Geometria Computacional

Retas: Algoritmos

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Classificação de retas

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
    1. nenhum ponto em comum (retas paralelas)

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
  1. nenhum ponto em comum (retas paralelas)
  2. um único ponto em comum (retas concorrentes)

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
  1. nenhum ponto em comum (retas paralelas)
  2. um único ponto em comum (retas concorrentes)
  3. todos os pontos em comum (retas coincidentes)

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
  1. nenhum ponto em comum (retas paralelas)
  2. um único ponto em comum (retas concorrentes)
  3. todos os pontos em comum (retas coincidentes)
- O coeficiente angular é a chave para tal classificação: retas com coeficientes angulares distintos são concorrentes

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
    1. nenhum ponto em comum (retas paralelas)
    2. um único ponto em comum (retas concorrentes)
    3. todos os pontos em comum (retas coincidentes)
- O coeficiente angular é a chave para tal classificação: retas com coeficientes angulares distintos são concorrentes
- Caso duas retas tenham coeficientes angulares iguais, é necessário verificar também o coeficiente linear: se iguais, as retas são coincidentes

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
  1. nenhum ponto em comum (retas paralelas)
  2. um único ponto em comum (retas concorrentes)
  3. todos os pontos em comum (retas coincidentes)
- O coeficiente angular é a chave para tal classificação: retas com coeficientes angulares distintos são concorrentes
- Caso duas retas tenham coeficientes angulares iguais, é necessário verificar também o coeficiente linear: se iguais, as retas são coincidentes
- Retas com coeficientes angulares iguais e coeficientes lineares distintos são paralelas

Retas paralelas, concorrentes e coincidentes

- Em relação às possíveis interseções entre duas retas, há três cenários possíveis:
    1. nenhum ponto em comum (retas paralelas)
    2. um único ponto em comum (retas concorrentes)
    3. todos os pontos em comum (retas coincidentes)
- O coeficiente angular é a chave para tal classificação: retas com coeficientes angulares distintos são concorrentes
- Caso duas retas tenham coeficientes angulares iguais, é necessário verificar também o coeficiente linear: se iguais, as retas são coincidentes
- Retas com coeficientes angulares iguais e coeficientes lineares distintos são paralelas
- A implementação destas verificações é trivial na representação baseada na equação reduzida, sendo necessário apenas o cuidado no trato do caso das retas verticais

Exemplo de implementação de classificação de retas em C++

```cpp
template<typename T>
struct Line {
    // Membros e construtores (equação reduzida)

    bool operator==(const Line& r) const    // Verdadeiro se coincidentes
    {
        if (vertical != r.vertical || !equals(m, r.m)) return false;

        return equals(b, r.b);
    }

    bool parallel(const Line& r) const      // Verdadeiro se paralelas
    {
        if (vertical && r.vertical) return b != r.b;
        if (vertical || r.vertical) return false;

        return equals(m, r.m) && !equals(b, r.b);
    }
};
```

Exemplo de implementação de classificação de retas em C++

```
template<typename T>
struct Line {
    // Membros e construtores (equação geral}

    bool operator==(const Line& r) const
    {
        auto k = a ? a : b;
        auto s = r.a ? r.a : r.b;

        return equals(a*s, r.a*k) && equals(b*s, r.b*k) && equals(c*s, r.c*k);
    }

    bool parallel(const Line& r) const
    {
        auto det = a*r.b - b*r.a;

        return det == 0 and !(*this == r);
    }
};
```

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1
- Outra maneira de checar se duas retas são perpendiculares é escolher dois pontos pertencentes a cada reta e montar dois vetores $\vec{u}$ e $\vec{v}$

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1
- Outra maneira de checar se duas retas são perpendiculares é escolher dois pontos pertencentes a cada reta e montar dois vetores $\vec{u}$ e $\vec{v}$
- Estes pontos podem ser escolhidos de forma eficiente, fazendo $x = 0$ e $y = 0$ (caso a reta não passe na origem)

