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

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

using namespace std;

#define pb push_back
#define mp make_pair
#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 = (n) - 1; i >= 0; --i)
#define forab(i, a, b) for (int i = (a); i < (b); ++i)
#define all(c) (c).begin(), (c).end()

using ll = long long;
using vi = vector<int>;
using pii = pair<int, int>;

#define FNAME ""

int main() {
#ifdef LOCAL
  freopen(FNAME".in", "r", stdin);
  freopen(FNAME".out", "w", stdout);
#endif
  cin.tie(0);
  ios_base::sync_with_stdio(0);

  return 0;
}
```

3 Stress

```
#!/bin/bash

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

4 Java

```
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 Int

```cpp
constexpr int BASE = 1000000000;
constexpr int BASE_DIGITS = 9;

struct BigInt {
  // value == 0 is represented by empty z
  vi z; // digits
  // sign == 1/-1 <==> value >=/< 0
  int sign;
    BigInt(): sign(1) {}
  BigInt(ll v) { *this = v; }
    BigInt& operator=(ll v) {
    sign = v < 0 ? -1 : 1; v *= sign;
    z.clear(); for (; v > 0; v = v / BASE) z.pb((int) (v % BASE));
    return *this;
  }
    BigInt& operator+=(const BigInt& other) {
    if (sign == other.sign) {
      for (int i = 0, carry = 0; i < sz(other.z) || carry; ++i) {
        if (i == sz(z)) z.pb(0);
        z[i] += carry + (i < sz(other.z) ? other.z[i] : 0);
        carry = z[i] >= BASE;
        if (carry) z[i] -= BASE;
      }
    } else if (other != 0 /* prevent infinite loop */) {
      *this -= -other;
    }
    return *this;
  }
    friend BigInt operator+(BigInt a, const BigInt& b) { return a += b; }
    BigInt& operator-=(const BigInt& other) {
    if (sign == other.sign) {
      if ((sign == 1 && *this >= other) || (sign == -1 && *this <= other)) {
        for (int i = 0, carry = 0; i < sz(other.z) || carry; ++i) {
          z[i] -= carry + (i < sz(other.z) ? other.z[i] : 0);
          carry = z[i] < 0;
          if (carry)
            z[i] += BASE;
        }
        trim();
      } else {
        *this = other - *this;
        this->sign = -this->sign;
      }
    } else
      *this += -other;
    return *this;
  }
    friend BigInt operator-(BigInt a, const BigInt& b) { return a -= b; }
    BigInt& operator*=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < sz(z) || carry; ++i) {
      if (i == sz(z))
        z.pb(0);
      ll cur = (ll) z[i] * v + carry;
      carry = (int) (cur / BASE);
      z[i] = (int) (cur % BASE);
    }
    trim();
    return *this;
  }
    BigInt operator*(int v) const { return BigInt(*this) *= v; }
    friend pair<BigInt, BigInt> divmod(const BigInt& a1, const BigInt& b1) {
    int norm = BASE / (b1.z.back() + 1);
    BigInt a = a1.abs() * norm;
    BigInt b = b1.abs() * norm;
    BigInt q, r;
    q.z.resize(sz(a.z));
    fornr (i, sz(a.z)) {
      r *= BASE, r += a.z[i];
      int s1 = sz(b.z) < sz(r.z) ? r.z[sz(b.z)] : 0;
      int s2 = sz(b.z) - 1 < sz(r.z) ? r.z[sz(b.z) - 1] : 0;
      int d = (int) (((ll) s1 * BASE + s2) / b.z.back());
      r -= b * d;
      while (r < 0) r += b, --d;
      q.z[i] = d;
    }
    q.sign = a1.sign * b1.sign, r.sign = a1.sign;
    q.trim(), r.trim();
    return {q, r / norm};
  }
    BigInt operator/(const BigInt& v) const { return divmod(*this, v).fst; }
    BigInt operator%(const BigInt& v) const { return divmod(*this, v).snd; }
    BigInt& operator/=(int v) {
    if (v < 0) sign = -sign, v = -v;
    int rem = 0;
    fornr (i, sz(z)) {
      ll cur = z[i] + rem * (ll) BASE;
      z[i] = (int) (cur / v);
      rem = (int) (cur % v);
    }
    trim();
    return *this;
  }
    BigInt operator/(int v) const { return BigInt(*this) /= v; }
    int operator%(int v) const {
    if (v < 0) v = -v;
    int m = 0;
    fornr (i, sz(z))
      m = (int) ((z[i] + m * (ll) BASE) % v);
    return m * sign;
  }
    BigInt& operator*=(const BigInt& v) { return *this = *this * v; }
  BigInt& operator/=(const BigInt& v) { return *this = *this / v; }
    bool operator<(const BigInt& v) const {
    if (sign != v.sign) return sign < v.sign;
    if (sz(z) != sz(v.z)) return sz(z) * sign < sz(v.z) * v.sign;
    fornr (i, sz(z))
      if (z[i] != v.z[i])
        return z[i] * sign < v.z[i] * sign;
    return false;
  }
    bool operator>(const BigInt& v) const { return v < *this; }
  bool operator<=(const BigInt& v) const { return !(v < *this); }
  bool operator>=(const BigInt& v) const { return !(*this < v); }
    bool operator==(const BigInt& v) const { return !(*this < v) && !(v < *this); }
    bool operator!=(const BigInt& v) const { return *this < v || v < *this; }
    void trim() {
    while (!z.empty() && z.back() == 0) z.pop_back();
    if (z.empty()) sign = 1;
  }
  bool isZero() const { return z.empty(); }
  friend BigInt operator-(BigInt v) {
    if (!v.z.empty()) v.sign = -v.sign;
    return v;
  }
  BigInt abs() const {
    return sign == 1 ? *this : -*this;
  }
  void read(const string& s) {
    sign = 1, z.clear();
    int pos = 0;
    while (pos < sz(s) && (s[pos] == '-' || s[pos] == '+')) {
      if (s[pos] == '-') sign = -sign;
      ++pos;
    }
    for (int i = sz(s) - 1; i >= pos; i -= BASE_DIGITS) {
      int x = 0;
      forab (j, max(pos, i - BASE_DIGITS + 1), i)
        x = x * 10 + s[j] - '0';
      z.pb(x);
    }
    trim();
  }
  friend ostream &operator<<(ostream& stream, const BigInt& v) {
```

```
    if (v.sign == -1)
      stream << '-';
    stream << (v.z.empty() ? 0 : v.z.back());
    fornr (i, sz(v.z) - 1)
      stream << setw(BASE_DIGITS) << setfill('0') << v.z[i];
    return stream;
  }
  static vi convertBase(const vi& a, int oldDigits, int
↪ newDigits) {
    vector<ll> p(max(oldDigits, newDigits) + 1);
    p[0] = 1;
    for (int i = 1; i < sz(p); i++)
      p[i] = p[i - 1] * 10;
    vi res;
    ll cur = 0;
    int curDigits = 0;
    for (int v : a) {
      cur += v * p[curDigits];
      curDigits += oldDigits;
      while (curDigits >= newDigits) {
        res.pb(int(cur % p[newDigits]));
        cur /= p[newDigits];
        curDigits -= newDigits;
      }
    }
    res.pb((int) cur);
    while (!res.empty() && res.back() == 0) res.pop_back();
    return res;
  }
  static vll karatsubaMultiply(const vll& a, const vll& b) {
    int n = sz(a);
    vll res(n + n);
    if (n <= 32) {
      for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
          res[i + j] += a[i] * b[j];
      return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k), a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k), b2(b.begin() + k, b.end());
    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);
    forn (i, k) a2[i] += a1[i];
    forn (i, k) b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    forn (i, sz(a1b1)) r[i] -= a1b1[i];
    forn (i, sz(a2b2)) r[i] -= a2b2[i];
    forn (i, sz(r)) res[i + k] += r[i];
    forn (i, sz(a1b1)) res[i] += a1b1[i];
    forn (i, sz(a2b2)) res[i + n] += a2b2[i];
    return res;
  }
  BigInt operator*(const BigInt& v) const {
    vi a6 = convertBase(this->z, BASE_DIGITS, 6);
    vi b6 = convertBase(v.z, BASE_DIGITS, 6);
    vll a(all(a6)), b(all(b6));
    while (sz(a) < sz(b)) a.pb(0);
    while (sz(b) < sz(a)) b.pb(0);
    while (sz(a) & (sz(a) - 1)) a.pb(0), b.pb(0);
    vll c = karatsubaMultiply(a, b);
    BigInt res;
    res.sign = sign * v.sign;
    int carry = 0;
    forn (i, sz(c)) {
      ll cur = c[i] + carry;
      res.z.push_back((int) (cur % 1000000));
      carry = (int) (cur / 1000000);
    }
    res.z = convertBase(res.z, 6, BASE_DIGITS);
    res.trim();
    return res;
  }
};
```

6 FFT

```
int rev[N];

//using Num = complex<dbl>;
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; }
};

Num rt[N];

