Algoritmos Geométricos

GIS (Geographic Information System)

Um sistema de informação geográfica (GIS) é um sistema de computador para capturar, armazenar, verificar e exibir dados relacionados às posições na superfície da Terra.

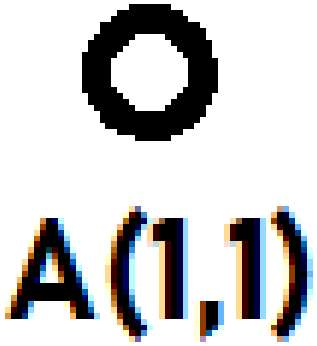
- National Geographic

Ementa laboratório

- Representação de pontos, vetores, retas e segmentos no plano em C++;
- Relações entre pontos e retas;
- Cálculos com vetores;
- Algoritmos para cálculo de área;
- Ordenação radial;
- Algoritmo do fecho convexo.

Informações de um ponto

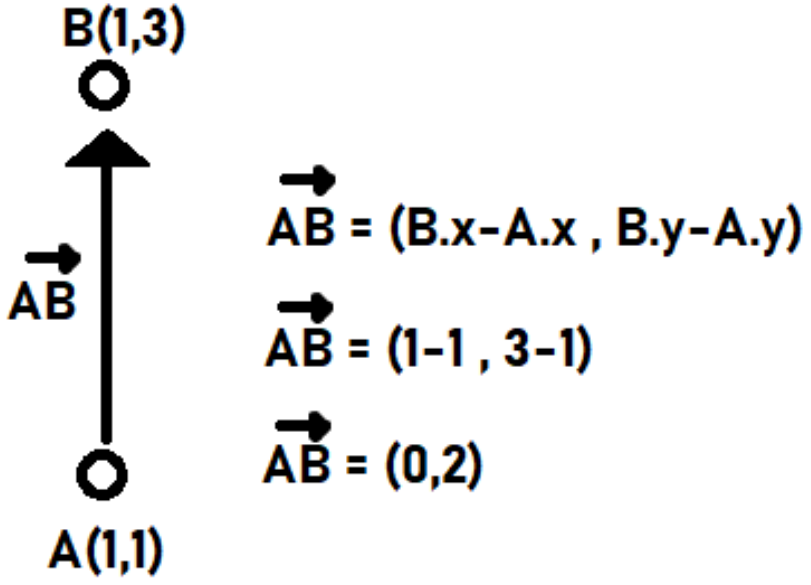
```
struct ponto
{
    float x;
    float y;
};
```



Informações de um vetor

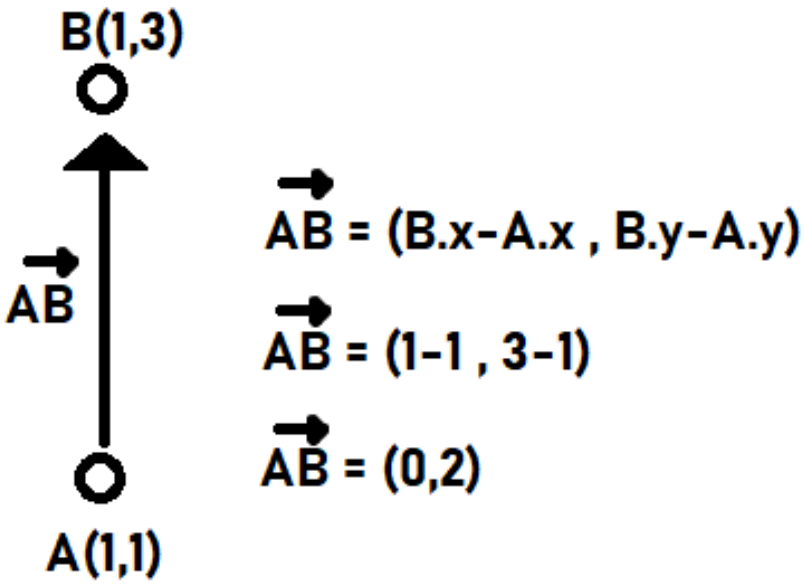
```
struct ponto
{
    float x;
    float y;
};
typedef ponto vetor;
```