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1
- Outra maneira de checar se duas retas são perpendiculares é escolher dois pontos pertencentes a cada reta e montar dois vetores $\vec{u}$ e $\vec{v}$
- Estes pontos podem ser escolhidos de forma eficiente, fazendo $x = 0$ e $y = 0$ (caso a reta não passe na origem)
- Se o produto interno dos dois vetores for igual a zero, as retas são perpendiculares

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1
- Outra maneira de checar se duas retas são perpendiculares é escolher dois pontos pertencentes a cada reta e montar dois vetores $\vec{u}$ e $\vec{v}$
- Estes pontos podem ser escolhidos de forma eficiente, fazendo $x = 0$ e $y = 0$ (caso a reta não passe na origem)
- Se o produto interno dos dois vetores for igual a zero, as retas são perpendiculares
- Importante notar, porém, é que os coeficientes a e b da equação geral de uma reta formam um vetor $\vec{v} = (a, b)$ perpendicular à reta

Retas perpendiculares

- Duas retas são perpendiculares se o produto de seus coeficientes angulares for igual a -1
- Outra maneira de checar se duas retas são perpendiculares é escolher dois pontos pertencentes a cada reta e montar dois vetores $\vec{u}$ e $\vec{v}$
- Estes pontos podem ser escolhidos de forma eficiente, fazendo $x = 0$ e $y = 0$ (caso a reta não passe na origem)
- Se o produto interno dos dois vetores for igual a zero, as retas são perpendiculares
- Importante notar, porém, é que os coeficientes a e b da equação geral de uma reta formam um vetor $\vec{v} = (a, b)$ perpendicular à reta
- Tais vetores, denominados normais, podem ser utilizados na comparação descrita anteriormente

Exemplo de verificação de retas perpendiculares em C++

```cpp
template<typename T>
struct Line
{
    // Membros e construtores (equação reduzida)

    bool orthogonal(const Line& r) const // Verdadeiro se perpendiculares
    {
        if (vertical && r.vertical)
            return false;

        if ((vertical && equals(r.m, 0)) || (equals(m, 0) && r.vertical))
            return true;

        if (vertical || r.vertical)
            return false;

        return equals(m * r.m, -1.0);
    }
};
```

Exemplo de verificação de retas perpendiculares em C++

```cpp
template<typename T>
struct Line
{
    // Membros e construtores (equação geral)

    bool orthogonal(const Line& r) const  // Verdadeiro se perpendiculares
    {
        return equals(a * r.a + b * r.b, 0);
    }
};
```

Relação entre retas

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:
  1. coincidentes (infinitas interseções),

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:
  1. coincidentes (infinitas interseções),
  2. paralelas (nenhuma interseção), ou

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:
  1. coincidentes (infinitas interseções),
  2. paralelas (nenhuma interseção), ou
  3. concorrentes (um único ponto de interseção)

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:
    1. coincidentes (infinitas interseções),
    2. paralelas (nenhuma interseção), ou
    3. concorrentes (um único ponto de interseção)
- Para encontrar o ponto de interseção, no caso de retas concorrentes, basta resolver o sistema linear resultante das equações gerais das duas retas:

$$\begin{cases} a_r x + b_r y + c_r = 0 \\ a_s x + b_s y + c_s = 0 \end{cases}$$

Interseção entre retas

- Dado um par de retas r e s, elas podem ser:
  1. coincidentes (infinitas interseções),
  2. paralelas (nenhuma interseção), ou
  3. concorrentes (um único ponto de interseção)
- Para encontrar o ponto de interseção, no caso de retas concorrentes, basta resolver o sistema linear resultante das equações gerais das duas retas:

$$\begin{cases} a_r x + b_r y + c_r = 0 \\ a_s x + b_s y + c_s = 0 \end{cases}$$

- As soluções são

$$x = (-c_r b_s + c_s b_r)/(a_r b_s - a_s b_r)$$

$$y = (-c_s a_r + c_r a_s)/(a_r b_s - a_s b_r)$$

Exemplo de implementação da interseção entre duas retas

```cpp
const int INF { -1 };

template<typename T>
std::pair<int, Point<T>> intersections(const Line<T>& r, const Line<T>& s)
{
    auto det = r.a * s.b - r.b * s.a;

    if (equals(det, 0))     // Coincidentes ou paralelas
    {
        return { (r == s) ? INF : 0, {} };
    }
    else                    // Concorrentes
    {
        auto x = (-r.c * s.b + s.c * r.b) / det;
        auto y = (-s.c * r.a + r.c * s.a) / det;

        return { 1, { x, y } };
    }
}
```

Ângulo entre retas

- Para mensurar o ângulo formado por duas retas (ou dois segmentos de reta), é preciso identificar os vetores $\vec{u}$ e $\vec{v}$ que estejam na mesma direção das duas retas e usar o produto interno

Ângulo entre retas

- Para mensurar o ângulo formado por duas retas (ou dois segmentos de reta), é preciso identificar os vetores $\vec{u}$ e $\vec{v}$ que estejam na mesma direção das duas retas e usar o produto interno
- Dados dois pontos distintos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$, o vetor direção da reta que passa por P e Q é dado por $\vec{u} = (x_q - x_p, y_q - y_p)$

Ângulo entre retas

- Para mensurar o ângulo formado por duas retas (ou dois segmentos de reta), é preciso identificar os vetores $\vec{u}$ e $\vec{v}$ que estejam na mesma direção das duas retas e usar o produto interno
- Dados dois pontos distintos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$, o vetor direção da reta que passa por P e Q é dado por $\vec{u} = (x_q - x_p, y_q - y_p)$
- De posse dos vetores de direção, o cosseno ângulo entre as retas é dado por

$$\cos\theta = \frac{u \cdot v}{|u||v|} = \frac{u_x v_x + u_y v_y}{\sqrt{u_x^2 + u_y^2}\sqrt{v_x^2 + v_y^2}}$$

Ângulo entre retas

- Para mensurar o ângulo formado por duas retas (ou dois segmentos de reta), é preciso identificar os vetores $\vec{u}$ e $\vec{v}$ que estejam na mesma direção das duas retas e usar o produto interno
- Dados dois pontos distintos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$, o vetor direção da reta que passa por P e Q é dado por $\vec{u} = (x_q - x_p, y_q - y_p)$
- De posse dos vetores de direção, o cosseno ângulo entre as retas é dado por

$$\cos\theta = \frac{u \cdot v}{|u||v|} = \frac{u_x v_x + u_y v_y}{\sqrt{u_x^2 + u_y^2}\sqrt{v_x^2 + v_y^2}}$$

- Para achar o ângulo, basta computar a função inversa do cosseno (`acos()`, na biblioteca de matemática padrão do C/C++) no lado direito da expressão acima

Exemplo de implementação do ângulo entre duas retas

```cpp
// Ângulo entre os segmentos de reta PQ e RS
template<typename T>
double angle(const Point<T>& P, const Point<T>& Q, const Point<T>& R, const Point<T>& S)
{
    auto ux = P.x - Q.x;
    auto uy = P.y - Q.y;

    auto vx = R.x - S.x;
    auto vy = R.y - S.y;

    auto num = ux * vx + uy * vy;
    auto den = hypot(ux, uy) * hypot(vx, vy);

    // Caso especial: se den == 0, algum dos vetores é degenerado: os dois
    // pontos são iguais. Neste caso, o ângulo não está definido

    return acos(num / den);
}
```

Interseção entre segmentos de reta

- Para determinar a interseção entre dois segmentos de reta é preciso resolver o problema para as duas retas que contém os respectivos segmentos e verificar se a interseção, se existir, pertence a ambos intervalos

