Seoul National University  
Team: MolaMola

Coach : Prof. Heon Young Yeom  
Contestant : Sangsoo Park, Sunggwan Park, Bumsoo Park

# Table of Contents

Table of Contents 1

Algorithm Checklist 2

Java Implementation Details 2

IO and BigInteger Usage Example 2

Find k-th element in set Example 3

Data Structures 3

Splay Tree 3

Link Cut Tree 4

Graph Algorithms 6

Dominator tree 6

Arborescence 6

Maximum Matching 7

Dinic’s Algorithm 8

Vizing’s Algorithm 9

Hungarian Method 9

Maximum clique 10

Stoer Wagner’s Mincut Algorithm 10

Mathematical Stuff 11

Kirchhoff’s Theorem 11

LP Duality 11

Simplex Algorithm 11

Polynomial multimplcation 12

Polynomial multimplcation with special modulo 13

Geometry 14

Delaunay triangulation 14

General Geometry Library Header 14

Tangent series 15

Intersect series 16

Distant series 16

Make circle series 17

3D Header 17

3D Intersect series 18

Convex Hull 18

Intersect area series 18

Smallest enclosing circle 19

3D Convex Hull 19

Dynamic convex hull 20

Halfplane Intersection 21

Polygon raycast 22

Miscellaneous 23

Newton Method 23

Root Finder using Bezier Curve 23

Pollard rho 23

Gauss Quadrature Table 24

# Algorithm Checklist

뒤집어서 생각하기

Greedy

DP – Knuth, Convex hull trick, Divide & Conquer Optimization

Binary Search (+with extra weight)

Divide & Conquer

Centroid Decomposition, Tree Diameter

Un/Unidirected Spanning tree

Linear Programming / Duality / Matroid

Hashing, KMP, Suffix Array, Manacher, Z-Algorithm, eertree

Min cut - Max flow, MCMF, Bipartite matching

Min cut – Resonable한 하한들로 고찰하기

a = b: a만 움직이기, b만 움직이기, 두 개 동시에 움직이기

답의 상한이 Resonable하게 작은가?

문제가 안풀릴 때 “당연하다고 생각한 것”을 의심하기.

말도 안되는 것을 한 번은 생각해보기.

Random Algorithm

LIS, LCS, Based on DP

Q Sqrt(N), ..

HLD, BCC, SCC

Double 안쓰기 (싫다아..)

Set, Map 쓰기 전에 생각하셨나요?

사막여우 껴안기

# Java Implementation Details

## IO and BigInteger Usage Example

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.Arrays;

import java.io.BufferedReader;

import java.io.OutputStream;

import java.io.PrintWriter;

import java.util.StringTokenizer;

import java.io.InputStream;

import java.math.BigInteger;

public class fileName {

public static void main(String[] args)

{ InputStream inputStream = System.in;

OutputStream outputStream = System.out;

InputReader in = new InputReader(inputStream);

PrintWriter out = new PrintWriter(outputStream);

TaskF solver = new taskName();

solver.solve(1, in, out);

out.close();

}

}

class taskName {

public void solve(int testNumber, InputReader in, PrintWriter out) {

int N = in.nextInt();

N++;

BigInteger ans = BigInteger.ONE;

for(int i = 2\*N, j = 1; i > N; i--, j++ ){

ans = ans.multiply( BigInteger.valueOf(i) );

ans = ans.divide( BigInteger.valueOf(j) );

}

ans = ans.divide( BigInteger.valueOf(N+1) );

out.println( ans );

}

}

class InputReader {

private final BufferedReader reader;

private StringTokenizer tokenizer;

public InputReader(InputStream stream) {

reader = new BufferedReader(new InputStreamReader(stream));

tokenizer = null;

}

public String nextLine() {

try {

return reader.readLine();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

public String next() {

while (tokenizer == null || !tokenizer.hasMoreTokens()) {

tokenizer = new StringTokenizer(nextLine());

}

return tokenizer.nextToken();

}

public int nextInt() {

return Integer.parseInt(next());

}

public double nextDouble() {

return Double.parseDouble(next());

}

}

## Find k-th element in set Example

#include<bits/stdc++.h>

#include <ext/pb\_ds/detail/standard\_policies.hpp>

#include <ext/pb\_ds/assoc\_container.hpp> // Common file

#include <ext/pb\_ds/tree\_policy.hpp> // Including tree\_order\_statistics\_node\_update

using namespace \_\_gnu\_pbds;

using namespace std;

typedef tree<

int,

null\_type,

less<int>,

rb\_tree\_tag,

tree\_order\_statistics\_node\_update>

ordered\_set;

int main()

{

    ordered\_set X;

    X.insert(1);

    X.insert(2);

    X.insert(4);

    X.insert(8);

    X.insert(16);

    cout<<\*X.find\_by\_order(1)<<endl; // 2

    cout<<\*X.find\_by\_order(2)<<endl; // 4

    cout<<\*X.find\_by\_order(4)<<endl; // 16

    cout<<(end(X)==X.find\_by\_order(6))<<endl; // true

    cout<<X.order\_of\_key(-5)<<endl;  // 0

    cout<<X.order\_of\_key(1)<<endl;   // 0

    cout<<X.order\_of\_key(3)<<endl;   // 2

    cout<<X.order\_of\_key(4)<<endl;   // 2

    cout<<X.order\_of\_key(400)<<endl; // 5

}

# Data Structures

## Splay Tree

const int N\_ = 2e5;

const int inf = ~0u>>1;

struct node{

inline void pushdown()

{

if( rev ){

if( link[0] ) link[0]->rev ^= 1;

if( link[1] ) link[1]->rev ^= 1;

swap( link[0], link[1] );

rev = 0;

}

if( add ){

if( link[0] ) link[0]->add += add, link[0]->mn += add, link[0]->val += add;

if( link[1] ) link[1]->add += add, link[1]->mn += add, link[1]->val += add;

add = 0;

}

}

inline void pushup()

{

cnt = (link[0]? link[0]->cnt:0) + (link[1]? link[1]->cnt:0) + 1;

mn = min( val, min(link[0]?link[0]->mn:inf, link[1]?link[1]->mn:inf));

}

int cnt, add, mn, val; //cnt: number of nodes

bool rev;

node \*link[2], \*par;

};

struct splaytree{

node N[ N\_ ];

node\* root;

int sz;

node\* operator[](int idx){ return N + idx; }

void clear(int s){

sz = 0;

for(int i=0;i<=s+2;i++){

N[i].link[0] = N[i].link[1] = N[i].par = 0, N[i].cnt = 1;

N[i].rev = false;

}

// dummy nodes can remove many null-pointer exceptions

root = N+s+1; root->cnt = 2;

N[s+2].par = N+s+1; N[s+1].link[1] = N+s+2;

}

inline int dir(node \*x){ return x->par->link[0] != x; }

inline int cnt(node\* p){ return p? p->cnt: 0; }

void rotate(node \*n) // To

{

n->par->pushdown(); n->pushdown();

node \*p = n->par;

int d = dir(n);

p->link[d] = n->link[!d]; if( n->link[!d] ) n->link[!d]->par = p;

n->par = p->par; if( p->par ) p->par->link[ dir(p) ] = n;

n->link[!d] = p; p->par = n;

p->pushup(); n->pushup();

}

void splay(node \*x, node \*f){

if( x == f ) return;

while(x->par != f){

x->par->pushdown();

if( x->par->par == f );

else if(dir(x) == dir(x->par)) rotate(x->par);

else rotate(x);

rotate(x);

}

x->pushdown();

if( f == NULL ) root = x;

}

// 1-index if dummy node exists

node\* kth\_splay(int k,node\* f)

{

node \*x = root;

x->pushdown();

while( cnt( x->link[0] ) != k ){

if( cnt( x->link[0] ) < k ){

if( !x->link[1] ) return x;

k -= cnt(x->link[0]) + 1, x = x->link[1];

}

else x = x->link[0];

x->pushdown();

}

splay( x, f );

return x;

}

// 1-index if dummy nodes exist

// recommend: don’t copy & paste code below.

