ACM-ICPC Team Reference Document

ICTIS SFU 1 (Bystrov, Ivanov, Lyz)

Contents

1 General

1.1 C++ Template

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:268435456")
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdio>
#include <cstdlib>
#include <cassert>
#include <climits>
#include <cmath>
#include <algorithm>
#include <cstring>
#include <string>
#include <vector>
```

```
#include <list>
#include <stack>
#include <set>
#include <bitset>
#include <queue>
#include <map>
#include <sstream>
#include <functional>
#include <unordered_map>
#include <unordered_set>
#include <complex>
#include <random>
#include <chrono>

using namespace std;

#define int long long
#define all(x) (x).begin(), (x).end()
#define sqr(cd) ((cd) * (cd))

#define y0 sdkfaslhagaklsldk
#define y1 aasdfasdfasdf
#define yn askfhwqriuperikldjk
#define j1 assdgsdgasghsf
#define tm sdfjahlfasfh
#define lr asgasgash
#define norm asdfasdgasdgsd
#define have adsgagshdshfhds
#define ends asdgahhfdsfshdshfd

template <typename T> void alert(const T& t) { cout << t
    << endl; exit(0); }
template <typename T> using min_heap = priority_queue<T,
    vector<T>, greater<T>>;
template <typename T> using max_heap = priority_queue<T,
    vector<T>, less<T>>;

typedef long long int64;
typedef unsigned long long uint64;
typedef long double ld;
typedef array<uint64, 2> hv;

// region rnd
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
#define rnd(a, b) (uniform_int_distribution<int>((a), (b))(rng
    ))
// endregion rnd

// region time
#include <time.h>
clock_t __clock__;
void startTime() {__clock__ = clock();}
void timeit(string msg) {cerr << "> " << msg << ": " <<
    precise(6) << ld(clock()-__clock__)/
    CLOCKS_PER_SEC << endl;}
// endregion time
const double PI = acos(-1);

signed main()
{
#ifdef _DEBUG
    freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
#endif // _DEBUG

    //srand(NULL);
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    // cout.precision(15);
    // cin >> hex >> n;

    return 0;
}
```

1.2 Troubleshoot

Pre-submit:
Write a few simple test cases, if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all datastructures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your team mates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all datastructures between test cases?

1.3 Python Template

```
import sys
import re
from math import ceil, log, sqrt, floor

__local_run__ = False
if __local_run__:
    sys.stdin = open('input.txt', 'r')
    sys.stdout = open('output.txt', 'w')

def main():
    a = int(input())
    b = int(input())
    print(a*b)

main()
```

1.4 Fast Input Output Template Java

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.lang.*;

public class Main {
  public static void main(String[] args) {
    InputReader in = new InputReader(System.in);
    OutputWriter out = new OutputWriter(System.out);
    // Do your thing
    out.close();
  }
```

```
}

class InputReader {
  private InputStream stream;
  private byte[] buf = new byte[1024];
  private int curChar;
  private int numChars;
  private SpaceCharFilter filter;

  public InputReader(InputStream stream) {
    this.stream = stream;
  }

  public int read() {
    if (numChars == -1) {
      throw new InputMismatchException();
    }

    if (curChar >= numChars) {
      curChar = 0;
      try {
        numChars = stream.read(buf);
      } catch (IOException e) {
        throw new InputMismatchException();
      }
      if (numChars <= 0) {
        return -1;
      }
    }

    return buf[curChar++];
  }

  public int readInt() {
    int c = read();
    while (isSpaceChar(c)) {
      c = read();
    }

    int sgn = 1;
    if (c == '-') {
      sgn = -1;
      c = read();
    }

    int res = 0;
    do {
      if (c < '0' || c > '9') {
        throw new InputMismatchException();
      }
      res *= 10;
      res += c - '0';
      c = read();
    } while (!isSpaceChar(c));

    return res * sgn;
  }

  public String readString() {
    int c = read();
    while (isSpaceChar(c)) {
      c = read();
    }

    StringBuilder res = new StringBuilder();
    do {
      res.appendCodePoint(c);
      c = read();
    } while (!isSpaceChar(c));
    return res.toString();
  }

  public boolean isSpaceChar(int c) {
    if (filter != null) {
      return filter.isSpaceChar(c);
    }
    return c == ' ' || c == '\n' || c == '\r' || c == '\t' || c ==
        -1;
  }

  public String next() {
    return readString();
  }

  public interface SpaceCharFilter {
    public boolean isSpaceChar(int ch);
  }
}

class OutputWriter {
  private final PrintWriter writer;

  public OutputWriter(OutputStream outputStream) {
    writer = new PrintWriter(
      new BufferedWriter(new OutputStreamWriter(
          outputStream))
    );
  }

  public OutputWriter(Writer writer) {
    this.writer = new PrintWriter(writer);
  }

  public void print(Object... objects) {
    for (int i = 0; i < objects.length; i++) {
      if (i != 0) {
        writer.print(' ');
      }
      writer.print(objects[i]);
    }
  }

  public void printLine(Object... objects) {
    print(objects);
    writer.println();
  }

  public void close() {
    writer.close();
  }

  public void flush() {
    writer.flush();
  }
}

class IOUtils {
  public static int[] readIntArray(InputReader in, int size) {
    int[] array = new int[size];
    for (int i = 0; i < size; i++) {
      array[i] = in.readInt();
    }
    return array;
  }
}
```

2 Data Structures

2.1 Disjoin Set Union

```
struct DSU
{
    vector<int> p;
    vector<int> sz;

    DSU(int n)
        {
        FOR(i, 0, n)
                {
            p.push_back(i);
            sz.push_back(1);
        }
    }

    int find(int a)
        {
        return p[a] = p[a] == a ? a : find(p[a]);
    }

    bool same(int a, int b)
        {
        return find(a) == find(b);
    }

    void unite(int a, int b)
        {
        a = find(a);
        b = find(b);
        if(sz[a] > sz[b]) swap(a, b);
        sz[b] += sz[a];
        p[a] = b;
```

```
    }
};
```

2.2 Fenwick Tree Point Update And Range Query

```
struct Fenwick {
    vector<ll> tree;
    int n;
    Fenwick(){}
    Fenwick(int _n) {
        n = _n;
        tree = vector<ll>(n+1, 0);
    }
    void add(int i, ll val) { // arr[i] += val
        for(; i <= n; i += i&(-i)) tree[i] += val;
    }
    ll get(int i) { // arr[i]
        return sum(i, i);
    }
    ll sum(int i) { // arr[1]+...+arr[i]
        ll ans = 0;
        for(; i > 0; i -= i&(-i)) ans += tree[i];
        return ans;
    }
    ll sum(int l, int r) {// arr[l]+...+arr[r]
        return sum(r) - sum(l-1);
    }
};
```

2.3 Fenwick Tree Range Update And Point Query

```
struct Fenwick {
    vector<ll> tree;
    vector<ll> arr;
    int n;
    Fenwick(vector<ll> _arr) {
        n = _arr.size();
        arr = _arr;
        tree = vector<ll>(n+2, 0);
    }
    void add(int i, ll val) { // arr[i] += val
        for(; i <= n; i += i&(-i)) tree[i] += val;
    }
    void add(int l, int r, ll val) {// arr[l..r] += val
        add(l, val);
        add(r+1, -val);
    }
    ll get(int i) { // arr[i]
        ll sum = arr[i-1]; // zero based
        for(; i > 0; i -= i&(-i)) sum += tree[i];
        return sum; // zero based
    }
};
```

2.4 Fenwick Tree Range Update And Range Query

```
struct RangedFenwick {
    Fenwick F1, F2; // support range query and point update
    RangedFenwick(int _n) {
        F1 = Fenwick(_n+1);
        F2 = Fenwick(_n+1);
    }
    void add(int l, int r, ll v) { // arr[l..r] += v
        F1.add(l, v);
        F1.add(r+1, -v);
        F2.add(l, v*(l-1));
        F2.add(r+1, -v*r);
    }
    ll sum(int i) { // arr[1..i]
        return F1.sum(i)*i-F2.sum(i);
    }
    ll sum(int l, int r) { // arr[l..r]
        return sum(r)-sum(l-1);
    }
};
```

2.5 Fenwick 2D

```
struct Fenwick2D {
    vector<vector<ll>> bit;
    int n, m;
    Fenwick2D(int _n, int _m) {
        n = _n; m = _m;
        bit = vector<vector<ll>>(n+1, vector<ll>(m+1, 0));
    }
    ll sum(int x, int y) {
        ll ret = 0;
        for (int i = x; i > 0; i -= i & (-i))
            for (int j = y; j > 0; j -= j & (-j))
                ret += bit[i][j];
        return ret;
    }
    ll sum(int x1, int y1, int x2, int y2) {
        return sum(x2, y2) - sum(x2, y1-1) - sum(x1-1, y2) +
            sum(x1-1, y1-1);
    }
    void add(int x, int y, ll delta) {
        for (int i = x; i <= n; i += i & (-i))
            for (int j = y; j <= m; j += j & (-j))
                bit[i][j] += delta;
    }
};
```

