OJ 10652

Board Wrapping

Prof. Edson Alves

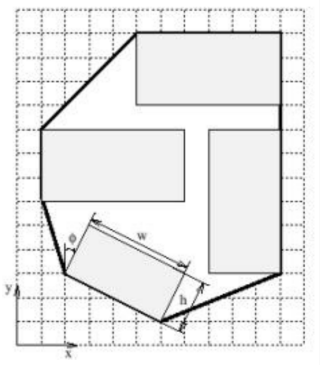
Faculdade UnB Gama

OJ 10652 – Board Wrapping

Problema

The small sawmill in Mission, British Columbia, has developed a brand new way of packaging boards for drying. By fixating the boards in special moulds, the board can dry efficiently in a drying room.

Space is an issue though. The boards cannot be too close, because then the drying will be too slow. On the other hand, one wants to use the drying room efficiently.



Problema

Looking at it from a 2-D perspective, your task is to calculate the fraction between the space occupied by the boards to the total space occupied by the mould. Now, the mould is surrounded by an aluminium frame of negligible thickness, following the hull of the boards' corners tightly. The space occupied by the mould would thus be the interior of the frame.

Entrada e saída

Input

On the first line of input there is one integer, $N \leq 50$, giving the number of test cases (moulds) in the input. After this line, N test cases follow. Each test case starts with a line containing one integer n, $1 < n \leq 600$, which is the number of boards in the mould. Then n lines follow, each with five floating point numbers x, y, w, h, ϕ where $0 \leq x, y, w, h \leq 10000$ and $-90^o < \phi \leq 90^o$. The x and y are the coordinates of the center of the board and w and h are the width and height of the board, respectively. ϕ is the angle between the height axis of the board to the y-axis in degrees, positive clockwise. That is, if $\phi = 0$, the projection of the board on the x-axis would be w. Of course, the boards cannot intersect.

Entrada e saída

Output

For every test case, output one line containing the fraction of the space occupied by the boards to the total space in percent. Your output should have one decimal digit and be followed by a space and a percent sign (‘%’).

Note: The Sample Input and Sample Output corresponds to the given picture

Exemplo de entradas e saídas

Sample Input

```
1
4
4 7.5 6 3 0
8 11.5 6 3 0
9.5 6 6 3 90
4.5 3 4.4721 2.2361 26.565
```

Sample Output

```
64.3 %
```

Solução $O(TN \log N)$

- A solução consiste em três etapas

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura
- Em seguida, é preciso determinar os vértices de cada placa

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura
- Em seguida, é preciso determinar os vértices de cada placa
- Pode-se assumir que eles estão inicialmente com o centro na origem, fazer a rotação em sentido horário e, em seguida, transladar os pontos para a posição correta

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura
- Em seguida, é preciso determinar os vértices de cada placa
- Pode-se assumir que eles estão inicialmente com o centro na origem, fazer a rotação em sentido horário e, em seguida, transladar os pontos para a posição correta
- Determinados estes pontos, os limites do polígono correspondem ao envoltório convexo

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura
- Em seguida, é preciso determinar os vértices de cada placa
- Pode-se assumir que eles estão inicialmente com o centro na origem, fazer a rotação em sentido horário e, em seguida, transladar os pontos para a posição correta
- Determinados estes pontos, os limites do polígono correspondem ao envoltório convexo
- A área do polígono pode ser determinada através da expressão que computa esta área por meio dos vértices do polígono

Solução $O(TN \log N)$

- A solução consiste em três etapas
- A primeira é determinar a área total ocupada pelas placas
- Basta somar a área individual de cada placa, que é o produto da base pela altura
- Em seguida, é preciso determinar os vértices de cada placa
- Pode-se assumir que eles estão inicialmente com o centro na origem, fazer a rotação em sentido horário e, em seguida, transladar os pontos para a posição correta
- Determinados estes pontos, os limites do polígono correspondem ao envoltório convexo
- A área do polígono pode ser determinada através da expressão que computa esta área por meio dos vértices do polígono
- A resposta será a diferença entre ambas áreas, em porcentagem

