パソコン甲子園 2009

【予選問題＊解説】

問題 01　十日市での人気の出店は？

問題のポイント

プログラミングの基礎能力を問う問題です。変数、入出力、四則演算、条件分岐等の基礎知識があれば解くことができます。

問題の解き方

最初に A の合計値を最大値としておき、残り 4 組分を読み込んでいく過程で最大値を更新していき、最終的な名前と最大値を出力します。

講評

提出数：981、正解数：181

不正解の原因の主な理由は次の通りです。

- 繰り返しミス

複数のデータセットを処理せずに一つのデータセットだけを読み込んでプログラムを終了しています。一つのデータセットは、「5 つの出店の販売数を入力としてとり、その中から最大のものを探して出力する」ための入力です。この問題では、これを複数回することを要求しているので、ループを使って繰り返すようにし、終了を示す入力を読み込むまで続ける必要があります。

また、入力の終わりを正しく判定することが出来ずに無限ループになってしまっているプログラムもありました。

- 読み込みミス

入力を正しく読み込んでいない解答も数件ありました。その中でも、特殊なケースですが、全てのデータセットの入力を読み込んでからはじめて処理を行う解答がありました。データセット毎に入力→処理→出力を行うようにしましょう。

- 出力形式の間違い

出力に余計な改行や空白等があったり、店の名前が小文字になっていたり、空白が全角の空白になっていたりするものがありました。出力する文字列は問題文で指定された通りになるように気を付けましょう。また、入力を促すようなメッセージ等も出力してはいけません。

- 最大値の初期化忘れ

不正解の解答で最も多かったのが変数の初期化でのミスでした。最大値を格納しておく変

数は、通常、格納される得る値より小さい値（今回の場合、販売数が 0 の場合が存在しないので 0 以下の数）で初期化すると良いでしょう。この時、この問題では複数のデータセットを処理する必要があるので、初期化の場所を間違えてしまうと、最初のデータセットの最大値が次のデータセットでの初期値になってしまいます。初期化に絡むミスはいろいろな問題で犯しやすいミスなので十分注意を払いましょう。

解答例

```
#include<iostream>
using namespace std;

main(){
   int x, y;
   while( cin >> x >> y && x && y ){
        int m = x+y;
        char ch = 'A';
        for ( int i = 1; i < 5; i++ ){
           cin >> x >> y;
           if ( x+y > m ){
                m = x+y;
                ch = (char)('A' + i);
           }
        }
        cout << ch << " " << m << endl;
   }
}
```

問題 02　野球大会

問題のポイント

多くのアルゴリズムの基本となる整列アルゴリズムを実装します。チーム数の最大値が比較的小さいのでバブルソート等の初等的なアルゴリズムでも解くことができます。

問題の解き方

同順位のチームが存在する場合は、入力順に出力しなければならないので、安定なソートアルゴリズムを選ぶ必要があります。つまり、チームを並び換える前と後で、同順位のチーム順序は変わってはいけないということになります。これをソートの安定性といい、順序が変わらないソートを安定なソート、変わってしまうソートを安定ではないソートと呼びます。安定なソートには、バブルソート、マージソート、挿入ソートなどがあり、安定ではないソートには、選択ソート、クイックソート等があります。従って、この問題を解くには安定なソートを選択しなければなりません。C++で実装する場合は、比較演算子を定義したクラス Team を作ると良いでしょう。または、整列基準を変えて２回安定なソートを行います。

講評

提出数：324、正解数：30

不正解となった解答にはC++のSTLで提供されるsort関数を用いているものがありました。このsort関数はクイックソートで実装されているので、上記の安定ではないソートになるのでそのまま使うと不正解となります。C++のSTLの場合は代わりにstable_sort関数（マージソートで実装されている）を使いましょう。

解答例

```
#include<iostream>
#include<algorithm>
using namespace std;
#define MAX 10
class Team{
   public:
   char name;
   int id, win, defeat, draw;
   Team( int id=0, char name=' ', int win=0, int defeat=0, int draw=0 ):
id(id), name(name), win(win), defeat(defeat), draw(draw){}
   bool operator < ( const Team &t ) const{
        if ( win == t.win ){
           if ( defeat == t.defeat ) return id < t.id;
           else return defeat < t.defeat;
        } else return win > t.win;
   }
};
```

```
main(){
    Team T[MAX];
    int n, x;
    while( cin >> n && n ){
        for ( int i = 0; i < n; i++ ) {
            T[i] = i;
            cin >> T[i].name;
            T[i].defeat = T[i].win = T[i].draw = 0;
            for ( int j = 0; j < n-1; j++ ){
                cin >> x;
                if ( x == 0 ) T[i].win++;
                else if ( x == 1 ) T[i].defeat++;
                else if ( x == 2 ) T[i].draw++;
            }
        }
        stable_sort( T, T + n );
        for ( int i = 0; i < n; i++ ){
            cout << T[i].name << endl;
        }
    }
}
```