2.6 Segment Tree

```
struct Node
{
        Node() : value(MIN_VALUE), push(MIN_VALUE) {}
        int64 value;
        int64 push;
};

struct ST
{
        ST(int n) : vec(4 * n),sz(4*n) {}

        void Set(int sl, int sr, int l, int r, int v, int64 value)
        {
                Refresh(v);

                if (sl == l && sr == r)
                {
                        vec[v].value = max(vec[v].value, value);
                        vec[v].push = value;
                        return;
                }

                int mid = MID(sl, sr);

                if (r <= mid) Set(sl, mid, l, r, LSON(v), value);
                else if (l > mid) Set(mid + 1, sr, l, r, RSON(v),
                    value);
                else
                {
                        Set(sl, mid, l, mid, LSON(v), value);
                        Set(mid + 1, sr, mid + 1, r, RSON(v),
                            value);
                }

                Refresh(LSON(v));
                Refresh(RSON(v));

                vec[v].value = min(vec[LSON(v)].value, vec[RSON
                    (v)].value);
        }

        void Refresh(int v)
        {
                if (v > sz) return;
```

```
        vec[v].value = max(vec[v].value, vec[v].push);

        if (RSON(v) >= sz || vec[v].push ==
            MIN_VALUE) return;

        vec[LSON(v)].push = max(vec[v].push,vec[LSON(
            v)].push);
        vec[RSON(v)].push = max(vec[v].push, vec[RSON
            (v)].push);
        vec[v].push = MIN_VALUE;
    }

    int64 Get(int sl, int sr, int l, int r, int v)
    {
        Refresh(v);

        if (sl == l && sr == r)
        {
            return vec[v].value;
        }

        int mid = MID(sl, sr);

        if (r <= mid) return Get(sl, mid, l, r, LSON(v));
        else if (l > mid) return Get(mid + 1, sr, l, r,
            RSON(v));
        else
        {
            int64 left = Get(sl, mid, l, mid, LSON(v));
            int64 right = Get(mid + 1, sr, mid + 1, r,
                RSON(v));

            return min(left, right);
        }
    }

    vector<Node> vec;
    int sz;
};
```

2.7 Treap

```
struct Treap
{
    int _size;
    static int sizeOf(const Treap *t)
    {
        return t ? t->_size : 0;
    }

    Treap *l, *r;
    int y;

    Treap() : l(0), r(0), y((rand() << 0xf) ^ rand()), _size
        (1)
    { }
    Treap(Treap *base, Treap *l, Treap *r) : l(l), r(r), y(base
        ->y), _size(1)
    {
        _size += sizeOf(l) + sizeOf(r);
        delete base;
    }

    static Treap* Merge(Treap *l, Treap *r)
    {
        if (!l) return r;
        if (!r) return l;
        if (l->y > r->y)
        {
            return new Treap(l, l->l, Merge(l->r, r));
        }
        else
        {
            return new Treap(r, Merge(l, r->l), r->r);
        }
    }

    void Split(Treap *&l, Treap *&r, int count)
    {
        Treap *nt;
        l = r = nt = 0;
        int leftCount = sizeOf(this->l);
        if (count <= leftCount)
        {
            if (this->l) this->l->Split(l, nt, count);
            r = new Treap(this, nt, this->r);
        }
        else
        {
            if (this->r) this->r->Split(nt, r, count -
                leftCount - 1);
            l = new Treap(this, this->l, nt);
        }
    }

    Treap* Insert(Treap *nt, int index)
    {
        Treap *l, *r;
        Split(l, r, index);
        return Merge(Merge(l, nt), r);
    }

    Treap* Remove(int index, Treap *&removed)
    {
        Treap *l, *r;
        Split(l, r, index);
        r->Split(removed, r, 1);
        return Merge(l, r);
    }

    Treap* Remove(int index)
    {
        Treap *l, *r, *m;
        Split(l, r, index);
        r->Split(m, r, 1);
        delete m;
        return Merge(l, r);
    }

    void ToVector(vint &v)
    {
        if (l) l->ToVector(v);
        v.push_back(y);
        if (r) r->ToVector(v);
    }

    static Treap* FromVector(const vint &v)
    {
        if (v.empty()) return 0;
        Treap *t = new Treap(/*v[0]*/);
        for(int i = 1; i < v.size(); i++)
        {
            t = Merge(t, new Treap(/*v[i]*/));
        }
        return t;
    }
};
```

3 Graphs

3.1 Floyd

```
vector<vector<int64>> g(n, vector<int64>(n));
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < n; ++j)
    {
        cin >> g[i][j];
    }
}

for (int k = 0; k < n; ++k)
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
        }
    }
}
```

3.2 Bridges

```
struct Node
{
	int in, lowLink;
	vector<int> e;
};

vector<Node> g;

int TIME = 0;
void DFS_T(int v, int p)
{
	g[v].in = g[v].lowLink = ++TIME;

	for (int i = 0; i < g[v].e.size(); ++i)
	{
		int to = g[v].e[i];

		if (to == p) continue;

		if (g[to].in == 0)
		{
			DFS_T(to, v);
			g[v].lowLink = min(g[v].lowLink, g[to].
				lowLink);

			if (g[v].in < g[to].lowLink)
			{
				ans.insert(ids[{v, to}]);
			}
		}
		else
		{
			g[v].lowLink = min(g[v].lowLink, g[to].in);
		}
	}
}
```

3.3 Cut Points

```
struct Node
{
	int in, lowLink;
	vector<int> e;
	bool ans = false;
};

int TIME = 0;
void DFS_T(int v, int p)
{
	g[v].in = g[v].lowLink = ++TIME;

	int children = 0;

	for (int i = 0; i < g[v].e.size(); ++i)
	{
		int to = g[v].e[i];

		if (to == p) continue;

		if (g[to].in == 0)
		{
			DFS_T(to, v);
			children++;
			g[v].lowLink = min(g[v].lowLink, g[to].
				lowLink);


			if (g[v].in <= g[to].lowLink && p != -1)
			{
				g[v].ans = true;
			}
		}
		else
		{
			g[v].lowLink = min(g[v].lowLink, g[to].in);
		}
	}

	if (children > 1 && p == -1) g[v].ans = true;
}
```

3.4 Condense

```
struct Node
{
	int in = 0;
	int lowLink = 0;
	int color = 0;
	int inStack = false;
	vector<int> e;
};

int TIME = 0;
int COLOR = 0;
vector<int> st;

void DFS_T(int f, int p)
{
	st.push_back(f);
	g[f].in = g[f].lowLink = ++TIME;
	g[f].inStack = true;

	for (auto& it : g[f].e)
	{
		//if (it == p) continue;
		if (g[it].in == 0)
		{
			DFS_T(it, f);
			g[f].lowLink = min(g[f].lowLink, g[it].
				lowLink);

			// bridges
			// cut-points
		}
		else if (g[it].inStack)
		{
			g[f].lowLink = min(g[f].lowLink, g[it].in);
		}
	}

	if (g[f].lowLink == g[f].in)
	{
		COLOR++;
		int x;
		do
		{
			x = st.back(); st.pop_back();
			g[x].color = COLOR;
			g[x].inStack = false;
		} while (x != f);
	}
}
```

3.5 Number Of Paths Of Fixed Length

Let G be the adjacency matrix of a graph. Then $C_k = G^k$ gives a matrix, in which the value $C_k[i][j]$ gives the number of paths between i and j of length k.

3.6 Shortest Paths Of Fixed Length

Define $A \bigodot B = C \iff C_{ij} = min_{p=1..n}(A_{ip} + B_{pj})$. Let G be the adjacency matrix of a graph. Also, let $L_k = G \bigodot \ldots \bigodot G = G^{\bigodot k}$. Then the value $L_k[i][j]$ denotes the length of the shortest path between i and j which consists of exactly k edges.