Fixamos a origem do vetor na origem do plano cartesiano

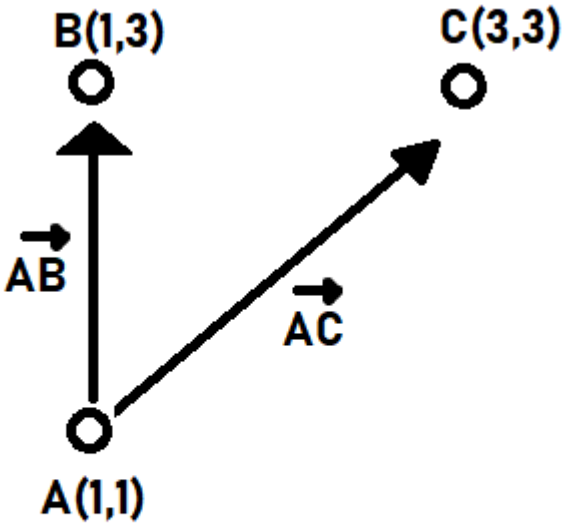


Criando um vetor

```
vetor cria_vetor(ponto A, ponto B)
{
   vetor AB;
   AB.x = B.x – A.x;
   AB.y = B.y – A.y;
   return AB;
}
```



Produto vetorial

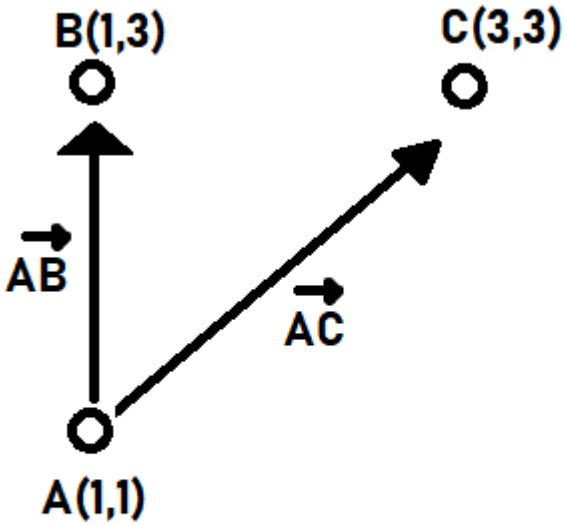


$\vec{AB} = (0,2)$ $\vec{AC} = (2,2)$

Produto vetorial

```
float vetorial(vetor A, vetor B)
{
   float resultado;
   resultado = A.x*B.y – A.y*B.x;
   return resultado;
}
```

