ACM ICPC2013 国内予選　問題 E つながれた風船

- 風船が最も高くあがるケースとして
  1. 一本のロープが垂直に延びて他の２本は緩んでいる
  2. 二本のロープがピンと張っており残りの１本は緩んでいる
  3. 三本のロープともピンとはっている

  の三つのケースが考えられる。ロープの本数は高々10 本なので、ケース１は高々$10\times{}_9C_2=360$ 通り、ケース２も高々${}_{10}C_2\times8=360$ 通り、ケース３は高々${}_{10}C_3=120$ 通りしかないので、これら全ての場合についてその実現可能性と実現可能な場合の風船と高さを求めることで、最も高く上がるケースの高さを求めることができる。
- ケース 1: i 番目のロープがピンと張った状態だとすると、風船の座標は(x_i, y_i, r_i)となる[1]。この位置から他の 2 本のロープの杭の位置までの距離がそのロープのロープ長以下であれば実現可能である。
- ケース 2: ピンと張っているロープの杭の位置を点$P_1 = (x_1, y_1, 0)$, 点$P_2 = (x_2, y_2, 0)$とする。点$P_1 = (x_1, y_1, 0)$, 点$P_2 = (x_2, y_2, 0)$を中心とする半径r_1, r_2の球の交点で z 座標が最大のものを$P = (x, y, z)$ とする。この時$P_0=(x, y, 0)$とすると P_0 は直線 P_1P_2 上の点であり、線分 PP_0 は点 P から x-y 平面への垂線となる。

  $|P_1P_2| = d$, $|P_1P| = d_1$, $|PP_2| = d_2$ とすると

  $$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 \qquad \text{(式 1.1)}$$
  $$r_1^2 = d_1^2 + z^2 \qquad \text{(式 1.2)}$$
  $$r_2^2 = d_2^2 + z^2 \qquad \text{(式 1.3)}$$
  $$d = d_1 + d_2 \qquad \text{(式 1.4)}$$

  (式 1.4)を(式 1.3)に代入し、(式 1.2)を引いて整理すると

  $$d_1 = (d^2 + r_1^2 - r_2^2)/2d \qquad \text{(式 2.1)}$$

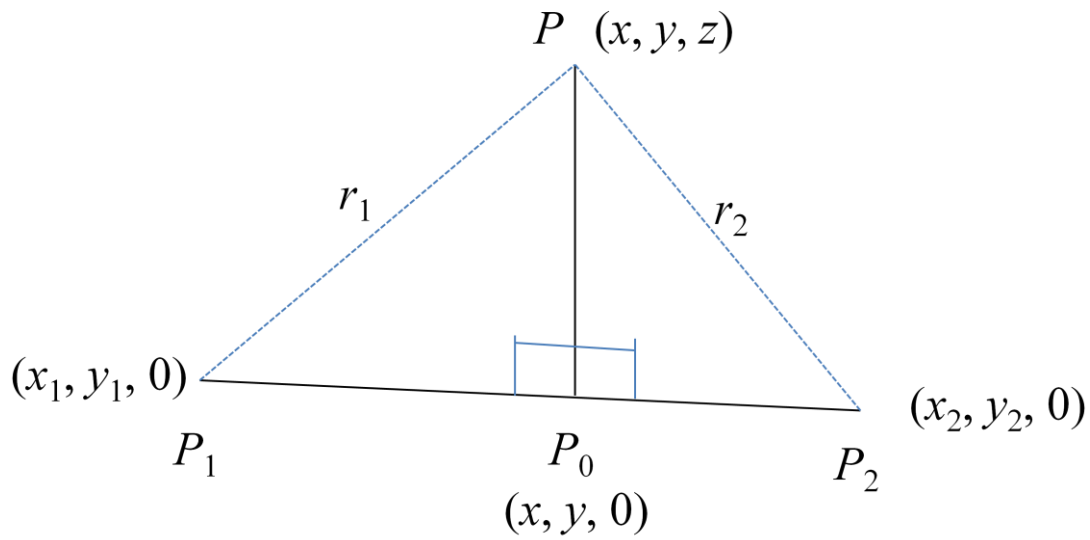
  これらを用いて

  $$z^2 = r_1^2 - d_1^2 = [4\, r_1^2\, r_2^2 - \{d^2 - (r_1^2 + r_2^2)\}^2] / 4d^2 \qquad \text{(式 3.1)}$$
  $$x = x_1 + (x_2 - x_1)\, d_1 / \, d \qquad \text{(式 3.2)}$$
  $$y = y_1 + (y_2 - y_1)\, d_1 / \, d \qquad \text{(式 3.3)}$$

