

# Содержание

<b>1 Common</b>	55 Primitive Root	
1 Setup	56 Simpson	<b>2</b>
2 Template	<b>10. Strings</b>	2
3 Stress	57 Aho-Corasick	2
4 Java	58 Prefix-function	2
	59 Z-function	2
<b>2 Big numbers</b>	60 Hashes	
5 Big Int	61 Manaker	<b>3</b>
6 FFT	62 Palindromic Tree	3
7 FFT by mod and FFT with digits up to $10^6$	63 Suffix Array (+stable)	5
	64 Suffix Automaton	5
	65 Suffix Tree	<b>6</b>
<b>3 Data Structures</b>	<b>11. C++ Tricks</b>	6
8 Centroid Decomposition	66 Fast allocation	6
9 Convex Hull Trick	67 Hash of pair	6
10 DSU	68 Ordered Set	7
11 Fenwick Tree	69 Hash Map	7
12 Hash Table	70 Fast I/O (short)	7
13 Heavy Light Decomposition	71 Fast I/O (long)	8
14 Next Greater in Segment Tree		8
15 Sparse Table	<b>12. Notes</b>	8
16 Fenwick Tree 2D	72 Работа с деревьями	
<b>4 Dynamic Programming</b>	73 Маски	<b>8</b>
17 LIS	74 Гранди	8
18 DP tree	75 Потоки	8
19 Masks tricks	76 ДП	8
<b>5 Flows</b>	77 Комбинаторика	
20 Utilities	78 Делители	<b>9</b>
21 Ford-Fulkerson	79 Числа Белла	9
22 Dinic		9
23 Hungarian		9
24 Min Cost Max Flow		10
<b>6 Games</b>		<b>10</b>
25 Retrograde Analysis		10
<b>7 Geometry</b>		<b>10</b>
26 ClosestPoints (SweepLine)		10
27 ConvexHull		11
28 GeometryBase		11
29 GeometryInterTangent		12
30 GeometrySimple		12
31 Halfplanes Intersection		13
<b>8 Graphs</b>		<b>14</b>
32 2-SAT		14
33 Bridges		14
34 Cut Points		14
35 Eulerian Cycle		15
36 Euler Tour Tree		15
37 Hamilton Cycle		16
38 Karp with cycle		16
39 Kuhn's algorithm		16
40 LCA		17
41 LCA offline (Tarjan)		17
<b>9 Math</b>		<b>17</b>
42 CRT (KTO)		17
43 Discrete Logarithm		17
44 Discrete Root		17
45 Eratosthenes		18
46 Factorial		18
47 Gauss		18
48 Gauss binary		18
49 Gcd		18
50 Gray		19
51 Miller-Rabin Test		19
52 Phi		19
53 Pollard		19
54 Power And Mul		19

# 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:

```
#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 fornrr(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

stress.sh:

```
#!/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

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 {
    IFastScanner in;
    IPrintWriter out;
```

```
    Ivoid solve() {
        Iint a = in.nextInt();
        Iint b = in.nextInt();
        Iout.print(a + b);
    }
```

```
    Ivoid run() {
        Itry {
            IIn = new FastScanner("input.txt");
            Iout = new PrintWriter("output.txt");
            Isolve();
            Iout.flush();
            Iout.close();
            Icatch (FileNotFoundException e) {
                Ie.printStackTrace();
                ISystem.exit(1);
            }
        }
```

```
    Iclass FastScanner {
        IBufferedReader br;
        IStringTokenizer st;
```

```
    Ipublic FastScanner() {
        Ibr = new BufferedReader(new
        InputStreamReader(System.in));
    }
```

```
    Ipublic FastScanner(String s) {
        Itry {
            Ibr = new BufferedReader(new FileReader(s));
            Icatch (FileNotFoundException e) {
                Ie.printStackTrace();
            }
        }
```

```
    IString nextToken() {
        Iwhile (st == null || !st.hasMoreElements()) {
            Itry {
                Ist = new StringTokenizer(br.readLine());
                Icatch (IOException e) {
                    Ie.printStackTrace();
                }
            }
            Ireturn st.nextToken();
        }
```

```
    Iint nextInt() {
        Ireturn Integer.parseInt(nextToken());
    }
```