問題 03　最大公約数～ユークリッドの互除法～

問題のポイント

説明通りにアルゴリズムを実装できるかが問われていますが、大会参加を通して、新しいアルゴリズムを学んでもらうための問題となっています。

問題の解き方

「X≧Y となるように」という条件を忘れずに、ユークリッドの互除法を実装します。再帰またはループ文で記述することができます。

講評

提出数：996、正解数：187

この問題で不正解だった解答の多くは問題 01 の講評に記載したのと同じミスを犯している解答が目立ちました。この問題特有のミスとしては、「X≧Y となるように値を変数 X,Y に代入」の手順を抜かしてしまっている解答が多かったです。この手順を抜かしてしまうと、X≦Y となるような場合はステップが 1 つ増えてしまいます（適当な値で確認してみてください）。実際にユークリッドの互除法を使う時にはこの手順は抜かしても問題ないのですが、この問題では指定された条件に正確に合わせる必要があります。問題文を正確に読む練習をしましょう。

解答例

```
#include<iostream>
#include<algorithm>
using namespace std;

main(){
   int x, y;
   while ( cin >> x >> y && !(x == 0 && y == 0) ){
        if ( y > x ) swap(x, y);
        int step = 0;
        while(y){
           int t = x % y;
           x = y;
           y = t;
           step++;
        }
        cout << x << " " << step << endl;
   }
}
```

問題 04　芸術家品川のピンチ

問題のポイント

立方体の同一性を全ての視点から調べるために立方体を回転するシミュレーションを実装できるかがポイントです。

問題の解き方

x 軸、y 軸、z 軸を中心に回転させるメソッドを定義したクラス Cube を定義し、与えられた立方体の中でユニークなものの数を数えます。立方体 i と立方体 j が等しいかどうかを判定するためには、立方体 j の向きを固定し、立方体 i を回転させ、全ての向き（24 通り）について立方体 j と比較し、1 つでも一致すれば、等しいと判断します。

講評

提出数：101、正解数：3

特別な知識を必要としない割には正解数が極端に少ない問題でした。提出された解答は、立方体を回転させなければならないことに気付いている解答と気付けていない解答の 2 グループに分かれました。気付いている解答の中でも、ほとんどの解答が全通りの回転を実装出来ていませんでした。

解答例

```
#include<iostream>
#include<string>
using namespace std;
#define MAX 100
class Cube{
   public:
   string f[6];
   Cube(){}
   void roll_z(){ roll(1, 2, 4, 3);}
   void roll_y(){ roll(0, 2, 5, 3);}
   void roll_x(){ roll(0, 1, 5, 4);}
   void roll(int i, int j, int k, int l){
        string t = f[i]; f[i] = f[j]; f[j] = f[k]; f[k] = f[l]; f[l] = t;
   }
};

Cube C[MAX];
int n;

bool eq( Cube c1, Cube c2 ){
   for(int i = 0;i<6;i++)
     if ( c1.f[i] != c2.f[i] ) return false;
   return true;
}
```

```
bool equal( Cube c1, Cube c2 ){
   for(int i = 0;i<6;i++){
       for(int j = 0;j<4;j++){
         if ( eq(c1, c2) ) return true;
           c1.roll_z();
       }
       if ( i % 2 == 0 ) c1.roll_y();
       else c1.roll_x();
   }
   return false;
}

main(){
   while( cin >> n && n ){
       for(int i = 0;i<n;i++) for(int j = 0;j<6;j++) cin >> C[i].f[j];
       int cnt = 0;
       for(int i = 0;i<n;i++) {
          bool hasSame = false;
          for(int j=i+1; j<=n-1;j++) if (equal(C[i],C[j])) {hasSame =
true;break;}
          if ( !hasSame ) cnt++;
       }
       cout << n - cnt << endl;
   }
}
```