  この時 z2 が非負で、点 P から残りのロープの杭の位置までの距離がそのロ

[1] ここでは i 番目のロープの長さを r_i としている。

ープのロープ長以下であれば実現可能である。



- ケース 3: 3 本のロープの杭の位置を点 $P_1 = (x_1, y_1, 0)$, 点 $P_2 = (x_2, y_2, 0)$, 点 $P_3 = (x_3, y_3, 0)$とする。点 $P_1 = (x_1, y_1, 0)$, 点 $P_2 = (x_2, y_2, 0)$, 点 $P_3 = (x_3, y_3, 0)$を中心とする半径 r_1, r_2, r_3 の球の交点を $P = (x, y, z)$ とすると

$$(x_1 - x)^2 + (y_1 - y)^2 + z^2 = r_1^2 \qquad \text{(式 1.1)}$$

$$(x_2 - x)^2 + (y_2 - y)^2 + z^2 = r_2^2 \qquad \text{(式 1.2)}$$

$$(x_3 - x)^2 + (y_3 - y)^2 + z^2 = r_3^2 \qquad \text{(式 1.3)}$$

(式 1.2)と(式 1.3)から(式 1.1)を引いて整理すると

$$(x_2 - x_1)\,x + (y_2 - y_1)\,y = -(A_2 - A_1)/2 \qquad \text{(式 2.1)}$$

$$(x_3 - x_1)\,x + (y_3 - y_1)\,y = -(A_3 - A_1)/2 \qquad \text{(式 2.1)}$$

ここで、$A_i = r_i^2 - x_i^2 - y_i^2$ $[\, i = 1, 2, 3]$ である。

また、$x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, $A_{i1} = -(A_i - A_1)/2$ とおくと(式 2.1)と(式 2.2)は

$$x_{21}x + y_{21}y = A_{21} \qquad \text{(式 3.1)}$$

$$x_{31}x + y_{31}y = A_{31} \qquad \text{(式 3.2)}$$

(式 3.1)、(式 3.2)より

$$x = (A_{21}y_{31} - A_{31}y_{21}) / D = B \qquad \text{(式 4.1)}$$

$$y = (A_{31}x_{21} - A_{21}x_{31}) / D = C \qquad \text{(式 4.2)}$$

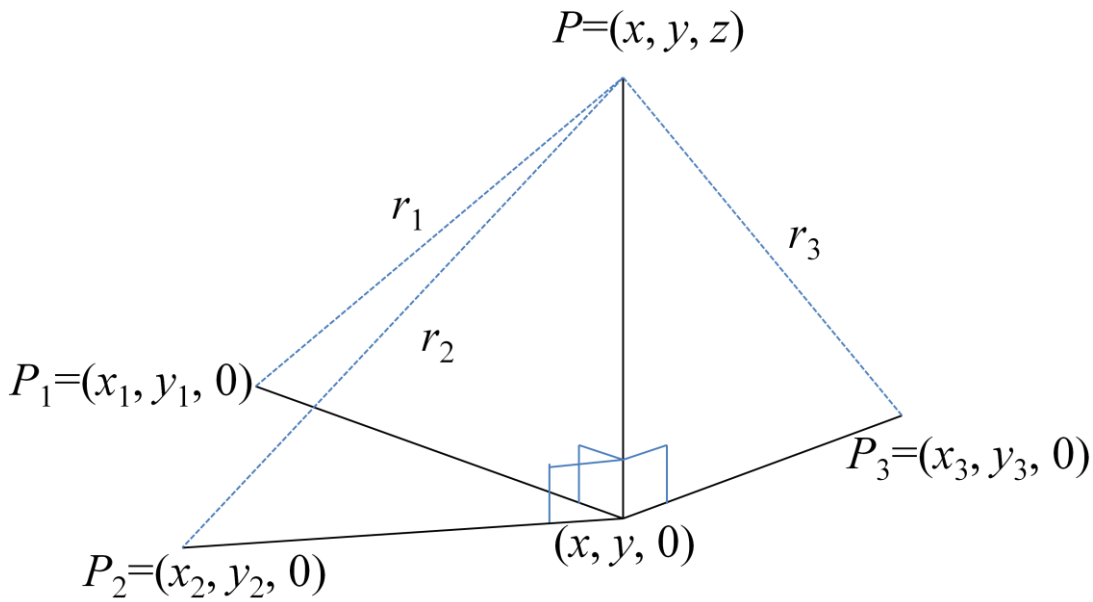
ここで、$D = x_{21}y_{31} - x_{31}y_{21} \neq 0$ と仮定している。