```
    Ilong nextLong() {
        Ireturn Long.parseLong(nextToken());
    }
```

```
    Idouble nextDouble() {
        Ireturn Double.parseDouble(nextToken());
    }
```

```
    Ichar nextChar() {
        Itry {
            Ireturn (char) (br.read());
        }
```

```

~I~I~I} catch (IOException e) {
~I~I~I~I~Ie.printStackTrace();
~I~I~I~I}
~I~I~I~Ireturn 0;
~I~I~I~I}

~I~I~I~IString nextLine() {
~I~I~I~I~Itry {
~I~I~I~I~I~Ireturn br.readLine();
~I~I~I~I~I} catch (IOException e) {
~I~I~I~I~I~Ie.printStackTrace();
~I~I~I~I~I~I}
~I~I~I~I~Ireturn "";
~I~I~I~I}
~I~I~I~I}

~I~I~I~Ipublic static void main(String[] args) {
~I~I~I~I~Inew Main().run();
~I~I~I~I}
}

```

## 2 Big numbers

### 5 Big Int

```

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] -= (1L)b[j] * k;
            forn(j, b.len)
                if (a[i + j] < 0) {
                    1L 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] += (1L)(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.comp(b) == 0; }
bool operator != ( const num &a, const num &b ) { return a.comp(b) != 0; }
bool operator < ( const num &a, const num &b ) { return a.comp(b) < 0; }
bool operator > ( const num &a, const num &b ) { return a.comp(b) > 0; }
bool operator <= ( const num &a, const num &b ) { return a.comp(b) <= 0; }
bool operator >= ( const num &a, const num &b ) { return a.comp(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

```

int rev[MAX_N];

//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; }
};

Num rt[MAX_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 = 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 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[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(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<sup>6</sup>

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);
for (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);
for (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[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);
    for (i, n)
        r[i] = tc[i] + (td[i] - tc[i] + (ll)MOD2) * MOD1 % MOD2 *
        ↪ MOD1;
}

```

### 3 Data Structures

#### 8 Centroid Decomposition

```

vi g[MAX_N];
int d[MAX_N], par[MAX_N], centroid;
//d par -

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; } //
    ↪ >
};

//      ,      (a,b)      (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; // >
            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[MAX_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) {
    for (i, n) pr[i] = i;
}
```

## 11 Fenwick Tree

```
int t[MAX_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 x){
    int k = get(hash);
    return ht[k] != 0;
}
```

## 13 Heavy Light Decomposition

```
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)
            for (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;
    for (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 * MAX_N], tSize = 1;

//                pos                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[MAX_N][MAX_LOG];
int lg[MAX_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][MAX_N][MAX_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);
}

```

## 4 Dynamic Programming

### 17 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;
}

```

### 18 DP tree

```

int dp[MAX_N][MAX_N], a[MAX_N];
vi g[MAX_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;
}

```

### 19 Masks tricks

```

int dp[(1 << MAX_MASK)][MAX_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

### 20 Utilities

```
vi g[MAX_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
}
```

### 21 Ford-Fulkerson

```
int used[MAX_N], pr[MAX_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;
}
```

### 22 Dinic

```
int pr[MAX_N], d[MAX_N], q[MAX_N], first[MAX_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;
}
```

### 23 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 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, lo = 1;
            forn (j, n)
                if (!u[j]) {
                    if ((tmp = a[x][j] + row[x] + col[j]) < d[j])
                        d[j] = tmp, la[j] = lo;
                    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;
}
```

## 24 Min Cost Max Flow

```

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));
}

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

### 25 Retrograde Analysis

```

int win[MAX_N], lose[MAX_N], outDeg[MAX_N];
vi rg[MAX_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

### 26 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);
for(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;
for(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;
}

```

## 27 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);

    /*
    (1)
    (2)
    (1):
    int k = p.size() - 1;
    while(k > 0 && eq((p[k] - 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;
}

```

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

struct Line{

```

```

    dbl a, b, c;
    Line(): a(0), b(0), c(0) {}
    Line(dbl aa, dbl bb, dbl cc): 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';
    }
};

```

## 29 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;
    dbl x = m.c * l.b - l.c * m.b;
    dbl 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();
    dbl d = fabs((p - a) % (b - a));
    return d * d / (b - a).len2();
}

```

## 30 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 "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);
}
```

