// UVa1601 The Morning after Halloween

// Rujia Liu

// This code implements the simpliest yet efficient-enough algorithm I'm aware of

// Readers are encouraged to experiment on other algorithms (especially for better efficiency!)

#include<cstdio>

#include<cstring>

#include<cctype>

#include<queue>

using namespace std;

const int maxs = 20;

const int maxn = 150; // 75% cells plus 2 fake nodes

const int dx[]={1,-1,0,0,0}; // 4 moves, plus "no move"

const int dy[]={0,0,1,-1,0};

inline int ID(int a, int b, int c) {

return (a<<16)|(b<<8)|c;

}

int s[3], t[3]; // starting/ending position of each ghost

int deg[maxn], G[maxn][5]; // target cells for each move (including "no move")

inline bool conflict(int a, int b, int a2, int b2) {

return a2 == b2 || (a2 == b && b2 == a);

}

int d[maxn][maxn][maxn]; // distance from starting state

int bfs() {

queue<int> q;

memset(d, -1, sizeof(d));

q.push(ID(s[0], s[1], s[2])); // starting node

d[s[0]][s[1]][s[2]] = 0;

while(!q.empty()) {

int u = q.front(); q.pop();

int a = (u>>16)&0xff, b = (u>>8)&0xff, c = u&0xff;

if(a == t[0] && b == t[1] && c == t[2]) return d[a][b][c]; // solution found

for(int i = 0; i < deg[a]; i++) {

int a2 = G[a][i];

for(int j = 0; j < deg[b]; j++) {

int b2 = G[b][j];

if(conflict(a, b, a2, b2)) continue;

for(int k = 0; k < deg[c]; k++) {

int c2 = G[c][k];

if(conflict(a, c, a2, c2)) continue;

if(conflict(b, c, b2, c2)) continue;

if(d[a2][b2][c2] != -1) continue;

d[a2][b2][c2] = d[a][b][c]+1;

q.push(ID(a2, b2, c2));

}

}

}

}

return -1;

}

int main() {

int w, h, n;

while(scanf("%d%d%d\n", &w, &h, &n) == 3 && n) {

char maze[20][20];

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

fgets(maze[i], 20, stdin);

// extract empty cells

int cnt, x[maxn], y[maxn], id[maxs][maxs]; // cnt is the number of empty cells

cnt = 0;

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

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

if(maze[i][j] != '#') {

x[cnt] = i; y[cnt] = j; id[i][j] = cnt;

if(islower(maze[i][j])) s[maze[i][j] - 'a'] = cnt;

else if(isupper(maze[i][j])) t[maze[i][j] - 'A'] = cnt;

cnt++;

}

// build a graph of empty cells

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

deg[i] = 0;

for(int dir = 0; dir < 5; dir++) {

int nx = x[i]+dx[dir], ny = y[i]+dy[dir];

// "Outermost cells of a map are walls" means we don't need to check out-of-bound

if(maze[nx][ny] != '#') G[i][deg[i]++] = id[nx][ny];

}

}

// add fakes nodes so that in each case we have 3 ghosts. this makes the code shorter

if(n <= 2) { deg[cnt] = 1; G[cnt][0] = cnt; s[2] = t[2] = cnt++; }

if(n <= 1) { deg[cnt] = 1; G[cnt][0] = cnt; s[1] = t[1] = cnt++; }

printf("%d\n", bfs());

}

return 0;

}

// UVa1602 Lattice Animals

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<set>

using namespace std;

struct Cell {

int x, y;

Cell(int x=0, int y=0):x(x),y(y) {};

bool operator < (const Cell& rhs) const {

return x < rhs.x || (x == rhs.x && y < rhs.y);

}

};

typedef set<Cell> Polyomino;

#define FOR\_CELL(c, p) for(Polyomino::const\_iterator c = (p).begin(); c != (p).end(); ++c)

inline Polyomino normalize(const Polyomino &p) {

int minX = p.begin()->x, minY = p.begin()->y;

FOR\_CELL(c, p) {

minX = min(minX, c->x);

minY = min(minY, c->y);

}

Polyomino p2;