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;
}

void fft(Num *a, int n) {
  assert(rev[1]); // don't forget to init
  int q = 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 fftInv(Num *a, int n) {
  fft(a, n);
  reverse(a + 1, a + n);
  forn (i, n)
    a[i] /= n;
}

void doubleFft(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[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, 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(PI / p), sin(PI / p));
    forab (i, p / 2, p)
      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[N], tb[N], tf[N], tg[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) << HALF) + ((cc % mod) << (2 *
↪   HALF))) % mod);
  }
}

int tc[N], td[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

```
vi g[N];
int d[N], par[N], centroid;
//d and par - in centroid tree

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

//fill(d, d + n, -1)
//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;
  calcInComponent(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 _k, int _b): k(_k), b(_b) {}
  ll get(int x) { return b + k * 1ll * x; }
  bool operator<(const Line &l) const { return k < l.k; } //
↪  change to > in case of different order
};

// Checks if intersection of (a, b) is on the left from (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);
}

struct Convex {
  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; // change to > in case of different
↪  order
      if (st[m - 1].get(x) < st[m].get(x))
        l = m;
      else
        r = m;
    }
    return l;
  }
  Convex() {}
  Convex(vector<Line> &lines) {
    st.clear();
    for(Line &l : lines)
      add(l);
  }
  Convex(Line line) { st.pb(line); }
  Convex(const Convex &a, const Convex &b) {
    vector<Line> lines;
    lines.resize(sz(a.st) + sz(b.st));
    merge(all(a.st), all(b.st), lines.begin());
    st.clear();
    for(Line &l : lines)
```

```
    add(l);
  }
};
```

10 DSU

```
int pr[N];

int get(int v) {
  return v == pr[v] ? v : pr[v] = get(pr[v]);
}

bool unite(int v, int u) {
  v = get(v), u = get(u);
  if (v == u) return 0;
  pr[u] = v;
  return 1;
}

void init(int n) {
  forn (i, n) pr[i] = i;
}
```

11 Fenwick Tree

```
int t[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 n, int val) {
  for (; ind < n; ind |= (ind + 1))
    t[ind] += val;
}

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

12 Hash Table

```
using H = ll;
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){
  int 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 k = get(hash);
  return ht[k] != 0;
}
```

13 Heavy Light Decomposition

```
vi g[N];
int size[N], comp[N], num[N], top[N], pr[N], tin[N], tout[N];
vi t[N], toPush[N], lst[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]
  }
}
```

14 Next Greater in Segment Tree

```
int t[4 * N], tSize = 1;

// Find position > pos with val > x.
int nextGreaterX(int v, int l, int r, int pos, int x) {
  if (r <= pos + 1 || t[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;
}
```

15 Sparse Table

```
int st[N][LOG];
int lg[N];

int get(int l, int r) { // [l, r)
  int curLog = lg[r - l];
  return min(st[l][curLog], st[r - (1 << curLog)][curLog]);
}

void initSparseTable(int *a, int n) {
  lg[1] = 0;
  forab (i, 2, n + 1) lg[i] = lg[i / 2] + 1;
  forn (i, n) st[i][0] = a[i];
  forn (j, lg[n])
    forn (i, n - (1 << (j + 1)) + 1)
      st[i][j + 1] = min(st[i][j], st[i + (1 << j)][j]);
}
```

16 Fenwick Tree 2D

```
ll a[4][N][N];
int n, m;

inline int f(int x) { return x & ~(x - 1); }

inline void add(int k, int x, int y, ll val) {
  for (; x <= n; x += f(x))
    for (int j = y; j <= m; j += f(j))
      a[k][x][j] += val;
}

inline ll get(int k, int x, int y) {
  ll s = 0;
  for (; x > 0; x -= f(x))
    for (int j = y; j > 0; j -= f(j))
      s += a[k][x][j];
  return s;
}

inline ll get(int x, int y) {
  return ll(x + 1) * (y + 1) * get(0, x, y) - (y + 1) * get(1, x,
↪ y)
      - (x + 1) * get(2, x, y) + get(3, x, y);
}

inline void add(int x, int y, ll val) {
  add(0, x, y, val);
  add(1, x, y, val * x);
  add(2, x, y, val * y);
  add(3, x, y, val * x * y);
}

inline ll get(int x_1, int y_1, int x_2, int y_2) {
  return get(x_2, y_2) - get(x_1 - 1, y_2) - get(x_2, y_1 - 1) +
↪ get(x_1 - 1, y_1 - 1);
}

// Adds val to corresponding rectangle
inline void add(int x_1, int y_1, int x_2, int y_2, ll val) {
  add(x_1, y_1, val);
  if (y_2 < m) add(x_1, y_2 + 1, -val);
  if (x_2 < n) add(x_2 + 1, y_1, -val);
  if (x_2 < n && y_2 < m) add(x_2 + 1, y_2 + 1, val);
}
```

17 Segment Tree 2D

```
int tSize = (1 << 10);

struct Node1D {
  Node1D *l, *r;
  ll val, need;
  Node1D(): l(nullptr), r(nullptr), val(0), need(0) {}
  inline void norm() {
    if(!l) l = new Node1D();
    if(!r) r = new Node1D();
  }
  ll get(int ql, int qr, int vl = 0, int vr = tSize) {
    if(vl >= qr || ql >= vr)
      return 0;
    if(ql <= vl && vr <= qr)
      return val;
    int a = max(vl, ql), b = min(vr, qr), vm = (vl + vr) / 2;
    norm();
    return l->get(ql, qr, vl, vm) + r->get(ql, qr, vm, vr) + need
↪ * ll(b - a);
  }
  void add(int ql, int qr, int x, int vl = 0, int vr = tSize) {
    if (ql >= vr || vl >= qr)
      return;
    if (ql <= vl && vr <= qr){
      need += x;
      val += x * ll(vr - vl);
      return;
    }
    int vm = (vl + vr) / 2;
    norm();
    l->add(ql, qr, x, vl, vm), r->add(ql, qr, x, vm, vr);
    val = l->val + r->val + need * (vr - vl);
  }
};

struct Node2D {
  Node2D *l, *r;
  Node1D *val, *need;
  Node2D(): l(nullptr), r(nullptr), val(new Node1D()), need(new
↪ Node1D()) {}
  inline void norm() {
    if(!l) l = new Node2D();
    if(!r) r = new Node2D();
  }
  ll get(int ql0, int qr0, int ql1, int qr1, int vl = 0, int vr =
↪ tSize) {
    if(vl >= qr0 || ql0 >= vr)
      return 0;
    if(ql0 <= vl && vr <= qr0)
      return val->get(ql1, qr1);
    int a = max(vl, ql0), b = min(vr, qr0), vm = (vl + vr) / 2;
    norm();
    return l->get(ql0, qr0, ql1, qr1, vl, vm) + r->get(ql0, qr0,
↪ ql1, qr1, vm, vr) + need->get(ql1, qr1) * ll(b - a);
  }
  void add(int ql0, int qr0, int ql1, int qr1, int x, int vl = 0,
↪ int vr = tSize) {
    if (ql0 >= vr || vl >= qr0)
      return;
    if (ql0 <= vl && vr <= qr0){
      need->add(ql1, qr1, x);
      val->add(ql1, qr1, x * ll(vr - vl));
      return;
    }
    int a = max(ql0, vl), b = min(qr0, vr), vm = (vl + vr) / 2;
    norm();
    l->add(ql0, qr0, ql1, qr1, x, vl, vm), r->add(ql0, qr0, ql1,
↪ qr1, x, vm, vr);
```

```
    val->add(ql1, qr1, x * ll(b - a));
  }
};
```

4 Dynamic Programming

18 LIS

```
int longestIncreasingSubsequence(vi a) {
  int n = sz(a);
  vi d(n + 1, INF);
  d[0] = -INF;
  forn (i, n)
    *upper_bound(all(d), a[i]) = a[i];
  fornr (i, n + 1) if (d[i] != INF) return i;
  return 0;
}
```

19 DP tree

```
int dp[N][N], a[N];
vi g[N];

int dfs(int v, int n) {
  forn (i, n + 1)
    dp[v][i] = -INF;
  dp[v][1] = a[v];
  int curSz = 1;
  for (int to : g[v]) {
    int toSz = dfs(to, n);
    for (int i = curSz; i >= 1; i--)
      fornr (j, toSz + 1)
        dp[v][i + j] = max(dp[v][i + j], dp[v][i] + dp[to][j]);
    curSz += toSz;
  }
  return curSz;
}
```

20 Masks tricks

```
int dp[(1 << MASK)][MASK];

void calcDP(int n) {
  forn(mask, 1 << n) {
    dp[mask][n] = 1;
    fornr(i, n) {
      dp[mask][i] = dp[mask][i + 1];
      if ((1 << i) & mask)
        dp[mask][i] += dp[mask ^ (1 << i)][i + 1];
    }
  }
}
```

5 Flows

21 Utilities

```
vi g[N];

// 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
}
```

22 Ford-Fulkerson

```
int used[N], pr[N];
int curTime = 1;

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 s, int t) {
  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;
}
```

23 Dinic

```
int pr[N], d[N], q[N], first[N];

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) {
  forn (i, n) 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 s, int t) {
  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;
}
```

24 Hungarian

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

// min = sum of a[pa[i],i]
// you may optimize speed by about 15%, just change all vectors to
↪  static arrays
vi Hungarian(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;
}
```