Solução com complexidade $O(TN \log N)$

```cpp
#include <bits/stdc++.h>

using namespace std;

const double EPS { 1e-5 }, PI { acos(-1.0) };

bool equals(double a, double b)
{
    return fabs(a - b) < EPS;
}

struct Point {
public:
    double x, y;

    double distance(const Point& P) const
    {
        return hypot(x - P.x, y - P.y);
    }
```

Solução com complexidade $O(TN \log N)$

```
21      Point translate(double dx, double dy) const
22      {
23          return Point { x + dx, y + dy };
24      }
25
26      Point rotate(double angle) const
27      {
28          return Point { x*cos(angle) - y*sin(angle), x*sin(angle) + y*cos(angle) };
29      }
30  };
31
32  double D(const Point& P, const Point& Q, const Point& R)
33  {
34      return (P.x * Q.y + P.y * R.x + Q.x * R.y) - (R.x * Q.y + R.y * P.x + Q.x * P.y);
35  }
```

Solução com complexidade $O(TN \log N)$

```
37 struct Polygon {
38     vector<Point> vs;
39     int n;
40 
41     Polygon(const vector<Point>& vertices) : vs(vertices), n(vs.size()) { vs.push_back(vs[0]); }
42 
43     double area() const
44     {
45         double a = 0;
46 
47         for (int i = 0; i < n; ++i)
48         {
49             a += vs[i].x * vs[i+1].y;
50             a -= vs[i+1].x * vs[i].y;
51         }
52 
53         return 0.5 * fabs(a);
54     }
55 };
```

Solução com complexidade $O(TN \log N)$

```
57 Point pivot(vector<Point>& P)
58 {
59     size_t idx = 0;
60
61     for (size_t i = 1; i < P.size(); ++i)
62         if (P[i].y < P[idx].y or (equals(P[i].y, P[idx].y) and P[i].x > P[idx].x))
63             idx = i;
64
65     swap(P[0], P[idx]);
66
67     return P[0];
68 }
69
70 void sort_by_angle(vector<Point>& P)
71 {
72     auto P0 = pivot(P);
73
74     sort(P.begin() + 1, P.end(), [&](const Point& A, const Point& B)
75         {
```

Solução com complexidade $O(TN \log N)$

```
76              if (equals(D(P0, A, B), 0))
77                  return A.distance(P0) < B.distance(P0);
78
79              auto alfa = atan2(A.y - P0.y, A.x - P0.x);
80              auto beta = atan2(B.y - P0.y, B.x - P0.x);
81
82              return alfa < beta;
83          }
84      );
85  }
86
87  Polygon convex_hull(const vector<Point>& points)
88  {
89      vector<Point> P(points);
90      auto n = P.size();
91
92      if (n <= 3)
93          return Polygon(P);
94
95      sort_by_angle(P);
```

Solução com complexidade $O(TN \log N)$

```
97      vector<Point> s;
98      s.push_back(P[n - 1]);
99      s.push_back(P[0]);
100     s.push_back(P[1]);
101
102     size_t i = 2;
103
104     while (i < n)
105     {
106         auto j = s.size() - 1;
107
108         if (D(s[j - 1], s[j], P[i]) > 0)
109             s.push_back(P[i++]);
110         else
111             s.pop_back();
112     }
113
114     s.pop_back();
115     return Polygon(s);
116 }
```

Solução com complexidade $O(TN \log N)$

```
118 int main()
119 {
120     int T;
121     cin >> T;
122 
123     while (T--)
124     {
125         int n;
126         cin >> n;
127 
128         vector<Point> points;
129         double boards_area = 0;
130 
131         while (n--)
132         {
133             double x, y, w, h, theta;
134             cin >> x >> y >> w >> h >> theta;
135 
136             theta /= 180.0;
137             theta *= PI;
```

Solução com complexidade $O(TN \log N)$

```
139             double xv = w / 2, yv = h / 2;
140
141             vector<Point> ps { Point { xv, yv }, Point { -xv, yv },
142                 Point { -xv, -yv }, Point { xv, -yv } };
143
144             for (auto p : ps)
145             {
146                 auto q = p.rotate(-theta);
147                 points.push_back(q.translate(x, y));
148             }
149
150             boards_area += w * h;
151         }
152
153         auto ch = convex_hull(points);
154         auto total = ch.area();
155         auto percent = 100.0 * boards_area / total;
156
157         printf("%.1f %%\n", percent);
158     }
```