**// be careful if dummy nodes don’t exist (ex. null-pointer exception)**

void insert(int wi, node \*n)

{

if( !root ){

root = n;

return;

}

kth\_splay(wi-1, 0);

kth\_splay(wi, root);

root->link[1]->link[0] = n; n->par = root->link[1];

root->link[1]->pushup(); root->pushup();

}

void Delete(int x){

kth\_splay(x-1,0);

kth\_splay(x+1,root);

root->link[1]->link[0] = NULL;

root->link[1]->pushup(); root->pushup();

}

void Reverse(int x,int y){

if( x > y ) return;

kth\_splay(x-1,0);

kth\_splay(y+1,root);

root->link[1]->link[0]->rev ^= 1;

}

void revolve(int x,int y,int T){ // rotate x~y T times

if( x >= y ) return;

int l = (y-x+1);

T = (T%l+l) % l;

Reverse(x,y-T);

Reverse(y-T+1,y);

Reverse(x,y);

}

int node\_address(int wi)

{

node \*p = N+wi;

splay(p, 0);

return cnt( p->link[0] );

}

int min(int x,int y){

kth\_splay(x-1,0);

kth\_splay(y+1,root);

return root->link[0]->link[1]->mn;

}

} pre, post;

## Link Cut Tree

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<algorithm>

#include<stdio.h>

using namespace std;

const int N\_ = 2e5;

struct node{

void pushup(){

cnt = (link[0]? link[0]->cnt:0) + (link[1]? link[1]->cnt:0) + 1;

mx = max( max( link[0]? link[0]->mx:0, link[1]? link[1]->mx:0 ), val);

}

int cnt, val, mx; //cnt: number of nodes

node \*link[2], \*par, \*path\_parent;

};

struct linkcuttree{

node N[ N\_ ];

void clear(int s){

for(int i=0;i<=s;i++)

N[i].link[0] = N[i].link[1] = N[i].par = N[i].path\_parent = 0, N[i].cnt = 1;

}

inline int dir(node \*x){ return x->par->link[0] != x; }

inline int cnt(node \*x){ return x?x->cnt:0; }

inline int mx(node \*x){ return x?x->mx:0; }

void rotate(node \*n) // To

{

if( !n->par ) return;

node \*p = n->par;

int d = dir(n);

n->path\_parent = p->path\_parent; p->path\_parent = NULL;

p->link[d] = n->link[!d]; if( n->link[!d] ) n->link[!d]->par = p;

n->par = p->par; if( p->par ) p->par->link[ dir(p) ] = n;

n->link[!d] = p; p->par = n;

p->pushup(); n->pushup();

}

void splay(node \*x){

while( x->par ){

if( !x->par->par );

else if(dir(x) == dir(x->par)) rotate(x->par);

else rotate(x);

rotate(x);

}

}

void access(node\* x)

{

splay(x);

if( x->link[1] ) x->link[1]->path\_parent = x, x->link[1]->par = NULL;

x->link[1] = NULL; x->pushup();

while( x->path\_parent ){

node \*pp = x->path\_parent, \*r;

splay(pp);

r = pp->link[1];

if( r ) r->par = NULL, r->path\_parent = pp;

pp->link[1] = x; pp->pushup(); x->par = pp;

x->path\_parent = NULL;

splay(x);

}

}

void cut(int u)

{

access(N+u);

if( N[u].link[0] ) N[u].link[0]->par = NULL;

N[u].link[0] = NULL; N[u].pushup();

}

void link(int u, int v) // u must be root.

{

if( u == v ) return;

access(N+u);

access(N+v);

//assert(!N[u].link[0]);

N[u].link[0] = N+v; N[v].par = N+u; N[u].pushup();

}

// recommend: don’t copy & paste code below.

int read(int u)

{

access( N+u );

return N[u].cnt;

}

int root(int u)

{

access( N+u );

node\* ans = N+u;

while( ans->link[0] ) ans = ans->link[0];

splay(ans);

return ans - N;

}

int mx(int u)

{

access( N+u );

return N[u].max;

}

bool chk()

{

for(int i=0;i<N\_;i++){

if( N[i].cnt == 0 ) return true;

if( N[i].cnt != cnt(N[i].link[0]) + cnt(N[i].link[1]) + 1) return false;

if( N[i].mx != max( max( mx(N[i].link[0]), mx(N[i].link[1]) ), N[i].val) ) return false;

if( N[i].par && N+i != N[i].par->link[dir(N+i)] ) return false;

if( N[i].link[0] && N+i != N[i].link[0]->par) return false;

if( N[i].link[1] && N+i != N[i].link[1]->par) return false;

}

return true;

}

}LCT;

# Graph Algorithms

## Dominator tree

#include<vector>

using namespace std;

#define pb(x) push\_back(x)

namespace dtree{

const int MAXN = 100010;

vector <int> E[MAXN];

vector <int> RE[MAXN], rdom[MAXN];

int S[MAXN], RS[MAXN], cs;

int par[MAXN], val[MAXN];

int sdom[MAXN], rp[MAXN];

int dom[MAXN];

int Find(int x, int c = 0) {

if(par[x] == x) return c ? -1 : x;

int p = Find(par[x], 1);

if(p == -1) return c ? par[x] : val[x];

if(sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]];

par[x] = p;

return c ? p : val[x];

}

void Union(int x, int y) {

par[x] = y;

}

void dfs(int x) {

RS[ S[x] = ++cs ] = x;

par[cs] = sdom[cs] = val[cs] = cs;

for(int e : E[x]) {

if(S[e] == 0) dfs(e), rp[S[e]] = S[x];

RE[S[e]].pb(S[x]);

}

}

int Do(int s, int \*up) {

dfs(s);

for(int i=cs;i;i--) {

for(int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]);

if(i > 1) rdom[sdom[i]].pb(i);

for(int e : rdom[i]) {

int p = Find(e);

if(sdom[p] == i) dom[e] = i;

else dom[e] = p;

}

if(i > 1) Union(i, rp[i]);

}

for(int i=2;i<=cs;i++) if(sdom[i] != dom[i]) dom[i] = dom[dom[i]];

for(int i=2;i<=cs;i++) {

up[RS[i]] = RS[dom[i]];

}

return cs;

}

void addE(int x, int y) { E[x].pb(y); }

}

## Arborescence

정점이 root인 트리 중 가중치 합이 최소인 것

namespace Arborescence{

  const int MX = 510, INF = 1e9;

  int e[MX][MX], lst[MX][MX];

  vector<int> v[MX], rev[MX], order;

  int was[MX], vst[MX], ans[MX], p[MX];

  vector<pii> G[MX];

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

  void set\_graph(int ee[MX][MX]){ memcpy(e, ee, sizeof e); }

  void go(int x) {

    if(vst[x]) return;

    vst[x] = 1;

    for (int to : v[x]) go(to);

    order.pb(x);

  }

  void col(int x, int o) {

    if (was[x]) return;

    was[x] = o;

    for (int to : rev[x]) col(to, o);

  }

  int run(int n, int root) {

    int ret = 0, done = 0;

    for(int i = 1; i <= n; i++) p[i] = i;

    memset(lst, 0, sizeof lst);

    for(int tt = 1;;tt++) {

      memset(was, 0, sizeof was);

      memset(vst, 0, sizeof vst);

      for (int i = 1; i <= n; i++) {

        v[i].clear();

        rev[i].clear();

      }

      order.clear();

      int mn[MX] = {};

      for(int i = 1; i <= n; i++) mn[i] = INF;

      for (int i = 1; i <= n; i++) if (find(i) != find(root))

        for (int j = 1; j <= n; j++) if(find(i) != find(j))

          mn[find(i)] = min(mn[find(i)], e[j][i]);

      for (int i = 1; i <= n; i++) if (find(i) != find(root)) {

        if(find(i) == i) ret += mn[i];

        for (int j = 1; j <= n; j++) if(find(i) != find(j)) e[j][i] -= mn[find(i)];

      }

      for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++){

        int a = find(i), b = find(j);

        if (a != b && e[i][j] == 0) {

          lst[i][j] = tt;

          v[a].pb(b);

          rev[b].pb(a);

        }

      }

      if (done) break;

      for (int i = 1; i <= n; i++) if (!vst[i]) go(i);

      reverse(order.begin(), order.end());

      for(int u : order) if (!was[u]) col(u, u);

      done = 1;

      for(int i = 1; i <= n; i++) if(was[i] != i) done = 0, p[i] = was[i];

    }

    priority\_queue<t3, vector<t3>, greater<t3>> Q;

    memset(ans, -1, sizeof ans);

    ans[root] = 0;

    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++)

      if(e[i][j] == 0) G[i].emplace\_back(lst[i][j], j);

    for(pii c : G[root]) Q.emplace(c.first, root, c.second);

    while(Q.size()){

      int a, b; tie(ignore, a, b) = Q.top(); Q.pop();

      if(ans[b] != -1) continue;

      ans[b] = a;

      for(pii c : G[b]) Q.emplace(c.first, b, c.second);

    }

    return ret;

  }

};

;

## Maximum Matching

namespace Matching{

//matching [1...n] <-> [1...m]

const int MX = 40040, MY = 40040;

vector <int> E[MX];

int xy[MX], yx[MY];

int n, m;

void addE(int x, int y) { E[x].pb(y); }

void setnm(int sn, int sm) { n = sn; m = sm; }

int tdis[MX], que[MX], \*dis = tdis + 1;

int bfs() {

int \*fr = que, \*re = que;

for(int i=1;i<=n;i++) {

if(xy[i] == -1) \*fr++ = i, dis[i] = 0;

else dis[i] = -1;

}

dis[-1] = -1;

while(fr != re) {

int t = \*re++;

if(t == -1) return 1;

for(int e : E[t]) {

if(dis[yx[e]] == -1) dis[yx[e]] = dis[t] + 1, \*fr++ = yx[e];

}

}

return 0;

}

int dfs(int x) {

for(int e : E[x]) {

if(yx[e] == -1 || (dis[yx[e]] == dis[x] + 1 && dfs(yx[e]))) {

xy[x] = e;

yx[e] = x;

return 1;