25 Min Cost Max Flow

```
int pr[N], in[N], q[N * M], used[N], d[N], pot[N];
vi g[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));
}

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];
      int 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 s, int t) {
  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;
}
```

6 Games

26 Retrograde Analysis

```
int win[N], lose[N], outDeg[N];
vi rg[N];

void retro(int n) {
  queue<int> q;
  forn (i, n)
    if (!outDeg[i])
      lose[i] = 1, q.push(i);
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int to : rg[v])
      if (lose[v]) {
        if (!win[to])
          win[to] = 1, q.push(to);
      } else {
        outDeg[to]--;
        if (!outDeg[to])
          lose[to] = 1, q.push(to);
      }
  }
}
```

7 Geometry

27 ClosestPoints (SweepLine)

```
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;
}
```

28 ConvexHull

```
using vpnt = vector<Pnt>;

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);

/*  To keep 180 angles (1)   (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;
}
```

29 GeometryBase

```
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 xx, dbl yy): x(xx), y(yy) {}

    inline Pnt operator +(const Pnt &p) const { return Pnt(x +
    p.x, y + p.y); }
    inline Pnt operator -(const Pnt &p) const { return Pnt(x -
    p.x, y - p.y); }
    inline dbl operator *(const Pnt &p) const { return x * p.x + y
    * p.y; } // ll
    inline dbl operator %(const Pnt &p) const { return x * p.y - y
    * p.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 &p) { x += p.x, y += p.y; }
    inline void operator -=(const Pnt &p) { x -= p.x, y -= p.y; }
    inline void operator *=(dbl k) { x*=k, y*=k; }

    inline bool operator ==(const Pnt &p) const { return
    abs(x-p.x)<=EPS && abs(y-p.y)<=EPS; }
    inline bool operator !=(const Pnt &p) const { return
    abs(x-p.x)>EPS || abs(y-p.y)>EPS; }
    inline bool operator <(const Pnt &p) const { return
    abs(x-p.x)<=EPS ? y<p.y-EPS : x<p.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 xx, yy;
    cin >> xx >> yy;
        x = xx, y = yy;
    }
    inline void write() const{
        cout << fixed << (double)x << " " << (double)y << '\n';
    }
    Pnt bmul(const Pnt& r) const {
    return Pnt(x*r.x - y*r.y, y*r.x + x*r.y);
  }
};

struct Line{
    dbl a, b, c;
    Line(): a(0), b(0), c(0) {}
    // normalizes
    Line(dbl aa, dbl bb, dbl cc) {
      dbl norm = sqrt(aa * aa + bb * bb);
      aa /= norm, bb /= norm, cc /= norm;
      a = aa, b = bb, c = cc;
    }

    Line(const Pnt &A, const Pnt &p){ // it normalizes (a,b),
    important in d(), normalToP()
```

```
        Pnt n = (p-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{
      cout << fixed << (double)a << " " << (double)b << " " <<
↪   (double)c << '\n';
    }
};
```

30 GeometryInterTangent

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


struct Circle {
    Pnt p;
    dbl r;
};

Pnt tangent(Pnt x, Circle y, int t = 0) {
  y.r = abs(y.r); // abs needed because internal calls y.s < 0
  if (y.r == 0) return y.p;
  dbl d = (x - y.p).len();
  Pnt a = (x - y.p) * pow(y.r / d, 2)  + y.p;
  Pnt b = ((x - y.p).getNorm() * sqrt(d * d - y.r * y.r) / d *
↪   y.r).bmul(Pnt(0, 1));
  return t == 0 ? a+b : a-b;
}

vector<pair<Pnt,Pnt>> external(const Circle &x, const Circle &y)
↪  {
  vector<pair<Pnt,Pnt>> v;
  if (x.r == y.r) {
    Pnt tmp = ((x.p-y.p).getNorm()*x.r).bmul(Pnt(0,1));
    v.pb(mp(x.p+tmp,y.p+tmp));
    v.pb(mp(x.p-tmp,y.p-tmp));
  } else {
    Pnt p = (x.p*y.r-y.p*x.r)/(y.r-x.r);
    forn(i,2) v.pb(mp(tangent(p,x,i),tangent(p,y,i)));
  }
  return v;
}
vector<pair<Pnt,Pnt>> internal(const Circle &x, const Circle &y)
↪  {
  return external({x.p,-x.r},y); }

vector<Pnt> line_line(const Line &l, const Line &m){
    dbl z = m.a * l.b - l.a * m.b;
  dbl x = m.c * l.b - l.c * m.b;
  dbl y = m.c * l.a - l.c * m.a;
    if(fabs(z) > EPS)
        return {Pnt(-x/z, y/z)};
    else if(fabs(x) > EPS || fabs(y) > EPS)
        return {}; // parallel lines
    else
        return {Pnt(0, 0), Pnt(0, 0)}; // same lines
}

vector<Pnt> 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) {
        return {c.p - l.no() * d};
    } else {
        dbl s = sqrt(fabs(sqr(c.r) - sqr(d)));
        return {c.p - l.no() * d + l.no().getRot90() * s,
            c.p - l.no() * d - l.no().getRot90() * s};
    }
}

vector<Pnt> circle_circle(const Circle &x, const Circle &y) {
  dbl d = (x.p-y.p).len(), a = x.r, b = y.r;
  if (eq(d, 0)) { assert(a != b); return {}; }
  dbl C = (a*a+d*d-b*b)/(2*a*d);
  if (abs(C) > 1+EPS) return {};
  dbl S = sqrt(max(1-C*C,(dbl)0)); Pnt tmp = (y.p-x.p)/d*x.r;
  if (eq(S, 0)) return {x.p+tmp.bmul(Pnt(C,0))};
  return {x.p+tmp.bmul(Pnt(C,S)),x.p+tmp.bmul(Pnt(C,-S))};
}

dbl circle_isect_area(const Circle &x, const Circle &y) {
  dbl d = (x.p-y.p).len(), a = x.r, b = y.r; if (a < b)
↪   swap(a,b);
  if (geq(d, a+b)) return 0;
  if (leq(d, a-b)) return PI*b*b;
  dbl ca = acos((a*a+d*d-b*b)/(2*a*d)), cb =
↪   acos((b*b+d*d-a*a)/(2*b*d));
  return (ca*a*a-0.5*a*a*sin(ca*2))+(cb*b*b-0.5*b*b*sin(cb*2));
}

// 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();
    dbl d = fabs((p - a) % (b - a));
    return d * d / (b - a).len2();
}
```

31 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 = sz(p);
    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 containsSlow(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 containsFast(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 "p" is in the triangle "abc" IFF triangle in
    CCW order
inline int isInTr(const Pnt &p, const Pnt &a, const Pnt &b, const
    Pnt &c){
    return
            gr((b - a) % (p - a), 0) &&
            gr((c - b) % (p - b), 0) &&
            gr((a - c) % (p - c), 0);
}
```

32 Halfplanes Intersection

```
namespace halfplanes {
Pnt st, v, p[N];
int n, sp, ss[N], ind[N], no[N], cnt[N], k = 0, a[N], b[N];
dbl ang[N];

Pnt Norm(int j) { return (p[a[j]] - p[b[j]]).getRot90(); }

void AddPlane( int i, int j ){
  a[k] = i, b[k] = j, ind[k] = k;
  ang[k] = Norm(k).angle();
  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 (g, k) {
    BUILD(a1, b1, c1, ind[g]);
    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;
}

void solve() {
  cin >> n;
  forn (i, n)
    cin >> 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;
    }
}
}
```

8 Graphs

33 2-SAT

```
// VAR - 2 * vars
int cntVar = 0, val[VAR], usedSat[VAR], comp[VAR];
vi topsortSat;

vi g[VAR], rg[VAR];

inline int newVar() {
  cntVar++;
  return (cntVar - 1) * 2;
}
```

```
inline int Not(int v) { return v ^ 1; }

inline void Implies(int v1, int v2) { g[v1].pb(v2),
↪  rg[v2].pb(v1); }

inline void Or(int v1, int v2) { Implies(Not(v1), v2),
↪  Implies(Not(v2), v1); }

inline void Nand(int v1, int v2) { Or(Not(v1), Not(v2)); }

inline void setTrue(int v) { Implies(Not(v), v); }

void dfs1(int v) {
  usedSat[v] = 1;
  for (int to : g[v])
    if (!usedSat[to]) dfs1(to);
  topsortSat.pb(v);
}

void dfs2(int v, int c) {
  comp[v] = c;
  for (int to : rg[v])
    if (!comp[to]) dfs2(to, c);
}

int getVal(int v) { return val[v]; }

// cntVar
bool solveSat() {
  forn(i, 2 * cntVar) usedSat[i] = 0;
  forn(i, 2 * cntVar)
    if (!usedSat[i]) dfs1(i);
  reverse(all(topsortSat));
  int c = 0;
  for (int v : topsortSat)
    if (!comp[v]) dfs2(v, ++c);
  forn(i, cntVar) {
    if (comp[2 * i] == comp[2 * i + 1]) return false;
    if (comp[2 * i] < comp[2 * i + 1]) val[2 * i + 1] = 1;
    else val[2 * i] = 1;
  }
  return true;
}
```