問題 05　難儀な人たちが座る椅子

問題のポイント

椅子に座る人達の動きをシミュレートする問題です。問題で定義されたやや複雑な仕様を理解しそれを間違いのないように実装できるかがポイントです。

問題の解き方

プログラムをきちんと構造化することが重要です。例えば、A、B、C、D それぞれの国の人の座り方をシミュレートする関数、setLeft(setA)、setB、setC、setD を定義します。また、椅子の状態を保持する配列の先頭と末尾に番兵（たとえば X という文字）を設置しておくことで、余分な条件式を記述する必要がなくなります。

講評

提出数：133、正解数：6

多くの提出がありながら正解数が少ない問題でした。不正解の原因としては、動きの実装ミスと問題の解釈ミスが最も多いものでした。特にD国人の人が座る際の処理で間違っている解答が目立ちました。また、配列を参照する機会が多いため、配列外参照をして不正解になる解答も多く見られました。シミュレーションの問題を解く際には問題の条件を正しく理解することが非常に重要になります。プログラミングを始める前にどんな機能を実装しなくてはならないのかを整理しておきましょう。

解答例

```
#include<iostream>
using namespace std;
#define MAX 10000
char C[MAX+2];
int n, m;

void setLeft(char ch){
   for ( int i = 1; ; i++ ){
        if ( C[i] == '#' ) { C[i] = ch; return; }
   }
}

void setB(){
   for ( int i = n; i >= 1; i-- ){
        if ( C[i] == '#' && C[i-1] != 'A' && C[i+1] != 'A' ){
           C[i] = 'B'; return;
        }
   }
   setLeft('B');
}

bool isEmpty(){
```

```
    for(int i = 1;i<=n;i++)
       if ( C[i] != '#' ) return false;
    return true;
}

void setC(){
    if ( isEmpty() ){ C[n/2+1] = 'C'; return; }
    for(int i = 1;i<=n;i++){
        if ( C[i] == '#' ) continue;
        if ( C[i+1] == '#' ) { C[i+1] = 'C'; return; }
        if ( C[i-1] == '#' ) { C[i-1] = 'C'; return; }
    }
}

int getDist( int p ){
    int l = 0, r = 0;
    while( C[p+l-1] == '#' ) l--;
    if ( C[p+l-1] == 'X' ) l = -1000000;
    while( C[p+r+1] == '#' ) r++;
    if ( C[p+r+1] == 'X' ) r = 1000000;
    return min( (-1)*l, r);
}

void setD(){
    if ( isEmpty() ){ C[1] = 'D'; return; }
    int d, mi, md = -1;
    for(int i = 1;i<=n;i++){
        if ( C[i] != '#' ) continue;
        d = getDist(i);
        if ( d > md ) {
           md = d; mi = i;
        }
    }
    C[mi] = 'D';
}

main(){
    char ch;
    while( cin >> n >> m && n && m ){
        for(int i = 0;i<=n+1;i++) C[i] = '#';
        C[0] = C[n+1] = 'X';
        for(int i = 1;i<=m;i++){
           cin >> ch;
           if ( ch == 'A' ) setLeft('A');
           else if ( ch == 'B' ) setB();
           else if ( ch == 'C' ) setC();
           else if ( ch == 'D' ) setD();
        }
        for(int i = 1;i<=n;i++) cout << C[i];
        cout << endl;
    }
}
```

問題 06　高校生一人旅～青春の片道切符編～

問題のポイント

典型的な最短経路を求めるためのアルゴリズムを実装できるかがポイントです。

問題の解き方

ダイクストラのアルゴリズム、またはワーシャルフロイドのアルゴリズムで解くことができます。

講評

提出数：51、正解数：11

最短経路問題を解くアルゴリズムを使っている解答はすんなり正解している一方、全ての経路をたどりその中から最短のものを採択しようとする解答などは不正解でした。最短経路問題を解くためのアルゴリズムはいくつか種類があります。提出された解答の中で一番多く使われていたのはワーシャルフロイド法でした。比較的高速ではないわりに実装量が非常に少ないこのアルゴリズムを選択したのは、この問題においては正しい選択でした。ただし、ノード（この問題の場合駅のこと）の数が増えればもっと高速なアルゴリズムを使う必要があります。従って、ダイクストラ法やベルマンフォード法などの最短経路問題を解くためのアルゴリズムをそれぞれの特徴を併せて覚え、使えるようにしておくと良いでしょう。