}

}

dis[x] = -1;

return 0;

}

int Do() {

memset(xy, -1, sizeof xy);

memset(yx, -1, sizeof yx);

int ans = 0;

while(bfs()) {

for(int i=1;i<=n;i++) if(xy[i] == -1 && dfs(i)) ++ans;

}

return ans;

}

}

void solve(){

int n, m;

scanf("%d%d", &n, &m);

Matching::setnm(n, m);

for(int i=1;i<=n;i++) {

int x; scanf("%d", &x);

while(x--) {

int y; scanf("%d", &y);

Matching::addE(i, y);

}

}

printf("%d\n", Matching::Do());

}

## Dinic’s Algorithm

namespace MaxFlow{

const int MV = 20020;

const int ME = 40040;

const int INF = 1e9;

int dis[MV];

int st[MV], en[ME<<1], nxt[ME<<1], flow[ME<<1], ce;

int start[MV];

void init() {

memset(st, 0, sizeof st);

ce = 1;

}

void addE(int s, int e, int f = 1) {

++ce; nxt[ce] = st[s]; st[s] = ce; en[ce] = e; flow[ce] = f;

++ce; nxt[ce] = st[e]; st[e] = ce; en[ce] = s; flow[ce] = 0;

}

int que[MV];

int bfs(int N, int S, int E) {

for(int i=1;i<=N;i++) dis[i] = -1;

dis[S] = 0;

int \*fr = que, \*re = que;

\*fr++ = S;

while(fr != re) {

int t = \*re++;

for(int i=st[t];i;i=nxt[i]) if(flow[i] > 0 && dis[en[i]] == -1) {

dis[en[i]] = dis[t] + 1;

\*fr++ = en[i];

}

}

return dis[E] != -1;

}

int dfs(int x, int E, int f) {

if(x == E) return f;

for(int &i=start[x],t;i;i=nxt[i]) if(flow[i] > 0 && dis[en[i]] == dis[x] + 1 && (t = dfs(en[i], E, min(f, flow[i]))) > 0){

flow[i] -= t;

flow[i^1] += t;

return t;

}

return 0;

}

int Get(int N, int S, int E) {

int res = 0;

while(bfs(N, S, E)) {

for(int i=1;i<=N;i++) start[i] = st[i];

while(1) {

int t = dfs(S, E, INF);

if(t == 0) break;

res += t;

}

}

return res;

}

}

int Do(int L) {

MaxFlow::init();

int S = n + m + 1, E = n + m + 2;

for(int i=1;i<=n;i++) MaxFlow::addE(S, i, L);

rep(i, m) {

rep(j, 2) MaxFlow::addE(p[i][j], n + 1 + i), g[i][j] = MaxFlow::ce - 1;

}

for(int i=1;i<=m;i++) MaxFlow::addE(n + i, E);

if(MaxFlow::Get(E, S, E) == m) {

for(int i=1;i<=m;i++) ans[i] = MaxFlow::flow[g[i-1][0]] == 0;

return 1;

}

return 0;

}

## Vizing’s Algorithm

G의 간선을 (max degree)+1개의 색으로 칠한다.

한 정점을 공유하는 간선 2개는 같은 색일 수 없다.

typedef pair<int,int> pii;

const int MX = 2505;

int C[MX][MX] = {}, G[MX][MX] = {};

void solve(vector<pii> &E, int N, int M){

int X[MX] = {}, a, b;

auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };

auto color = [&](int u, int v, int c){

int p = G[u][v];

G[u][v] = G[v][u] = c;

C[u][c] = v; C[v][c] = u;

C[u][p] = C[v][p] = 0;

if( p ) X[u] = X[v] = p;

else update(u), update(v);

return p; };

auto flip = [&](int u, int c1, int c2){

int p = C[u][c1], q = C[u][c2];

swap(C[u][c1], C[u][c2]);

if( p ) G[u][p] = G[p][u] = c2;

if( !C[u][c1] ) X[u] = c1;

if( !C[u][c2] ) X[u] = c2;

return p; };

for(int i = 1; i <= N; i++) X[i] = 1;

for(int t = 0; t < E.size(); t++){

int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d;

vector<pii> L;

int vst[MX] = {};

while(!G[u][v0]){

L.emplace\_back(v, d = X[v]);

if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c);

else if(!C[u][d])for(a=(int)L.size()-1;a>=0;a--)color(u,L[a].first,L[a].second);

else if( vst[d] ) break;

else vst[d] = 1, v = C[u][d];

}

if( !G[u][v0] ){

for(;v; v = flip(v, c, d), swap(c, d));

if(C[u][c0]){

for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);

for(; a >= 0; a--) color(u, L[a].first, L[a].second);

} else t--;

}

}

}

## Hungarian Method

int in[505][505];

int mats[505], matt[505];

int Ls[505], Lt[505];

int revs[505], revt[505];

int valt[505];

bool chks[505], chkt[505];

vector <int> Vu;

void vpush(int p, int N) {

chks[p] = true;

for (int i = 1; i <= N; i++) {

if (!valt[i]) continue;

if (valt[i] > Ls[p] + Lt[i] - in[p][i]) {

valt[i] = Ls[p] + Lt[i] - in[p][i];

revt[i] = p;

if (!valt[i]) Vu.push\_back(i);

}

}

}

int main() {

int N, i, j, k;

scanf("%d", &N);

for (i = 1; i <= N; i++) {

for (j = 1; j <= N; j++) {

scanf("%d", &in[i][j]);

in[i][j] = -in[i][j];

}

}

for (i = 1; i <= N; i++) Lt[i] = -INF;

for (i = 1; i <= N; i++) for (j = 1; j <= N; j++) Lt[j] = max(Lt[j], in[i][j]);

for (i = 1; i <= N; i++) {

for (j = 1; j <= N; j++) chks[j] = chkt[j] = false;

for (j = 1; j <= N; j++) valt[j] = INF;

for (j = 1; j <= N; j++) revs[j] = revt[j] = 0;

int p = 0;

for (j = 1; j <= N; j++) if (!mats[j]) break;

p = j;

vpush(p, N);

while (1) {

if (!Vu.empty()) {

int t = Vu.back();

Vu.pop\_back();

chkt[t] = true;

if (!matt[t]) {

vector <int> Vu2;

Vu2.push\_back(t);

while (1) {

Vu2.push\_back(revt[Vu2.back()]);

if (Vu2.back() == p) break;

Vu2.push\_back(revs[Vu2.back()]);

}

reverse(all(Vu2));

for (j = 0; j < Vu2.size(); j += 2) {

int s = Vu2[j], t = Vu2[j + 1];

mats[s] = t;

matt[t] = s;

}

break;

}

else {

int s = matt[t];

revs[s] = t;

vpush(s, N);

}

}

else {

int mn = INF;

for (j = 1; j <= N; j++) if (!chkt[j]) mn = min(mn, valt[j]);

for (j = 1; j <= N; j++) {

if (chks[j]) Ls[j] -= mn;

if (chkt[j]) Lt[j] += mn;

else {

valt[j] -= mn;

if (valt[j] == 0) Vu.push\_back(j);

}

}

}

}

Vu.clear();

}

int ans = 0;

for (i = 1; i <= N; i++) ans += Ls[i] + Lt[i];

return !printf("%d\n", -ans);

}

## Maximum clique

ll G[40]; // 0-index

    void get\_clique(int R = 0, ll P = (1ll<<N)-1, ll X = 0){

      if((P|X) == 0){

        cur = max(cur, R);

        return;

      }

      int u = \_\_builtin\_ctzll(P|X);

      ll c = P&~G[u];

      while(c){

        int v = \_\_builtin\_ctzll(c);

        get\_clique(R + 1, P&G[v], X&G[v]);

        P ^= 1ll << v;

        X |= 1ll << v;

        c ^= 1ll << v;

      }

    }

## Stoer Wagner’s Mincut Algorithm

가중치 그래프를 두조각내는 최소 min-cut을 구한다.

namespace stoer\_wagner{

const int MX = 505;

int G[MX][MX], vst[MX];

void init(int n){ memset(G, 0, sizeof G); }

void add\_edge(int a, int b, int d){ if(a != b) G[a][b] = G[b][a] = d; }

pii minimum\_cut\_phase(int n, int st, int &res){

int dist[MX] = {}, vis[MX];

int cur = 1e9, s = st, e = -1;

memcpy(vis, vst, sizeof vst);

dist[st] = 1e9;

while(1){

int mx = 0;

for(int i=1;i<=n;i++) if(!vis[i] && (!mx || dist[mx] < dist[i])) mx = i;

if(mx == 0) break;

cur = dist[mx]; e = s; s = mx; vis[mx] = 1;

for(int i = 1; i <= n; i++) dist[i] += G[mx][i];

}

res = min(res, cur);

return pii(s, e);