34 Bridges

```
int up[N], tIn[N], timer;
vector<vi> comps;
vi st;

struct Edge {
  int to, id;
  Edge(int _to, int _id) : to(_to), id(_id) {}
};

vector<Edge> g[N];

void newComp(int size = 0) {
  comps.emplace_back(); // new empty
  while (sz(st) > size) {
    comps.back().pb(st.back());
    st.pop_back();
  }
}

void findBridges(int v, int parentEdge = -1) {
  if (up[v]) // visited
    return;
  up[v] = tIn[v] = ++timer;
  st.pb(v);
  for (Edge e : g[v]) {
    if (e.id == parentEdge)
      continue;
    int u = e.to;
    if (!tIn[u]) {
      int size = sz(st);
      findBridges(u, e.id);
      if (up[u] > tIn[v])
        newComp(size);
    }
    up[v] = min(up[v], up[u]);
  }
}

// after find_bridges newComp() for root
void run(int n) {
  forn (i, n)
    if (!up[i]) {
      findBridges(i);
      newComp();
    }
}
```

35 Cactus

```
int used[N];

struct Edge {
   ll l;
   Edge() {}
   Edge(int _l): l(_l) {}
};

vector<pair<int, Edge>> g[N], rev[N], path;
pair<int, Edge> pr[N];

void dfsInit(int v, int p, Edge prE) {
  used[v] = 1;
  pr[v] = mp(p, prE);
  for (auto e : g[v]) {
    int u = e.fst;
    if (u == p)
      continue;
    if (used[u] == 1)
      rev[u].pb(mp(v, e.snd));
    else if (used[u] != 2)
      dfsInit(u, v, e.snd);
  }
  used[v] = 2;
}

void calc(int v) {
  used[v] = 1;
  for (auto e : rev[v]) {
     path.clear();
     int u = e.fst;
      while (u != v) {
         calc(u);
         path.pb(mp(u, pr[u].snd));
         u = pr[u].fst;
      }
      // Calculate answer for cycle -- path and vertex v
   }
   for (auto e : g[v])
     if (!used[e.fst] && e.fst != pr[v].fst) {
       calc(e.fst);
       // Update answer for tree edges
     }
}
```

36 Cut Points

```
bool used[M];
int tIn[N], timer, isCut[N], color[M], compCnt;
vi st;

struct Edge {
  int to, id;
  Edge(int _to, int _id) : to(_to), id(_id) {}
};

vector<Edge> g[N];

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])
        used[id] = 1, st.push_back(id);
      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)
    isCut[v] = 1; // v is cut vertex
  return up;
}
```

37 Dominator Tree

```
// clean: forn(i, n+1)!!!
vi adj[N], ans[N]; // input edges, edges of dominator tree
vi radj[N], child[N], sdomChild[N];
int label[N], rlabel[N], sdom[N], dom[N], co = 0;
int par[N], bes[N];
int get(int x) { // DSU with path compression
  // get vertex with smallest sdom on path to root
  if (par[x] != x) {
    int t = get(par[x]); par[x] = par[par[x]];
    if (sdom[t] < sdom[bes[x]]) bes[x] = t;
  }
  return bes[x];
}
void dfs(int x) { // create DFS tree
  label[x] = ++co; rlabel[co] = x;
  sdom[co] = par[co] = bes[co] = co;
  for(auto y : adj[x]) {
    if (!label[y]) {
      dfs(y); child[label[x]].pb(label[y]); }
    radj[label[y]].pb(label[x]);
  }
}
void init(int root) {
  dfs(root);
  for(int i = co; i >= 1; i--) {
    for(auto j : radj[i]) sdom[i] = min(sdom[i], sdom[get(j)]);
    if (i > 1) sdomChild[sdom[i]].pb(i);
    for(auto j : sdomChild[i]) {
      int k = get(j);
      if (sdom[j] == sdom[k]) dom[j] = sdom[j];
      else dom[j] = k;
    }
    for(auto j : child[i]) par[j] = i;
  }
  forab(i,2,co+1) {
    if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
    ans[rlabel[dom[i]]].pb(rlabel[i]);
  }
}
```

38 Eulerian Cycle

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

vector<Edge> edges;
vi g[N], res, ptr;
// don't forget to clear 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); // edges
    }
  }
  res.pb(v); // res contains vertices
}
```

39 Euler Tour Tree

```
mt19937 rng(239);

struct Edge {
   int v, u;
   Edge(int _v, int _u): v(_v), u(_u) {}
};

struct Node {
  Node *l, *r, *p;
  Edge e;
  int y, size;
  Node(Edge _e): l(nullptr), r(nullptr), p(this), e(_e), y(rng()),
↪  size(1) {}
};

inline int getSize(Node* root) { return root ? root->size : 0; }

inline void recalc(Node* root) { root->size = getSize(root->l) +
↪  getSize(root->r) + 1; }

set<pair<int, Node*>> edges[N];

Node* merge(Node *a, Node *b) {
  if (!a) return b;
  if (!b) return a;
  if (a->y < b->y) {
    a->r = merge(a->r, b);
    if (a->r) a->r->p = a;
    recalc(a);
    return a;
  }
  b->l = merge(a, b->l);
  if (b->l) b->l->p = b;
  recalc(b);
  return b;
}

void split(Node *root, Node *&a, Node *&b, int size) {
  if (!root) {
    a = b = nullptr;
    return;
  }
  int lSize = getSize(root->l);
  if (lSize >= size) {
    split(root->l, a, root->l, size);
    if (root->l) root->l->p = root;
    b = root, b->p = b;
  } else {
    split(root->r, root->r, b, size - lSize - 1);
    if (root->r) root->r->p = root;
    a = root, a->p = a;
    a->p = a;
  }
  recalc(root);
}

inline Node* rotate(Node* root, int k) {
  if (k == 0) return root;
  Node *l, *r;
  split(root, l, r, k);
  return merge(r, l);
}

inline pair<Node*, int> goUp(Node* root) {
  int pos = getSize(root->l);
  while (root->p != root)
    pos += (root->p->r == root ? getSize(root->p->l) + 1 : 0),
↪    root = root->p;
  return mp(root, pos);
}
```

```
inline Node* deleteFirst(Node* root) {
  Node* a;
  split(root, a, root, 1);
  edges[a->e.v].erase(mp(a->e.u, a));
  return root;
}

inline Node* getNode(int v, int u) {
  return edges[v].lower_bound(mp(u, nullptr))->snd;
}

inline void cut(int v, int u) {
  auto pV = goUp(getNode(v, u));
  auto pU = goUp(getNode(u, v));
  int l = min(pV.snd, pU.snd), r = max(pV.snd, pU.snd);
  Node *a, *b, *c;
  split(pV.fst, a, b, l);
  split(b, b, c, r - l);
  deleteFirst(b);
  merge(a, deleteFirst(c));
}

inline pair<Node*, int> getRoot(int v) {
  return !sz(edges[v]) ? mp(nullptr, 0) :
↪  goUp(edges[v].begin()->snd);
}

inline Node* makeRoot(int v) {
  auto root = getRoot(v);
  return rotate(root.fst, root.snd);
}

inline Node* makeEdge(int v, int u) {
  Node* e = new Node(Edge(v, u));
  edges[v].insert(mp(u, e));
  return e;
}

inline void link(int v, int u) {
  Node *vN = makeRoot(v), *uN = makeRoot(u);
  merge(merge(merge(vN, makeEdge(v, u)), uN), makeEdge(u, v));
}
```

40 Hamilton Cycle

```
// DP in O(n*2^n) for Ham cycle
vi g[MASK];
int adj[MASK], dp[1 << MASK];

vi hamiltonCycle(int n) {
  fill(dp, dp + (1 << n), 0);
  forn (v, n) {
    adj[v] = 0;
    for (int to : g[v])
      adj[v] |= (1 << to);
  }
  dp[1] = 1;
  forn (mask, (1 << n))
    forn(v, n)
      if (mask & (1 << v) && dp[mask ^ (1 << v)] & adj[v])
        dp[mask] |= (1 << v);
  vi ans;
  int mask = (1 << n) - 1, v;
  if (dp[mask] & adj[0]) {
    forab (i, 1, n)
      if ((1 << i) & (mask & adj[0]))
        v = i;
    ans.pb(v);
    mask ^= (1 << v);
    while(v) {
      forn(i, n)
        if ((dp[mask] & (1 << i)) && (adj[i] & (1 << v))) {
          v = i;
          break;
        }
      mask ^= (1 << v);
      ans.pb(v);
    }
  }
  return ans;
}
```

41 Karp with cycle