解答例

```
#include<iostream>
using namespace std;
#define INFTY (1<<21)
#define MAX 100

void floyd(int m, int C[MAX][MAX], int T[MAX][MAX]){
   for(int k = 0;k<m;k++)
     for(int i = 0;i<m;i++)
        for(int j = 0;j<m;j++){
        C[i][j] = min( C[i][j], C[i][k] + C[k][j] );
        T[i][j] = min( T[i][j], T[i][k] + T[k][j] );
   }
}

main(){
   int n, m, a, b, c, t, k, q, C[MAX][MAX], T[MAX][MAX];
   while( cin >> n >> m && !(n == 0 && m == 0) ){
        for(int i = 0;i<m;i++)
          for(int j = 0;j<m;j++) { C[i][j] = T[i][j] = INFTY; }
        for(int i = 0;i<n;i++){
           cin >> a >> b >> c >> t;
           C[a-1][b-1] = C[b-1][a-1] = c;
           T[a-1][b-1] = T[b-1][a-1] = t;
        }
        floyd(m, C, T);
        cin >> k;
        for(int i = 0;i<k;i++){
```

```
            cin >> a >> b >> q;
            if ( q ) cout << T[a-1][b-1] << endl;
            else cout << C[a-1][b-1] << endl;
        }
    }
}
```

問題 07　錬金マスター

問題のポイント

文字列の扱い、再帰関数を応用したアルゴリズムが実装できるかがポイントです。

問題の解き方

レシピの連鎖にサイクルが無いため、再帰によって最小金額を求めることができます。例えば、getCost(string name)で、名前が name である item を作る最小の金額を求めます。最小金額は、item の金額と item をレシピで作る金額のうち、小さい方になります。以下に示すプログラムでは、１つのアイテムが複数のレシピを持てるように実装されています。

講評

提出数：27、正解数：11

解答の指針としては、再帰的に品物の最安値を求めていく方法、または動的計画法によって最安値を更新していく方法の二通りの方法が一般的な解法でしょう。正解解答の殆どは再帰的に解いていました。全体的に文字列の扱いに苦労していたようです。C++の STL 等の機能を用いると簡単にプログラミングすることが出来るので調べて使えるようにしておくと便利です。C を使っている人は C++や Java を使っている人に比べて文字列処理を助けてくれる機能が少ないと言えます。その点で、C++か Java を使えるようになった方が有利でしょう。

解答例

```
#include<iostream>
#include<string>
#include<map>
#include<vector>
using namespace std;

#define MAX 200

class Item{
   public:
   string name;
   int cost;
   vector<vector<string> > recipes;

   Item(){}
   Item( string name, int cost ): name(name),
cost(cost){ recipes.clear(); }

   void addRecipe(vector<string> r ){ recipes.push_back(r);}
};

int n;
map<string, int> N_I;
Item items[MAX];

int getCost( string name ){
   Item item = items[ N_I[name] ];
```

```
    int cost = item.cost;

    for ( int i = 0; i < item.recipes.size(); i++ ){
        vector<string> recipe = item.recipes[i];
        int sum = 0;
        for ( int j = 0; j < recipe.size(); j++ ){
            sum += getCost( recipe[j] );
        }
        cost = min( cost, sum );
    }

    return cost;
}

main(){
    int q, cost, nr, k;
    string name, source;
    while( cin >> n && n){
        for ( int i = 0; i < n; i++ ){
            cin >> name >> cost;
            items[i] = Item(name, cost);
            N_I[name] = i;
        }
        cin >> nr;
        for ( int i = 0; i < nr; i++ ){
            cin >> name >> k;
            vector<string> recipe;
            for ( int j = 0; j < k; j++ ){
                cin >> source;
                recipe.push_back(source);
            }
            items[ N_I[name] ].addRecipe(recipe);
        }
        cin >> name;
        cout << getCost( name ) << endl;
    }
}
```

問題 08　上司のおごり

問題のポイント

素数判定と動的計画法の実装ができないと解けない問題です。

問題の解き方

まずは、エラトステネスの篩によって予め素数表を作っておきます。K[i]を i 番目の品物の金額とし、T^i[j]を i 番目までの金額を使って j を支払えるかのフラグ（1 の場合に支払えるものとする）を記録した配列とすると、以下の式が成り立ちます：