}

int run(int n){

if(n <= 1) return 0;

memset(vst, 0, sizeof vst);

int res = 1e9, t = 1, u;

for(int i = 0; i < n-1; i++){

tie(t, u) = minimum\_cut\_phase(n, t, res);

vst[u] = 1;

for(int i = 1; i <= n; i++){

if(vst[i] || t == i) continue;

G[t][i] += G[u][i]; G[i][t] += G[u][i];

}

}

return res;

}

};

# Mathematical Stuff

## Kirchhoff’s Theorem

주어진 그래프에서 가능한 신장트리의 경우의 수를 구한다.

m[i][j] := **-**(i-j 간선 개수) (i != j)

m[i][i] := 정점 i의 degree

(개수) = (m의 첫 번째 행과 첫 번째 열을 없앤 (n-1) by (n-1) matrix의 행렬식)

## LP Duality

tableu를 대각선으로 뒤집고 음수 부호를 붙인 답 = -(원 문제의 답)

ex) n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]], b = [24, 60], c = [6, 14, 13]

⬄ n = 3, m = 2, a = [[-0.5, -1], [-2, -2], [-1, -4]], b = [-6, -14, -13], c = [-24, -60]

## Simplex Algorithm

n := number of variables

m := number of constraints

a[1~m][1~n] := constraints

b[1~m] := constraints value (b[i] can be negative)

c[1~n] := maximum coefficient

v := results

sol[i] := 등호조건, i번째 변수의 값

ex) Maximize p = 6x + 14y + 13z

Constraints: 0.5x + 2y + z ≤ 24

x + 2y + 4z ≤ 60

n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]], b = [24, 60], c = [6, 14, 13]

namespace simplex {

  using T = long double;

  const int N = 410, M = 30010;

  const T eps = 1e-7;

  int n, m;

  int Left[M], Down[N];

  T a[M][N], b[M], c[N], v, sol[N];

  bool eq(T a, T b) { return fabs(a - b) < eps; }

  bool ls(T a, T b) { return a < b && !eq(a, b); }

  void init(int p, int q) {

    n = p; m = q; v = 0;

    for(int i = 1; i <= m; i++){

      for(int j = 1; j <= n; j++) a[i][j]=0;

    }

    for(int i = 1; i <= m; i++) b[i]=0;

    for(int i = 1; i <= n; i++) c[i]=sol[i]=0;

  }

  void pivot(int x,int y) {

    swap(Left[x], Down[y]);

    T k = a[x][y]; a[x][y] = 1;

    vector<int> nz;

    for(int i = 1; i <= n; i++){

      a[x][i] /= k;

      if(!eq(a[x][i], 0)) nz.push\_back(i);

    }

    b[x] /= k;

    for(int i = 1; i <= m; i++){

      if(i == x || eq(a[i][y], 0)) continue;

      k = a[i][y]; a[i][y] = 0;

      b[i] -= k\*b[x];

      for(int j : nz) a[i][j] -= k\*a[x][j];

    }

    if(eq(c[y], 0)) return;

    k = c[y]; c[y] = 0;

    v += k\*b[x];

    for(int i : nz) c[i] -= k\*a[x][i];

  }

  // 0: found solution, 1: no feasible solution, 2: unbounded

  int solve() {

    for(int i = 1; i <= n; i++) Down[i] = i;

    for(int i = 1; i <= m; i++) Left[i] = n+i;

    while(1) { // Eliminating negative b[i]

      int x = 0, y = 0;

      for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b[i] < b[x])) x = i;

      if(x == 0) break;

      for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0 || a[x][i] < a[x][y])) y = i;

      if(y == 0) return 1;

      pivot(x, y);

    }

    while(1) {

      int x = 0, y = 0;

      for(int i = 1; i <= n; i++)

        if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;

      if(y == 0) break;

      for(int i = 1; i <= m; i++)

        if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])) x = i;

      if(x == 0) return 2;

      pivot(x, y);

    }

    for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b[i];

    return 0;

  }

}

## Polynomial multimplcation

#include <complex>

#include <vector>

using namespace std;

#define pb(x) push\_back(x)

// blog.myungwoo.kr/54

namespace FFT{

typedef complex<double> base;

typedef long long ll;

#define all(x) (x).begin(), (x).end()

#define sz(x) ((int)(x).size())

const double C\_PI = acos(-1);

void fft(vector <base> &a, bool invert){

int n = sz(a);

for(int i=0,j=0;i<n;++i) {

if(i>j) swap(a[i],a[j]);

for(int k=n>>1;(j^=k)<k;k>>=1);

}

for (int len=2;len<=n;len<<=1){

double ang = 2\*C\_PI/len\*(invert?-1:1);

base wlen(cos(ang), sin(ang));

for (int i=0;i<n;i+=len){

base w(1);

for (int j=0;j<len/2;j++){

if((j & 511) == 511)w = base(cos(ang \* j), sin(ang \* j));

//오차가 클 경우 이 빈도를 늘린다. cos, sin 함수는 시간 부담이 있으니 주의

base u = a[i+j], v = a[i+j+len/2]\*w;

a[i+j] = u+v;

a[i+j+len/2] = u-v;

w \*= wlen;

}

}

}

if (invert){

for (int i=0;i<n;i++) a[i] /= n;

}

}

void multiply(const vector<int> &a,const vector<int> &b,vector<int> &res){

int L = sz(a) + sz(b) + 1;

vector <base> fa(all(a)), fb(all(b));

int n = 1;

while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;

fa.resize(n); fb.resize(n);

fft(fa,false); fft(fb,false);

for (int i=0;i<n;i++) fa[i] \*= fb[i];

fft(fa,true);

res.resize(L);

for(int i=0;i<L;i++)res[i]=((int)(fa[i].real()+(fa[i].real()>0?0.5:-0.5))) & 1;

}

void multiply\_with\_modulo(const vector<int> &a,const vector<int> &b,vector<int> &res, const int MOD){

int n = 1;

while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;

vector <base> A(n), B(n);

int L\_BLOCK = 15; //2^L\_BLOCK ~= sqrt(MOD).

for(int i=0;i<n;i++)

A[i] = (i < sz(a) ? base(a[i] & ((1<<L\_BLOCK)-1),a[i] >> L\_BLOCK) : base(0));

for(int i=0;i<n;i++)

B[i] = (i < sz(b) ? base(b[i] & ((1<<L\_BLOCK)-1),b[i] >> L\_BLOCK) : base(0));

fft(A, false); fft(B, false);

vector <base> f1(n), f2(n), f3(n), f4(n);

for(int i=0;i<n;i++) {

int j=(n-i)&(n-1);

f2[i]=(A[i]+conj(A[j]))\*base(0.5,0);

f1[i]=(A[i]-conj(A[j]))\*base(0,-0.5);

f4[i]=(B[i]+conj(B[j]))\*base(0.5,0);

f3[i]=(B[i]-conj(B[j]))\*base(0,-0.5);

}

for(int i=0;i<n;i++) {

A[i]=f1[i]\*f3[i]+f1[i]\*f4[i]\*base(0,1);

B[i]=f2[i]\*f4[i]\*base(0,1)+f2[i]\*f3[i];

}

fft(A, true); fft(B, true);

res.resize(n);

for(int i=0;i<n;i++) {

ll g1=(ll)(A[i].real()+0.5) % MOD; //A[i].real > 0 이어야 함.

ll g2=(ll)(A[i].imag()+0.5) % MOD;

ll g3=(ll)(B[i].real()+0.5) % MOD;

ll g4=(ll)(B[i].imag()+0.5) % MOD;

res[i] = (g4 + ((g2+g3)<<L\_BLOCK) + (g1<<(L\_BLOCK<<1))) % MOD;

}

}

void multiply\_big(const vector<int> &a,const vector<int> &b, vector <ll> &res){

// 단순히 오차가 심해 구하지 못하는 경우

// 결과값은 long long 범위 안

int n = 1;

while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;

vector <base> A(n), B(n);

int L\_BLOCK = 10;

for(int i=0;i<n;i++)

A[i] = (i < sz(a) ? base(a[i] & ((1<<L\_BLOCK)-1),a[i] >> L\_BLOCK) : base(0));

for(int i=0;i<n;i++)

B[i] = (i < sz(b) ? base(b[i] & ((1<<L\_BLOCK)-1),b[i] >> L\_BLOCK) : base(0));

fft(A, false); fft(B, false);

vector <base> f1(n), f2(n), f3(n), f4(n);

for(int i=0;i<n;i++) {

int j=(n-i)&(n-1);

f2[i]=(A[i]+conj(A[j]))\*base(0.5,0);

f1[i]=(A[i]-conj(A[j]))\*base(0,-0.5);

f4[i]=(B[i]+conj(B[j]))\*base(0.5,0);

f3[i]=(B[i]-conj(B[j]))\*base(0,-0.5);

}

for(int i=0;i<n;i++) {

A[i]=f1[i]\*f3[i]+f1[i]\*f4[i]\*base(0,1);

B[i]=f2[i]\*f4[i]\*base(0,1)+f2[i]\*f3[i];