Produto escalar

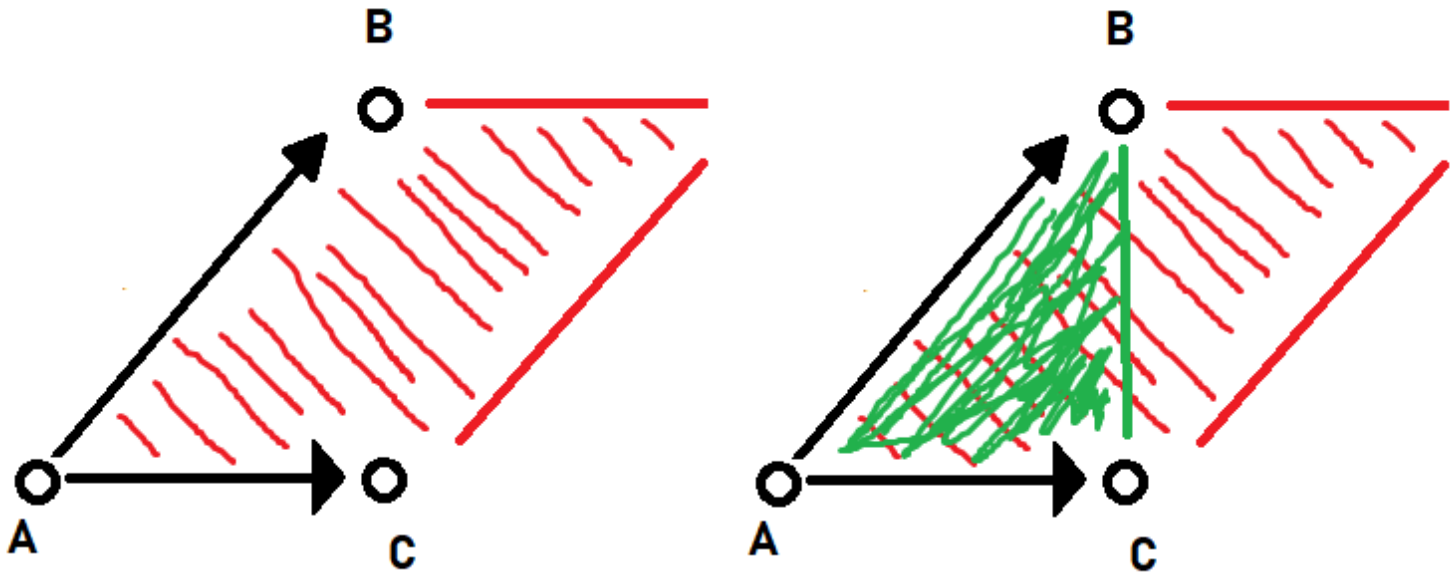


$\vec{AB} = (0,2)$ $\vec{AC} = (2,2)$

Produto escalar

```
float escalar(vetor A, vetor B)
{
   float resultado;
   resultado = A.x*B.x + A.y*B.y;
   return resultado;
}
```

Área triângulo



$$\Delta ABC = |AB \times AC| / 2$$

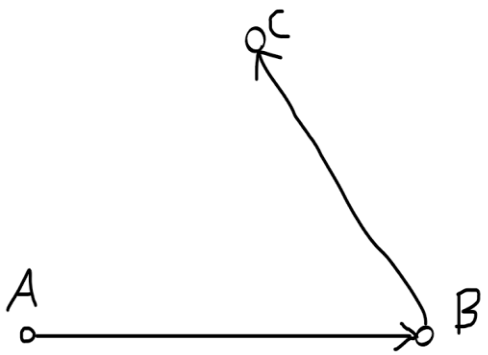
Área triângulo

```
float area_triangulo(ponto A,ponto B,ponto C)
{
        vetor AB;
        vetor AC;
        float area;
        AB = cria_vetor(A,B);
        AC = cria_vetor(A,C);
        area = fabs(vetorial(AB,AC)) / 2;
        return area;
}
```

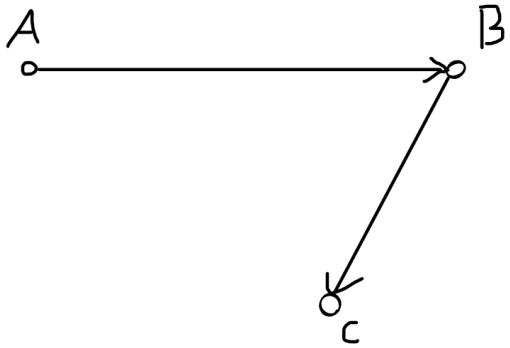
Sentido

Consiste em verificar se, dado 3 pontos, se o percurso faz uma curva ou se se mantém em frente.