## 31 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[ind[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;
}

```

## 8 Graphs

### 32 2-SAT

```

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

vi g[MAXVAR], rg[MAXVAR];

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() {
  for (i, 2 * cntVar) usedSat[i] = 0;
  for (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;
}

```

### 33 Bridges

```

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

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

vector<Edge> g[MAX_N];

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

void findBridges(int v, int parentEdge = -1) {
  if (up[v]) //
    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]);
  }
}

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

```

### 34 Cut Points

```

bool used[MAX_M];
int tIn[MAX_N], timer, isCut[MAX_N], color[MAX_M], compCnt;
vi st;

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

vector<Edge> g[MAX_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;
}

```

## 35 Eulerian Cycle

```

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

vector<Edge> edges;
vi g[MAX_N], res, ptr;
// 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); //
        }
    }
    res.pb(v); // res
}

```

## 36 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[MAX_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));
}

```

## 37 Hamilton Cycle

```

//          -          n*2^n
vi g[MAX_MASK];
int adj[MAX_MASK], dp[1 << MAX_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;
}

```

## 38 Karp with cycle

```

int d[MAX_N][MAX_N], p[MAX_N][MAX_N];
vi g[MAX_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;
}

```

## 39 Kuhn's algorithm

```

//          - n          ,          - m
//
int n, m, paired[2 * MAX_N], used[2 * MAX_N];
vi 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;
fill(used, used + n + m, 0);
for (i, n)
    if (!used[i] && paired[i] == -1 && dfs(i))
        ans++, run = 1;
}
return ans;
}

//
// Max      -- A+, B-
// Min      -- 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]);
}

//
void findCoverIndependent() {
    fill(used, used + n + m, 0);
    for (i, n)
        if (paired[i] == -1)
            dfsCoverIndependent(i);
    for (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);
}

```

## 40 LCA

```

int tin[MAX_N], tout[MAX_N], up[MAX_N][MAX_LOG];
vi g[MAX_N];
int curTime = 0;

void dfs(int v, int p) {
    up[v][0] = p;
    for (i, MAX_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;
    for (i, MAX_LOG)
        if (!isUpper(up[u][i], v))
            u = up[u][i];
    return up[u][0];
}

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

```

## 41 LCA offline (Tarjan)

```

vi g[MAX_N], q[MAX_N];
int pr[MAX_N], ancestor[MAX_N], used[MAX_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) {
    for (i, n) pr[i] = i, ancestor[i] = i;
    dfs(0);
}

```

## 9 Math

### 42 CRT (KTO)

```

vi crt(vi a, vi mod) {
    int n = sz(a);
    vi x(n);
    for (i, n) {
        x[i] = a[i];
        for (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;
}

```

### 43 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 = sqrt(mod);
    int sq2 = mod / sq + (mod % sq ? 1 : 0);
    vector<pii> powers(sq2);
    for (i, sq2)
        powers[i] = mp(power(a, (i + 1) * sq, mod), i + 1);
    sort(all(powers));
    for (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->fst == cur)
            return it->snd * sq - i;
    }
    return -1;
}

```

### 44 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);
}

```

## 45 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;
}

```

## 46 Factorial

```

// Returns pair (rem, deg), where rem = n! % mod,
// deg = k: mod^k | n!, mod is prime, 0(mod log mod)
pii fact(int n, int mod) {
    int rem = 1, deg = 0, nCopy = n;
    while (nCopy) nCopy /= mod, deg += nCopy;
    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 % mod, deg);
}

```

## 47 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;
        for (i, m + 1) swap(a[row][i], a[best][i]);
        for (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;
for (i, m)
    if (par[i] != -1) ans[i] = a[par[i]][m] / a[par[i]][i];
    else single = 0;
for (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;
}

```

## 48 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]);
        for (i, n)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        row++;
    }
    vi ans(m, 0);
    for (i, m)
        if (par[i] != -1)
            ans[i] = a[par[i]][n] / a[par[i]][i];
    bool ok = 1;
    for (i, n) {
        int cur = 0;
        for (j, m) cur ^= (ans[j] & a[i][j]);
        if (cur != a[i][n]) ok = 0;
    }
    return ok;
}

```

## 49 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;
}

```

## 50 Gray

```

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

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