```
int d[N][N], p[N][N];
vi g[N], ans;

struct Edge {
  int a, b, w;
  Edge(int _a, int _b, int _w): a(_a), b(_b), w(_w) {}
};

vector<Edge> edges;

void fordBellman(int s, int n) {
  forn (i, n + 1)
    forn (j, n + 1)
      d[i][j] = INF;
  d[0][s] = 0;
  forab (i, 1, n + 1)
    for (auto &e : edges)
      if (d[i - 1][e.a] < INF && d[i][e.b] > d[i - 1][e.a] + e.w)
↪
        d[i][e.b] = d[i - 1][e.a] + e.w, p[i][e.b] = e.a;
}

ld karp(int n) {
  int s = n++;
  forn (i, n - 1)
    g[s].pb(sz(edges)), edges.pb(Edge(s, i, 0));
  fordBellman(s, n);
  ld ansValue = INF;
  int curV = -1, dist = -1;
  forn (v, n - 1)
    if (d[n][v] != INF) {
      ld curAns = -INF;
      int curPos = -1;
      forn(k, n)
        if (curAns <= (d[n][v] - d[k][v]) * (ld) (1) / (n - k))
          curAns = (d[n][v] - d[k][v]) * (ld) (1) / (n - k),
↪  curPos = k;
      if (ansValue > curAns)
        ansValue = curAns, dist = curPos, curV = v;
    }
  if (curV == -1) return ansValue;
  for (int iter = n; iter != dist; iter--)
    ans.pb(curV), curV = p[iter][curV];
  reverse(all(ans));
  return ansValue;
}
```

42 Kuhn's algorithm

```
// sz(LEFT) = n, sz(RIGHT) = m
// numbered consequently
int n, m, paired[2 * N], used[2 * N];
vi g[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;
    fill(used, used + n + m, 0);
    forn(i, n)
      if (!used[i] && paired[i] == -1 && dfs(i))
        ans++, run = 1;
  }
  return ans;
```

```
}

// Start from unpaired vertex in Left part, go from Left anywhere,
↪  from Right only to pair
// Max Independent -- A+, B-
// Min Cover       -- A-, B+

vi 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]);
}

// Kuhn first!
void findCoverIndependent() {
  fill(used, used + n + m, 0);
  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);
}
```

43 Blossom algorithm

```
mt19937 rng(239017);
template<int SZ> struct UnweightedMatch {
  int match[SZ], N;
  vi adj[SZ];

  void ae(int u, int v) {
    adj[u].pb(v);
    adj[v].pb(u);
  }

  queue<int> q;
  int par[SZ], vis[SZ], orig[SZ], aux[SZ];

  void augment(int u, int v) { // toggle edges on u-v path
    while (1) { // one more matched pair
      int pv = par[v], nv = match[pv];
      match[v] = pv; match[pv] = v;
      v = nv; if (u == pv) return;
    }
  }

  int lca(int u, int v) { // find LCA of supernodes in O(dist)
    static int t = 0;
    for (++t;;swap(u,v)) {
      if (!u) continue;
      if (aux[u] == t) return u; // found LCA
      aux[u] = t; u = orig[par[match[u]]];
    }
  }

  void blossom(int u, int v, int a) { // go other way
    for (; orig[u] != a; u = par[v]) { // around cycle
      par[u] = v; v = match[u]; // treat u as if vis[u] = 1
      if (vis[v] == 1) vis[v] = 0, q.push(v);
      orig[u] = orig[v] = a; // merge into supernode
    }
  }

  bool bfs(int u) { // u is initially unmatched
    forn (i,N+1)
      par[i] = 0, vis[i] = -1, orig[i] = i;
    q = queue<int>();
    vis[u] = 0;
    q.push(u);
    while (sz(q)) { // each node is pushed to q at most once
      int v = q.front(); q.pop(); // 0 -> unmatched vertex
      for (int x : adj[v]) {
        if (vis[x] == -1) { // neither of x, match[x] visited
          vis[x] = 1; par[x] = v;
          if (!match[x])
            return augment(u,x),1;
          vis[match[x]] = 0;
          q.push(match[x]);
        } else if (vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v],orig[x]); // odd cycle
          blossom(x,v,a), blossom(v,x,a);
        } // contract O(n) times
      }
    }
    return 0;
  }

  int calc(int _N) { // rand matching -> constant improvement
    N = _N;
    forn (i, N+1)
      match[i] = aux[i] = 0;
    int ans = 0; vi V(N); iota(all(V),1); shuffle(all(V),rng); //
↪   find rand matching
    for (int x : V) {
      if (!match[x]) {
        for (int y : adj[x]) {
          if (!match[y]) {
            match[x] = y, match[y] = x; ++ans;
            break;
          }
        }
      }
    }
    forab (i, 1, N+1)
      if (!match[i] && bfs(i))
        ++ans;
    return ans;
  }
};
```

44 LCA

```
int tin[N], tout[N], up[N][LOG], curTime = 0;;
vi g[N];

void dfs(int v, int p) {
  up[v][0] = p;
  forn (i, LOG - 1)
    up[v][i + 1] = up[up[v][i]][i];
  tin[v] = curTime++;
  for (int u : g[v])
    if (u != p)
      dfs(u, v);
  tout[v] = curTime++;
}

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

int lca(int v, int u) {
  if (isUpper(u, v)) return u;
  fornr (i, LOG)
    if (!isUpper(up[u][i], v))
      u = up[u][i];
  return up[u][0];
}

void init() {
  dfs(0, 0);
}
```

45 LCA offline (Tarjan)

```
vi g[N], q[N];
int pr[N], ancestor[N], used[N];

int get(int v) {
  return v == pr[v] ? v : pr[v] = get(pr[v]);
}

void unite(int v, int u, int anc) {
```

```
  v = get(v), u = get(u);
  pr[u] = v, ancestor[v] = anc;
}

void dfs(int v) {
  used[v] = 1;
  for (int u : g[v])
    if (!used[u])
      dfs(u), unite(v, u, v);
  for (int u : q[v])
    if (used[u])
      ancestor[get(u)]; // handle answer somehow
}

void init(int n) {
  forn (i, n) pr[i] = i, ancestor[i] = i;
  dfs(0);
}
```

46 2 Chinese

```
struct Edge {
    int fr, to, w, id;
    bool operator<(const Edge& o) const { return w < o.w; }
};

// find oriented mst (tree)
// there are no edge --> root (root is 0)
// 0 .. n - 1, weights and vertices will be changed, but ids are
↪  ok
vector<Edge> work(const vector<vector<Edge>>& graph) {
    int n = sz(graph);
    vi color(n), used(n, -1);
    forn (i, n)
        color[i] = i;
    vector<Edge> e(n);
    forn (i, n) {
        if (graph[i].empty())
            e[i] = {-1, -1, -1, -1};
        else
            e[i] = *min_element(graph[i].begin(),
↪   graph[i].end());

    }
    vector<vi> cycles;
    used[0] = -2;
    forn (s, n) {
        if (used[s] != -1)
            continue;
        int x = s;
        while (used[x] == -1) {
            used[x] = s;
            x = e[x].fr;
        }
        if (used[x] != s)
            continue;
        vi cycle = {x};
        for (int y = e[x].fr; y != x; y = e[y].fr)
            cycle.push_back(y), color[y] = x;
        cycles.push_back(cycle);
    }
    if (cycles.empty())
        return e;
    vector<vector<Edge>> next_graph(n);
    forn (s, n) {
        for (const Edge& edge : graph[s]) {
            if (color[edge.fr] != color[s])
                next_graph[color[s]].push_back({
                    color[edge.fr], color[s], edge.w - e[s].w,
↪   edge.id
                });
        }
    }
    vector<Edge> tree = work(next_graph);
    for (const auto& cycle : cycles) {
        int cl = color[cycle[0]];
        Edge next_out = tree[cl], out{};
        int from = -1;
        for (int v : cycle) {
            tree[v] = e[v];
            for (const Edge& edge : graph[v])
                if (edge.id == next_out.id)
                    from = v, out = edge;
        }
        tree[from] = out;
    }
    return tree;
}
```

47 Matroid Intersection

```
struct Gmat { // graphic matroid
  int V = 0; vector<pii> ed; vi par;
  Gmat(vector<pii> _ed):ed(_ed) {
    map<int,int> m;
    for(auto &t : ed) m[t.fst] = m[t.snd] = 0;
    for(auto &t : m) t.snd = V++;
    for(auto &t : ed) t.fst = m[t.fst], t.snd = m[t.snd];
  }
  int p(int v) {
    return par[v] == v ? v : par[v] = p(par[v]);
  }
  bool unite(int v, int u) {
    v = p(v), u = p(u);
    if (v != u) { par[v] = u; return true; }
    return false;
  }
  void clear() {
    par.resize(V);
    forn(i,V) par[i] = i;
  }
  void ins(int i) { assert(unite(ed[i].fst,ed[i].snd)); }
  bool indep(int i) { return p(ed[i].fst) != p(ed[i].snd); }
};
struct Cmat { // colorful matroid
  int C = 0; vi col; vi used;
  Cmat(vi _col):col(_col) {for(auto t : col) C = max(C, t+1);}
  void clear() { used.assign(C,0); }
  void ins(int i) { used[col[i]] = 1; }
  bool indep(int i) { return !used[col[i]]; }
};
template<class M1, class M2> struct MatroidIsect {
  int n; vi iset; M1 m1; M2 m2;
  bool augment() {
    vi pre(n+1,-1); queue<int> q({n});
    while (sz(q)) {
      int x = q.front(); q.pop();
      if (iset[x]) {
        m1.clear(); forn(i,n) if (iset[i] && i != x) m1.ins(i);
        forn(i,n) if (!iset[i] && pre[i] == -1 && m1.indep(i))
          pre[i] = x, q.push(i);
      } else {
        auto backE = [&]() { // back edge
          m2.clear();
          forn(c,2)forn(i,n)
↪   if((x==i||iset[i])&&(pre[i]==-1)==c){
            if (!m2.indep(i))return c?pre[i]=x,q.push(i),i:-1;
            m2.ins(i); }
          return n;
        };
        for (int y; (y = backE()) != -1;) if (y == n) {
          for(; x != n; x = pre[x]) iset[x] = !iset[x];
          return 1; }
      }
    }
    return 0;
  }
  MatroidIsect(int _n, M1 _m1, M2 _m2):n(_n), m1(_m1), m2(_m2) {
    iset.assign(n+1,0); iset[n] = 1;
    m1.clear(); m2.clear(); // greedily add to basis
    fornr(i,n) if (m1.indep(i) && m2.indep(i))
      iset[i] = 1, m1.ins(i), m2.ins(i);
    while (augment());
  }
};
```

9 Math

48 Berlekamp