3.7 Dijkstra

```
struct Edge
{
	Edge(int64 a, int b) : cost(a), to(b) {};
	Edge() {};
```

```
	int64 cost;
	int to;
};

int64 INF = 1e11 + 77;

struct Node
{
	vector<Edge> edges;
	int64 dist = INF;
};
vector<Node> graph;

void Dij(int from)
{
	graph[from].dist = 0;

	priority_queue<pair<int64, int64>, vector<pair<int64,
	    int64>>, greater<pair<int64, int64>>> que;

	que.push({ 0,from });

	while (!que.empty())
	{
		auto top = que.top(); que.pop();
		int v = top.second;
		int dis = top.first;

		if (graph[v].dist != dis) continue;

		for (auto& it : graph[v].edges)
		{
			int64 now = it.cost + dis;

			if (now < graph[it.to].dist)
			{
				graph[it.to].dist = now;
				que.push({ now,it.to });
			}
		}
	}
}
```

3.8 Bellmanford

```
struct Edge
{
	Edge(int64 t = 0, int64 c = 0) : t(t), c(c) {}
	int64 t, c;
};
int64 INF = 1e17+17;
struct Node
{
	int64 d;
	vector<Edge> e;
};

vector<Node> g;
void Refresh()
{
	for (auto& v : g) v.d = INF;
}

void FB(int64 s,bool again = false)
{
	if (!again)
	{
		Refresh();
		g[s].d = 0;
	}

	for (int64 step = 0; step < g.size() - 1; ++step)
	{
		for (int64 i = 0; i < g.size(); ++i)
		{
			if (g[i].d == INF) continue;
			for (const auto& e : g[i].e)
			{
				int64 u = e.t;
				int64 c = e.c;

				if (g[u].d > g[i].d + c)
				{
					if (!again) g[u].d = min(INF,
					    g[i].d + c);
					else g[u].d = -INF; //
					    negative cycle
				}
			}
		}
	}
}

FB(s);
FB(s, true);
```

3.9 Kruskal

```
struct Edge
{
	Edge(int a, int b, int c)
		: v(a),u(b),cost(c)
	{
	}

	int v;
	int u;
	int cost;

	bool operator<(const Edge& obj) const
	{
		return cost < obj.cost;
	}
};

vector<Edge> edges;

vector<int> p;
int dsu_get(int v)
{
	return (v == p[v]) ? v : (p[v] = dsu_get(p[v]));
}

void dsu_unite(int v, int u)
{
	if (rand() & 1) swap(v, u);

	if (v != u) p[v] = u;
}

map<int, int> ids;
forn(i, m)
{
	int a, b, c;
	cin >> a >> b >> c;
	a--; b--;

	if (ids.find(a) == ids.end())
	{
		ids.insert(make_pair(a, ids.size()));
	}

	if (ids.find(b) == ids.end())
	{
		ids.insert(make_pair(b, ids.size()));
	}

	edges.push_back(Edge(a, b, c));
}

sort(all(edges));

forn(i, m)
{
	int v = ids.find(edges[i].v)->second;
	int u = ids.find(edges[i].u)->second;
	int cost = edges[i].cost;
	if (dsu_get(v) != dsu_get(u))
	{
		ans += cost;
		dsu_unite(dsu_get(v), dsu_get(u));
	}
}
```

3.10 Topo Sort

```
void dfs (int v)
{
	used[v] = true;
	for (size_t i=0; i<g[v].size(); ++i)
	{
		int to = g[v][i];
		if (!used[to])
			dfs (to);
	}
	ans.push_back (v);
}

void topological_sort()
{
	for (int i=0; i<n; ++i)
		used[i] = false;
	ans.clear();
	for (int i=0; i<n; ++i)
		if (!used[i])
			dfs (i);
	reverse (ans.begin(), ans.end());
}
```

3.11 Khun

```
bool try_kuhn (int v) {
	if (used[v]) return false;
	used[v] = true;
	for (size_t i=0; i<g[v].size(); ++i) {
		int to = g[v][i];
		if (mt[to] == -1 || try_kuhn (mt[to])) {
			mt[to] = v;
			return true;
		}
	}
	return false;
}

for (int v=0; v<n; ++v) {
	used.assign (n, false);
	try_kuhn (v);
}

for (int i=0; i<k; ++i)
	if (mt[i] != -1)
		printf ("%d %d\n", mt[i]+1, i+1);
```

3.12 Khun Ex

```
mt.assign (k, -1);
vector<char> used1 (n);
for (int i=0; i<n; ++i)
	for (size_t j=0; j<g[i].size(); ++j)
		if (mt[g[i][j]] == -1) {
			mt[g[i][j]] = i;
			used1[i] = true;
			break;
		}

for (int i=0; i<n; ++i) {
	if (used1[i]) continue;
	used.assign (n, false);
	try_kuhn (i);
}
```

3.13 Lca

```
typedef vector < vector<int> > graph;
typedef vector<int>::const_iterator const_graph_iter;

vector<int> lca_h, lca_dfs_list, lca_first, lca_tree;
vector<char> lca_dfs_used;

void lca_dfs (const graph & g, int v, int h = 1)
{
	lca_dfs_used[v] = true;
	lca_h[v] = h;
	lca_dfs_list.push_back (v);
	for (const_graph_iter i = g[v].begin(); i != g[v].end();
	    ++i)
		if (!lca_dfs_used[*i])
		{
			lca_dfs (g, *i, h+1);
			lca_dfs_list.push_back (v);
		}
}

void lca_build_tree (int i, int l, int r)
{
	if (l == r)
		lca_tree[i] = lca_dfs_list[l];
	else
	{
		int m = (l + r) >> 1;
		lca_build_tree (i+i, l, m);
		lca_build_tree (i+i+1, m+1, r);
		if (lca_h[lca_tree[i+i]] < lca_h[lca_tree[i+i+1]])
			lca_tree[i] = lca_tree[i+i];
		else
			lca_tree[i] = lca_tree[i+i+1];
	}
}

void lca_prepare (const graph & g, int root)
{
	int n = (int) g.size();
	lca_h.resize (n);
	lca_dfs_list.reserve (n*2);
	lca_dfs_used.assign (n, 0);

	lca_dfs (g, root);

	int m = (int) lca_dfs_list.size();
	lca_tree.assign (lca_dfs_list.size() * 4 + 1, -1);
	lca_build_tree (1, 0, m-1);

	lca_first.assign (n, -1);
	for (int i = 0; i < m; ++i)
	{
		int v = lca_dfs_list[i];
		if (lca_first[v] == -1)
			lca_first[v] = i;
	}
}

int lca_tree_min (int i, int sl, int sr, int l, int r)
{
	if (sl == l && sr == r)
		return lca_tree[i];
	int sm = (sl + sr) >> 1;
	if (r <= sm)
		return lca_tree_min (i+i, sl, sm, l, r);
	if (l > sm)
		return lca_tree_min (i+i+1, sm+1, sr, l, r);
	int ans1 = lca_tree_min (i+i, sl, sm, l, sm);
	int ans2 = lca_tree_min (i+i+1, sm+1, sr, sm+1, r);
	return lca_h[ans1] < lca_h[ans2] ? ans1 : ans2;
}

int lca (int a, int b)
{
	int left = lca_first[a],
		right = lca_first[b];
	if (left > right) swap (left, right);
	return lca_tree_min (1, 0, (int)lca_dfs_list.size()-1, left,
	       right);
}
```

3.14 Bipartite Checking

```
vector<char> part (n, -1);
bool ok = true;
vector<int> q (n);
for (int st=0; st<n; ++st)
	if (part[st] == -1) {
		int h=0, t=0;
```

```
            q[t++] = st;
            part[st] = 0;
            while (h<t) {
                int v = q[h++];
                for (size_t i=0; i<g[v].size(); ++i) {
                    int to = g[v][i];
                    if (part[to] == -1)
                        part[to] = !part[v], q[t++]
                            = to;
                    else
                        ok &= part[to] != part[v];
                }
            }
        }

puts (ok ? "YES" : "NO");
```

4 Math

4.1 Linear Sieve

```
ll minDiv[MAXN+1];
vector<ll> primes;

void sieve(ll n){
    FOR(k, 2, n+1){
        minDiv[k] = k;
    }
    FOR(k, 2, n+1) {
        if(minDiv[k] == k) {
            primes.pb(k);
        }
        for(auto p : primes) {
            if(p > minDiv[k]) break;
            if(p*k > n) break;
            minDiv[p*k] = p;
        }
    }
}
```

4.2 Extended Euclidean Algorithm

```
// ax+by=gcd(a,b)
void solveEq(ll a, ll b, ll& x, ll& y, ll& g) {
    if(b==0) {
        x = 1;
        y = 0;
        g = a;
        return;
    }
    ll xx, yy;
    solveEq(b, a%b, xx, yy, g);
    x = yy;
    y = xx-yy*(a/b);
}
// ax+by=c
bool solveEq(ll a, ll b, ll c, ll& x, ll& y, ll& g) {
    solveEq(a, b, x, y, g);
    if(c%g != 0) return false;
    x *= c/g; y *= c/g;
    return true;
}
// Finds a solution (x, y) so that x >= 0 and x is minimal
bool solveEqNonNegX(ll a, ll b, ll c, ll& x, ll &y, ll& g) {
    if(!solveEq(a, b, c, x, y, g)) return false;
    ll k = x*g/b;
    x = x - k*b/g;
    y = y + k*a/g;
    if(x < 0) {
        x += b/g;
        y -= a/g;
    }
    return true;
}
```

All other solutions can be found like this:

$$x' = x - k\frac{b}{g}, y' = y + k\frac{a}{g}, k \in \mathbb{Z}$$

4.3 Chinese Remainder Theorem

Let's say we have some numbers m_i, which are all mutually coprime. Also, let $M = \prod_i m_i$. Then the system of congruences

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

is equivalent to $x \equiv A \pmod{M}$ and there exists a unique number A satisfying $0 \le A \le M$.