```

## 51 Miller-Rabin Test

```
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), 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;
}

```

## 52 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);
}

```

## 53 Pollard

```

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;
                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.fst != ans.back().fst)
                        ans.pb(np);
                    else
                        ans.back().snd += np.snd;
                return ans;
            }
        }
    }
    assert(0);
}

```

## 54 Power And Mul

```

inline ll fix(ll a, ll mod) { // a in [0, 2 * mod)
    if (a >= mod) a -= mod;
    return a;
}

// Returns (a * b) % mod, 0 <= a < mod, 0 <= b < mod
ll mulSlow(ll a, ll b, ll mod) {
    if (!b) return 0;
    ll c = fix(mulSlow(a, b / 2, mod) * 2, mod);
    return b & 1 ? fix(c + a, mod) : c;
}

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

```

```

}

int power(int a, int n, int mod) {
    if (!n) return 1;
    int b = power(a, n / 2, mod);
    b = (b * 111 * b) % mod;
    return n & 1 ? (a * 111 * b) % mod : b;
}

ll powerLL(ll a, ll n, ll mod) {
    if (!n) return 1;
    ll b = power(a, n / 2, mod);
    b = mul(b, b, mod);
    return n & 1 ? mul(a, b, mod) : b;
}

int powerFast(int a, int n, int mod) {
    int res = 1;
    while (n) {
        if (n & 1)
            res = (res * 111 * a) % mod;
        a = (a * 111 * a) % mod;
        n /= 2;
    }
    return res;
}

```

## 55 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;
}

```

## 56 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;
}

```

## 10 Strings

### 57 Aho-Corasick

```

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

```

```

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[MAX_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];
}

```

## 58 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;
}

```

## 59 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;
}

```

## 60 Hashes

```
const int P = 239017;

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

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

const int MOD_X = 1e9 + 9, MOD_Y = 1e9 + 7;

// using H = unsigned long long;
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& 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*(ll k) const { return H(int((x * k) %
        ↪ MOD_X), int((y * k) % MOD_Y)); }
    inline H operator*(const H& h) const { return H(int((ll(x) *
        ↪ h.x) % MOD_X), int((ll(y) * h.y) % MOD_Y)); }
    inline bool operator==(const H& h) const { return x == h.x
        ↪ && y == h.y; }
    inline bool operator!=(const H& h) const { return x != h.x
        ↪ || y != h.y; }
    inline bool operator<(const H& h) const { return x < h.x ||
        ↪ (x == h.x && y < h.y); }
    explicit inline operator ll() const { return ll(x) * MOD_Y +
        ↪ y + 1; } // > 0
};

H deg[MAX_N], h[MAX_N];

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

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

## 61 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;
        }
    }
}
```

}

## 62 Palindromic Tree

```
const int ALPHA = 26;

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

int vn, v;
Vertex t[MAX_N + 2];
int n, s[MAX_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;
}
```

## 63 Suffix Array (+stable)

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

inline int mod(int x) {
    return (x + sLen) % sLen;
}

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

```

}

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

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

```

## 64 Suffix Automaton

```

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 (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;
    }
};

```

## 65 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);
}

```

## 11 C++ Tricks

### 66 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*) {}

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

```

### 67 Hash of pair

```

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

```

### 68 Ordered Set

```

#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) << " " << X.order_of_key(1) <<
        ↪ "\n";
}

```

### 69 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 = 1l(2e18 * PI) + 71; // large odd number
    const int RANDOM = 912387491;

```

```

    1l operator()(1l 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) {
    return u.find(x) == end(u) ? 0 : u[x];
}

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

```

## 70 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 = readChar(), x = 0;
    if (c == '-')
        s = 1, c = readChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = readChar();
    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]);
}

```

## 71 Fast I/O (long)

```

template <class T = int> inline T readInt();
inline double readDouble();
inline int readUInt();
inline int readChar();
inline void readWord(char *s);
inline bool readLine(char *s); // do not save '\n'
inline bool isEof();
inline int peekChar();
inline bool seekEof();

template <class T> inline void writeInt(T x, int len);
template <class T> inline void writeUInt(T x, int len);
template <class T> inline void writeInt(T x) { writeInt(x,
    ↪ -1); };
template <class T> inline void writeUInt(T x) { writeUInt(x,
    ↪ -1); };
inline void writeChar(int x);
inline void writeWord(const char *s);
inline void writeDouble(double x, int len = 0);
inline void flush();

const int BUF_SIZE = 4096;