```
using T = int;
using poly = vector<int>;

void remz(poly& p) { while (sz(p)&&p.back()==T(0)) p.pop_back();
↪ }

poly operator*(const poly& l, const poly& r) {
  if (!min(sz(l),sz(r))) return {};
  poly x(sz(l)+sz(r)-1);
  forn(i,sz(l)) forn(j,sz(r)) x[i+j] += l[i]*r[j];
  return x;
}
pair<poly,poly> quoRem(poly a, poly b) {
  remz(a); remz(b); assert(sz(b));
  T lst = b.back(), B = T(1)/lst; for(auto &t : a) t *= B;
  for(auto &t : b) t *= B;
  poly q(max(sz(a)-sz(b)+1,0));
  for (int dif; (dif=sz(a)-sz(b)) >= 0; remz(a)) {
    q[dif] = a.back(); forn(i,sz(b)) a[i+dif] -= q[dif]*b[i]; }
  for(auto &t : a) t *= lst;
  return {q,a}; // quotient, remainder
}
poly operator%(const poly& a, const poly& b) {
  return quoRem(a,b).snd; }

struct LinRec {
  poly s, C, rC;
  void BM() { // find smallest C such that C[0]=1 and
    // for all i >= sz(C)-1, sum_{j=0}^{sz(C)-1}C[j]*s[i-j]=0
    // If we treat C and s as polynomials in D, then
    // for all i >= sz(C)-1, [D^i]C*s=0
    int x = 0; T b = 1;
    poly B; B = C = {1}; // B is fail vector
    /// for all sz(B)+x-1 <= j < i, [D^j](B<<x)*s=0
    /// but [D^i](B<<x)*s=b
    /// invariant: sz(B)+x = M
    forn(i,sz(s)) { // update C after adding a term of s
      ++x; int L = sz(C), M = i+3-L;
      T d = 0; forn(j,L) d += C[j]*s[i-j]; // [D^i]C*s
      if (d == 0) continue; // [D^i]C*s=0
      poly _C = C; T coef = d/b; /// d-coef*b = 0
      /// set C := C-coef*(B<<x) to satisfy condition
      C.resize(max(L,M)); forn(j,sz(B)) C[j+x] -= coef*B[j];
      if (L < M) B = _C, b = d, x = 0;
    } /// replace B<<x with C<<0
  }
  void init(const poly& _s) {
    s = _s; BM();
    rC = C; reverse(all(rC)); // poly for getPow
    C.erase(begin(C)); for(auto &t : C) t *= -1;
  } // now s[i]=sum_{j=0}^{sz(C)-1}C[j]*s[i-j-1]
  poly getPow(ll p) { // get x^p mod rC
    if (p == 0) return {1};
    poly r = getPow(p/2); r = (r*r)%rC;
    return p&1?(r*poly{0,1})%rC:r;
  }
  T dot(poly v) { // dot product with seq
    T ans = 0; forn(i,sz(v)) ans += v[i]*s[i];
    return ans; } // get p-th term of rec
  T eval(ll p) { assert(p >= 0); return dot(getPow(p)); }
};
```

49 CRT (KTO)

```
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;
}
```

50 Discrete Logarithm

```
// Returns x: a^x = b (mod mod) or -1, if no such x exists
int discreteLogarithm(int a, int b, int mod) {
  int sq = (int) 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 = mul(cur, b, mod);
    auto it = lower_bound(all(powers), mp(cur, 0));
    if (it != powers.end() && it->fst == cur)
      return it->snd * sq - i;
  }
  return -1;
}
```

51 Discrete Root

```
// 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 = discreteLogarithm(power(g, k, mod), a, mod);
  return power(g, y, mod);
}
```

52 Eratosthenes

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

vi eratosthenesLinear(int n) {
  vi minDiv(n + 1, 0), primes;
  minDiv[1] = 1;
  forab (i, 2, n + 1) {
    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;
}
```

53 Factorial

```
// Returns pair (rem, power), where rem = n! % mod,
// power = k: mod^k | n!, mod is prime, O(mod log mod)
pii fact(int n, int mod) {
  int rem = 1, power = 0, nCopy = n;
  while (nCopy) nCopy /= mod, power += nCopy;
  while (n > 1) {
    rem = (rem * ((n / mod) % 2 ? -1 : 1) + mod) % mod;
    for (int i = 2; i <= n % mod; i++)
      rem = mul(rem, i, mod);
    n /= mod;
  }
  return mp(rem % mod, power);
}
```

54 Gauss

```
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;
    forn (i, m + 1) swap(a[row][i], a[best][i]);
    forn (i, n)
      if (i != row) {
        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;
  forn (i, m)
    if (par[i] != -1) ans[i] = a[par[i]][m] / a[par[i]][i];
    else single = 0;
  forn (i, n) {
    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;
}
```

55 Gauss binary

```
const int MAX = 1024;

int gaussBinary(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]);
    forn (i, n)
      if (i != row && a[i][col])
        a[i] ^= a[row];
    row++;
  }
  vi ans(m, 0);
  forn (i, m)
    if (par[i] != -1)
      ans[i] = a[par[i]][n] / a[par[i]][i];
  bool ok = 1;
  forn (i, n) {
    int cur = 0;
    forn (j, m) cur ^= (ans[j] & a[i][j]);
    if (cur != a[i][n]) ok = 0;
  }
  return ok;
}
```

56 Gcd

```
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;
  }
  int g = gcd(b, a % b, x, y), 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;
}
```

57 Gray

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

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

58 Miller-Rabin Test

```
bool isPrimeMillerRabin(ull n) { // not ll!
  if (n < 2 || n % 6 % 4 != 1)
    return n - 2 < 2;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
  ull s = __builtin_ctzll(n - 1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = power(a, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = mul(p, p, n);
    if (p != n - 1 && i != s) return 0;
  }
  return 1;
}
```

59 Phi

```
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);
}
```

60 Pollard

```
ull pollard(ull n) { // return some nontrivial factor of n
  auto f = [n](ull x) { return mul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) { /// speedup: don't take
↪ gcd every it
    if (x == y) x = ++i, y = f(x);
```

```
    if ((q = mul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}

void factorize(ull n, map<ull,int>& cnt) {
  if (n == 1) return;
  if (isPrimeMillerRabin(n)) {
    ++cnt[n];
    return;
  }
  ull u = pollard(n);
  factorize(u, cnt), factorize(n / u, cnt);
}
```

61 Power And Mul

```
template <typename T>
inline T add(T a, T b, T mod) {
  a += b;
  return a >= mod ? a - mod : a;
}

template <typename T>
inline T sub(T a, T b, T mod) {
  a -= b;
  return a < 0 ? a + mod : a;
}

template <typename T>
T mul(T a, T b, T mod) {
  return T((a * 1ll * b) % mod);
}

template <>
ll mul<ll>(ll a, ll b, ll mod) {
  ll q = ll((ld) a * b / mod);
  ll r = a * b - mod * q;
  while (r < 0) r += mod;
  while (r >= mod) r -= mod;
  return r;
}

template <typename T>
T power(T a, T n, T mod) {
  if (!n) return 1;
  T b = power(a, n / 2, mod);
  b = mul(b, b, mod);
  return n & 1 ? mul<T>(a, b, mod) : b;
}

int powerFast(int a, int n, int mod) {
  int res = 1;
  while (n) {
    if (n & 1)
      res = mul(res, a, mod);
    a = mul(a, a, mod);
    n /= 2;
  }
  return res;
}
```