Interseção entre segmentos de reta

- Para determinar a interseção entre dois segmentos de reta é preciso resolver o problema para as duas retas que contém os respectivos segmentos e verificar se a interseção, se existir, pertence a ambos intervalos
- Embora esta abordagem permita conhecer as coordenadas das possíveis interseções, ela traz alguns problemas em potencial:

Interseção entre segmentos de reta

- Para determinar a interseção entre dois segmentos de reta é preciso resolver o problema para as duas retas que contém os respectivos segmentos e verificar se a interseção, se existir, pertence a ambos intervalos
- Embora esta abordagem permita conhecer as coordenadas das possíveis interseções, ela traz alguns problemas em potencial:
    1. mesmo que as retas sejam coincidentes, não há garantias que os segmentos tenham interseção

Interseção entre segmentos de reta

- Para determinar a interseção entre dois segmentos de reta é preciso resolver o problema para as duas retas que contém os respectivos segmentos e verificar se a interseção, se existir, pertence a ambos intervalos
- Embora esta abordagem permita conhecer as coordenadas das possíveis interseções, ela traz alguns problemas em potencial:
  1. mesmo que as retas sejam coincidentes, não há garantias que os segmentos tenham interseção
  2. a concorrência também não garante interseção: ainda é preciso verificar se o ponto pertence a ambos intervalos

Interseção entre segmentos de reta

- Para determinar a interseção entre dois segmentos de reta é preciso resolver o problema para as duas retas que contém os respectivos segmentos e verificar se a interseção, se existir, pertence a ambos intervalos
- Embora esta abordagem permita conhecer as coordenadas das possíveis interseções, ela traz alguns problemas em potencial:
    1. mesmo que as retas sejam coincidentes, não há garantias que os segmentos tenham interseção
    2. a concorrência também não garante interseção: ainda é preciso verificar se o ponto pertence a ambos intervalos
- Para identificar apenas se há interseção entre ambos segmentos, sem determinar as coordenadas de tal interseção, o problema fica simplificado, e será abordado mais adiante

Rotina que verifica se um ponto P pertence ao segmento AB

```cpp
// Verifica se o ponto P pertence ao segmento de reta AB
template<typename T>
bool contains(const Point<T>& A, const Point<T>& B, const Point<T>& P)
{
    if (P == A || P == B)
        return true;

    auto xmin = min(A.x, B.x);
    auto xmax = max(A.x, B.x);
    auto ymin = min(A.y, B.y);
    auto ymax = max(A.y, B.y);

    if (P.x < xmin || P.x > xmax || P.y < ymin || P.y > ymax)
        return false;

    // Verifica relação de semelhança no triângulo
    return equals((P.y - A.y)*(B.x - A.x), (P.x - A.x)*(B.y - A.y));
}
```

Relação entre retas e pontos

Distância entre ponto e reta

- A distância de um ponto P a uma reta r é definida como a menor distância possível entre todos os pontos de r e P:

$$d(P, r) = \min\{d(P, Q), Q \in r\}$$

- Contudo, não é necessário computar as infinitas distâncias possíveis: a menor distância será aquela entre P e o ponto de interseção Q de r com a reta perpendicular a r que passa por P
- Seja usando álgebra, geometria ou álgebra linear, é possível mostrar que esta distância d entre $P = (x_p, y_p)$ e a reta $ax + by + c = 0$ é dada por

$$d(P, r) = \frac{|ax_p + by_p + c|}{\sqrt{a^2 + b^2}}$$

- As coordenadas de $Q = (x_q, y_q)$ podem ser obtidas utilizando-se as expressões

$$x_q = \frac{b(bx_p - ay_p) - ac}{a^2 + b^2}, \quad y_q = \frac{a(-bx_p + ay_p) - bc}{a^2 + b^2}$$

