Geometria Computacional

Objetos Tridimensionais: problemas resolvidos

Prof. Edson Alves

2018

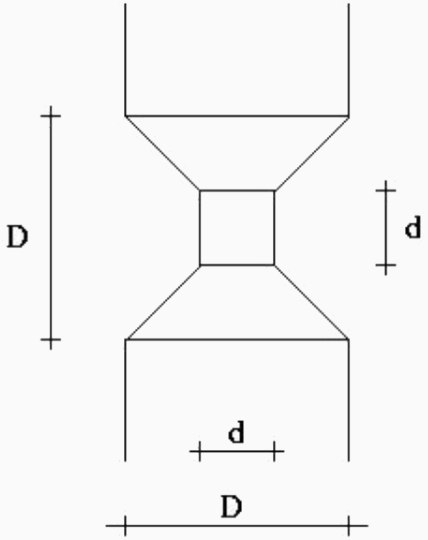
Faculdade UnB Gama

Sumário

UVA 10297 – Beavergnaw

Problema

When chomping a tree the beaver cuts a very specific shape out of the tree trunk. What is left in the tree trunk looks like two frustums of a cone joined by a cylinder with the diameter the same as its height. A very curious beaver tries not to demolish a tree but rather sort out what should be the diameter of the cylinder joining the frustums such that he chomped out certain amount of wood. You are to help him to do the calculations.



Problema

We will consider an idealized beaver chomping an idealized tree. Let us assume that the tree trunk is a cylinder of diameter D and that the beaver chomps on a segment of the trunk also of height D. What should be the diameter d of the inner cylinder such that the beaver chmped out V cubic units of wood?

Entrada e saída

Input

Input contains multiple cases each presented on a separate line. Each line contains two integer numbers D and V separated by whitespace. D is the linear units and V is in cubic units. V will not exceed the maximum volume of wood that the beaver can chomp. A line with $D = 0$ and $V = 0$ follows the last case.

Output

For each case, one line of output should be produced containing one number rounded to three fractional digits giving the value of d measured in linear units.

Exemplo de entradas e saídas

Sample Input

```
10 250
20 2500
25 7000
50 50000
0 0
```

Sample Output

```
8.054
14.775
13.115
30.901
```

Solução $O(T \log D)$

- Um cone reto com diâmetro D tem volume igual a

$$V_C = D\pi \left(\frac{D}{2}\right)^2 = \frac{\pi D^3}{4}$$

- Após o castor roer um volume igual a V, restará um volume $L = V_c - V$
- Este volume deve ser composto por dois troncos de cone com altura $(D-d)/2$ e bases com diâmetro D e d, mais um cone reto de altura e diâmetros iguais a d
- Em notação matemática

$$L = 2\left[\frac{1}{3}\pi\left(\frac{D-d}{2}\right)\left(\frac{D^2}{4} + \frac{Dd}{4} + \frac{d^2}{4}\right)\right] + \frac{\pi d^3}{4}$$

Solução $O(T \log D)$

- Caso o castor não coma nada, d deve ter valor máximo igual a D
- Caso o castor coma toda a região, d assume o valor mínimo igual a zero
- Assim, dado o valor V, é possível determinar o valor de d por meio de uma busca binária
- A complexidade desta solução é $O(T \log D)$, onde T é o número de casos de teste

Solução com complexidade $O(T \log D)$

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

using namespace std;

const double PI { acos(-1.0) }, EPS { 1e-6 };

double solve(double D, double V)
{
    auto L = (PI * D * D * D)/4.0 - V;
    auto a = 0.0, b = D, d = 0.0;

    while (a <= b)
    {
        d = (a + b)/2;
        auto R = 2*((PI/3.0)*((D - d)/2.0)*(D*D + D*d + d*d)/4.0)
            + PI*(d * d * d)/4.0;

        if (fabs(L - R) < EPS)
            break;
        else if (L > R)
            a = d;
```

Solução com complexidade $O(T \log D)$