$T^0[j] = 0$

$T^i[j] = T^{i-1}[j] \mid T^{i-1}[j - K[i]]$

この式を基に、動的計画法により T の要素を計算していきます。

最後に、i が素数で T[i]が 1 である i の最大値を求めます。

講評

提出数：37、正解数：6

品物の金額を組み合わせる方法ですが、選び方を全て試す方法だと時間がかかり過ぎますので動的計画法で効率よく求める必要があります。動的計画法は慣れていないと問題に適用するのは難しいので類題を数多く練習して慣れておきましょう。

解答例

```
#include<iostream>
#include<cmath>
using namespace std;
#define KMAX 30
#define VMAX 1000000

int K[KMAX];
unsigned char T[KMAX][VMAX+1];

void eratos ( int n, bool prime[]){
   for ( int i = 0; i <= n; i++ ) prime[i] = false;
   for ( int i = 3; i <= n; i += 2 ) prime[i] = true;
   prime[2] = true;
   int limit = (int)sqrt((double)n) + 1;
   for ( int i = 3; i <= limit; i += 2 ){
       if ( !prime[i] ) continue;
       for ( int j = i + i; j <= n; j += i ) prime[j] = false;
   }
}
main(){
   bool prime[VMAX + 1];
   eratos( VMAX, prime );

   int k, v;
   while( cin >> k >> v && !(k == 0 && v == 0) ){
       for ( int i = 0; i < k; i++ ) cin >> K[i];
```

```
        for ( int j = 0; j <= v; j++ ) T[0][j] = 0;
        for ( int j = K[0]; j <= v; j += K[0] ) T[0][j] = 1;

        for ( int i = 0; i < k; i++ ) T[i][0] = 1;
        for ( int i = 1; i < k; i++ ){
           for ( int j = 1; j < K[i]; j++ ) T[i][j] = T[i-1][j];
           for ( int j = K[i]; j <= v; j++ ){
               T[i][j] = (T[i-1][j] | T[i][j-K[i]]);
           }
        }
        int ans = -1;
        for ( int i = v; i >= 2; i-- ){
           if ( prime[i] && T[k-1][i] ) { ans = i; break; }
        }

        if ( ans == -1 ) cout << "NA" << endl;
        else cout << ans << endl;
    }
}
```

問題 09　会津山スキー場の新企画

問題のポイント

スキー場の上から下までの滑り方の総数を計算できるかがポイントです。

問題の解き方

以下に示すように、動的計画法によって各セルに到達する場合の数を求めます。
マス(y, x)にたどり着く場合の数は、以下のものの合計になります：
マス(y-1, x) にたどり着く場合の数(そこが移動可能なマスの場合)
マス(y-1, x-1) にたどり着く場合の数(そこが移動可能なマスの場合)
マス(y-1, x+1) にたどり着く場合の数(そこが移動可能なマスの場合)
マス(y-2, x) にたどり着く場合の数(そこがジャンプ台の場合)
ただし、ここでは y が増えていく方向に（順番に）計算を行います。
マス（y, x）が移動可能な場所であれば、
マス(y+1, x)
マス(y+1, x-1)
マス(y+1, x+1)
にそれぞれ マス(y, x)の値を加算していきます（移動可能な場合)。
マス(y, x)がジャンプ台の場合は、
マス(y+2, x)
にマス(y, x)の値を加算します（移動可能な場合)。

講評

提出数：38、正解数：8
典型的な動的計画法の問題ですが、スキー場の大きさがそれ程大きくないので全通りの滑り方を試すような方法でも解くことができます。実際、提出された解答のうち動的計画法で解いている解答と全通り試している解答は同じくらいの数でした。不正解となった解答のほとんどがジャンプ台付近の処理でミスがありました。ジャンプ台に入る前、ジャンプ台から出るとき、ジャンプ台の一つ前、などこのあたりのマスの処理を注意すれば正解に近づきます。他には、配列外参照をして不正解になっている解答も多少ありました。
　一般的に全通り試す方法の方が実装は簡単です。問題の要件にあってさえいれば、必ずしも最速の解法でなくても良いので、より楽に実装できる解法を選択することが素早く問題を解くコツです。