Solution for two: $x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2}$. Let $x = a_1 + km_1$. Substituting into the second congruence: $km_1 \equiv a_2 - a_1 \pmod{m_2}$. Then, $k = (m_1)^{-1}_{m_2}(a_2 - a_1) \pmod{m_2}$. and we can easily find x. This can be extended to multiple equations by solving them one-by-one.

If the moduli are not coprime, solve the system $y \equiv 0 \pmod{\frac{m_1}{g}}, y \equiv \frac{a_2-a_1}{g} \pmod{\frac{m_2}{g}}$ for y. Then let $x \equiv gy + a_1 \pmod{\frac{m_1 m_2}{g}}$.

4.4 Euler Totient Function

```
// Number of numbers x < n so that gcd(x, n) = 1
ll phi(ll n) {
    if(n == 1) return 1;
    auto f = factorize(n);
    ll res = n;
    for(auto p : f) {
        res = res - res/p.first;
    }
    return res;
}
```

4.5 Factorization With Sieve

```
// Use linear sieve to calculate minDiv
vector<pll> factorize(ll x) {
    vector<pll> res;
    ll prev = -1;
    ll cnt = 0;
    while(x != 1) {
        ll d = minDiv[x];
        if(d == prev) {
            cnt++;
        } else {
            if(prev != -1) res.pb({prev, cnt});
            prev = d;
            cnt = 1;
        }
        x /= d;
    }
    res.pb({prev, cnt});
    return res;
}
```

4.6 Modular Inverse

```
bool invWithEuclid(ll a, ll m, ll& aInv) {
    ll x, y, g;
    if(!solveEqNonNegX(a, m, 1, x, y, g)) return false;
    aInv = x;
    return true;
}
// Works only if m is prime
ll invFermat(ll a, ll m) {
    return pwr(a, m-2, m);
}
// Works only if gcd(a, m) = 1
ll invEuler(ll a, ll m) {
    return pwr(a, phi(m)-1, m);
}
```

4.7 Simpson Integration

```
const int N = 1000 * 1000; // number of steps (already
        multiplied by 2)

double simpsonIntegration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) {
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

4.8 Burnside's Lemma

Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g. Burnside's lemma asserts the following formula for the number of orbits:

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

Example. Coloring a cube with three colors.

Let X be the set of 3^6 possible face color combinations. Let's count the sizes of the fixed sets for each of the 24 rotations:

- one 0-degree rotation which leaves all 3^6 elements of X unchanged
- six 90-degree face rotations, each of which leaves 3^3 elements of X unchanged
- three 180-degree face rotation, each of which leaves 3^4 elements of X unchanged
- eight 120-degree vertex rotations, each of which leaves 3^2 elements of X unchanged
- six 180-degree edge rotations, each of which leaves 3^3 elements of X unchanged

The average is then $\frac{1}{24}(3^6 + 6\cdot 3^3 + 3\cdot 3^4 + 8\cdot 3^2 + 6\cdot 3^3) = 57$. For n colors: $\frac{1}{24}(n^6+3n^4+12n^3+8n^2)$.

Example. Coloring a circular stripe of n cells with two colors.

X is the set of all colored striped (it has 2^n elements), G is the group of rotations (n elements - by 0 cells, by 1 cell, ..., by $(n-1)$ cells). Let's fix some K and find the number of stripes that are fixed by the rotation by K cells. If a stripe becomes itself after rotation by K cells, then its 1st cell must have the same color as its $(1+K \mod n)$-th cell, which is in turn the same as its $(1+2K \mod n)$-th cell, etc., until $mK \mod n = 0$. This will happen when $m = n/gcd(K,n)$. Therefore, we have $n/gcd(K,n)$ cells that must all be of the same color. The same will happen when starting from the second cell and so on. Therefore, all cells are separated into $gcd(K,n)$ groups, with each group being of one color, and that yields $2^{gcd(K,n)}$ choices. That's why the answer to the original problem is $\frac{1}{n}\sum_{k=0}^{n-1} 2^{gcd(k,n)}$.

4.9 FFT

```
namespace FFT {
    int n;
    vector<int> r;
    vector<complex<ld>> omega;
    int logN, pwrN;

    void initLogN() {
        logN = 0;
        pwrN = 1;
        while (pwrN < n) {
            pwrN *= 2;
            logN++;
        }
        n = pwrN;
    }

    void initOmega() {
        FOR(i, 0, pwrN) {
            omega[i] = { cos(2 * i*PI / n), sin(2 * i*PI / n) };
        }
    }

    void initR() {
        r[0] = 0;
        FOR(i, 1, pwrN) {
            r[i] = r[i / 2] / 2 + ((i & 1) << (logN - 1));
        }
    }

    void initArrays() {
        r.clear();
        r.resize(pwrN);
        omega.clear();
        omega.resize(pwrN);
    }

    void init(int n) {
        FFT::n = n;
        initLogN();
        initArrays();
        initOmega();
        initR();
    }

    void fft(complex<ld> a[], complex<ld> f[]) {
        FOR(i, 0, pwrN) {
            f[i] = a[r[i]];
        }
        for (ll k = 1; k < pwrN; k *= 2) {
            for (ll i = 0; i < pwrN; i += 2 * k) {
                for (ll j = 0; j < k; j++) {
                    auto z = omega[j*n / (2 * k)] * f[i + j + k];
                    f[i + j + k] = f[i + j] - z;
                    f[i + j] += z;
                }
            }
        }
    }
}
```

4.10 FFT With Modulo

```
bool isGenerator(ll g) {
    if (pwr(g, M - 1) != 1) return false;
    for (ll i = 2; i*i <= M - 1; i++) {
        if ((M - 1) % i == 0) {
            ll q = i;
            if (isPrime(q)) {
                ll p = (M - 1) / q;
                ll pp = pwr(g, p);
                if (pp == 1) return false;
            }
            q = (M - 1) / i;
            if (isPrime(q)) {
                ll p = (M - 1) / q;
                ll pp = pwr(g, p);
                if (pp == 1) return false;
            }
        }
    }
    return true;
}

namespace FFT {
    ll n;
    vector<ll> r;
    vector<ll> omega;
    ll logN, pwrN;

    void initLogN() {
        logN = 0;
        pwrN = 1;
        while (pwrN < n) {
            pwrN *= 2;
            logN++;
        }
        n = pwrN;
    }

    void initOmega() {
        ll g = 2;
        while (!isGenerator(g)) g++;
        ll G = 1;
        g = pwr(g, (M - 1) / pwrN);
        FOR(i, 0, pwrN) {
            omega[i] = G;
            G *= g;
            G %= M;
        }
    }

    void initR() {
        r[0] = 0;
        FOR(i, 1, pwrN) {
            r[i] = r[i / 2] / 2 + ((i & 1) << (logN - 1));
        }
    }

    void initArrays() {
        r.clear();
        r.resize(pwrN);
        omega.clear();
        omega.resize(pwrN);
    }

    void init(ll n) {
        FFT::n = n;
        initLogN();
        initArrays();
        initOmega();
        initR();
    }

    void fft(ll a[], ll f[]) {
        for (ll i = 0; i < pwrN; i++) {
            f[i] = a[r[i]];
        }
        for (ll k = 1; k < pwrN; k *= 2) {
            for (ll i = 0; i < pwrN; i += 2 * k) {
                for (ll j = 0; j < k; j++) {
                    auto z = omega[j*n / (2 * k)] * f[i + j + k] %
                        M;
                    f[i + j + k] = f[i + j] - z;
                    f[i + j] += z;
                    f[i + j + k] %= M;
                    if (f[i + j + k] < 0) f[i + j + k] += M;
                    f[i + j] %= M;
                }
            }
        }
    }
}
```

4.11 Big Integer Multiplication With FFT