FOR\_CELL(c, p)

p2.insert(Cell(c->x - minX, c->y - minY));

return p2;

}

inline Polyomino rotate(const Polyomino &p) {

Polyomino p2;

FOR\_CELL(c, p)

p2.insert(Cell(c->y, -c->x));

return normalize(p2);

}

inline Polyomino flip(const Polyomino &p) {

Polyomino p2;

FOR\_CELL(c, p)

p2.insert(Cell(c->x, -c->y));

return normalize(p2);

}

const int dx[] = {-1,1,0,0};

const int dy[] = {0,0,-1,1};

const int maxn = 10;

set<Polyomino> poly[maxn+1];

int ans[maxn+1][maxn+1][maxn+1];

// add a cell to p0 and check whether it's new. If so, add to the polyonimo set

void check\_polyomino(const Polyomino& p0, const Cell& c) {

Polyomino p = p0;

p.insert(c);

p = normalize(p);

int n = p.size();

for(int i = 0; i < 4; i++) {

if(poly[n].count(p) != 0) return;

p = rotate(p);

}

p = flip(p);

for(int i = 0; i < 4; i++) {

if(poly[n].count(p) != 0) return;

p = rotate(p);

}

poly[n].insert(p);

}

void generate() {

Polyomino s;

s.insert(Cell(0, 0));

poly[1].insert(s);

// generate

for(int n = 2; n <= maxn; n++) {

for(set<Polyomino>::iterator p = poly[n-1].begin(); p != poly[n-1].end(); ++p)

FOR\_CELL(c, \*p)

for(int dir = 0; dir < 4; dir++) {

Cell newc(c->x + dx[dir], c->y + dy[dir]);

if(p->count(newc) == 0) check\_polyomino(\*p, newc);

}

}

// precompute answers

for(int n = 1; n <= maxn; n++)

for(int w = 1; w <= maxn; w++)

for(int h = 1; h <= maxn; h++) {

int cnt = 0;

for(set<Polyomino>::iterator p = poly[n].begin(); p != poly[n].end(); ++p) {

int maxX = 0, maxY = 0;

FOR\_CELL(c, \*p) {

maxX = max(maxX, c->x);

maxY = max(maxY, c->y);

}

if(min(maxX, maxY) < min(h, w) && max(maxX, maxY) < max(h, w))

++cnt;

}

ans[n][w][h] = cnt;

}

}

int main() {

generate();

int n, w, h;

while(scanf("%d%d%d", &n, &w, &h) == 3) {

printf("%d\n", ans[n][w][h]);

}

return 0;

}

// UVa1603 Square Destroyer

// Rujia Liu

// This code implements a variant of an algorithm presented in a book. It's simple yet efficient.

// Readers are encouraged to experiment on other algorithms.

// However, it's still slow for n=5 and m=0 (which is NOT in judge input)

// If you really want an efficient solution, learn DLX (Algorithm X with dancing links)

// DLX is well expained (with code) in my other book <<Beginning Algorithm Contests -- Training Guide>>

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<algorithm>

using namespace std;

const int maxs = 60; // number of squares: 25+16+9+4+1=55

const int maxm = 60; // number of matches: 2\*5\*(5+1)=60

int n, exists[maxm]; // matches

int s, size[maxs], fullsize[maxs], contains[maxs][maxm]; // squares

int best;

inline int row\_match(int x, int y) {

return (2\*n+1)\*x+y;

}

inline int col\_match(int x, int y) {

return (2\*n+1)\*x+n+y;

}

// number of matches in a full n\*n grid

inline int match\_count(int n) {

return 2\*n\*(n+1);

}

void init() {

int m, v;

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

for(int i = 0; i < match\_count(n); ++i) exists[i] = 1;

while(m--) {

scanf("%d", &v);

exists[v-1] = 0;