Implementação de distância entre ponto e reta em C++

```cpp
#include <cmath>
#include <iostream>

template<typename T>
struct Point {
    T x, y;
};

template<typename T>
struct Line {
    T a, b, c;

    double distance(const Point<T>& p) const   // Distância de p à reta
    {
        return fabs(a*p.x + b*p.y + c)/hypot(a, b);
    }

    Point<T> closest(const Point<T>& p) const // Ponto mais próximo de p
    {
        auto den = (a*a + b*b);
```

Implementação de distância entre ponto e reta em C++

```cpp
        auto x = (b*(b*p.x - a*p.y) - a*c)/den;
        auto y = (a*(-b*p.x + a*p.y) - b*c)/den;

        return Point<T> { x, y };
    }
};

int main()
{
    Point<double> P { 1.0, 4.0 };
    Line<double> r { 1.0, -1.0, 0 };

    std::cout << "Distance: " << r.distance(P) << '\n';

    auto Q = r.closest(P);

    std::cout << "Closest: Q = (" << Q.x << ", " << Q.y << ")\n";

    return 0;
}
```

Reta mediatriz

- Dado o segmento de reta PQ, a mediatriz é a reta perpendicular a PQ que passa pelo ponto médio do segmento
- Qualquer ponto da reta mediatriz é equidistante a P e Q, e esta propriedade permite a dedução dos coeficientes a, b, c da mediatriz
- Seja $R = (x, y)$ um ponto qualquer da mediatriz. Então

$$d^2(P, R) = d^2(Q, R),$$

isto é,

$$(x - x_p)^2 + (y - y_p)^2 = (x - x_q)^2 + (y - y_q)^2$$

Logo os coeficientes são

$$a = 2(x_q - x_p),\ b = 2(y_q - x_q),\ c = (x_p^2 + y_p^2) - (y_p^2 + y_q^2)$$

Exemplo de implementação da reta mediatriz em C++

```cpp
// Definição das classes Point e Line

typename<template T>
Line<T> perpendicular_bisector(const Point<T>& P, const  Point<T>& Q)
{
    auto a = 2*(Q.x - P.x);
    auto b = 2*(Q.y - P.y);
    auto c = (P.x * P.x + P.y * P.y) - (Q.x * Q.x + Q.y * Q.y);

    return Line<T>(a, b, c);
}
```

Orientação entre ponto e reta

- O determinante utilizado para o cálculo dos coeficientes da equação geral da reta também identifica a orientação de um ponto em relação a uma reta
- Sejam P, Q, R três pontos no plano e considere r a reta que passa por P e Q
- Logo o vetor $\vec{v} = (x_p - x_q, y_p - y_q)$ tem mesma direção que r
- Seja $\vec{u} = (x_r - x_q, y_r - y_q)$. O vetor $\vec{n} = (y_q - y_r, x_r - x_q)$ é perpendicular ao vetor $\vec{u}$ (pois $\vec{u} \cdot \vec{n} = 0$)
- Assim, $\vec{u}$ e $\vec{v}$ serão paralelos (e, como consequência, P, Q e R serão colineares) se o produto interno entre $\vec{v}$ e $\vec{n}$ for igual a zero
- Daí,

  $$0 = \vec{v} \cdot \vec{n} = (x_p - x_q)(y_q - y_r) + (y_p - y_q)(x_r - x_q),$$

  isto é,

  $$0 = (x_p y_q - x_p y_r - x_q y_p + x_q y_r) + (y_p x_r - y_p x_q - y_q x_r + y_q x_p)$$

Orientação entre ponto e reta

- Portanto,

$$0 = (x_p y_q + x_q y_r + x_r y_p) - (y_p x_q + y_q x_r + y_r x_p),$$

expressão que pode ser reescrita como

$$\det \begin{bmatrix} x_p & y_p & 1 \\ x_q & y_q & 1 \\ x_r & y_r & 1 \end{bmatrix} = 0$$

- Deste modo, o discriminante $D(P, Q, R)$ é definido como o determinante acima
- Se r é uma reta que passa pelos pontos P e Q, e R é um ponto qualquer, então
  1. R pertence a reta se $D = 0$,
  2. R está no semiplano à esquerda da reta, se $D > 0$, ou
  3. R no semiplano à direita da reta, se $D < 0$
- A orientação (esquerda ou direita) diz respeito à direção que vai de P a Q

