Ponto 2D / Vetor

Representação de pontos e vetores como pontos.

Estrutura

Descrição

- **(double)** norm: Calcula norma do vetor em relação a origem
- **(double)** normalized: Retorna vetor normalizado (Norma 1)
- **(double)** angle: Angulo do vetor com a origem
- **(double)** polarAngle: Angulo polar em relação a origem

```
const double EPS = 1e-9;
const double PI = acos(-1);

struct point{
    double x, y;

    point(double _x, double _y) : x(_x), y(_y) {} ;
    point(){ x=y=0.0; };

    bool operator <( point other ) const {
        return fabs(x-other.x)<EPS ? y<other.y : x<other.x;
    }
    bool operator ==(point other) const {
        return fabs(x-other.x) < EPS && fabs(y-other.y)<EPS;
    }
    point operator +(point other) const {
        return point( x+other.x, y+other.y);
    }
    point operator -(point other) const {
        return point( x-other.x, y-other.y );
    }
    point operator *(double k) const {
        return point(x*k, y*k);
    }

    double norm() { return hypot(x, y); }
    point normalized(){ return point(x, y)*(1.0/norm()); }
    double angle() { return atan2(y, x); }
    double polar_angle(){
        double a = atan2(y, x);
        return a < 0 ? a + 2*PI : a;
    }
};
```

Funções

Descrição

- **(double)** dist: Distância entre 2 pontos
- **(double/int)** inner: Produto interno entre 2 vetores
- **(double/int)** cross: Produto vetorial entre 2 vetores (componente z)
- **(double/int)** ccw: Verifica se os pontos estão no sentido antihorário (DISCRIMINANTE)
- **(double/int)** collinear: Teste de colinearidade entre os pontos
- **(double)** rotate: Rotação do ponto em relação a origem
- **(double)** angle: Angulo formado entre vetores a e b com o de origem
- **(double)** proj: Projeção de u sobre v
- **(double/int)** between: Verifica se o ponto q está dentro no segmento p r incluindo as bordas; Para desconsiderar as bordas basta mudar inn<=0 para inn<0
- **(double)** line_intersect: Retorna ponto formado pela intersecção das retas p q e A B **Se as retas forem coolineares c é zero e a função retorna nan**
- **(double/int)** parallel: Teste de paralelidade
- **(double)** seg_intersects: Verifica se segmentos a b e p q se interceptam
- **(double)** closet_point: Ponto mais próximo entre p e o segmento de reta a b

```
double dist(point p1, point p2){
    return hypot(p1.x-p2.x, p1.y-p2.y);
}
double inner(point p1, point p2){
    return p1.x*p2.x+p1.y*p2.y;
}
double cross(point p1, point p2){
    return p1.x*p2.y-p1.y*p2.x;
}
bool ccw(point p, point q, point r){
    point k1=q-p, k2=r-p;
    double d = k1.x*k2.y-k1.y*k2.x;
    return d>0;
}
bool collinear(point p, point q, point r){
    point k1=q-p, k2=r-p;
    double d = k1.x*k2.y-k1.y*k2.x;
    return fabs(d)<EPS;
}
point rotate(point p, double rad){
    return point(p.x*cos(rad)-p.y*sin(rad),
                 p.x*sin(rad)+p.y*cos(rad));
}
double angle(point a, point o, point b){
    double den = hypot(o.x-a.x, o.y-a.y)*hypot(o.x-b.x, o.y-b.y);
    point k1=a-o, k2=b-o;
    double num = k1.x*k2.x+k1.y*k2.y;
    return acos(num/den);
}
```

```
point proj(point u, point v){
    double num = u.x*v.x+u.y*v.y;
    double den = v.x*v.x+v.y*v.y;
    return v*(num/den);
}
bool between(point p, point q, point r){
    point k1=q-p, k2=r-p;
    double col = k1.x*k2.y-k1.y*k2.x;
    k1=p-q, k2=r-q;
    double inn = k1.x*k2.x+k1.y*k2.y;
    return fabs(col)<EPS && inn<=0;
}
point line_intersect(point p, point q, point A, point B){
    double c = cross(A-B, p-q);
    double a = cross(A, B);
    double b = cross(p, q);
    return ( (p-q)*(a/c) ) - ( (A-B)*(b/c) );
}
bool parallel(point a, point b){
    double c = a.x*b.y-a.y*b.x;
    return fabs(c)<EPS;
}
bool seg_intersects(point a, point b, point p, point q){
    if(parallel(a-b, p-q)){
        return between(a, p, b) || between(a, q, b) ||
               between(p, a, q) || between(p, b, q);
    }
    point i = line_intersect(a, b, p, q);
    return between(a, i, b) && between(p, i, q);
}
point closet_point(point p, point a, point b){
    if(a==b) return a;
    point k1=p-a, k2=b-a;
    double num = k1.x*k2.x+k1.y*k2.y;
    double dem = k2.x*k2.x+k2.y*k2.y;
    double u = num/dem;
    if(u < 0.0) return a;
    if(u > 1.0) return b;
    return a+((b-a)*u);
}
```