}

// collect full squares

s = 0;

memset(contains, 0, sizeof(contains));

for(int i = 1; i <= n; i++) // side length

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

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

size[s] = 0;

fullsize[s] = 4\*i; // number of matches in a complete square

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

int a = row\_match(x, y+j); // up

int b = row\_match(x+i, y+j); // down

int c = col\_match(x+j, y); // left

int d = col\_match(x+j, y+i); // right

contains[s][a] = 1;

contains[s][b] = 1;

contains[s][c] = 1;

contains[s][d] = 1;

size[s] += exists[a] + exists[b] + exists[c] + exists[d]; // number of matches now

}

++s;

}

}

int find\_square() {

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

if(size[i] == fullsize[i]) return i;

return -1;

}

void dfs(int dep) {

if(dep >= best) return;

int k = find\_square();

if(k == -1) {

best = dep;

return;

}

// remove a match in that square

for(int i = 0; i < match\_count(n); i++)

if(contains[k][i]) {

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

if(contains[j][i]) size[j]--;

dfs(dep + 1);

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

if(contains[j][i]) size[j]++;

}

}

int main() {

int T;

scanf("%d", &T);

while(T--) {

init();

best = n\*n;

dfs(0);

printf("%d\n", best);

}

return 0;

}

// UVa10603 Fill

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<queue>

using namespace std;

struct Node {

int v[3], dist;

bool operator < (const Node& rhs) const {

return dist > rhs.dist;

}

};

const int maxn = 200 + 5;

int mark[maxn][maxn], dist[maxn][maxn], cap[3], ans[maxn];

void update\_ans(const Node& u) {

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

int d = u.v[i];

if(ans[d] < 0 || u.dist < ans[d]) ans[d] = u.dist;

}

}

void solve(int a, int b, int c, int d) {

cap[0] = a; cap[1] = b; cap[2] = c;

memset(ans, -1, sizeof(ans));

memset(mark, 0, sizeof(mark));

memset(dist, -1, sizeof(dist));

priority\_queue<Node> q;

Node start;

start.dist = 0;

start.v[0] = 0; start.v[1] = 0; start.v[2] = c;

q.push(start);

dist[0][0] = 0;

while(!q.empty()) {

Node u = q.top(); q.pop();

if(mark[u.v[0]][u.v[1]]) continue;

mark[u.v[0]][u.v[1]] = 1;

update\_ans(u);

if(ans[d] >= 0) break;

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

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

if(u.v[i] == 0 || u.v[j] == cap[j]) continue;

int amount = min(cap[j], u.v[i] + u.v[j]) - u.v[j];

Node u2;

memcpy(&u2, &u, sizeof(u));

u2.dist = u.dist + amount;

u2.v[i] -= amount;

u2.v[j] += amount;

int& D = dist[u2.v[0]][u2.v[1]];

if(D < 0 || u2.dist < D){

D = u2.dist;

q.push(u2);

}

}

}

while(d >= 0) {

if(ans[d] >= 0) {

printf("%d %d\n", ans[d], d);

return;

}

d--;

}

}

int main() {

int T, a, b, c, d;

scanf("%d", &T);

while(T--) {

scanf("%d%d%d%d", &a, &b, &c, &d);

solve(a, b, c, d);

}

return 0;

}

// UVa10976 Fractions Again?!

// Rujia Liu

#include<cstdio>

#include<vector>

using namespace std;

int main() {

int k;

while(scanf("%d", &k) == 1 && k) {

vector<int> X, Y;

for(int y = k+1; y <= k\*2; y++) {

// 1/k = 1/x + 1/y => x = ky/(y-k)

if(k\*y%(y-k) == 0)

{ X.push\_back(k\*y/(y-k)); Y.push\_back(y); }

}

printf("%d\n", X.size());

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

printf("1/%d = 1/%d + 1/%d\n", k, X[i], Y[i]);

}

return 0;

}

// UVa11059 Maximum Product

// Rujia Liu

#include<iostream>

using namespace std;

int main() {

int S[20], kase = 0, n;

while(cin >> n && n) {

for(int i = 0; i < n; i++) cin >> S[i];

long long ans = 0;

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

long long v = 1;

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

v \*= S[j];

if(v > ans) ans = v;

}

}

cout << "Case #" << ++kase << ": The maximum product is " << ans << ".\n\n";

}

return 0;