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

```

```

inline bool isEof() {
    if (pos == bufLen) {
        pos = 0, bufLen = fread(buf, 1, BUF_SIZE, stdin);
        if (pos == bufLen)
            return 1;
    }
    return 0;
}

inline int getChar() {
    return isEof() ? -1 : buf[pos++];
}

inline int peekChar() {
    return isEof() ? -1 : buf[pos];
}

inline bool seekEof() {
    int c;
    while ((c = peekChar()) != -1 && c <= 32)
        pos++;
    return c == -1;
}

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

inline int readUInt() {
    int c = readChar(), x = 0;
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    return x;
}

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 double readDouble() {
    int s = 1, c = readChar();
    double x = 0;
    if (c == '-')
        s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    if (c == '.') {
        c = getChar();
        double coef = 1;
        while ('0' <= c && c <= '9')
            x += (c - '0') * (coef *= 1e-1), c = getChar();
    }
    return s == 1 ? x : -x;
}

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

```

```

}

inline bool readLine(char *s) {
    int c = getChar();
    while (c != '\n' && c != -1)
        *s++ = c, c = getChar();
    *s = 0;
    return c != -1;
}

int writePos = 0;
char writeBuf[BUF_SIZE];

inline void writeChar(int x) {
    if (writePos == BUF_SIZE)
        fwrite(writeBuf, 1, BUF_SIZE, stdout), writePos = 0;
    writeBuf[writePos++] = x;
}

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

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

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

template <class T>
inline void writeUInt(T x, int outputLen) {
    char s[24];
    int n = 0;
    while (x || !n)
        s[n++] = '0' + char(x % 10), x /= 10;
    while (n < outputLen)
        s[n++] = '0';
    while (n--)
        writeChar(s[n]);
}

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

inline void writeDouble(double x, int outputLen) {
    if (x < 0)
        writeChar('-'), x = -x;
    int t = (int) x;
    writeUInt(t, x -= t);
    writeChar('.');
    for (int i = outputLen - 1; i > 0; i--) {
        x *= 10;
        t = std::min(9, (int) x);
        writeChar('0' + t), x -= t;
    }
    x *= 10;
    t = std::min(9, (int)(x + 0.5));
    writeChar('0' + t);
}

```



}

## 12 Notes

### 72 Работа с деревьями

Приемы для работы с деревьями:

1. Двоичные подьемы
2. Поддеревья как отрезки Эйлера 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  $O(n)$

### 73 Маски

Считаем динамику по маскам за  $O(2^n \cdot n)$   $f[mask] = \text{sum}$  по  $\text{submask}$   $g[\text{submask}]$ .  $dp[mask][i]$  — 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  $dp[mask][i] = dp[mask][i+1] + dp[mask \setminus (1 \ll i)][i+1]$ .

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)
```

### 74 Гранди

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

### 75 Поток

Потоки:

Name	Asymptotic
Ford-Fulkerson	$O( f  \cdot E)$
Ford-Fulkerson with scaling	$O(\log  f  \cdot E^2)$
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  $e$  ( $v \rightarrow u$ ) 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.

### 76 ДП

Табличка с оптимизациями для динамики:

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

### 77 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:  $O(\log p)$ .
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:  $O(\log p)$ .

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

### 78 Делители

- $\leq 20 : d(12) = 6$
- $\leq 50 : d(48) = 10$
- $\leq 100 : d(60) = 12$
- $\leq 1000 : d(840) = 32$

- $\leq 10^4 : d(9240) = 64$
- $\leq 10^5 : d(83160) = 128$
- $\leq 10^6 : d(720720) = 240$
- $\leq 10^7 : d(8648640) = 338$
- $\leq 10^8 : d(91891800) = 768$
- $\leq 10^9 : d(931170240) = 1344$
- $\leq 10^{11} : d(97772875200) = 4032$
- $\leq 10^{12} : d(963761198400) = 6720$
- $\leq 10^{15} : d(866421317361600) = 15360$
- $\leq 10^{18} : d(897612484786617600) = 103680$

79 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