62 Primitive Root

```
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;
}
```

63 Simpson

```
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;
}
```

64 Simplex

```
/**
   * maximize c^T x subject to Ax <= b, x >= 0.
   * -inf / inf / max c^T x
  * define variables such that x = 0 is viable.
  * 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: O(NM \cdot \#pivots), where a pivot may be e.g. an edge
↪  relaxation. O(2^N) in the general case.
*/

using vi = vector<int>;
using dbl = double;
using vd = vector<dbl>;
using vvd = vector<vd>;
const dbl eps = 1e-8, inf = 1/.0;

#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; // # contraints, # variables
  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)) {
    forn(i,m) forn(j,n) D[i][j] = A[i][j];
    forn(i,m) { // B[i]: add basic variable for each constraint,
      B[i] = n+i, D[i][n] = -1, D[i][n+1] = b[i];
      // convert ineqs to eqs
    } // D[i][n]: artificial variable for testing feasibility
    forn(j,n) {
      N[j] = j; // non-basic variables, all zero
      D[m][j] = -c[j]; // minimize -c^T x
    }
    N[n] = -1; D[m+1][n] = 1;
  }
  void pivot(int r, int s) { // r = row, c = column
    dbl *a = D[r].data(), inv = 1/a[s];
    forn(i,m+2) if (i != r && abs(D[i][s]) > eps) {
      dbl *b = D[i].data(), binv = b[s]*inv;
      forn(j,n+2) b[j] -= a[j]*binv;
      // make column corresponding to s all 0s
      b[s] = a[s]*binv; // swap N[s] with B[r]
    }
    // equation for r scaled so x_r coefficient equals 1
    forn(j,n+2) if (j != s) D[r][j] *= inv;
    forn(i,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv; swap(B[r], N[s]); // swap basic w/ non-basic
  }
  bool simplex(int phase) {
    int x = m+phase-1;
    while (1) {
      int s = -1; forn(j,n+1) if (N[j] != -phase) ltj(D[x]);
      // find most negative col for nonbasic (nb) variable
      if (D[x][s] >= -eps) return 1;
      // can't get better sol by increasing nb variable
      int r = -1;
      forn(i,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;
      // find smallest positive ratio
```

```
    } // -> max increase in nonbasic variable
    if (r == -1) return 0; // unbounded
    pivot(r,s);
  }
}
dbl solve(vd& x) {
  int r = 0; forab(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
  if (D[r][n+1] < -eps) { // run simplex, find feasible x!=0
    pivot(r, n); // N[n] = -1 is artificial variable
    // initially set to smth large
    if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
    // D[m+1][n+1] is max possible value of the negation of
    // artificial variable, optimal value should be zero
    // if exists feasible solution
    forn(i,m) if (B[i] == -1) { // ?
      int s = 0; forab(j,1,n+1) ltj(D[i]);
      pivot(i,s);
    }
  }
  bool ok = simplex(1); x = vd(n);
  forn(i,m) if (B[i] < n) x[B[i]] = D[i][n+1];
  return ok ? D[m][n+1] : inf;
}
};
```

65 Euclidean Burunduk-1

```
/**
 * Sergey Kopeliovich (burunduk30@gmail.com)
 */

#include <iostream>

using namespace std;

// finds x:
//    a+k*x mod m --> min, 0 <= x <= r (0 <= a, k < m, 0 <= r)
//     +k costs pk, -m costs pm
//    return r-x
int go(int a, int k, int m, int pk, int pm, int r) {
  if (!k) return r;
  if (a >= k) { // make a: 0 <= a < k
    int add = (m - a + k - 1) / k;
    if ((int64_t)add * pk + pm > r) return r;
    a += (int64_t)add * k - m, r -= add * pk + pm;
  }
  int m1 = m % k, pm1 = (m / k) * pk + pm;
  if (!m1) return r;
  int k1 = k % m1, pk1 = (k / m1) * pm1 + pk;
  if (pm1 * (a / m1) > r) return r % pm1;
  return go(a % m1, k1, m1, pk1, pm1, r - (a / m1) * pm1);
}

// finds x: a+k*x mod m --> min, 0 <= a, k < m, 0 <= r
int go(int a, int k, int m, int r) {
  return r - go(a, k, m, 1, 0, r);
}

int main() {
  ios_base::sync_with_stdio(false), cin.tie(0);

  int a, k, m, r;
  while (cin >> a >> k >> m >> r) {
    int x = go(a, k, m, r);
    cout << ((int64_t)x * k + a) % m << ' ' << x << '\n';
  }
}
```

66 Euclidean Burunduk-2

```
/**
 * Sergey Kopeliovich (burunduk30@gmail.com)
 */

#include <iostream>

using namespace std;

// finds min x:
//    a+k*x mod m \in [l..r]
//     +k costs pk, -m costs pm
//    l <= r < a, first tries -m then +k
int go(int a, int k, int m, int pk, int pm, int l, int r) {
  int ans = 0, steps;
  while (1) {
    steps = (a - r + m - 1) / m;
    ans += steps * pm, a -= steps * m;
    if (l <= a) return ans;
    if (!k) return -1;
    steps = (l - a + k - 1) / k;
    ans += steps * pk, a += steps * k;
    if (a <= r) return ans;
    int m1 = m % k, pm1 = (m / k) * pk + pm;
    if (!m1) return -1;
    int k1 = k % m1, pk1 = (k / m1) * pm1 + pk;
    k = k1, m = m1, pk = pk1, pm = pm1; // recursion =)
  }
}

int go(int a, int k, int m, int l, int r) {
  if (a < r)
    a += ((r - a) / m + 1) * m;
  return go(a, k, m, 1, 0, l, r);
}

int main() {
  ios_base::sync_with_stdio(false), cin.tie(0);

  int a, k, m, l, r;
  while (cin >> a >> k >> m >> l >> r)
    cout << go(a, k, m, l, r) << '\n';
}
```

10 Strings

67 Aho-Corasick

```
struct Node {
  int next[ALPHA], term; //
  int go[ALPHA], suf, p, pCh; //
  Node(): term(0), suf(-1), p(-1) {
    fill(next, next + ALPHA, -1);
    fill(go, go + ALPHA, -1);
  }
};

Node g[N];
int last;

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

int go(int v, int c);

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

68 Prefix-function

```
vi prefix(const string &s) {
  int n = sz(s);
  vi pr(n);
  forab (i, 1, n + 1) {
    int j = pr[i - 1];
    while (j > 0 && s[i] != s[j]) j = pr[j - 1];
    if (s[i] == s[j]) j++;
    pr[i] = j;
  }
  return pr;
}
```

69 Z-function

```
vi z(const string& s) {
   int n = sz(s);
   vi z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

70 Hashes

```
#include "../math/PowerAndMul.cpp"
const int P = 239017, MOD_X = 1e9 + 7, MOD_Y = 1e9 + 9;

// using H = unsigned long long;
struct H {
  int x, y;
  H() = default;
  H(int _x): x(_x), y(_x) {}
  H(int _x, int _y): x(_x), y(_y) {}
  inline H operator+(const H& h) const { return H(add(x, h.x,
↪  MOD_X), add(y, h.y, MOD_Y)); }
  inline H operator-(const H& h) const { return H(sub(x, h.x,
↪  MOD_X), sub(y, h.y, MOD_Y)); }
  inline H operator*(const H& h) const { return H(mul(x, h.x,
↪  MOD_X), mul(y, h.y, MOD_Y)); }
  inline bool operator==(const H& h) const { return x == h.x && y
↪  == h.y; }
};

H p[N], h[N];

inline H get(int l, int r) { return h[r] - h[l] * p[r - l]; }

void init(const string& s) {
  int n = sz(s);
  p[0] = 1;
  forn (i, n)
    h[i + 1] = h[i] * P + s[i], p[i + 1] = p[i] * P;
}
```

71 Manaker

```
void manaker(const string& s, int *z0, int *z1) {
  int n = sz(s);
  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;
    }
  }
}
```

72 Palindromic Tree

```
struct Vertex {
  int suf, len, next[ALPHA];
  Vertex() { fill(next, next + ALPHA, 0); }
};

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

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

void init() {
  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;
}
```

73 Suffix Array (+stable)

```
int sLen, num[N + 1], p[N], col[N], inv[N], lcp[N];
char s[N + 1];

inline int add(int a, int b) {
  a += b;
  return a >= sLen ? a - sLen : a;
}

inline int sub(int a, int b) {
  a -= b;
  return a < 0 ? a + sLen : a;
}

void buildArray(int n) {
  sLen = n;
  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)
      inv[num[col[sub(p[i], k)]]++] = sub(p[i], k);
    int cc = 0;
    forn (i, n) {
      bool flag = col[inv[i]] != col[inv[i - 1]];
      flag |= col[add(inv[i], k)] != col[add(inv[i - 1], k)];
      if (i && flag) cc++;
      num[inv[i]] = cc;
    }
    forn (i, n) p[i] = inv[i], col[i] = num[i];
  }

  memset(num, 0, sizeof(num));
  forn (i, n) num[col[i] + 1]++;
  forn (i, ma) num[i + 1] += num[i];
  forn (i, n) inv[num[col[i]]++] = i;
  forn (i, n) p[i] = inv[i];
  forn (i, n) inv[p[i]] = i;
```

```
}

void buildLCP(int n) {
  int len = 0;
  forn (ind, n){
    int i = inv[ind];
    len = max(0, len - 1);
    if (i != n - 1)
      while (len < n && s[add(p[i], len)] == s[add(p[i + 1],
↪   len)])
        len++;
    lcp[i] = len;
    if (i != n - 1 && p[i + 1] == n - 1) len = 0;
  }
}
```

74 Suffix Automaton

```
struct Vx {
    int len, suf;
    int next[ALPHA];
    Vx() {}
    Vx(int l, int s): len(l), suf(s) {}
};

struct SA {
    static const int V = 2 * LEN;
    int last, vcnt;
    Vx v[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 (p && !v[p].next[c]) // added p &&
            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;
    }
};
```

11 C++ Tricks

75 Fast allocation

```
const int MEM = 100 << 20;
static char buf[MEM];
inline void* operator new(size_t n) {
  static size_t i = sizeof buf;
  assert(n < i);
  return (void*) &buf[i -= n];
}
inline void operator delete(void*) {}
inline void* operator new[](size_t) { assert(0); }
inline void operator delete[](void*) { assert(0); }
```