}

fft(A, true); fft(B, true);

res.resize(n);

for(int i=0;i<n;i++) {

ll g1=(ll)(A[i].real()+0.5);

ll g2=(ll)(A[i].imag()+0.5);

ll g3=(ll)(B[i].real()+0.5);

ll g4=(ll)(B[i].imag()+0.5);

res[i] = (g4 + ((g2+g3)<<(L\_BLOCK)) + (g1<<(L\_BLOCK<<1)));

}

}

}

## Polynomial multimplcation with special modulo

정수만을 사용하여 다항식 곱셈을 한 것. (writer: cubelover)

#include <cstdio>

const int A = 7, B = 26, P = A << B | 1, R = 3;

const int SZ = 20, N = 1 << SZ;

int Pow(int x, int y) {

int r = 1;

while (y) {

if (y & 1) r = (long long)r \* x % P;

x = (long long)x \* x % P;

y >>= 1;

}

return r;

}

void FFT(int \*a, bool f) {

int i, j, k, x, y, z;

j = 0;

for (i = 1; i < N; i++) {

for (k = N >> 1; j >= k; k >>= 1) j -= k;

j += k;

if (i < j) {

k = a[i];

a[i] = a[j];

a[j] = k;

}

}

for (i = 1; i < N; i <<= 1) {

x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);

for (j = 0; j < N; j += i << 1) {

y = 1;

for (k = 0; k < i; k++) {

z = (long long)a[i | j | k] \* y % P;

a[i | j | k] = a[j | k] - z;

if (a[i | j | k] < 0) a[i | j | k] += P;

a[j | k] += z;

if (a[j | k] >= P) a[j | k] -= P;

y = (long long)y \* x % P;

}

}

}

if (f) {

j = Pow(N, P - 2);

for (i = 0; i < N; i++) a[i] = (long long)a[i] \* j % P;

}

}

int X[N];

int main() {

int i, n;

scanf("%d", &n);

for (i = 0; i <= n; i++) scanf("%d", &X[i]);

FFT(X, false);

for (i = 0; i < N; i++) X[i] = (long long)X[i] \* X[i] % P;

FFT(X, true);

for (i = 0; i <= n + n; i++) printf("%d ", X[i]);

}

# Geometry

## Delaunay triangulation

struct triple {

int i, j, k;

triple() {}

triple(int i, int j, int k) : i(i), j(j), k(k) {}

};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {

int n = x.size();

vector<T> z(n);

vector<triple> ret;

for (int i = 0; i < n; i++)

z[i] = x[i] \* x[i] + y[i] \* y[i];

for (int i = 0; i < n-2; i++) {

for (int j = i+1; j < n; j++) {

for (int k = i+1; k < n; k++) {

if (j == k) continue;

double xn = (y[j]-y[i])\*(z[k]-z[i]) - (y[k]-y[i])\*(z[j]-z[i]);

double yn = (x[k]-x[i])\*(z[j]-z[i]) - (x[j]-x[i])\*(z[k]-z[i]);

double zn = (x[j]-x[i])\*(y[k]-y[i]) - (x[k]-x[i])\*(y[j]-y[i]);

bool flag = zn < 0;

for (int m = 0; flag && m < n; m++)

flag = flag && ((x[m]-x[i])\*xn +(y[m]-y[i])\*yn + (z[m]-z[i])\*zn <= 0);

if (flag) ret.push\_back(triple(i, j, k));

}

}

}

return ret;

}

## General Geometry Library Header

#include<stdio.h>

#include<set>

#include<assert.h>

#include<queue>

#include<algorithm>

#include<vector>

#include<cmath>

using namespace std;

typedef long long ll;

typedef \_\_int128 llll;

typedef pair<int, int> pii;

typedef pair<double, double> pdd;

typedef pair<double, int> pdi;

const double EPS = 1e-8;

const double PI = acos(-1);

double sq(double x){ return x\*x; }

ll sq(ll x){ return x\*x; }

int sign(ll x){ return x < 0? -1 : x > 0? 1 : 0; }

int sign(int x){ return x < 0? -1 : x > 0? 1 : 0; }

double sign(double x){ return abs(x) < EPS? 0 : x < 0? -1 : 1; }

pii operator+(const pii &l, const pii &r){ return pii(l.first + r.first, l.second + r.second); }

pii operator-(const pii &l, const pii &r){ return pii(l.first - r.first, l.second - r.second); }

ll operator^(const pii &l, const pii &r){ return (ll)l.first \* r.second - (ll)l.second \* r.first; }

ll operator/(const pii &l, const pii &r){ return (ll)l.first \* r.second - (ll)l.second \* r.first; }

ll operator\*(const pii &l, const pii &r){ return (ll)l.first \* r.first + (ll)l.second \* r.second; }

pii operator\*(const pii &l, const int &r){ return pii(l.first \* r, l.second \* r); }

pii operator-(const pii &l){ return pii(-l.first, -l.second); }

pdd operator+(const pdd &l, const pdd &r){ return pdd(l.first + r.first, l.second + r.second); }

pdd operator-(const pdd &l, const pdd &r){ return pdd(l.first - r.first, l.second - r.second); }

double operator^(const pdd &l, const pdd &r){ return l.first \* r.second - l.second \* r.first; }

double operator/(const pdd &l, const pdd &r){ return l.first \* r.second - l.second \* r.first; }

double operator\*(const pdd &l, const pdd &r){ return l.first \* r.first + l.second \* r.second; }

pdd operator\*(const pdd &l, const double &r){ return pdd(l.first \* r, l.second \* r); }

pdd operator-(const pdd &l){ return pdd(-l.first, -l.second); }

double size(pdd x){ return hypot(x.first, x.second); }

double size2(pdd x){ return sq(x.first) + sq(x.second); }

ll size2(pii x){ return sq((ll)x.first) + sq((ll)x.second); }

double polar(pdd x){ return atan2(x.second, x.first); }

pdd unit(double a){ return pdd(cos(a), sin(a)); }

pdd norm(pdd a){ return a \* (1.0 / size(a)); }

pdd rotate(pdd v, double a){ return unit(a) \* v.first + unit(a + PI / 2) \* v.second; }

pdd r90(pdd v){ return pdd(-v.second, v.first); }

void normalize(double &a){

while (a < 0) a += 2 \* PI;

while (a >= 2 \* PI) a -= 2 \* PI;

}

struct circle{

circle(pdd O, double r):O(O), r(r){}

circle(){}

pdd O;

double r;

};

## Tangent series

int tangent(circle &A, circle &B, pdd des[4]){

// outer

int top = 0;

double d = size(A.O - B.O), a = polar(B.O - A.O), b = PI + a;

double t = sq(d) - sq(A.r - B.r);

if (t >= 0){

t = sqrt(t);

double p = atan2(B.r - A.r, t);

des[top++] = pdd(a + p + PI / 2, b + p - PI / 2);

des[top++] = pdd(a - p - PI / 2, b - p + PI / 2);

}

// inner

t = sq(d) - sq(A.r + B.r);

if (t >= 0){

t = sqrt(t);

double p = atan2(B.r + A.r, t);

des[top++] = pdd(a + p - PI / 2, b + p - PI / 2);

des[top++] = pdd(a - p + PI / 2, b - p + PI / 2);

}

return top;

}

// C : conter\_clockwise(C[0] == C[N])

// return highest point in C <- P(clockwise) or -1 in convex

// recommend : strongly convex, C.size() >= 3, C[i] != P

// up^down == 0 : on line

int convex\_tangent(vector<pii> &C, pii P, int up = 1){

auto sign = [&](ll c){ return c > 0 ? up : c == 0 ? 0 : -up; };

auto local = [&](pii P, pii a, pii b, pii c) {

return sign((a - P) ^ (b - P)) <= 0 && sign((b - P) ^ (c - P)) >= 0;

};

assert(C.size() >= 2);

int N = C.size()-1, s = 0, e = N, m;

if( local(P, C[1], C[0], C[N-1]) ) return 0;

// for(int i = 1; i < N; i++) if( local(P, C[i-1], C[i], C[i+1])) return i;

while(s+1 < e){

m = (s+e) / 2;

if( local(P, C[m-1], C[m], C[m+1]) ) return m;

if( sign((C[s]-P) ^ (C[s+1]-P)) < 0 ){ // up

if( sign((C[m]-P) ^ (C[m+1]-P)) > 0 ) e = m;

else if( sign((C[m]-P) ^ (C[s]-P)) > 0 ) s = m;

else e = m;

}

else{ // down

if( sign((C[m]-P) ^ (C[m+1]-P)) < 0 ) s = m;

else if( sign((C[m]-P) ^ (C[s]-P)) < 0 ) s = m;

else e = m;

}

}

if( s && local(P, C[s-1], C[s], C[s+1]) ) return s;

if( e != N && local(P, C[e-1], C[e], C[e+1]) ) return e;

return -1;