```
22          else
23              b = d;
24      }
25
26      return d;
27  }
28
29  int main()
30  {
31      int D, V;
32
33      while (cin >> D >> V, D | V)
34      {
35          auto ans = solve(D, V);
36
37          cout.precision(3);
38          cout << fixed << ans << '\n';
39      }
40
41      return 0;
42  }
```

Solução $O(T)$

- Para resolver cada caso de teste com complexidade $O(1)$, é preciso interpretar o volume restante no tronco de forma ligeiramente diferente
- Observe que este volume pode ser visto como a união de dois cones retos de altura $D/2$ e bases com diâmetro D e um sólido S, composto por um cone reto de altura e diâmetros iguais a d, subtraído de dois cones retos de altura e raio iguais a $d/2$.
- Em notação matemática

$$L = 2\left[\frac{1}{3}\pi\left(\frac{D}{2}\right)\left(\frac{D}{2}\right)^2\right] + \left\{\pi d\left(\frac{d}{2}\right)^2 - 2\left[\frac{1}{3}\pi\left(\frac{d}{2}\right)\left(\frac{d}{2}\right)^2\right]\right\}$$

Solução $O(T)$

- Esta expressão pode ser reescrita como

$$\begin{aligned} L &= \frac{1}{12}\pi D^3 + \frac{1}{4}\pi d^3 - \frac{1}{12}\pi d^3 \\ \frac{1}{4}\pi D^3 - V &= \frac{1}{12}\pi D^3 + \frac{1}{6}\pi d^3 \\ \frac{1}{6}\pi D^3 - V &= \frac{1}{6}\pi d^3 \end{aligned}$$

- Assim,

$$d = \sqrt[3]{\frac{6}{\pi}\left(\frac{1}{6}\pi D^3 - V\right)} = \sqrt[3]{D^3 - \frac{6V}{\pi}}$$

Solução com complexidade $O(T)$

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

const double PI { acos(-1.0) };

double solve(double D, double V)
{
    return cbrt(D*D*D - 6.0*V/PI);
}

int main()
{
    int D, V;

    while (std::cin >> D >> V, D | V)
    {
        std::cout.precision(3);
        std::cout << std::fixed << solve(D, V) << '\n';
    }

    return 0;
}
```

URI 1730 – Global Roaming

Problema

Hoje em dia vários dispositivos móveis de comunicação dependem de uma vista direta para um satélite. Portanto, para os provedores de comunicação é crucial saber onde os seus serviços estão disponíveis. Sua tarefa é identificar os locais que têm uma vista direta para um satélite particular, ou seja, este satélite deve estar acima do horizonte. Para facilitar as coisas, você pode assumir que a Terra é uma esfera perfeita com um raio de 6378km (montanhas serão adicionadas no próximo ano...). O satélite é um objeto *pointlike* acima da superfície terrestre.

Entrada e saída

Entrada

A entrada consiste de vários casos de teste. Em cada caso de teste, a primeira linha contém o número de localizações N a serem verificados, seguido pela a posição do satélite: a sua latitude, a longitude (ambas em grau) e sua altura (em km) acima da superfície terrestre.

Cada uma das seguintes linhas N contém um local na superfície terrestre: o nome da localidade (uma string com menos de 60 caracteres ASCII que não contém espaços em branco), seguido por sua latitude e longitude (ambos em graus). A entrada é terminada por $N = 0$.

Entrada e saída

Saída

Para cada caso de teste o seu programa deverá imprimir uma linha dizendo "Test case K:", onde K é o número da instância atual. Então nas seguintes linhas, imprimir em linhas separadas, os nomes das localidades onde o satélite é visível na mesma ordem em que aparecem no arquivo de entrada. Imprima uma linha em branco após cada instância.