```
complex<ld> a[MAX_N], b[MAX_N];
complex<ld> fa[MAX_N], fb[MAX_N], fc[MAX_N];
complex<ld> cc[MAX_N];

string mul(string as, string bs) {
    int sgn1 = 1;
    int sgn2 = 1;
    if (as[0] == '-') {
        sgn1 = -1;
        as = as.substr(1);
    }
    if (bs[0] == '-') {
        sgn2 = -1;
        bs = bs.substr(1);
    }
    int n = as.length() + bs.length() + 1;
    FFT::init(n);
    FOR(i, 0, FFT::pwrN) {
        a[i] = b[i] = fa[i] = fb[i] = fc[i] = cc[i] = 0;
    }
    FOR(i, 0, as.size()) {
        a[i] = as[as.size() - 1 - i] - '0';
    }
    FOR(i, 0, bs.size()) {
        b[i] = bs[bs.size() - 1 - i] - '0';
    }
    FFT::fft(a, fa);
    FFT::fft(b, fb);
    FOR(i, 0, FFT::pwrN) {
        fc[i] = fa[i] * fb[i];
    }
    // turn [0,1,2,...,n-1] into [0, n-1, n-2, ..., 1]
    FOR(i, 1, FFT::pwrN) {
        if (i < FFT::pwrN - i) {
            swap(fc[i], fc[FFT::pwrN - i]);
        }
    }
    FFT::fft(fc, cc);
    ll carry = 0;
    vector<int> v;
    FOR(i, 0, FFT::pwrN) {
        int num = round(cc[i].real() / FFT::pwrN) + carry;
        v.pb(num % 10);
        carry = num / 10;
    }
    while (carry > 0) {
        v.pb(carry % 10);
        carry /= 10;
    }
    reverse(v.begin(), v.end());
    bool start = false;
    ostringstream ss;
    bool allZero = true;
    for (auto x : v) {
        if (x != 0) {
            allZero = false;
            break;
        }
    }
    if (sgn1*sgn2 < 0 && !allZero) ss << "-";
    for (auto x : v) {
        if (x == 0 && !start) continue;
        start = true;
        ss << abs(x);
    }
    if (!start) ss << 0;
    return ss.str();
}
```

4.12 Gaussian Elimination

```
// The last column of a is the right-hand side of the system.
// Returns 0, 1 or oo - the number of solutions.
// If at least one solution is found, it will be in ans
int gauss (vector < vector<ld> > a, vector<ld> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < eps)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                ld c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        ld sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > eps)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return oo;
    return 1;
}
```

4.13 Sprague Grundy Theorem

We have a game which fulfills the following requirements:

- There are two players who move alternately.
- The game consists of states, and the possible moves in a state do not depend on whose turn it is.
- The game ends when a player cannot make a move.
- The game surely ends sooner or later.
- The players have complete information about the states and allowed moves, and there is no randomness in the game.

Grundy Numbers. The idea is to calculate Grundy numbers for each game state. It is calculated like so: $mex(\{g_1, g_2, ..., g_n\})$, where $g_1, g_2, ..., g_n$ are the Grundy numbers of the states which are reachable from the current state. mex gives the smallest nonnegative number that is not in the set ($mex(\{0, 1, 3\}) = 2, mex(\emptyset) = 0$). If the Grundy number of a state is 0, then this state is a losing state. Otherwise it's a winning state.

Grundy's Game. Sometimes a move in a game divides the game into subgames that are independent of each other. In this case, the Grundy number of a game state is $mex(\{g_1, g_2, ..., g_n\}), g_k = a_{k,1} \oplus a_{k,2} \oplus ... \oplus a_{k,m}$ meaning that move k divides the game into m subgames whose Grundy numbers are $a_{i,j}$.

Example. We have a heap with n sticks. On each turn, the player chooses a heap and divides it into two nonempty heaps such that the heaps are of different size. The player who makes the last move wins the game. Let $g(n)$ denote the Grundy number of a heap of size n. The Grundy number can be calculated by going though all possible ways to divide the heap into two parts. E.g. $g(8) = mex(\{g(1) \oplus g(7), g(2) \oplus g(6), g(3) \oplus g(5)\})$. Base case: $g(1) = g(2) = 0$, because these are losing states.

4.14 Formulas

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$; $\sum_{i=1}^{n} i^2 = \frac{n(2n+1)(n+1)}{6}$; $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$; $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$; $\sum_{i=a}^{b} c^i = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$; $\sum_{i=1}^{n} a_1 + (i-1)d = \frac{n(a_1+a_n)}{2}$; $\sum_{i=1}^{n} a_1 r^{i-1} = \frac{a_1(1-r^n)}{1-r}, r \neq 1$; $\sum_{i=1}^{\infty} ar^{i-1} = \frac{a_1}{1-r}, |r| \leq 1$.

5 Geometry

5.1 2d Vector

```
template <typename T>
struct Vec {
    T x, y;
    Vec(): x(0), y(0) {}
    Vec(T _x, T _y): x(_x), y(_y) {}
    Vec operator+(const Vec& b) {
        return Vec<T>(x+b.x, y+b.y);
    }
    Vec operator-(const Vec& b) {
        return Vec<T>(x-b.x, y-b.y);
    }
    Vec operator*(T c) {
        return Vec(x*c, y*c);
    }
    T operator*(const Vec& b) {
        return x*b.x + y*b.y;
    }
    T operator^(const Vec& b) {
        return x*b.y-y*b.x;
    }
    bool operator<(const Vec& other) const {
        if(x == other.x) return y < other.y;
        return x < other.x;
    }
    bool operator==(const Vec& other) const {
        return x==other.x && y==other.y;
    }
    bool operator!=(const Vec& other) const {
        return !(*this == other);
    }
    friend ostream& operator<<(ostream& out, const Vec& v) {
        return out << "(" << v.x << ", " << v.y << ")";
    }
    friend istream& operator>>(istream& in, Vec<T>& v) {
        return in >> v.x >> v.y;
    }
    T norm() { // squared length
        return (*this)*(*this);
    }
    ld len() {
        return sqrt(norm());
    }
    ld angle(const Vec& other) { // angle between this and
        other vector
```

```
        return acosl((*this)*other/len()/other.len());
    }
    Vec perp() {
        return Vec(-y, x);
    }
};
/* Cross product of 3d vectors: (ay*bz-az*by, az*bx-ax*bz, ax*
     by-ay*bx)
 */
```

5.2 Line

```
template <typename T>
struct Line { // expressed as two vectors
    Vec<T> start, dir;
    Line() {}
    Line(Vec<T> a, Vec<T> b): start(a), dir(b-a) {}

    Vec<ld> intersect(Line l) {
        ld t = ld((l.start-start)^l.dir)/(dir^l.dir);
        // For segment-segment intersection this should be in
              range [0, 1]
        Vec<ld> res(start.x, start.y);
        Vec<ld> dirld(dir.x, dir.y);
        return res + dirld*t;
    }
};
```

5.3 Convex Hull Gift Wrapping

```
vector<Vec<int>> buildConvexHull(vector<Vec<int>>& pts)
      {
    int n = pts.size();
    sort(pts.begin(), pts.end());
    auto currP = pts[0]; // choose some extreme point to be on
          the hull

    vector<Vec<int>> hull;
    set<Vec<int>> used;
    hull.pb(pts[0]);
    used.insert(pts[0]);
    while(true) {
        auto candidate = pts[0]; // choose some point to be a
              candidate

        auto currDir = candidate-currP;
        vector<Vec<int>> toUpdate;
        FOR(i, 0, n) {
            if(currP == pts[i]) continue;
            // currently we have currP->candidate
            // we need to find point to the left of this
            auto possibleNext = pts[i];
            auto nextDir = possibleNext - currP;
            auto cross = currDir ^ nextDir;
            if(candidate == currP || cross > 0) {
                candidate = possibleNext;
                currDir = nextDir;
            } else if(cross == 0 && nextDir.norm() > currDir.
                  norm()) {
                candidate = possibleNext;
                currDir = nextDir;
            }
        }
        if(used.find(candidate) != used.end()) break;
        hull.pb(candidate);
        used.insert(candidate);
        currP = candidate;
    }
    return hull;
}
```

5.4 Convex Hull With Graham's Scan

```
// Takes in >= 3 points
// Returns convex hull in clockwise order
// Ignores points on the border
vector<Vec<int>> buildConvexHull(vector<Vec<int>> pts) {
    if(pts.size() <= 3) return pts;
    sort(pts.begin(), pts.end());
    stack<Vec<int>> hull;
    hull.push(pts[0]);
    auto p = pts[0];
    sort(pts.begin()+1, pts.end(), [&](Vec<int> a, Vec<int> b)
          -> bool {
        // p->a->b is a ccw turn
        int turn = sgn((a-p)^(b-a));
        //if(turn == 0) return (a-p).norm() > (b-p).norm();
        // ^ among collinear points, take the farthest one
        return turn == 1;
    });
    hull.push(pts[1]);
    FOR(i, 2, (int)pts.size()) {
        auto c = pts[i];
        if(c == hull.top()) continue;
        while(true) {
            auto a = hull.top(); hull.pop();
            auto b = hull.top();
            auto ba = a-b;
            auto ac = c-a;
            if((ba^ac) > 0) {
                hull.push(a);
                break;
            } else if((ba^ac) == 0) {
                if(ba*ac < 0) c = a;
                // ^ c is between b and a, so it shouldn't be
                      added to the hull
                break;
            }
        }
        hull.push(c);
    }
    vector<Vec<int>> hullPts;
    while(!hull.empty()) {
        hullPts.pb(hull.top());
        hull.pop();
    }
    return hullPts;
}
```

5.5 Circle Line Intersection