解答例

```
#include<iostream>
using namespace std;
#define MAX 15
#define TREE 1
#define JUMP 2
#define SNOW 0
int X, Y;
int G[MAX+2][MAX+2];

int compute(){
   int T[MAX+2][MAX+2];
   for(int y = 0,y<=Y+1;y++) for(int x = 0;x<=X+1;x++) T[y][x] = 0;

   for(int x = 1;x<=X;x++){
    if ( G[0][x] == TREE || G[0][x] == JUMP ) continue;
    int s = -1, t = 1;
    if ( 0 == Y-1 ) s = t = 0;
    for(int dx = s;dx<=t;dx++){ // G[0][x] == SNOW
       if ( G[1][x+dx] == SNOW ) T[1][x+dx]++;
       if ( G[1][x+dx] == JUMP && dx == 0 ) T[1][x+dx]++;
    }
   }

   for(int y = 1;y<=Y-1;y++) for(int x = 1;x<=X;x++){
    if ( G[y][x] == SNOW ){
       int s = -1, t = 1;
       if ( y == Y-1 ) s = t = 0;
       for(int dx = s;dx<=t;dx++){
       if ( G[y+1][x+dx] == SNOW ) T[y+1][x+dx] += T[y][x];
       if ( G[y+1][x+dx] == JUMP && dx == 0 ) T[y+1][x+dx] += T[y][x];
       }
    } else if ( G[y][x] == JUMP ){
       if ( G[y+2][x] != TREE ) T[y+2][x] += T[y][x];
    }
   }

   int sum = 0;
   for(int y = Y;y<=Y+1;y++)
    for(int x = 1;x<=X;x++) sum += T[y][x];
   return sum;
}

main(){
   while( cin >> X >> Y && !(X == 0 && Y == 0) ){
    for(int y = 0;y<=Y-1;y++)
       for(int x = 0;x<=X+1;x++)
              G[y][x] = TREE;

    for(int y = Y;y<=Y+1;y++)
       for(int x = 0;x<=X+1;x++)
               G[y][x] = SNOW;

    for(int y = 0;y<=Y-1;y++)
       for(int x = 1;x<=X;x++)
              cin >> G[y][x];

    cout << compute() << endl;
   }
}
```

問題 10　UFO 撃墜作成

問題のポイント

円と線分の交差判定を行う基本的な幾何学アルゴリズムの実装と、やや複雑なシミュレーションの実装力が問われる問題です。このような幾何の問題では事前にどれだけ知識があるかが問われます。線分と線分の交差判定、点と線分の距離など使いそうなプログラムを事前に用意して持っておくことがポイントです。

問題の解き方

UFO とレーザーが接触したかどうかの判定で点と半直線の距離を算出しなければなりません。UFO の中心点とレーザーとの距離が UFO の半径以下ならば UFO とレーザーは触れ合っていることになります。レーザーの効果がない範囲で UFO と接触しても撃墜されないことに注意しましょう。