}

## Intersect series

int intersect(circle &A, circle &B, pdd des[2]){

double d = size(A.O - B.O), t = (sq(A.r) + sq(d) - sq(B.r)) / 2 / A.r / d, u = (sq(B.r) + sq(d) - sq(A.r)) / 2 / B.r / d;

if (abs(d) < EPS) return 0;

if (1 - t\*t < 0 || 1 - u\*u < 0) return 0;

double a = atan2(sqrt(1 - t\*t), t), b = atan2(sqrt(1 - u\*u), u), p = polar(B.O - A.O), q = PI + p;

des[0] = pdd(p + a, q - b);

des[1] = pdd(p - a, q + b);

return 2;

}

int intersect(circle A, pdd s, pdd d, pdd des[2]){

double c = size2(A.O - s) - sq(A.r), b = d \* (s - A.O), a = size2(d);

if (b\*b - a\*c < 0) return 0;

des[0].second = (-b - sqrt(b\*b - a\*c)) / a;

des[1].second = (-b + sqrt(b\*b - a\*c)) / a;

des[0].first = polar(s + d\*des[0].second - A.O);

des[1].first = polar(s + d\*des[1].second - A.O);

return 2;

}

int intersect(pdd a, pdd b, pdd u, pdd v, pdd &des){

if (abs(b^v) < EPS) return 0;

des = pdd(((a - u) ^ v) / (v^b), ((a - u) ^ b) / (v^b));

return 1;

}

bool isintersect(const pii &a, const pii &b, const pii &u, const pii &v){

if( b/v != 0 ) return sign((u-a)/b) \* sign((u+v-a)/b) <= 0 && sign((a-u)/v) \* sign((a+b-u)/v) <= 0;

else return (a-u)/v == 0 && (0 <= v\*(a-u) && v\*(a-u) <= v\*v || 0 <= b\*(u-a) && b\*(u-a) <= b\*b);

}

## Distant series

double dist(const pdd &A, const pdd &p, const pdd &d){

if( size(A-p) <= EPS ) return 0;

else if( size(d) <= EPS ) return size(A-p);

double sina = ((A-p) ^ d) / size(A-p) / size(d);

double cosa = ((A-p) \* d) / size(A-p) / size(d);

double r = abs(size(A - p) \* sina), e = size(A - p) \* cosa;

if (0 < e && e < size(d));

else r = min(size(A - p), size(A - p - d));

return r;

}

## Make circle series

int get\_circle(pdd a, pdd b, double R, circle des[2]){

pdd m = (a+b) \* 0.5, t = (b-a);

double d = (R\*R - size2(m-a));

if( d < 0 ) return 0;

d = sqrt(d);

pdd p = norm(pdd(t.second, -t.first));

des[0] = circle(m + p\*d, R);

des[1] = circle(m - p\*d, R);

return 2;

}

int get\_circle(pdd p0, pdd p1, pdd p2, circle &des){

pdd a = (p0+p1) \* 0.5, b = r90(p0-p1);

pdd u = (p0+p2) \* 0.5, v = r90(p0-p2), R;

if( !intersect(a, b, u, v, R) ) return 0;

des = circle(a+b\*R.first, size(a+b\*R.first - p0));

return 1;

}

## 3D Header

struct v3{

double x, y, z;

v3(){}

v3(double x, double y, double z) :x(x), y(y), z(z){}

v3 operator-()const{ return v3(-x, -y, -z); }

v3 operator-(const v3 &l)const{ return v3(x - l.x, y - l.y, z - l.z); }

v3 operator+(const v3 &l)const{ return v3(x + l.x, y + l.y, z + l.z); }

v3 operator\*(const double c)const{ return v3(x\*c, y\*c, z\*c); }

double operator\*(const v3 &l)const{ return x\*l.x + y\*l.y + z\*l.z; }

v3 operator^(const v3 &l)const{ return v3(y\*l.z - z\*l.y, z\*l.x - x\*l.z, x\*l.y - y\*l.x); }

v3 operator/(const v3 &l)const{ return v3(y\*l.z - z\*l.y, z\*l.x - x\*l.z, x\*l.y - y\*l.x); }

double size(){ return sqrt(sq(x) + sq(y) + sq(z)); }

double size2(){ return sq(x) + sq(y) + sq(z); }

v3 norm(){

double p = size();

return v3(x / p, y / p, z / p);

}

void print(){ printf("%lf %lf %lf\n", x, y, z); }

bool operator < (const v3 &l) const {

if (abs(x - l.x) >= EPS) return x < l.x;

if (abs(y - l.y) >= EPS) return y < l.y;

if (abs(z - l.z) >= EPS) return z < l.z;

return false;

}

bool operator == (const v3 &l) const {

return abs(x - l.x) < EPS && abs(y - l.y) < EPS && abs(z - l.z) < EPS;

}

};

struct Quad{

double a;

v3 v;

Quad(double a, v3 v) :a(a), v(v){}

Quad operator \* (const double &c)const{ return Quad(a \* c, v \* c); }

Quad operator~() const { return Quad(-a, -v); }

Quad operator-() const { return Quad(a, -v) \* (1 / (sq(a) + sq(v.x) + sq(v.y) + sq(v.z))); }

Quad operator \* (const Quad &l)const{ return Quad(a\*l.a - v\*l.v, l.v\*a + v\*l.a + (v^l.v)); } // 순서 조심, (u 회전 후 v) = v\*u

v3 apply(v3 p){

return ((\*this) \* Quad(0, p) \* -(\*this)).v;

}

};

double size(v3 a){ return a.size(); }

double size2(v3 a){ return a.size2(); }

v3 norm(v3 a){ return a.norm(); }

v3 unit(double a, double b){ return v3(cos(a)\*cos(b),sin(a)\*cos(b),sin(b)); }

Quad set\_rotate(v3 axis, double a){

return Quad(cos(a / 2), axis.norm() \* sin(a / 2));

}

struct sphere{

sphere(v3 O, double r):O(O), r(r){}

v3 O;

double r;

};

## 3D Intersect series

// line - line

int intersect(v3 a, v3 b, v3 u, v3 v, pdd &des){

v3 p = b^v, q = (u-a)^v;

v3 r = v^b, s = (a-u)^b;

// if (size(p) < EPS || size(p^q) > EPS) return 0;

des = pdd(p\*q/size2(p), r\*s/size2(r));

return 1;

}

//sphere - line

int intersect(sphere A, v3 s, v3 d, double des[2]){

double c = (A.O - s).size2() - sq(A.r), b = d \* (s - A.O), a = d.size2();

if (b\*b - a\*c < 0) return 0;

des[0] = (-b + sqrt(b\*b - a\*c)) / a;

des[1] = (-b - sqrt(b\*b - a\*c)) / a;

return 2;

}

// face - face

int intersect(v3 u, v3 v, v3 p, v3 q, v3 &s, v3 &d){ // v.size(), q.size() == 1

if( abs(v\*q-1) < EPS ) return 0;

d = v^q;

double t = v\*q;

s = v\*((u\*v - p\*q\*t) / (1-t\*t)) + q\*((u\*v\*t - p\*q) \* 1.0 / (t\*t-1));

return 1;

}

// face - line

int intersect(v3 u, v3 v, v3 p, v3 q, double &s){ // v.size(), q.size() == 1

if( abs(q\*v) <= EPS ) return 0;

s = -((p-u)\*v) / (q\*v);

// printf("intersect %.10lf\n", (u-(p+q\*s))\*v);

return 1;

}

double area(circle C, double s, double e)

{

double p = C.O.first, q = C.O.second, r = C.r;

return (p\*r\*(sin(e) - sin(s)) + q\*r\*(cos(s)-cos(e)) + r\*r\*(e-s)) \* 0.5;

}

## Convex Hull

template<typename T>

void convex\_hull(vector<T> &L, vector<T> &R){

int mn = 0;

for(int i = 1; i < L.size(); i++){

if( L[mn] > L[i] ) mn = i; // I changed this after checking above codes...

}

swap(L[mn], L[0]);

T t = L[0];

for(int i = 1; i < L.size(); i++) L[i] = L[i] - L[0];

L[0] = T(0, 0);

sort(L.begin()+1, L.end(), [](T l, T r){

if( sign(l^r) != 0 ) return sign(l^r) < 0;

return size(l) < size(r);

});

for(T c : L){

while(R.size() >= 2 && sign((R[R.size()-2] - R.back()) ^ (c - R.back())) <= 0 ) R.pop\_back();

R.push\_back(c);

}

for(T &c : R) c = c + t;

}

## Intersect area series

double area(pdd A[4], pdd B[4]){

vector<pdd> L;

for(int i = 0; i < 3; i++) L.push\_back(A[i]);

for(int i = 0; i < 3; i++) L.push\_back(B[i]);

for(int i = 0; i < 3; i++){

for(int j = 0; j < 3; j++){

pdd R;

if( !intersect(A[i], A[i+1]-A[i], B[j], B[j+1] - B[j], R) ) continue;

if( R.first < -EPS || R.first > 1+EPS || R.second < -EPS || R.second > 1+EPS ) continue;

L.push\_back(A[i] + (A[i+1]-A[i]) \* R.first);

}

}

vector<pdd> tmp;

swap(tmp, L);

for(pdd c : tmp){

bool ch = 1;

for(int i = 0; ch && i < 3; i++){

if( ((A[i+1]-A[i]) ^ (c-A[i])) > EPS ) ch = 0;