```
double r, a, b, c; // ax+by+c=0, radius is at (0, 0)
// If the center is not at (0, 0), fix the constant c to translate
      everything so that center is at (0, 0)
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
if (c*c > r*r*(a*a+b*b)+eps)
    puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < eps) {
    puts ("1 point");
    cout << x0 << ' ' << y0 << '\n';
}
else {
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << ' ' << by
          << '\n';
}
```

5.6 Circle Circle Intersection

Let's say that the first circle is centered at $(0,0)$ (if it's not, we can move the origin to the center of the first circle and adjust the coordinates), and the second one is at (x_2, y_2). Then, let's construct a line $Ax + By + C = 0$, where $A = -2x_2, B = -2y_2, C = x_2^2 + y_2^2 + r_1^2 - r_2^2$. Finding the intersection between this line and the first circle will give us the answer.

The only tricky case: if both circles are centered at the same point. We handle this case separately.

5.7 Common Tangents To Two Circles

```
struct pt {
    double x, y;

    pt operator- (pt p) {
        pt res = { x-p.x, y-p.y };
        return res;
    }
};
struct circle : pt {
    double r;
};
struct line {
    double a, b, c;
};
void tangents (pt c, double r1, double r2, vector<line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -eps) return;
    d = sqrt (abs (d));
    line l;
    l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z;
    l.c = r1;
    ans.push_back (l);
}
vector<line> tangents (circle a, circle b) {
    vector<line> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b-a, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    return ans;
}
```

5.8 Number Of Lattice Points On Segment

Let's say we have a line segment from (x_1, y_1) to (x_2, y_2). Then, the number of lattice points on this segment is given by

$$gcd(x_2 - x_1, y_2 - y_1) + 1.$$

5.9 Pick's Theorem

We are given a lattice polygon with non-zero area. Let's denote its area by S, the number of points with integer coordinates lying strictly inside the polygon by I and the number of points lying on the sides of the polygon by B. Then:

$$S = I + \frac{B}{2} - 1.$$

5.10 Usage Of Complex

```
typedef long long C; // could be long double
typedef complex<C> P; // represents a point or vector
#define X real()
#define Y imag()
...
P p = {4, 2}; // p.X = 4, p.Y = 2
P u = {3, 1};
P v = {2, 2};
P s = v+u; // {5, 3}
P a = {4, 2};
P b = {3, -1};
auto l = abs(b-a); // 3.16228
auto plr = polar(1.0, 0.5); // construct a vector of length 1 and
    angle 0.5 radians
v = {2, 2};
auto alpha = arg(v); // 0.463648
v *= plr; // rotates v by 0.5 radians counterclockwise. The
    length of plt must be 1 to rotate correctly.
auto beta = arg(v); // 0.963648
a = {4, 2};
b = {1, 2};
C p = (conj(a)*b).Y; // 6 <- the cross product of a and b
```

5.11 Misc

Distance from point to line.

We have a line $l(t) = \vec{a} + \vec{b}t$ and a point $\vec{p}$. The distance from this point to the line can be calculated by expressing the area of a triangle in two different ways. The final formula: $d = \frac{(\vec{p}-\vec{a})\times(\vec{p}-\vec{b})}{|\vec{b}-\vec{a}|}$

Point in polygon.

Send a ray (half-infinite line) from the points to an arbitrary direction and calculate the number of times it touches the boundary of the polygon. If the number is odd, the point is inside the polygon, otherwise it's outside.

Using cross product to test rotation direction.

Let's say we have vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$. Let's define $\vec{ab} = b - a, \vec{bc} = c - b$ and $s = sgn(\vec{ab} \times \vec{bc})$. If $s = 0$, the three points are collinear. If $s = 1$, then $\vec{bc}$ turns in the counterclockwise direction compared to the direction of $\vec{ab}$. Otherwise it turns in the clockwise direction.

Line segment intersection.

The problem: to check if line segments ab and cd intersect. There are three cases:

1. **The line segments are on the same line.** Use cross products and check if they're zero - this will tell if all points are on the same line. If so, sort the points and check if their intersection is non-empty. If it is non-empty, there are an infinite number of intersection points.
2. **The line segments have a common vertex.** Four possibilities: $a = c, a = d, b = c, b = d$.
3. **There is exactly one intersection point that is not an endpoint.** Use cross product

to check if points c and d are on different sides of the line going through a and b and if the points a and b are on different sides of the line going through c and d.

Angle between vectors.

$arccos(\frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\cdot|\vec{b}|})$.

Dot product properties.

If the dot product of two vectors is zero, the vectors are orthogonal. If it is positive, the angle is acute. Otherwise it is obtuse.

Lines with line equation.

Any line can be described by an equation $ax + by + c = 0$.

- Construct a line using two points A and B:
  1. Take vector from A to B and rotate it 90 degrees ($(x, y) \to (-y, x)$). This will be (a, b).
  2. Normalize this vector. Then put A (or B) into the equation and solve for c.
- Distance from point to line: put point coordinates into line equation and take absolute value. If (a, b) is not normalized, you still need to divide by $\sqrt{a^2 + b^2}$.
- Distance between two parallel lines: $|c_1 - c_2|$ (if they are not normalized, you still need to divide by $\sqrt{a^2 + b^2}$).
- Project a point onto a line: compute signed distance d between line L and point P. Answer is $P - d\vec{(a, b)}$.
- Build a line parallel to a given one and passing through a given point: compute the signed distance d between line and point. Answer is $ax + by + (c - d) = 0$.
- Intersect two lines: $d = a_1b_2 - a_2b_1, x = \frac{c_2b_1 - c_1b_2}{d}, y = \frac{c_1a_2 - c_2a_1}{d}$. If $abs(d) < \epsilon$, then the lines are parallel.

Half-planes.

Definition: define as line, assume a point (x, y) belongs to half plane iff $ax + by + c \geq 0$.

Intersecting with a convex polygon:

1. Start at any point and move along the polygon's traversal.
2. Alternate points and segments between consecutive points.
3. If point belongs to half-plane, add it to the answer.
4. If segment intersects the half-plane's line, add it to the answer.

Some more techniques.

- Check if point A lies on segment BC:
  1. Compute point-line distance and check if it is 0 (abs less than ϵ).
  2. $\vec{BA}\cdot\vec{BC} \geq 0$ and $\vec{CA}\cdot\vec{CB} \geq 0$.
- Compute distance between line segment and point: project point onto line formed by the segment. If this point is on the segment, then the distance between it and original point is the answer. Otherwise, take minimum of distance between point and segment endpoints.

6 Strings

6.1 Hashing

```
struct HashedString {
        // 630911, 933494437)
    const ll A1 = 999999929, B1 = 1000000009, A2 =
          1000000087, B2 = 1000000097;
    vector<ll> A1pwrs, A2pwrs;
    vector<pll> prefixHash;
    HashedString(const string& _s) {
        init(_s);
        calcHashes(_s);
    }
    void init(const string& s) {
        ll a1 = 1;
        ll a2 = 1;
        FOR(i, 0, (int)s.length()+1) {
            A1pwrs.pb(a1);
            A2pwrs.pb(a2);
            a1 = (a1*A1)%B1;
            a2 = (a2*A2)%B2;
        }
    }
    void calcHashes(const string& s) {
        pll h = {0, 0};
        prefixHash.pb(h);
        for(char c : s) {
            ll h1 = (prefixHash.back().first*A1 + c)%B1;
            ll h2 = (prefixHash.back().second*A2 + c)%B2;
            prefixHash.pb({h1, h2});
        }
    }
    pll getHash(int l, int r) {
        ll h1 = (prefixHash[r+1].first - prefixHash[l].first*A1pwrs
              [r+1-l]) % B1;
        ll h2 = (prefixHash[r+1].second - prefixHash[l].second*
              A2pwrs[r+1-l]) % B2;
        if(h1 < 0) h1 += B1;
        if(h2 < 0) h2 += B2;
        return {h1, h2};
    }
};
```

6.2 Prefix Function