※ $D = 0$ の場合、3 点 P_1, P_2, P_3 は一直線上に並ぶため、交点 P が存在する場合は 2 円の交点（ケース 2）として求めることが出来る。

(式 4.1)、(式 4.2)を(式 1.1)に代入して整理すると

$$z^2 = r_1^2 - (x_1 - B)^2 - (y_1 - C)^2$$

※ $z^2 < 0$ の時は、交点は存在しない



【プログラム例】

```
// ACM ICPC2013 Domestic Problem E
// http://sparth.u-aizu.ac.jp/icpc2013/d_problems.php?lang=jp#section_E
// Filename:        pe.c
// Compile:         cc -lm pe.c
// Execution:        ./a.out < E0 > E0.result
// Check:  diff E0.result E0.ans

#include <stdio.h>
#include <math.h>
#define MAX_N 10 // 杭の本数の最大値
#define MAX_R 300            // ロープの長さの最大値

// 杭のデータを表す構造体
typedef struct {
   int x;   // 杭の位置の x 座標
   int y;   // 杭の位置の y 座標
   int r;   // ロープの長さ
} kui_t;

// 点の座標を表す構造体
typedef struct {
   double x;
   double y;
   double z;
} point_t;

kui_t kui[MAX_N]; // 杭のデータを格納する配列
```

```
int n;                                  // 杭の本数

// 2 点間の距離を求める関数
double len(point_t p1, point_t p2)
{
   double len;
   len = (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) +
      (p1.z - p2.z)*(p1.z - p2.z);
   return sqrt(len);
}

//  2 つの杭を用いてそのロープがたるんでいない状態の最高点座標を求める関数
// そのような状態が存在しない場合は、z 座標を-1 とする
point_t h2(int kn1, int kn2)    // kn1, kn2 =  杭番号
{
   double d,d1;
   double r1,r2, h2;
   double r12, r22, d2;
   point_t p1, p2, p3;
   p1.x = kui[kn1].x;
   p1.y = kui[kn1].y;
   p1.z = 0;
   r1 = kui[kn1].r;
   p2.x = kui[kn2].x;
   p2.y = kui[kn2].y;
   p2.z = 0;
   r2 = kui[kn2].r;
   d = len(p1, p2);
   if(r1 > d+r2 || r2 > d+r1 || r1 + r2 < d){
      p3.z = -1;
      return p3;
   }
   d2 = d*d; r12 = r1*r1; r22 = r2*r2;
   h2 = (-r12*r12-r22*r22-d2*d2+2*r12*r22+2*r12*d2+2*r22*d2)/(4*d2);
   p3.z = sqrt(h2);
   d1 = (r12 - r22 + d2)/(2*d);
   p3.x = ((d-d1)*p1.x + d1*p2.x)/d;
   p3.y = ((d-d1)*p1.y + d1*p2.y)/d;
   return p3;
}

// 3 つの杭を用いてそのロープがたるんでいない状態の最高点座標を求める関数
// そのような状態が存在しない場合は、z 座標を-1 とする
point_t h3(int kn1, int kn2, int kn3)      // kn1, kn2, kn3  は杭番号
{
   point_t ans;
   double a1, a2, a3;
   double a21, a31, d;
   double r, x, y, z, z2;
   double x1, y1, r1;
   double x2, y2, r2;
```

```
    double x3, y3, r3;
    double x21, y21, x31, y31;

    x1 = kui[kn1].x;
    y1 = kui[kn1].y;
    r1 = kui[kn1].r;
    a1 = r1*r1 - x1*x1 - y1*y1;

    x2 = kui[kn2].x;
    y2 = kui[kn2].y;
    r2 = kui[kn2].r;
    a2 = r2*r2 - x2*x2 - y2*y2;

    x3 = kui[kn3].x;
    y3 = kui[kn3].y;
    r3 = kui[kn3].r;
    a3 = r3*r3 - x3*x3 - y3*y3;

    x21 = x2 - x1;
    y21 = y2 - y1;
    x31 = x3 - x1;
    y31 = y3 - y1;
    a21 = -(a2 - a1)/2;
    a31 = -(a3 - a1)/2;

    d = x21*y31 - x31*y21;
    if(d!=0){
        x = (a21*y31 - a31*y21)/d;
        y = (a31*x21 - a21*x31)/d;
        z2 = r1*r1 - (x1 - x)*(x1 - x) - (y1 - y)*(y1 - y);
        if(z2 < 0) z=-1;
        else z = sqrt(z2);
    }
    else
        z = -1;
    ans.x = x;
    ans.y = y;
    ans.z = z;
    return ans;