}

// UVa11212 Editing a Book

// Rujia Liu

// This implementation is not very fast, but easy to understand

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn = 9;

int n, a[maxn];

bool is\_sorted() {

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

if(a[i] >= a[i+1]) return false;

return true;

}

// the number of integers with incorrect successor

int h() {

int cnt = 0;

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

if(a[i]+1 != a[i+1]) cnt++;

if(a[n-1] != n) cnt++;

return cnt;

}

bool dfs(int d, int maxd) {

if(d\*3 + h() > maxd\*3) return false;

if(is\_sorted()) return true;

int b[maxn], olda[maxn];

memcpy(olda, a, sizeof(a));

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

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

// cut

int cnt = 0;

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

if(k < i || k > j) b[cnt++] = a[k];

// insert before position k

for(int k = 0; k <= cnt; k++) {

int cnt2 = 0;

for(int p = 0; p < k; p++) a[cnt2++] = b[p];

for(int p = i; p <= j; p++) a[cnt2++] = olda[p];

for(int p = k; p < cnt; p++) a[cnt2++] = b[p];

if(dfs(d+1, maxd)) return true;

memcpy(a, olda, sizeof(a));

}

}

return false;

}

int solve() {

if(is\_sorted()) return 0;

int max\_ans = 5; // after experiments, we found ans <= 5 for n <= 9

for(int maxd = 1; maxd < max\_ans; maxd++)

if(dfs(0, maxd)) return maxd;

return max\_ans;

}

int main() {

int kase = 0;

while(scanf("%d", &n) == 1 && n) {

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

printf("Case %d: %d\n", ++kase, solve());

}

return 0;

}

// UVa12325 Zombie's Treasure Chest

// Rujia Liu

#include<cstdio>

#include<algorithm>

using namespace std;

typedef long long LL;

int main(){

int T;

scanf("%d", &T);

for(int kase = 1; kase <= T; kase++) {

int n, s1, v1, s2, v2;

scanf("%d%d%d%d%d", &n, &s1, &v1, &s2, &v2);

if(s1 > s2){

swap(s1, s2);

swap(v1, v2);

}

LL ans = 0;

if(n / s2 >= 65536){ // both s1 and s2 are small

for(LL i = 0; i <= s1; i++){

ans = max(ans, v2\*i + (n-s2\*i)/s1\*v1);

}

for(LL i = 0; i <= s2; i++){

ans = max(ans, v1\*i + (n-s1\*i)/s2\*v2);

}

}else{ // s2 is large

for(LL i = 0; s2\*i <= n; i++)

ans = max(ans, v2\*i + (n-s2\*i)/s1\*v1);

}

printf("Case #%d: %lld\n", kase, ans);

}

return 0;

}

// UVa129 Krypton Factor

// Rujia Liu

#include<stdio.h>

int n, L, cnt;

int S[100];

int dfs(int cur) { // 返回0表示已经得到解，无须继续搜索

if(cnt++ == n) {

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

if(i % 64 == 0 && i > 0) printf("\n");

else if(i % 4 == 0 && i > 0) printf(" ");

printf("%c", 'A'+S[i]); // 输出方案

}

printf("\n%d\n", cur);

return 0;

}

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

S[cur] = i;

int ok = 1;

for(int j = 1; j\*2 <= cur+1; j++) { // 尝试长度为j\*2的后缀

int equal = 1;

for(int k = 0; k < j; k++) // 检查后一半是否等于前一半

if(S[cur-k] != S[cur-k-j]) { equal = 0; break; }

if(equal) { ok = 0; break; } // 后一半等于前一半，方案不合法

}

if(ok) if(!dfs(cur+1)) return 0; // 递归搜索。如果已经找到解，则直接退出

}

return 1;

}

int main() {

while(scanf("%d%d", &n, &L) == 2 && n > 0) {

cnt = 0;

dfs(0);

}

return 0;

}

// UVa140 Bandwidth

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<vector>

#include<algorithm>

using namespace std;

const int maxn = 10;

int id[256], letter[maxn];

int main() {

char input[1000];

while(scanf("%s", input) == 1 && input[0] != '#') {

// 计算结点个数并给字母编号

int n = 0;

for(char ch = 'A'; ch <= 'Z'; ch++)

if(strchr(input, ch) != NULL) {

id[ch] = n++;

letter[id[ch]] = ch;