```
// pi[i] is the length of the longest proper prefix of the substring
      s[0..i] which is also a suffix
// of this substring
vector<int> prefixFunction(const string& s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

6.3 Prefix Function Automaton

```
// aut[oldPi][c] = newPi
vector<vector<int>> computeAutomaton(string s) {
    const char BASE = 'a';
    s += "#";
    int n = s.size();
    vector<int> pi = prefixFunction(s);
    vector<vector<int>> aut(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && BASE + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + (BASE + c == s[i]);
        }
    }
    return aut;
}
vector<int> findOccurs(const string& s, const string& t) {
    auto aut = computeAutomaton(s);
    int curr = 0;
    vector<int> occurs;
    FOR(i, 0, (int)t.length()) {
        int c = t[i]-'a';
        curr = aut[curr][c];
        if(curr == (int)s.length()) {
            occurs.pb(i-s.length()+1);
        }
    }
    return occurs;
}
```

6.4 KMP

```
// Knuth-Morris-Pratt algorithm
vector<int> findOccurences(const string& s, const string& t) {
    int n = s.length();
    int m = t.length();
    string S = s + "#" + t;
    auto pi = prefixFunction(S);
    vector<int> ans;
    FOR(i, n+1, n+m+1) {
        if(pi[i] == n) {
            ans.pb(i-2*n);
        }
    }
    return ans;
}
```

6.5 Aho Corasick Automaton

```
// alphabet size
const int K = 70;

// the indices of each letter of the alphabet
int intVal[256];
void init() {
    int curr = 2;
    intVal[1] = 1;
    for(char c = '0'; c <= '9'; c++, curr++) intVal[(int)c] =
            curr;
    for(char c = 'A'; c <= 'Z'; c++, curr++) intVal[(int)c] =
            curr;
    for(char c = 'a'; c <= 'z'; c++, curr++) intVal[(int)c] =
            curr;
}

struct Vertex {
    int next[K];
    vector<int> marks;
    // ^ this can be changed to int mark = -1, if there will be
            no duplicates
    int p = -1;
    char pch;
    int link = -1;
    int exitLink = -1;
    // ^ exitLink points to the next node on the path of suffix
            links which is marked
    int go[K];

    // ch has to be some small char
    Vertex(int __p=-1, char ch=(char)1) : p(__p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void addString(string const& s, int id) {
    int v = 0;
    for (char ch : s) {
        int c = intVal[(int)ch];
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].marks.pb(id);
}

int go(int v, char ch);

int getLink(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(getLink(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int getExitLink(int v) {
    if(t[v].exitLink != -1) return t[v].exitLink;
    int l = getLink(v);
    if(l == 0) return t[v].exitLink = 0;
    if(!t[l].marks.empty()) return t[v].exitLink = l;
    return t[v].exitLink = getExitLink(l);
}

int go(int v, char ch) {
    int c = intVal[(int)ch];
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(getLink(v), ch);
    }
    return t[v].go[c];
}

void walkUp(int v, vector<int>& matches) {
    if(v == 0) return;
    if(!t[v].marks.empty()) {
        for(auto m : t[v].marks) matches.pb(m);
    }
    walkUp(getExitLink(v), matches);
}

// returns the IDs of matched strings.
// Will contain duplicates if multiple matches of the same string
        are found.
vector<int> walk(const string& s) {
    vector<int> matches;
    int curr = 0;
    for(char c : s) {
        curr = go(curr, c);
        if(!t[curr].marks.empty()) {
            for(auto m : t[curr].marks) matches.pb(m);
        }
        walkUp(getExitLink(curr), matches);
    }
    return matches;
}
/* Usage:
 * addString(strs[i], i);
 * auto matches = walk(text);
 * .. do what you need with the matches - count, check if some
        id exists, etc ..
 * Some applications:
 * - Find all matches: just use the walk function
```

```
 * - Find lexicographically smallest string of a given length that
        doesn't match any of the given strings:
 * For each node, check if it produces any matches (it either
       contains some marks or walkUp(v) returns some marks).
 * Remove all nodes which produce at least one match. Do DFS
        in the remaining graph, since none of the remaining
       nodes
 * will ever produce a match and so they're safe.
 * - Find shortest string containing all given strings:
 * For each vertex store a mask that denotes the strings which
       match at this state. Start at (v = root, mask = 0),
 * we need to reach a state (v, mask=2^n-1), where n is the
       number of strings in the set. Use BFS to transition
       between states
 * and update the mask.
 */
```

6.6 Suffix Array

```
vector<int> sortCyclicShifts(string const& s) {
    int n = s.size();
    const int alphabet = 256; // we assume to use the whole
         ASCII range
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n
                 ]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h))
                  % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
vector<int> constructSuffixArray(string s) {
    s += "$"; // <- this must be smaller than any character in
         s
    vector<int> sorted_shifts = sortCyclicShifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

6.7 Z Func

```
vi Z(string S) {
        vi z(sz(S));
        int l = -1, r = -1;
        rep(i,1,sz(S)) {
                z[i] = i >= r ? 0 : min(r - i, z[i - l]);
                while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
                        z[i]++;
                if (i + z[i] > r)
                        l = i, r = i + z[i];
        }
        return z;
}
```

7 Dynamic Programming

7.1 Convex Hull Trick

```
/*
Let's say we have a relation:
dp[i] = min(dp[j] + h[j+1]*w[i]) for j<=i
Let's set k_j = h[j+1], x = w[i], b_j = dp[j]. We get:
dp[i] = min(b_j+k_j*x) for j<=i.
This is the same as finding a minimum point on a set of lines.
After calculating the value, we add a new line with
k_i = h[i+1] and b_i = dp[i].
*/
struct Line {
    int k;
    int b;

    int eval(int x) {
        return k*x+b;
    }

    int intX(Line& other) {
        int x = b-other.b;
        int y = other.k-k;
        int res = x/y;
        if(x%y != 0) res++;
        return res;
    }
};

struct BagOfLines {
    vector<pair<Line, int>> lines;

    void addLine(int k, int b) {
        Line current = {k, b};
        if(lines.empty()) {
            lines.pb({current, -OO});
            return;
        }
        int x = -OO;
        while(true) {
            auto line = lines.back().first;
            int from = lines.back().second;
            x = line.intX(current);
            if(x > from) break;
            lines.pop_back();
        }
        lines.pb({current, x});
    }

    int findMin(int x) {
        int lo = 0, hi = (int)lines.size()-1;
        while(lo < hi) {
            int mid = (lo+hi+1)/2;
            if(lines[mid].second <= x) {
                lo = mid;
            } else {
                hi = mid-1;
            }
        }
        return lines[lo].first.eval(x);
    }
};
```

7.2 Divide And Conquer

```
/*
Let A[i][j] be the optimal answer for using i objects to satisfy j
     first
```

```
requirements.
The recurrence is:
A[i][j] = min(A[i-1][k] + f(i, j, k)) where f is some function that
        denotes the
cost of satisfying requirements from k+1 to j using the i-th
        object.
Consider the recursive function calc(i, jmin, jmax, kmin, kmax),
        that calculates
all A[i][j] for all j in [jmin, jmax] and a given i using known A[i
        -1][*].
*/

void calc(int i, int jmin, int jmax, int kmin, int kmax) {
    if(jmin > jmax) return;
    int jmid = (jmin+jmax)/2;
    // calculate A[i][jmid] naively (for k in kmin...min(jmid,
        kmax){...})
    // let kmid be the optimal k in [kmin, kmax]
    calc(i, jmin, jmid-1, kmin, kmid);
    calc(i, jmid+1, jmax, kmid, kmax);
}

int main() {
    // set initial dp values
    FOR(i, start, k+1){
        calc(i, 0, n-1, 0, n-1);
    }
    cout << dp[k][n-1];
}
```

7.3 Optimizations

1. Convex Hull 1:
   - Recurrence: $dp[i] = \min_{j<i}\{dp[j] + b[j]\cdot a[i]\}$
   - Condition: $b[j] \geq b[j+1], a[i] \leq a[i+1]$
   - Complexity: $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n)$
2. Convex Hull 2:
   - Recurrence: $dp[i][j] = \min_{k<j}\{dp[i-1][k] + b[k]\cdot a[j]\}$
   - Condition: $b[k] \geq b[k+1], a[j] \leq a[j+1]$
   - Complexity: $\mathcal{O}(kn^2) \rightarrow \mathcal{O}(kn)$
3. Divide and Conquer:
   - Recurrence: $dp[i][j] = \min_{k<j}\{dp[i-1][k] + C[k][j]\}$
   - Condition: $A[i][j] \leq A[i][j+1]$
   - Complexity: $\mathcal{O}(kn^2) \rightarrow \mathcal{O}(kn\log(n))$
4. Knuth:
   - Recurrence: $dp[i][j] = \min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$
   - Condition: $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
   - Complexity: $\mathcal{O}(n^3) \rightarrow \mathcal{O}(n^2)$

Notes:

- $A[i][j]$ - the smallest k that gives the optimal answer
- $C[i][j]$ - some given cost function

8 Misc

8.1 Mo's Algorithm

Mo's algorithm processes a set of range queries on a static array. Each query is to calculate something base on the array values in a range $[a, b]$. The queries have to be known in advance. Let's divide the array into blocks of size $k = O(\sqrt{n})$. A query $[a_1, b_1]$ is processed before query $[a_2, b_2]$ if $\lfloor \frac{a_1}{k} \rfloor < \lfloor \frac{a_2}{k} \rfloor$ or $\lfloor \frac{a_1}{k} \rfloor = \lfloor \frac{a_2}{k} \rfloor$ and $b_1 < b_2$.

Example problem: counting number of distinct values in a range. We can process the queries in the described order and keep an array count, which knows how many times a certain value has appeared. When moving the boundaries back and forth, we either increase count$[x_i]$ or decrease it. According to value of it, we will know how the number of distinct values has changed (e.g. if count$[x_i]$ has just become 1, then we add 1 to the answer, etc.).

8.2 Ternary Search