}

// 風船の位置が点 p になりうるかをチェック
// なりうる場合は 1 を返し、不可能な場合は 0 を返す
// 但し、指定の杭(k1,k2,k3)のロープの長さはチェックしない
int is_ok(point_t p, int k1, int k2, int k3)
{
    int k;    // 杭番号
    point_t kp;          // 杭の座標
    double d;            // 杭と p の距離
    int ok=1;
    for(k=0; k<n; k++){
```

```
      if(k==k1 || k==k2 || k==k3) continue;        // これらの杭はチェックしない
      kp.x = kui[k].x;// 杭の座標をセット
      kp.y = kui[k].y;
      kp.z = 0;
      d = len(p, kp);  // d = 点 p と杭との距離
      if(d > kui[k].r){//  2 点間の距離はロープの長さより長い
        ok = 0;                        // 実現不可能
        break;
      }
    }
    return ok;
}

// ロープが一本張っている状態をチェックし、最高の高さを返す
// ロープ長の最小値が、最高の高さとなる可能性がある
// 可能ならそのロープ長を返し、不可能な場合は 0 を返す
double r1h(void)
{
    double h;
    int i,k;
    point_t p;
    // ロープ長が最小の杭を求める
    // ロープ長が最小の杭が複数ある場合、その最小値は最高の高さにはならないので
    // ロープ長が最小の杭を一つだけチェックすれば良い
    h = MAX_R + 1;
    for(i=0; i<n; i++){
      if(kui[i].r < h){
        h = kui[i].r;
        k = i;
      }
    }
    // k = ロープ長が最小の杭番号, h = そのロープ長
    p.x = kui[k].x;
    p.y = kui[k].y;
    p.z = h;                          // ロープが垂直に延びている場合の高さが最高
    if(is_ok(p, k, k, k))                    // これが実現可能ならその高さを返す
      return h;
    else                                     // 実現不可能なら 0 を返す
      return 0;
}

// ロープが二本張っている状態で、可能な最高の高さを返す
// そのような状態が不可能なら 0 を返す
double r2h(void)
{
    double h = 0;
    int i, j;
    point_t p;
    // 杭 i と杭 j からのロープが張っている状態をチェック
    for(i=0; i<n; i++){
```

```
    for(j=i+1; j<n; j++){
      p = h2(i,j);                  // 杭 i と杭 j からのロープが張っている状態の最高点の位置
      if(p.z < 0) continue;         // そのような状態は存在しないので次の杭をチェック
      if(is_ok(p, i, j, j)){        // 他の杭のロープ長も考慮してその状態が実現可能で
        if(p.z > h)// これまで求まっている高さよりその位置が高ければ
          h = p.z;// 高さの情報を更新
      }
    }
  }
  return h;
}

// ロープが三本張っている状態で、可能な最高の高さを返す
// そのような状態が不可能なら 0 を返す
double r3h(void)
{
  int i, j, k;
  point_t p;
  double h = 0;
  // 杭 i,j,k からのロープが張っている状態をチェック
  for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
      for(k=j+1; k<n; k++){
        p = h3(i,j,k);          // p = 杭 i,j,k からのロープが張っている位置
        if(p.z == -1) continue;         // そのような位置がなければ次の杭をチェック
        if(is_ok(p, i, j, k)){  // 他の杭のロープ長も考慮してその状態が実現可能なら
          if(p.z > h)           // これまで求まっている高さよりその位置が高ければ
            h = p.z;                    // 高さの情報を更新
        }
      }
    }
  }
  return h;
}

int main()
{
  int i,j,k,m;
  double max_h, h;
  while(1){
    scanf("%d¥n", &n);
    if(n==0) break;
    for(i=0; i<n; i++)
      scanf("%d %d %d¥n", &kui[i].x, &kui[i].y, &kui[i].r);
    // ロープ一本が張っている状態
    // ロープ長の最小値が、最高の高さとなる可能性がある
    max_h = r1h();
    if(max_h == 0) {
      // ロープ２本が張っている状態をチェック
      max_h = r2h();
```

```
            h = r3h();
            if(h > max_h) max_h = h;
        }
        printf("%.7lf¥n", max_h);
    }
}
```