

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

```
@echo off

for /L %i in (1,1,1000000) do (
gen.exe || exit
main.exe || exit
stupid.exe || exit
fc .out 2.out || exit
echo Test %i OK
)
```

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

```
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 < s.size() && (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[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);
for (i, n)
    r[i] = (T) round(tmp[i].real());
}

void init() { // don't forget to init
    for (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));
        for (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[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;
    for (i, n) {
        int x = int(a[i] % mod);
        ta[i] = Num(x & tw, x >> HALF);
    }
    for (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;

// 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[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);
}

```

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[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;
}

```

20 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

21 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
}

```

22 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;
}

```

```

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[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;
}

```

24 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, 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

```
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

26 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

27 ClosestPoints (SweepLine)

```
const int N = 2e5;

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



```

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

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

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

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

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

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

```

28 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);

    /* To keep 180 angles (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] - r[rn - 1]) % (q - r[rn - 1]), 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';
    }
};

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; }

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
int circle_line(const Circle &c, const Line &l){
    dbl d = l.d(c.p);
    if(fabs(d) > c.r + EPS)
        return 0;
    if(fabs(fabs(d) / c.r - 1) < EPS) {
        v.pb(c.p - l.no() * d);
        return 1;
    } 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);
        return 2;
    }
}

```

```

// Intersection of two circles, 3 = inf
int circle_circle(const Circle &a, const Circle &b) {
    if (a.p == b.p && eq(a.r, b.r))
        return 3;
    Pnt diff = b.p - a.p;
    dbl dist = diff.len();
    if (ls(a.r + dist, b.r) || ls(b.r + dist, a.r))
        return 0;
}

```

```

Line line(diff.x * 2, diff.y * 2, a.p.len2() - b.p.len2() +
↪ sqr(b.r) - sqr(a.r));
return circle_line(a, line);
}

// 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);
}

```

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() {
    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;
}

```

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(); // 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();
        }
}

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;
// 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
}

```

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

```

// DP in O(n*2^n) for Ham cycle
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

```

// sz(LEFT) = n, sz(RIGHT) = m
// numbered consequently
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);
        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);
}

```

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

42 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 = (int) graph.size();
    vector<int> color(n), used(n, -1);
    for (int i = 0; i < n; i++)
        color[i] = i;
    vector<Edge> e(n);
    for (int i = 0; i < n; i++) {
        if (graph[i].empty())
            e[i] = {-1, -1, -1, -1};
        else {
            e[i] = *min_element(graph[i].begin(),
                               ↪ graph[i].end());
        }
    }
    vector<vector<int>> cycles;
    used[0] = -2;
    for (int s = 0; s < n; s++) {
        if (used[s] != -1)
            continue;
        int x = s;
        while (used[x] == -1) {
            used[x] = s;
            x = e[x].fr;
        }
        if (used[x] != s)
            continue;
        vector<int> 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);
    for (int s = 0; s < n; s++) {
        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;
}
```

9 Math

43 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;
}
```

44 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);
    forn (i, sq2)
        powers[i] = mp(power(a, (i + 1) * sq, mod), i + 1);
    sort(all(powers));
    forn (i, sq + 1) {
        int cur = power(a, i, mod);
        cur = (cur * 111 * 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;
}
```

45 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);
}
```

46 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;
}

```

47 Factorial

```

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

```

48 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;
}

```

49 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;
}

```

50 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;
}

```

51 Gray

```

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

```

```

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

```


52 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 = powerLL(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;
    }
}
```

53 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);
}
```

54 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);
    }
}
```

55 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 * 1ll * b) % mod;
    return n & 1 ? (a * 1ll * b) % mod : b;
}
```

```
ll powerLL(ll a, ll n, ll mod) {
    if (!n) return 1;
    ll b = powerLL(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 * 1ll * a) % mod;
        a = (a * 1ll * a) % mod;
        n /= 2;
    }
    return res;
}
```

56 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;
}
```

57 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

58 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];
}
```

59 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;
}
```

60 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;
}
```

61 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;
}
```

62 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 = 1 + (r - i - t);
                k = min(z[j], j - 1);
            }
            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;
    }
}
}

```

63 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;
}

```

64 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 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;
    }
}

```

65 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;
    }
};

```

66 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;
map<int,int> 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 L-R;}
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{
            map<int,int>::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);
    for(int i=0;i<L;i++)
        P=add_char(P,i);
}

```

11 C++ Tricks

67 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); }

```

68 Hash of pair

```

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

```

69 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";
}

```

70 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;
    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) {
    return u.find(x) == end(u) ? 0 : u[x];
}

```

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

71 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]);
}

```

72 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

73 Работа с деревьями

Приемы для работы с деревьями:

- 1. Двоичные подъемы
- 2. Поддеревья как отрезки Эйлераова 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 → дерево блоков, размеры ∈ [K..2K]
- 11. У вершины не более O(√N) разных поддеревьев
- 12. Сумма размеров поддеревьев без тяжелого ребенка O(n log n)
- 13. Сумма глубин поддеревьев без глубокого ребенка O(n)

74 Маски

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

75 Гранди

Теорема Шпрага-Гранди: берем тех всех значений функции Гранди по состояни-ям, 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 xor по чему-нибудь.

76 Потоки

Потоки:

Name	Asympthotic
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 фиктив-ные 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 мак-симальный 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 мак-симальный поток по найденному циклу.

77 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 $\lambda \cdot k$	$O(k^{(2)}n)$ $O(n \log C)$

78 Комбинаторика

Биномиальные коэффициенты:
 Теорема Люка для биномиальных коэффициентов: Хотим посчитать C_n^k , раз-ложим 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 оста-ток
 precalc: $O(p \log p)$, query: $O(\log p)$.
 $C_n^{\frac{n}{2}} = \frac{2^n}{\sqrt{\frac{\pi n}{2}}}$

79 Делители

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

80 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

81 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	$\sim 2\text{e}5$	$\sim 2\text{e}8$

82 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:

$$f(x) = \text{sum}(x_i) \rightarrow \text{max}, \; \text{Ans} = 1/f(x)$$

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

Для 2×2 , p первый игрок, q — второй:

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

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

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

83 Mixed

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

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