}

for(int i = 0; ch && i < 3; i++){

if( ((B[i+1]-B[i]) ^ (c-B[i])) > EPS ) ch = 0;

}

if( ch ) L.push\_back(c);

}

if( L.size() == 0 ) return 0;

sort(L.begin(), L.end());

L.resize(unique(L.begin(), L.end()) - L.begin());

vector<pdd> R; convex\_hull(L, R);

double ans = 0;

R.push\_back(R[0]);

for(int i = 0; i+1 < R.size(); i++) ans += R[i+1] ^ R[i];

return ans;

}

double area(pdd A, pdd B, double R) {

auto helper = [](pdd A, pdd B, double R){ return R\*R\*atan2(A^B, A\*B) / 2; };

auto is\_valid = [](double x) { return 0 <= x && x <= 1; };

double ans = 0, rv = 1;

pdd C, D, res[2];

if( size2(A) > size2(B) ) swap(A, B), rv = -1;

if (size2(B) <= R\*R) ans = (A^B) / 2;

else if (size2(A) <= R\*R) {

if (!intersect(circle(pdd(0, 0), R), A, B-A, res)) ans = (A^B) / 2;

else C = A + (B-A) \* (is\_valid(res[1].second)? res[1].second : res[0].second), ans = (A^C) / 2 + helper(C, B, R);

}

else {

if (!intersect(circle(pdd(0, 0), R), A, B-A, res) ||

res[0].second < 0 && res[1].second < 0 || res[0].second > 1 && res[1].second > 1) ans = helper(A, B, R);

else C = A + (B-A) \* res[0].second, D = A + (B-A) \* res[1].second, ans = helper(A, C, R) + (C^D) / 2 + helper(D, B, R);

}

return ans \* rv;

}

## Smallest enclosing circle

circle make\_circle(vector<pdd> Q){

if( Q.size() == 0 ) return circle(pdd(0, 0), 0);

if( Q.size() == 1 ) return circle(Q[0], 0);

circle res;

for(int i = 0; i < Q.size(); i++){

swap(Q.back(), Q[i]);

res = circle((Q[0]+Q[1]) \* 0.5, size(Q[0]-Q[1])/2);

bool ch = 1; for(pdd c : Q) if( size2(c-res.O) > sq(res.r) + EPS ) ch = 0;

if( ch ) return res;

swap(Q.back(), Q[i]);

}

get\_circle(Q[0], Q[1], Q[2], res);

return res;

}

circle smallest\_circle(vector<pdd> &P, vector<pdd> &Q, int N)

{

circle c = make\_circle(Q);

if( N == 0 || Q.size() >= 3 ) return c;

for(int i = 0; i < N; i++){

if( size2(c.O - P[i]) > sq(c.r) ){

Q.push\_back(P[i]);

c = smallest\_circle(P, Q, i);

Q.pop\_back();

}

}

return c;

}

## 3D Convex Hull

void convex\_hull(vector<v3> &L, vector<v3> &R, vector<v3> &O, vector<v3> &D, vector<v3> &X)

{

int N = L.size();

static bool chk[1005][1005];

static bool in[1005];

for(int i = 0; i < N; i++) for(int j = 0; j < N; j++) chk[i][j] = 0, in[i] = 0;

int q1 = 0, q2 = 1;

queue<pii> Q;

vector<pair<pii, int>> F;

for(int i = 1; i < N; i++) if( L[i].z < L[q1].z ) q1 = i, q2 = 0;

for(int i = 0; i < N; i++){

if( i == q1 ) continue;

if( (norm(L[i] - L[q1]) \* v3(0, 0, 1)) < (norm(L[q2]-L[q1]) \* v3(0, 0, 1)) ) q2 = i;

}

Q.push(pii(q1, q2)); chk[q1][q2] = 1;

while(Q.size()){

pii f = Q.front(); Q.pop();

int a = f.first, b = f.second, c = -1;

in[a] = in[b] = 1;

for(int i = 0; i < N; i++){

if( i == a || i == b ) continue;

if( c == -1 || ((L[i]-L[a])^(L[c]-L[a]))\*(L[b]-L[a]) < 0 ) c = i;

}

if( !chk[c][b] ) Q.push(pii(c, b)), chk[c][b] = 1;

if( !chk[a][c] ) Q.push(pii(a, c)), chk[a][c] = 1;

if( a > b ) swap(a, b);

if( b > c ) swap(b, c);

if( a > b ) swap(a, b);

F.emplace\_back(pii(a, b), c);

}

sort(F.begin(), F.end());

F.resize(unique(F.begin(), F.end()) - F.begin());

for(auto f : F){

int a = f.first.first, b = f.first.second, c = f.second;

O.push\_back(L[a]); X.push\_back(norm(L[a] - L[b]));

D.push\_back(norm((L[a]-L[b])^(L[c]-L[b])));

}

for(int i = 0; i < N; i++) if( in[i] ) R.push\_back(L[i]);

}

## Dynamic convex hull

// lo = 1 : lower\_convex

int cmp\_flag = 0, cmp\_lo = 0;

struct Convex{

struct line{

line(pii x){

if( !cmp\_flag ) s = x;

else d = x;

}

line(pii s, pii d):s(s), d(d){}

pii s, d;

bool operator<(const line &l)const{

if( !cmp\_flag ) return s < l.s;

return cmp\_lo ? d/l.d > 0 : d/l.d < 0;

}

};

int lo;

set<line> C;

void add(pii p){

cmp\_flag = 0;

auto pos = [&](ll x){ return lo ? x >= 0 : x <= 0; };

auto neg = [&](ll x){ return lo ? x <= 0 : x >= 0; };

auto e = C.upper\_bound(line(p)), d = e;

if( e != C.end() && d != C.begin() ){ d--;

if( pos((e->s - d->s)/(p - d->s)) ) return;

}

if( lo && e == C.end() && C.size() && C.rbegin()->s.first == p.first ) return;

if( !lo && e == C.begin() && C.size() && e->s.first == p.first ) return;

while(1){

d = C.upper\_bound(line(p)), e = d;

if( d == C.end() || ++e == C.end()) break;

if( neg((d->s - p)/(e->s - p)) ) C.erase(d);

else break;

}

while(1){

d = C.upper\_bound(line(p)), e = d;

if( d == C.begin() || --e == C.begin()) break;

d = e; --e; if( pos((d->s - p)/(e->s - p)) ) C.erase(d);

else break;

}

if( lo && C.size() && C.rbegin()->s.first == p.first) C.erase(--C.end());

if( !lo && C.size() && C.begin()->s.first == p.first) C.erase(C.begin());

d = C.upper\_bound(line(p));

if( d != C.begin() ){

e = prev(d);

line t = line(e->s, p - e->s);

C.erase(e); C.insert(t);

}

if( d == C.end() ) C.insert(line(p, pii(0, lo? 1 : -1)));

else C.insert(line(p, d->s - p));

}

// x \* -m.second + y \* m.first minimize

ll get\_mn(pii m){

cmp\_flag = 1; cmp\_lo = lo;

auto d = C.lower\_bound(m);

return (ll)d->s.first \* -m.second + (ll)d->s.second \* m.first;

}

};

## Halfplane Intersection

namespace halfplane\_intersection{

const int INF = 1e9;

struct line{ //

line(pii u, pii v):u(u), v(v){}

pii u, v;

};

double C(line A, line B){

return B.v / A.v == 0 ? 1e18 : ((A.u - B.u)/B.v) / (double)(B.v/A.v); //

}

bool V(line A, line B, double cur){

double l = (cur - A.u.first) / A.v.first \* A.v.second + A.u.second;

double r = (cur - B.u.first) / B.v.first \* B.v.second + B.u.second;

return l-EPS > r; // EPS!!