}

// 处理输入

int len = strlen(input), p = 0, q = 0;

vector<int> u, v;

for(;;) {

while(p < len && input[p] != ':') p++;

if(p == len) break;

while(q < len && input[q] != ';') q++;

for(int i = p+1; i < q; i++) {

u.push\_back(id[input[p-1]]);

v.push\_back(id[input[i]]);

}

p++; q++;

}

// 枚举全排列

int P[maxn], bestP[maxn], pos[maxn], ans = n;

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

do {

for(int i = 0; i < n; i++) pos[P[i]] = i; // 每个字母的位置

int bandwidth = 0;

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

bandwidth = max(bandwidth, abs(pos[u[i]] - pos[v[i]])); // 计算带宽

if(bandwidth < ans) {

ans = bandwidth;

memcpy(bestP, P, sizeof(P));

}

} while(next\_permutation(P, P+n));

// 输出

for(int i = 0; i < n; i++) printf("%c ", letter[bestP[i]]);

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

}

return 0;

}

// UVa524 Prime Ring Problem

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int is\_prime(int x) {

for(int i = 2; i\*i <= x; i++)

if(x % i == 0) return 0;

return 1;

}

int n, A[50], isp[50], vis[50];

void dfs(int cur) {

if(cur == n && isp[A[0]+A[n-1]]) {

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

if(i != 0) printf(" ");

printf("%d", A[i]);

}

printf("\n");

}

else for(int i = 2; i <= n; i++)

if(!vis[i] && isp[i+A[cur-1]]) {

A[cur] = i;

vis[i] = 1;

dfs(cur+1);

vis[i] = 0;

}

}

int main() {

int kase = 0;

while(scanf("%d", &n) == 1 && n > 0) {

if(kase > 0) printf("\n");

printf("Case %d:\n", ++kase);

for(int i = 2; i <= n\*2; i++) isp[i] = is\_prime(i);

memset(vis, 0, sizeof(vis));

A[0] = 1;

dfs(1);

}

return 0;

}

// UVa725 Division

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int main() {

int n, kase = 0;

char buf[99];

while(scanf("%d", &n) == 1 && n) {

int cnt = 0;

if(kase++) printf("\n");

for(int fghij = 1234; ; fghij++) {

int abcde = fghij \* n;

sprintf(buf, "%05d%05d", abcde, fghij);

if(strlen(buf) > 10) break;

sort(buf, buf+10);

bool ok = true;

for(int i = 0; i < 10; i++)

if(buf[i] != '0' + i) ok = false;

if(ok) {

cnt++;

printf("%05d / %05d = %d\n", abcde, fghij, n);

}

}

if(!cnt) printf("There are no solutions for %d.\n", n);

}

return 0;

}

// UVa1343 The Rotation Game

// Rujia Liu

// This solutions uses IDA\* instead of BFS described in the book, because it's shorter 8-)

// It's shorter because no need for lookup tables and "automatically" lexicographically smallest solution.

#include<cstdio>

#include<algorithm>

using namespace std;

/\*

00 01

02 03

04 05 06 07 08 09 10

11 12

13 14 15 16 17 18 19

20 21

22 23

\*/

// lines E~H are computed with the help of rev[]

int line[8][7]={

{ 0, 2, 6,11,15,20,22}, // A

{ 1, 3, 8,12,17,21,23}, // B

{10, 9, 8, 7, 6, 5, 4}, // C

{19,18,17,16,15,14,13}, // D

};

const int rev[8] = {5, 4, 7, 6, 1, 0, 3, 2}; // reverse lines of each line

// center squares

const int center[8] = {6, 7, 8, 11, 12, 15, 16, 17};

int a[24];

char ans[1000];

bool is\_final() {

for(int i = 0; i < 8; i++)

if (a[center[i]] != a[center[0]]) return false;

return true;