Exemplo de implementação do discriminante D em C++

```cpp
// Definição da classe Point

// D = 0: R pertence a reta PQ
// D > 0: R à esquerda da reta PQ
// D < 0: R à direita da reta PQ
template<typename T>
T D(const Point<T>& P, const Point<T>& Q, const Point<T>& R)
{
    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);
}
```

Ponto mais próximo a um segmento de reta

- Para determinar o ponto do segmento AB mais próximo de um ponto P dado, é preciso, inicialmente, determinar o ponto Q da reta r que contém A e B mais próximo de P
- Em seguida, é preciso avaliar também os extremos A e B do segmento, pois o ponto Q pode estar fora do segmento
- Assim, o ponto mais próximo (e a respectiva distância) será, dentre A, B e Q, o mais próximo de P que pertença ao intervalo

Implementação do ponto mais próximo de P em AB

```cpp
// Definição das classes Point e Line, e da função equals()

template<typename T>
struct Segment {
    Point<T> A, B;

    // Verifica se o ponto P da reta r que contém _A_ e _B_
    // pertence ao segmento
    bool contains(const Point<T>& P) const
    {
        if (equals(A.x, B.x))
            return min(A.y, B.y) <= P.y and P.y <= max(A.y, B.y);
        else
            return min(A.x, B.x) <= P.x and P.x <= max(A.x, B.x);
    }

```

Implementação do ponto mais próximo de P em AB

```cpp
    // Ponto mais próximo de P no segmento AB
    Point<T> closest(const Point<T>& P)
    {
        Line<T> r(A, B);
        auto Q = r.closest(P);

        if (this->contains(Q))
            return Q;

        auto distA = P.distanceTo(A);
        auto distB = P.distanceTo(B);

        if (distA <= distB)
            return A;
        else
            return B;
    }
}
```

Relação entre segmentos

Interseção entre segmentos

- Para se determinar se dois segmentos AB e PQ se intersectam pode-se utilizar um algoritmo baseado no discriminante D
- A ideia central é que dois segmentos se interceptam se a reta que passa por um dos segmento separa os dois pontos do outro segmento em semiplanos distintos
- É preciso, contudo, tomar cuidado com o caso onde um dos pontos de um segmento (por exemplo, A) é colinear em relação aos pontos do outro segmento (P e Q)
- Neste caso especial, o discriminante será igual a zero, e será necessário verificar se o ponto A pertence ou não a PQ

Implementação da interseção de segmentos em C++

```cpp
// Definição da classe Point e do discriminante D()

template<typename T>
class Segment {
public:
    Point<T> A, B;

    // Definição do método contains()

    bool intersect(const Segment& s) const
    {
        auto d1 = D(A, B, s.A);
        auto d2 = D(A, B, s.B);

        if ((equals(d1, 0) && contains(s.A)) ||
           (equals(d2, 0) && contains(s.B)))
            return true;
```

Implementação da interseção de segmentos em C++

```cpp

        auto d3 = D(s.A, s.B, A);
        auto d4 = D(s.A, s.B, B);

        if ((equals(d3, 0) && s.contains(A)) ||
           (equals(d4, 0) && s.contains(B)))
            return true;

        return (d1 * d2 < 0) && (d3 * d4 < 0);
    }
}
```

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **De BERG**, Mark; **CHEONG**, Otfried. *Computational Geometry: Algorithms and Applications*, 2008.
4. David E. Joyce. *Euclid's Elements*. Acesso em 15/02/2019[1]
5. Wikipédia. *Geometria Euclidiana*. Acesso em 15/02/2019[2].

[1] https://mathcs.clarku.edu/ djoyce/elements/bookI/defI1.html
[2] https://pt.wikipedia.org/wiki/Geometria_euclidiana