Exemplo de entradas e saídas

Exemplo de Entrada

```
3 20.0 -60.0 150000000.0
Ulm 48.406 10.002
Jakarta -6.13 106.75
Honolulu 21.32 -157.83
2 48.4 10 0.5
Ulm 48.406 10.002
Honolulu 21.32 -157.83
0 0.0 0.0 0.0
```

Exemplo de Saída

```
Test case 1:
Ulm
Honolulu

Test case 2:
Ulm
```

Solução

- Para computar o ângulo que o satélite faz com a linha do horizonte, é preciso converter as coordenadas geográficas (latitude, longitue e altura) em coordenadas esféricas
- É preciso atentar ao fato de que a latitude λ se relaciona com colatitude φ das coordenadas esféricas através da relação $\lambda = 90^o - \varphi$.
- Assim, sendo θ a longitude, a mudança de variáveis é dada por

$$\begin{aligned} x &= r\cos\theta\cos\lambda \\ y &= r\sin\theta\cos\lambda \\ z &= r\sin\lambda, \end{aligned}$$

  pois $\sin(90^o - \lambda) = \cos\lambda$ e $\cos(90^o - \lambda) = \sin\lambda$
- Se $\vec{P}$ é o vetor posição da cidade e $\vec{Q}$ o vetor posição do satélite, o ângulo em relação ao horizonte é o ângulo entre $\vec{P}$ e $\vec{u} = \vec{Q} - \vec{P}$
- É preciso tomar cuidado com possíveis erros de precisão no argumento da função `acos()`, cujo domínio é o intervalo $[-1, 1]$

Solução AC

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

using namespace std;

const double R { 6378 };
const double PI { acos(-1) };

struct Query {
    string name;
    double lat, lon;
};

struct Point3D {
    double x, y, z;

    Point3D operator-(const Point3D& P) const
    {
        return Point3D { x - P.x, y - P.y, z - P.z };
    }

```

Solução AC

```
21      double operator*(const Point3D& P) const
22      {
23          return x * P.x + y * P.y + z * P.z;
24      }
25
26      void normalize()
27      {
28          auto len = length();
29
30          x /= len;
31          y /= len;
32          z /= len;
33      }
34
35      double length() const
36      {
37          return sqrt(x * x + y * y + z * z);
38      }
39
```

Solução AC

```
40      static Point3D from_polar(double R, double lat, double lon)
41      {
42          auto gamma = lat * PI / 180.0;
43          auto theta = lon * PI / 180.0;
44
45          // latitude = (90 - gamma), em relação às coordenadas esféricas
46          auto x = R*cos(theta)*cos(gamma);
47          auto y = R*sin(theta)*cos(gamma);
48          auto z = R*sin(gamma);
49
50          return Point3D { x, y, z };
51      }
52  };
53
54  double angle(Point3D& P, Point3D& Q)
55  {
56      P.normalize();
57      Q.normalize();
58
59      auto dot = P * Q;
60
```

Solução AC

```
61      if (dot > 0)
62          dot = min(dot, 1.0);
63
64      if (dot < 0)
65          dot = max(dot, -1.0);
66
67      return acos(dot);
68  }
69
70  vector<string>
71  solve(double lat, double lon, double h, const vector<Query>& qs)
72  {
73      vector<string> ans;
74
75      auto Q = Point3D::from_polar(h + R, lat, lon);
76
77      for (const auto& q : qs)
78      {
79          auto P = Point3D::from_polar(R, q.lat, q.lon);
80          auto u = Q - P;
81          auto alpha = angle(P, u);
```

Solução AC

```
82
83          if (alpha < PI/2.0)
84              ans.push_back(q.name);
85      }
86
87      return ans;
88  }
89
90  int main()
91  {
92      ios::sync_with_stdio(false);
93      int N, test = 0;
94
95      while (cin >> N, N)
96      {
97          double lat, lon, h;
98          cin >> lat >> lon >> h;
99
100         vector<Query> qs;
101
102         while (N--)
```

Solução AC

```
103         {
104             string name;
105             double x, y;
106 
107             cin >> name >> x >> y;
108 
109             qs.push_back({ name, x, y });
110         }
111 
112         auto ans = solve(lat, lon, h, qs);
113 
114         cout << "Test case " << ++test << ":\n";
115 
116         for (const auto& name : ans)
117             cout << name << '\n';
118 
119         cout << '\n';
120     }
121 
122     return 0;
123 }
```

Referências

1. UVA 10297 – Beavergnaw
2. URI 1730 – Global Roaming