}

int diff(int target) {

int ans = 0;

for(int i = 0; i < 8; i++)

if(a[center[i]] != target) ans++;

return ans;

}

inline int h() {

return min(min(diff(1), diff(2)), diff(3));

}

inline void move(int i) {

int tmp = a[line[i][0]];

for(int j = 0; j < 6; j++) a[line[i][j]] = a[line[i][j+1]];

a[line[i][6]] = tmp;

}

bool dfs(int d, int maxd) {

if(is\_final()) {

ans[d] = '\0';

printf("%s\n", ans);

return true;

}

if(d + h() > maxd) return false;

for(int i = 0; i < 8; i++) {

ans[d] = 'A' + i;

move(i);

if(dfs(d+1, maxd)) return true;

move(rev[i]);

}

return false;

}

int main() {

for(int i = 4; i < 8; i++)

for(int j = 0; j < 7; j++) line[i][j] = line[rev[i]][6-j];

while(scanf("%d", &a[0]) == 1 && a[0]) {

for(int i = 1; i < 24; i++) scanf("%d", &a[i]);

for(int i = 0; i < 24; i++) if(!a[i]) return 0;

if(is\_final()) {

printf("No moves needed\n");

} else {

for(int maxd = 1; ; maxd++)

if(dfs(0, maxd)) break;

}

printf("%d\n", a[6]);

}

return 0;

}

// UVa1354 Mobile Computing

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<vector>

using namespace std;

struct Tree {

double L, R; // distance from the root to the leftmost/rightmost point

Tree():L(0),R(0) {}

};

const int maxn = 6;

int n, vis[1<<maxn];

double r, w[maxn], sum[1<<maxn];

vector<Tree> tree[1<<maxn];

void dfs(int subset) {

if(vis[subset]) return;

vis[subset] = true;

bool have\_children = false;

for(int left = (subset-1)&subset; left; left = (left-1)&subset) {

have\_children = true;

int right = subset^left;

double d1 = sum[right] / sum[subset];

double d2 = sum[left] / sum[subset];

dfs(left); dfs(right);

for(int i = 0; i < tree[left].size(); i++)

for(int j = 0; j < tree[right].size(); j++) {

Tree t;

t.L = max(tree[left][i].L + d1, tree[right][j].L - d2);

t.R = max(tree[right][j].R + d2, tree[left][i].R - d1);

if(t.L + t.R < r) tree[subset].push\_back(t);

}

}

if(!have\_children) tree[subset].push\_back(Tree());

}

int main() {

int T;

scanf("%d", &T);

while(T--) {

scanf("%lf%d", &r, &n);

for(int i = 0; i < n; i++) scanf("%lf", &w[i]);

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

sum[i] = 0;

tree[i].clear();

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

if(i & (1<<j)) sum[i] += w[j];

}

int root = (1<<n)-1;

memset(vis, 0, sizeof(vis));

dfs(root);

double ans = -1;

for(int i = 0; i < tree[root].size(); i++)

ans = max(ans, tree[root][i].L + tree[root][i].R);

printf("%.10lf\n", ans);

}

return 0;

}

// UVa1374 Power Calculus

// Rujia Liu

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxans = 13; // we got it by experimenting

int n, a[maxans+1];

bool dfs(int d, int maxd) {

if(a[d] == n) return true;

if(d == maxd) return false;

int maxv = a[0];

for(int i = 1; i <= d; i++) maxv = max(maxv, a[i]);

if((maxv << (maxd-d)) < n) return false;

// Always use the last value.

// I can't prove it, but we haven't found a counter-example for n <= 1000

for(int i = d; i >= 0; i--) {

a[d+1] = a[d] + a[i];

if(dfs(d+1, maxd)) return true;

a[d+1] = a[d] - a[i];

if(dfs(d+1, maxd)) return true;

}

return false;

}

int solve(int n) {

if(n == 1) return 0;

a[0] = 1;

for(int maxd = 1; maxd < maxans; maxd++) {

if(dfs(0, maxd)) return maxd;

}

return maxans;

}

int main() {

while(scanf("%d", &n) == 1 && n) {

printf("%d\n", solve(n));

}

return 0;

}