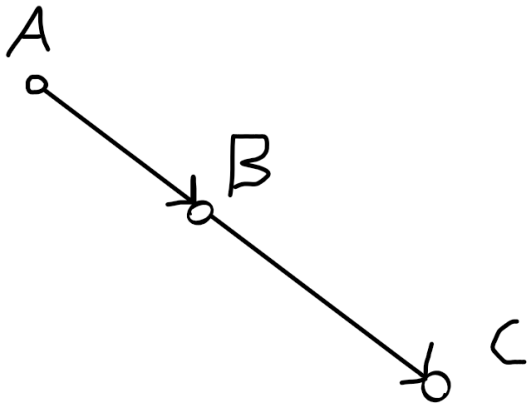
Curva à esquerda



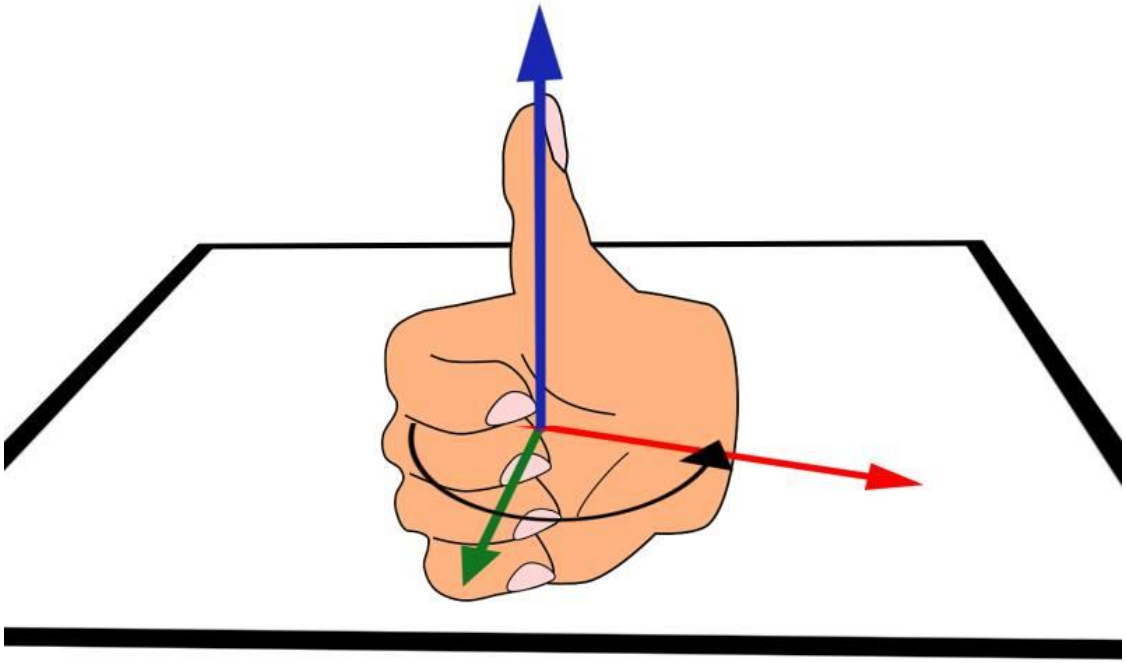
Curva à direita



Colinear

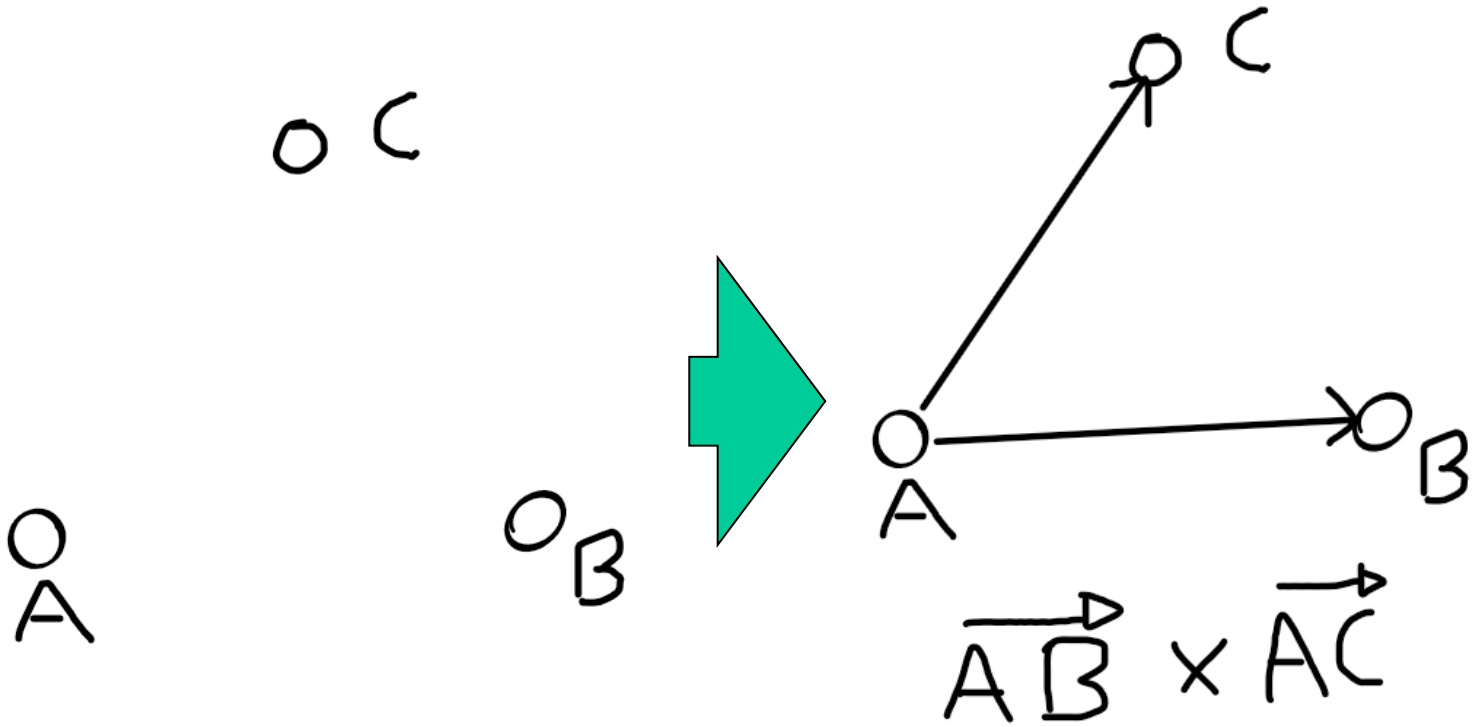


Sentido



$$\vec{A} \times \vec{B} = \vec{C}$$

Sentido



Sentido

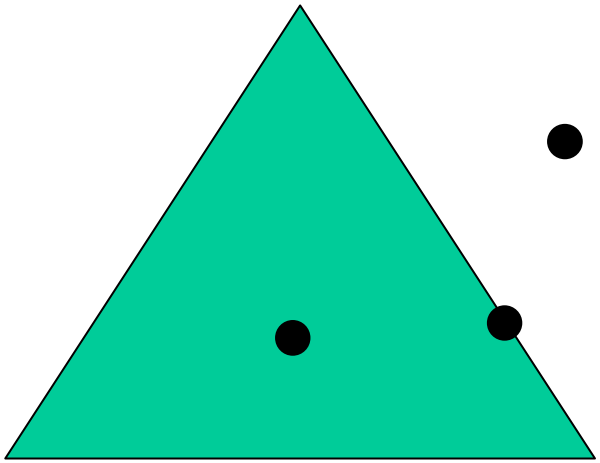
- Se AB x AC > 0 ⟶ Vetor “para cima” = curva à esquerda;
- Se AB x AC < 0 ⟶ Vetor “para baixo” = curva à direita;
- Se AB x AC = 0 ⟶ Vetor de tamanho zero = colinear