講評

提出数：26、正解数：0

解答例

```
#include<iostream>
#include<cfloat>
#include<cmath>
#include<vector>

using namespace std;

#define EPS (1e-8)
#define equals(a, b) (fabs((a) - (b)) < EPS )
#define dle(a, b) (equals(a, b) || a < b )
static const double PI = acos(-1);

class Point{
   public:
   double x, y;

   Point ( double x = 0, double y = 0): x(x), y(y){}

   Point operator + ( Point p ){ return Point(x + p.x, y + p.y); }
   Point operator - ( Point p ){ return Point(x - p.x, y - p.y); }
   Point operator * ( double a ){ return Point(x*a, y*a); }

   double abs() { return sqrt(norm());}
   double norm() { return x*x + y*y; }

   bool operator < ( const Point &p ) const {
        return x != p.x ? x < p.x : y < p.y;
   }

   bool operator == ( const Point &p ) const {
        return fabs(x-p.x) < EPS && fabs(y-p.y) < EPS;
   }
};

typedef Point Vector;
```

```
class Segment{
    public:
    Point p1, p2;
    Segment(Point s = Point(), Point t = Point()): p1(s), p2(t){}
};

typedef Segment Line;

class Circle{
    public:
    Point c;
    double r;
    Circle(Point c = Point(), double r = 0.0): c(c), r(r){}
};

typedef vector<Point> Polygon;

double norm( Vector a ){ return a.x*a.x + a.y*a.y; }
double abs( Vector a ){ return sqrt(norm(a)); }
Point polar( double a, double r ){ return Point(cos(r)*a, sin(r)*a);}
double getDistance( Vector a, Vector b ){ return abs(a - b); }
double dot( Vector a, Vector b ){ return a.x*b.x + a.y*b.y; }
double cross( Vector a, Vector b ){ return a.x*b.y - a.y*b.x; }

static const int COUNTER_CLOCKWISE = 1;
static const int CLOCKWISE = -1;
static const int ONLINE_BACK = 2;
static const int ONLINE_FRONT = -2;
static const int ON_SEGMENT = 0;

int ccw( Point p0, Point p1, Point p2 ){
    Vector a = p1 - p0;
    Vector b = p2 - p0;
    if ( cross(a, b) > EPS ) return COUNTER_CLOCKWISE;
    if ( cross(a, b) < -EPS ) return CLOCKWISE;
    if ( dot(a, b) < -EPS ) return ONLINE_BACK;
    if ( norm(a) < norm(b) ) return ONLINE_FRONT;
    return ON_SEGMENT;
}

double arg(Vector p){
    return atan2(p.y, p.x);
}

double getDistanceLP(Line s, Point p){
    return abs(cross(s.p2 - s.p1, p - s.p1)/abs(s.p2 - s.p1));
}

bool isIntersect( Circle c1, Line l ){
    double d = getDistanceLP(l, c1.c);
    return ( equals(d, c1.r) || d < c1.r );
}

pair<Point, Point> getCrossPoints(Circle c1, Line l ){
    double d = getDistanceLP(l, c1.c);
    Point v = l.p2 - l.p1;
    if ( ccw(l.p1, l.p2, c1.c) == COUNTER_CLOCKWISE ){
        v = polar(abs(v), arg(v)-acos(-1)/2);
    } else {
        v = polar(abs(v), arg(v)+acos(-1)/2);
    }
    double th = acos(d/c1.r);
    Point v1 = polar(c1.r, arg(v) + th );
    Point v2 = polar(c1.r, arg(v) - th );
    return make_pair(c1.c+v1, c1.c+v2);
}


#define NON 0
#define IN 1
#define OUT 2
```

```
double R;

class UFO{
   public:
   Circle body;
   double v, dx, dy;
   int state;

   UFO( Circle c = Circle(Point(0,0),0), double v=0):body(c), v(v){
        Point ct = Point(0, 0);
        Vector a = ct - body.c;
        dx = v*cos(arg(a));
        dy = v*sin(arg(a));
        state = NON;
   }

   void move(){
        if ( state != NON ) return;
        double pred = getDistance(Point(0, 0), body.c);
        if ( v > pred || equals(v, pred)){ state = IN; return; }
        body.c.x += dx;
        body.c.y += dy;
        double d = getDistance(Point(0, 0), body.c);
        if ( R > d || equals(R, d) ) state = IN;
   }
};

#define MAX 100

UFO U[MAX];
int n;

void attack( Point p1, Point p2){
   Line l = Line(p1, p2);
   for ( int i = 0; i < n; i++ ){
        if ( U[i].state != NON ) continue;
        if ( !isIntersect(U[i].body, l) ) continue;
        pair<Point, Point> pp = getCrossPoints(U[i].body, l);
        if ( equals( arg(p2 - p1), arg(pp.first - p1)) && getDistance(p1,
pp.first) > R ) {
           U[i].state = OUT;
        }
        if ( equals( arg(p2 - p1), arg(pp.second - p1)) && getDistance(p1,
pp.second) > R ) {
           U[i].state = OUT;
        }
   }
}

int simulate(){
   while(1){
        int life = 0;
        for ( int i = 0; i < n; i++ ) {
           U[i].move();
           if ( U[i].state == NON ) life++;
        }
        if ( life == 0 ) break;

        int target = -1;
        double md = 1000000;
        for ( int i = 0; i < n; i++ ){
           if ( U[i].state != NON ) continue;
           double d = getDistance(Point(0, 0), U[i].body.c);
           if ( d < md ){
                md = d;
                target = i;
           }
        }
        attack(Point(0, 0), U[target].body.c);
   }

   int cnt = 0;
   for ( int i = 0; i < n; i++ ) if ( U[i].state == IN ) cnt++;
```

```
    return cnt;
}

main(){
    double x, y, r, v;
    while( cin >> R && R ){
        cin >> n;
        for ( int i = 0; i < n; i++ ){
            cin >> x >> y >> r >> v;
            U[i] = UFO(Circle(Point(x, y), r), v);
        }
        cout << simulate() << endl;
    }
}
```