}

pdd W(line A, line B){

return (pdd)A.v \* C(A, B) + (pdd)A.u;

}

void chain(vector<line> &X, int l, int r, int up){ //

vector<line> R;

sort(X.begin(), X.end(), [&](line l, line r){ return l.v/r.v\*up < 0; });

R.emplace\_back(pii(l, 0), pii(0, up)); //

X.emplace\_back(pii(r, 0), pii(0, -up)); //

for(line c : X){

while(R.size() >= 2 && C(R[R.size()-2], R.back()) >= C(R[R.size()-2], c)) R.pop\_back();

R.push\_back(c);

} swap(R, X);

X.pop\_back(); X.erase(X.begin());

}

int reduce(vector<line> &X, vector<line> &Y, double &cur, int left){

while(X.size() && Y.size()){

if( V(X.back(), Y.back(), cur) ) break;

if( X.back().v/Y.back().v\*left >= 0 ) return 0;

cur = W(X.back(), Y.back()).first;

int ch = 0;

while(X.size() >= 2 && !V(X[X.size()-2], X.back(), cur)) X.pop\_back(), ch = 1;

while(Y.size() >= 2 && V(Y[Y.size()-2], Y.back(), cur)) Y.pop\_back(), ch = 1;

if( !ch ) break;

}

return 1;

}

// U + kV, CCW(Ex. ->: up, <-: down)

// 0 : empty, 1 : convex

// R -> 0: empty, 1: ccw order closed points

// added -1e9 ~ 1e9 x -1e9 ~ 1e9 bounding box

int run(vector<pii> &U, vector<pii> &V, vector<pdd> &R, int l = -INF, int r = INF, int u = -INF, int d = INF){ //

U.emplace\_back(0, u); V.emplace\_back(1, 0);

U.emplace\_back(0, d); V.emplace\_back(-1, 0);

vector<line> X, Y; // X : up convex, Y : down convex

int N = V.size();

for(int i = 0; i < N; i++){

if( V[i].first == 0 ){

if( V[i].second > 0 ) r = min(r, U[i].first);

else l = max(l, U[i].first);

}

else if( V[i].first < 0 ) X.emplace\_back(U[i], -V[i]);

else Y.emplace\_back(U[i], V[i]);

}

if( l > r ) return 0;

chain(X, l, r, 1);

chain(Y, l, r, -1);

double left = l, right = r;

auto rv = [](vector<line> &t){ reverse(t.begin(), t.end()); };

if( !reduce(X, Y, right, -1) ) return 0; rv(X), rv(Y);

if( !reduce(X, Y, left, 1) ) return 0; rv(Y);

if( left > right ) return 0;

vector<line> L;

if( left == l ) L.emplace\_back(pii(l, 0), pii(0, 1)); //

for(line c : Y) L.push\_back(c);

if( right == r ) L.emplace\_back(pii(r, 0), pii(0, -1)); //

for(line c : X) L.push\_back(c);

for(int i = 0; i+1 < L.size(); i++) R.push\_back(W(L[i], L[i+1]));

R.push\_back(W(L.back(), L[0]));

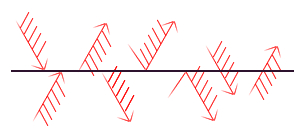
R.resize(unique(R.begin(), R.end()) - R.begin());

return 1;

}

};

## Polygon raycast



struct frac{

frac(){}

frac(ll a, ll b){

if( b < 0 ) first = -a, second = -b;

else first = a, second = b;

}

ll first, second;

bool operator<(const frac &r)const{ return (\_\_int128)first \* r.second < (\_\_int128)second \* r.first; }

bool operator<=(const frac &r)const{ return (\_\_int128)first \* r.second <= (\_\_int128)second \* r.first; }

bool operator==(const frac &r)const{ return (\_\_int128)first \* r.second == (\_\_int128)second \* r.first; }

double v(){ return first / (double)second; }

};

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

frac raypoints(vector<pii> &C, pii A, pii d, vector<pair<frac, int>> &R){

assert(d != pii(0, 0));

int g = gcd(abs(d.first), abs(d.second));

d.first /= g, d.second /= g;

vector<pair<frac, int>> L;

for(int i = 0; i+1 < C.size(); i++){

pii v = C[i+1] - C[i];

int a = sign(d/(C[i]-A)), b = sign(d/(C[i+1]-A));

if( a == 0 ) L.emplace\_back(frac(d\*(C[i]-A)/size2(d), 1), b);

if( b == 0 ) L.emplace\_back(frac(d\*(C[i+1]-A)/size2(d), 1), a);

if( a\*b == -1 ) L.emplace\_back(frac((A-C[i])/v, v/d), 6);

}

sort(L.begin(), L.end());

int sz = 0;

for(int i = 0; i < L.size(); i++){

// assert(i+2 >= L.size() || !(L[i].first == L[i+2].first));

if( i+1 < L.size() && L[i].first == L[i+1].first && L[i].second != 6){

int a = L[i].second, b = L[i+1].second;

R.emplace\_back(L[i++].first, a\*b ? a\*b > 0? 4: 6: (1-a-b)/2);

}

else R.push\_back(L[i]);

}

int state = 0; // 0: out, 1: in, 2: line+ccw, 3: line+cw

// R.second -> 0: out->line, 1: in->line, 2: line->out, 3: line->in, 4: convex, 5: concave, 6: out->in, 7: in->out

for(auto &e : R){

int &n = e.second;

if( n == 6 ) n ^= state, state ^= 1;

else if( n == 4 ) n ^= state;

else if( n == 0 ) n = state, state ^= 2;

else if( n == 1 ) n = state^(state>>1), state ^= 3;

}

return frac(g, 1);

}

bool visible(vector<pii> &C, pii A, pii B)

{

if( A == B ) return true;

char I[4] = "356", O[4] = "157";

vector<pair<frac, int>> R;

vector<frac> E;

frac s = frac(0, 1), e = raypoints(C, A, B-A, R);

for(auto e : R){

int &n = e.second, m;

if(\*find(O, O+3, n+'0')) E.emplace\_back(e.first);

if(\*find(I, I+3, n+'0')) E.emplace\_back(e.first);

}

for(int j = 0; j < E.size(); j += 2) if( !(e <= E[j] || E[j+1] <= s) ) return false;

return true;

}

# Miscellaneous

## Newton Method

의 root 하나를 구한다.

## Root Finder using Bezier Curve

n차 다항식에서 0과 1사이의 모든 root를 구한다.

struct state{

state(double s, double e, double\* t):s(s), e(e){

memcpy(coeff, t, sizeof coeff);

}

double s, e, coeff[27];

};

queue<state> Qu;

int n, rc;

double C[27][27]; // pascal's triangle

double coeff\_mono[27], coeff\_bern[27];

double roots[27]; // answer

double matrix[27][27]; // de casteljau matrix

// It is invalid that all coefficients are same sign

int is\_valid(double \*coeff)

{

int posi\_cnt = 0, nega\_cnt = 0;

for(int i = 0; i <= n; i++){

if(coeff[i] > 1e-9) posi\_cnt++;

if(coeff[i] < -1e-9) nega\_cnt++;

}

return posi\_cnt != n+1 && nega\_cnt != n+1;

}

int main()

{

scanf("%d", &n);

for(int i = 0; i <= n; i++)

scanf("%lf", &coeff\_mono[i]);

for(int i = 0; i <= 25; i++){

C[i][0] = 1.0;

for(int j = 1; j <= i; j++)

C[i][j] = C[i-1][j-1] + C[i-1][j];

}

for(int i = 0; i <= n; i++)

for(int j = 0; j <= i; j++)

coeff\_bern[i] += C[i][j] \* coeff\_mono[j] / C[n][j];

Qu.emplace(0.0, 1.0, coeff\_bern);

while(Qu.size()){

state p = Qu.front(); Qu.pop();

double m = (p.s + p.e) / 2;

if(p.e - p.s < 1e-9){

roots[++rc] = m;

continue;

}

for(int i = 0; i <= n; i++) matrix[0][i] = p.coeff[i];

for(int i = 1; i <= n; i++) // de casteljau algorithm

for(int j = 0; j <= n-i; j++)

matrix[i][j] = (matrix[i-1][j] + matrix[i-1][j+1]) / 2.0;

double left\_sub[27] = {}, right\_sub[27] = {};

for(int i = 0; i <= n; i++){

left\_sub[i] = matrix[i][0];

right\_sub[i] = matrix[n-i][i];

}

if(is\_valid(left\_sub)) Qu.emplace(p.s, m, left\_sub);

if(is\_valid(right\_sub)) Qu.emplace(m, p.e, right\_sub);

}

for(int i = 1; i <= rc; i++){

if(i > 1 && roots[i-1]+1e-9 > roots[i]) continue;

printf("%.10lf\n", roots[i]);

}

}

## Pollard rho

// 3,215,031,751

// ll p[] = {2,3,5,7};

// 3,825,123,056,546,413,051

ll p[]={2,3,5,7,11,13,17,19,23};

ll sqa(ll a,ll b,ll c){

ll ret = 1;

while(b){

if(b&1)ret=(\_\_int128)ret\*a%c;

a=(\_\_int128)a\*a%c;

b>>=1;

}

return ret;

}

bool ML(ll x)

{

if(x==2)return true;

if(x<2||!(x&1))return false;

ll y = x-1;

int s = 0;

while((y&1)==0)y>>=1,s++;

for(ll c : p){

if(c>=x)return true;

ll d = sqa(c,y,x);

if(d==1)continue;

for(int j=0;j<s;j++){

ll t = (\_\_int128)d\*d%x;

if(t==1 && d!=1 && d!=x-1)return false;

d = t;

}

if(d!=1)return false;

}

return true;

}

ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}

ll div(ll x)

{

ll a=rand()%(x-1)+1,b=a,i=1,k=2;

ll c=1;

do{

i++;

ll d = gcd(b-a+x,x);

if(d!=1 && d!=x)return d;

if(i==k)b=a,k<<=1;

a=((\_\_int128)a\*a+x-c)%x;

}while(a!=b);

return x;

}

void PR(ll x)

{

if(x==1)return;

if(ML(x)){/\* x is prime factor \*/printf("%lld\n", x);return;}

ll p = div(x);

PR(x/p);

PR(p);

}

## Gauss Quadrature Table