```
double ternary_search(double l, double r) {
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l); //return the maximum of f(x) in [l, r]
}
```

8.3 Big Integer

```
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;
    int size() {
        if (a.empty()) return 0;
        int ans = (a.size() - 1) * base_digits;
        int ca = a.back();
        while (ca) ans++, ca /= 10;
        return ans;
    }
    bigint operator^(const bigint &v) {
        bigint ans = 1, x = *this, y = v;
        while (!y.isZero()) {
            if (y % 2) ans *= x;
            x *= x, y /= 2;
        }
        return ans;
    }
    string to_string() {
        stringstream ss;
        ss << *this;
        string s;
        ss >> s;
        return s;
    }
    int sumof() {
        string s = to_string();
        int ans = 0;
        for (auto c : s) ans += c - '0';
        return ans;
    }
    bigint() : sign(1) {}
    bigint(long long v) {
        *this = v;
    }
    bigint(const string &s) {
        read(s);
    }
    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }
    void operator=(long long v) {
        sign = 1;
```

```
    a.clear();
    if (v < 0)
        sign = -1, v = -v;
    for (; v > 0; v = v / base)
        a.push_back(v % base);
}
bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;
        for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.
                size()) || carry; ++i) {
            if (i == (int)res.a.size()) res.a.push_back(0);
            res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
            carry = res.a[i] >= base;
            if (carry) res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}
bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int)v.a.size() ||
                    carry; ++i) {
                res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] :
                        0);
                carry = res.a[i] < 0;
                if (carry) res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}
void operator*=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
    }
    trim();
}
bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}
void operator*=(long long v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
    }
    trim();
}
bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
}
friend pair<bigint, bigint> divmod(const bigint &a1, const
        bigint &b1) {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());
    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size()
            - 1];
        int d = ((long long)base * s1 + s2) / b.a.back();
        r -= b * d;
        while (r < 0) r += b, --d;
        q.a[i] = d;
    }
    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}
bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
}
bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
}
void operator/=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long)base;
        a[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
}
bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}
int operator%(int v) const {
    if (v < 0) v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long)base) % v;
    return m * sign;
}
void operator+=(const bigint &v) {
    *this = *this + v;
}
void operator-=(const bigint &v) {
    *this = *this - v;
}
void operator*=(const bigint &v) {
    *this = *this * v;
}
void operator/=(const bigint &v) {
    *this = *this / v;
}
bool operator<(const bigint &v) const {
    if (sign != v.sign) return sign < v.sign;
    if (a.size() != v.a.size())
        return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[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 (!a.empty() && !a.back()) a.pop_back();
    if (a.empty()) sign = 1;
}
bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
}
bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
}
bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}
long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--) res = res * base + a[i
        ];
```

```
    return res * sign;
}
friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}
void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int)s.size() && (s[pos] == '-' || s[pos] ==
            '+')) {
        if (s[pos] == '-') sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i; j
            ++)
            x = x * 10 + s[j] - '0';
        a.push_back(x);
    }
    trim();
}
friend istream &operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}
friend ostream &operator<<(ostream &stream, const bigint
        &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int)v.a.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') << v.a[i
            ];
    return stream;
}
static vector<int> convert_base(const vector<int> &a, int
        old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back()) res.pop_back();
    return res;
}
typedef vector<long long> vll;
static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    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);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

    for (int i = 0; i < k; i++) a2[i] += a1[i];
    for (int i = 0; i < k; i++) b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];
    for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
    for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i
        ];
    for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] +=
        a2b2[i];
    return res;
}
bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll x(a6.begin(), a6.end());
    vll y(b6.begin(), b6.end());
    while (x.size() < y.size()) x.push_back(0);
    while (y.size() < x.size()) y.push_back(0);
    while (x.size() & (x.size() - 1)) x.push_back(0), y.
        push_back(0);
    vll c = karatsubaMultiply(x, y);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int)c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int)(cur % 1000000));
        carry = (int)(cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
};
```

8.4 Binary Exponentiation

```
ll pwr(ll a, ll b, ll m) {
    if(a == 1) return 1;
    if(b == 0) return 1;
    a %= m;
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```

8.5 Tortoise Hare

```
int64 a, b, c;

int64 f(int64 x)
{
        return (a*x + x%b) % c;
}

int main()
{
        cin >> a >> b >> c;

        int64 x0 = 1;
        int64 tortoise = f(x0);
        int64 hare = f(f(x0));

        for (int64 i = 0; i < 2e7; ++i)
        {
                if (tortoise == hare)
                {
                        // At this point the tortoise position,  ,
                            which is also equal
                        // to the distance between hare and
                            tortoise, is divisible by
                        // the period  . So hare moving in circle
                            one step at a time,
                        // and tortoise (reset to x0) moving
                            towards the circle, will
                        // intersect at the beginning of the circle.
                            Because the
                        // distance between them is constant at  2,
                             a multiple of  ,
```

```
                // they will agree as soon as the tortoise
                    reaches index  .

                // Find the position  of first repetition.
                int64 len = 0;
                tortoise = x0;

                while (tortoise != hare)
                {
                    tortoise = f(tortoise);
                    hare = f(hare);
                    len++;
                }

                // Find the length of the shortest cycle
                    starting from  x_
                // The hare moves one step at a time
                    while tortoise is still.
                // lam is incremented until  is found.
                int64 lam = 1;
                hare = f(tortoise);
                while (tortoise != hare)
                {
                    hare = f(hare);
                    lam += 1;
                }


                alert(lam+len);
            }
            else
            {
                // Main phase of algorithm: finding a
                    repetition x_i = x_2i.
                // The hare moves twice as quickly as the
                    tortoise and
                // the distance between them increases by
                    1 at each step.
                // Eventually they will both be inside the
                    cycle and then,
                // at some point, the distance between
                    them will be
                // divisible by the period  .
                tortoise = f(tortoise);
                hare = f(f(hare));
            }
        }

        cout << -1 << endl;

        return 0;
}
```

8.6 Builtin GCC Stuff

- __builtin_clz(x): the number of zeros at the beginning of the bit representation.
- __builtin_ctz(x): the number of zeros at the end of the bit representation.
- __builtin_popcount(x): the number of ones in the bit representation.
- __builtin_parity(x): the parity of the number of ones in the bit representation.
- __gcd(x, y): the greatest common divisor of two numbers.
- __int128_t: the 128-bit integer type. Does not support input/output.

9 Kactl

9.1 Math

Equations

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

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

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

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

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

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

Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

Trigonometry

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

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

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

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

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

Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two): $s_a =$

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

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

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

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

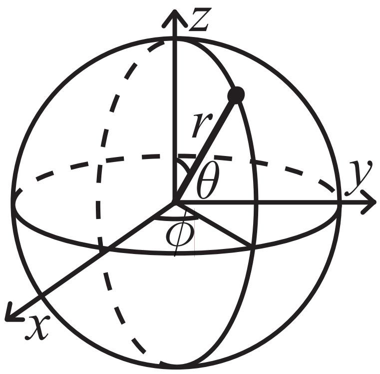
Quadrilaterals

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

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

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

Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \mathrm{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \mathrm{atan2}(y, x)$$

Derivatives/Integrals

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

Integration by parts:

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

Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

9.2 Combinatorial

Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

Cycles

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

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

Derangements

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

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

Burnside's lemma

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

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

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

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

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

Partition function

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

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

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

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

Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able). $B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

Eulerian numbers

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

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

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

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

Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

Bell numbers

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

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

Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

Catalan numbers

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

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

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

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