76 Hash of pair

```
struct PairHasher {
  size_t operator()(const pair<int, int>& p) const { return p.fst
↪   * 239017 + p.snd; }
};
```

77 Ordered Set

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

using namespace __gnu_pbds;

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

void example() {
  ordered_set<int> s;
  s.insert(1), s.insert(3);
  assert(s.order_of_key(3) == 1 && s.order_of_key(4) == 2 &&
↪   *s.find_by_order(0) == 1);
}
```

78 Hash Map

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

using namespace __gnu_pbds;

struct chash { // To use most bits rather than just the lowest
↪   ones:
  const uint64_t C = ll(2e18 * PI) + 71; // large odd number
  const int RANDOM = 912387491;
  ll operator()(ll x) const { return __builtin_bswap64((x ^
↪   RANDOM) * C); }
};
template<class K, class V> using ht = gp_hash_table<K, V, chash>;
template<class K, class V> V get(ht<K, V>& u, K x) {
  auto it = u.find(x); return it == end(u) ? 0 : it->snd;
}

ht<ll, int> h({}, {}, {}, {}, {1<<20});
```

79 Fast I/O

```
const int BUF_SIZE = 4096;

char buf[BUF_SIZE];
int bufLen = 0, pos = 0;

inline int getChar() {
  if (pos == bufLen) {
    pos = 0, bufLen = (int) fread(buf, 1, BUF_SIZE, stdin);
    if (!bufLen)
      return -1;
  }
  return buf[pos++];;
}

inline int readChar() {
  int c = getChar();
  while (c != -1 && c <= 32)
    c = getChar();
  return c;
}

template <class T>
inline T readInt() {
  int s = 1, c = readChar();
  T x = 0;
  if (c == '-')
    s = -1, c = getChar();
  while ('0' <= c && c <= '9')
    x = x * 10 + c - '0', c = getChar();
  return s == 1 ? x : -x;
}

inline void readWord(char *s) {
  int c = readChar();
  while (c > 32)
    *s++ = (char) c, c = getChar();
  *s = 0;
}

int writePos = 0;
char writeBuf[BUF_SIZE];
```

```
inline void flush() {
  if (writePos)
    fwrite(writeBuf, 1, writePos, stdout), writePos = 0;
}

inline void writeChar(int x) {
  if (writePos == BUF_SIZE)
    flush();
  writeBuf[writePos++] = (char) x;
}

template <class T>
inline void writeInt(T x, char after = '\0') {
  if (x < 0)
    writeChar('-'), x = -x;

  char s[24];
  int n = 0;
  while (x || !n)
    s[n++] = '0' + x % 10, x /= 10;
  while (n--)
    writeChar(s[n]);
  if (after)
    writeChar(after);
}

inline void writeWord(const char *s) {
  while (*s)
    writeChar(*s++);
}
```

12 Notes

80 Работа с деревьями

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
8. Корневая по запросам
9. Тяжелые/легкие вершины
10. DFS $\rightarrow$ дерево блоков, размеры $\in [K..2K]$
11. У вершины не более $O(\sqrt{N})$ разных поддеревьев
12. Сумма размеров поддеревьев без тяжелого ребенка $O(n \log n)$
13. Сумма глубин поддеревьев без глубокого ребенка $O(n)$

81 Маски

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++)
^^Isubmask : for (int s = mask; s; s = (s - 1) & mask)
^^Isupmask : for (int s = mask; s < (1 << n); s = (s + 1) | mask)
```

82 Гранди

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.

83 Потоки

Потоки:

Name	Asympthotic
Ford-Fulkerson	$O(\lvert f \rvert \cdot E)$
Ford-Fulkerson with scaling	$O(\log \lvert f \rvert \cdot E^2)$
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.

84 ДП

Табличка с оптимизациями для динамики:

Name	Original recurrence Sufficient Condition	From To
CHT1	$dp[i] = \min\limits_{j<i} dp[j] + b[j] \cdot a[i]$ $b[j] \geqslant b[j+1] \mid\mid a[i] \leqslant a[i+1]$	$O(n^2)$ $O(n)$
CHT2	$dp[i][j] = \min\limits_{k<j} dp[i-1][k] + b[k] \cdot a[j]$ $b[k] \geqslant b[k+1] \mid\mid a[j] \leqslant a[j+1]$	$O(kn^2)$ $O(kn)$
D&C	$dp[i][j] = \min\limits_{k<j} dp[i-1][k] + c[k][j]$ $p[i,j] \leqslant p[i,j+1]$	$O(kn^2)$ $O(kn \log n)$
Knuth	$dp[i][j] = \min\limits_{i<k<j} dp[i][k] + dp[k][j] + c[i][j]$ $p[i,j-1] \leqslant p[i,j] \leqslant p[i+1,j]$	$O(n^3)$ $O(n^2)$
IOI	$f_n(k)$ — best for fixed k f_n — convex, add penalty $\lambda \cdot k$	$O(k^{(2)}n)$ $O(n \log C)$

85 Комбинаторика

Биномиальные коэффициенты:

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

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: $O(logp)$.

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

86 Делители

- $\leq 20 : d(12) = 6$
- $\leq 50 : d(48) = 10$
- $\leq 100 : d(60) = 12$
- $\leq 1000 : d(840) = 32$
- $\leq 10^4 : d(9\,240) = 64$
- $\leq 10^5 : d(83\,160) = 128$
- $\leq 10^6 : d(720\,720) = 240$
- $\leq 10^7 : d(8\,648\,640) = 338$
- $\leq 10^8 : d(91\,891\,800) = 768$
- $\leq 10^9 : d(931\,170\,240) = 1344$
- $\leq 10^{11} : d(97\,772\,875\,200) = 4032$
- $\leq 10^{12} : d(963\,761\,198\,400) = 6720$
- $\leq 10^{15} : d(866\,421\,317\,361\,600) = 15360$
- $\leq 10^{18} : d(897\,612\,484\,786\,617\,600) = 103680$

87 Числа Белла

i	B_i	i	B_i
0	1	12	4,213,597
1	1	13	27,644,437
2	2	14	190,899,322
3	5	15	1,382,958,545
4	15	16	10,480,142,147
5	52	17	82,864,869,804
6	203	18	682,076,806,159
7	877	19	5,832,742,205,057
8	4,140	20	51,724,158,235,372
9	21,147	21	474,869,816,156,751
10	115,975	22	4,506,715,738,447,323
11	678,570	23	44,152,005,855,084,346

88 Разбиения

Число неупорядоченных разбиений n на положительные слагаемые.

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

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

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

89 Матричные игры

Пишем матрицу стратегий $A_{i,j}$ это выигрыш первого и проигрыш второго, i стратегия 1-го. Седловая точка есть для несмешанной стратегии если $\max_i \min A_{i,*} = \min_j \max A_{*,j}$. Иначе:

$$f(x) = sum(x_i) \to max,\ \ Ans = 1/f(x)$$

$$Ax \leq 1_n,\ \ x_i \geq 0$$

Для 2×2, p первый игрок, q — второй:

$$p^* = \left(\frac{a_{22} - a_{21}}{a_{22} - a_{12} + a_{11} - a_{21}}; \frac{a_{11} - a_{12}}{a_{22} - a_{12} + a_{11} - a_{21}} \right)$$

$$q^* = \left(\frac{a_{22} - a_{12}}{a_{22} - a_{12} + a_{11} - a_{21}}; \frac{a_{11} - a_{21}}{a_{22} - a_{12} + a_{11} - a_{21}} \right)$$

$$Ans = \frac{a_{22}a_{11} - a_{12}a_{21}}{a_{11} + a_{22} - a_{12} - a_{21}}$$

90 Mixed

- Формула Пика: $S = Inside + Edge/2 - 1$
- Теорема Люка: $0 \leq n, m \in \mathbb{Z}$, p простое. $n = n_k p^k + ... + n_1 p + n_0$ и $m = m_k p^k + ... + m_1 p + m_0$. Тогда $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.
- Лемма Бернсайда: $|X/G|$ число орбит G. $X^g = \{x \in X | gx = x\}$

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

91 Ideas

- **Generic:** binary search, ternary search, sort, dp, meet-in-the-middle, divide&conquer, greedy, sqrt-decomposition, matroids, Gauss, FFT, suffix array, suffix automaton, DSU;
- **Graphs:** build graph, add vertices / edges, 2-SAT, flows / cut, matching, Hall's theorem, topsort, HLD, centroid decomposition, MST, Euler cycle, Binary lifting, LCA;
- **Tricks:** consider the process from the end / from the middle, try any one, draw on 2D plane, simplify the problem / consider special case / consider more general case, simplify solution, prefix sums, differences of adjacent elements, consider min/max, analyze why a straightforward solution doesn't work, check limitations, consider contribution of separate element, small answer, different solutions for different limitations, consider complement set, maintain sum / sum of squares, convex function, store O(1) top candidates, inversions, inclusion-exclusion formula, bounding box, angle sort, Grundy function, Eucklid, Mo's algorithm, iterate over divisors, matrix exponentiation;