Sentido

```
int sentido(ponto A, ponto B, ponto C)
{
        vetor AB;
        vetor AC;
        float vet;
        AB = cria_vetor(A,B);
        AC = cria_vetor(A,C);
        vet = vetorial(AB,AC);
        if(vet > 0)
                return 1;
        else if(vet < 0)
                return -1;
        else
                return 0;
}
```

Dentro triângulo

Consiste em verificar se um ponto está dentro, na borda ou fora de um triângulo

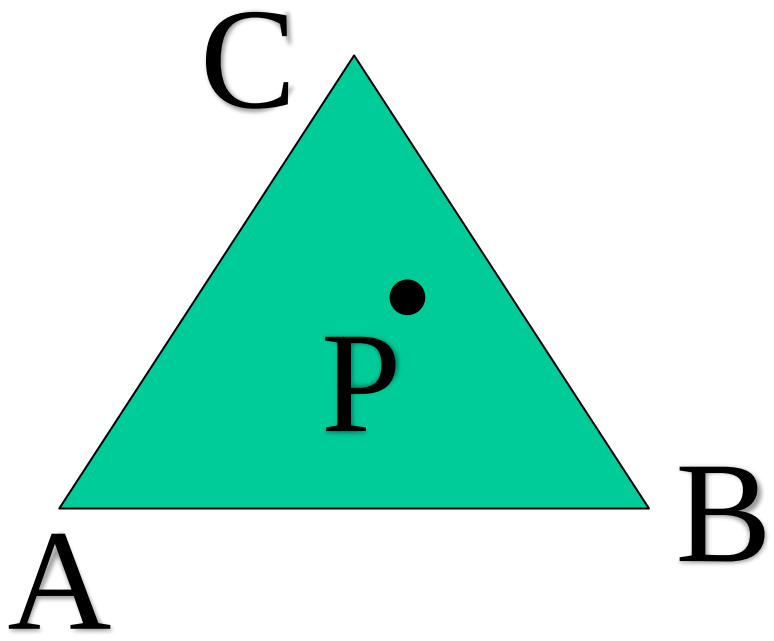


Dentro triângulo

- Verifica-se o sentido com base nos 3 lados do triângulo e o ponto de estudo;
- Se estiver à esquerda de todos os lados, ou à direita de todos os lados, o ponto esta **dentro do triângulo;**
- Se estiver à direita de um lado e à esquerda de qualquer outro lado, o ponto está **fora do triângulo**;
- Caso contrário, está na **borda do triângulo**.

