)$

Чтобы по степени двойки получить логарифм, можно воспользоваться тем, что все степени двойки имеют разный остаток по модулю 67.

```
for (int mask = 0; mask < (1 << n); mask++)
  submask : for (int s = mask; s; s = (s - 1) & mask)
  supmask : for (int s = mask; s < (1 << n); s = (s + 1) |
   ↪  mask)
```

67 Flows

Потоки:

Name	Asympthotic
Ford-Fulkerson	$O(\vert f\vert \cdot E)$
Ford-Fulkerson with scaling	$O(\log \vert f\vert \cdot E^2)$
Edmonds-Karp	$O(V \cdot E^2)$
Dinic	$O(V^2 \cdot E)$
Dinic with scaling	$O(V \cdot E \cdot \log C)$
Dinic on bipartite graph	$O(E\sqrt{V})$
Dinic on unit network	$O(E\sqrt{E})$

L-R потоки: Есть граф с недостатками или избытками в каждой вершине. Создаем фиктивные исток и сток (из истока все ребра в избытки, из недостатков все ребра в сток). Теперь пусть у нас есть L-R граф, для каждого ребра e $(v \to u)$ известны L_e и R_e. Добавим в v избыток L_e, в u недостаток L_e, а пропускную способность сделаем $R_e - L_e$. Получили решение задачи о LR-циркуляции. Если у нас обычный граф с истоком и стоком, то добавляем бесконечное ребро из стока в сток и ищем циркуляцию. Таким образом нашли удовлетворяющий условиям LR-поток. Если хотим максимальный поток, то на остаточной сети запускаем поиск максимального потока. В новом графе в прямую сторону пропускная способность равна $R_e - f_e$, в обратную $f_e - L_e$.

MinCostCirculation: Пока есть цикл отрицательного веса, запускаем алгоритм Карпа и пускаем максимальный поток по найденному циклу.

68 Grandi

Теорема Шпрага-Гранди: берем mex всех значений функции Гранди по состояниям, в которые можем перейти из данного. Если сумма независимых игр, то значение функции Гранди равно xor значений функций Гранди по всем играм. Бывает полезно вывести первые n значений и поискать закономерность. Часто сводится к xor по чему-нибудь.