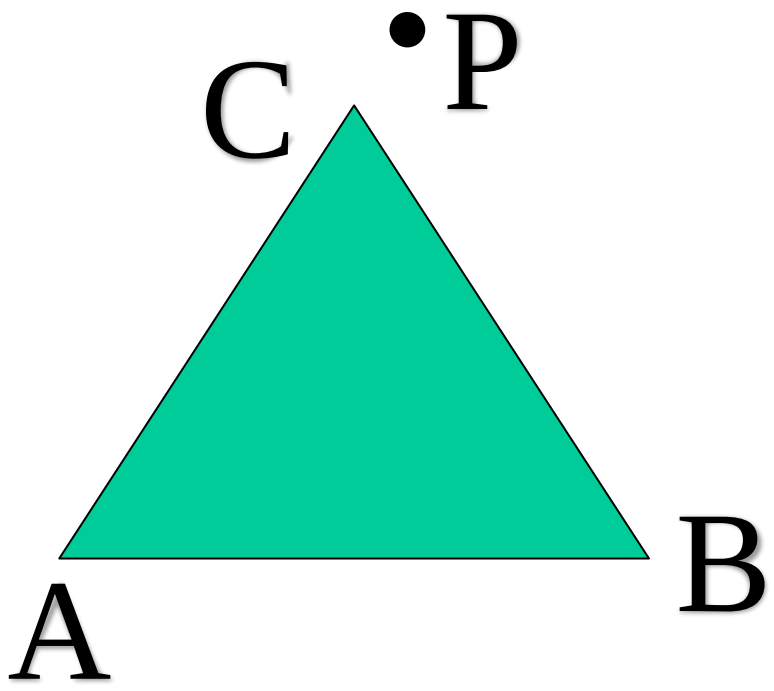
Dentro triângulo

Está dentro, pois está à esquerda de todos os lados



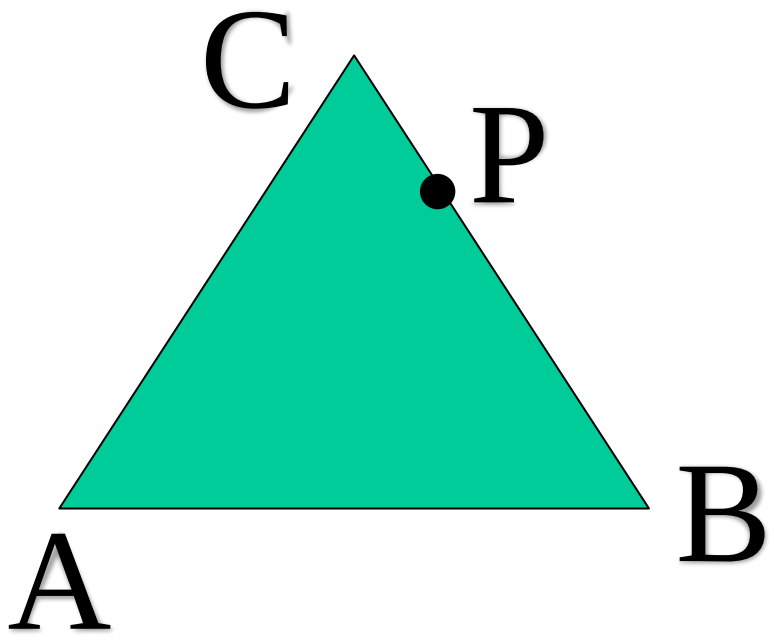
Dentro triângulo

Está fora, pois está à esquerda AB, mas está à direita de BC



Dentro triângulo

Está na borda, pois todas as condições anteriores não foram satisfeitas



Dentro triângulo

```
int dentro_triangulo(ponto A, ponto B, ponto C, ponto P)
{
        int s1,s2,s3;
        s1 = sentido(A,B,P);
        s2 = sentido(B,C,P);
        s3 = sentido(C,A,P);
        if(s1 == s2 && s2 == s3)
                return 1;
        else if(s1*s3 == -1 || s1*s2 == -1 || s2*s3 == -1)
                return -1;
        else
                return 0;
}
```

Área do polígono

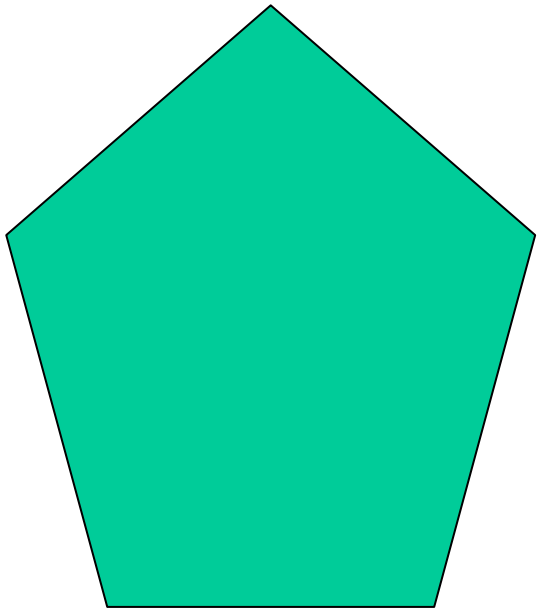
- Divide-se o polígono em triângulos;
- Soma-se as áreas dos triângulos divididos;

Atenção:

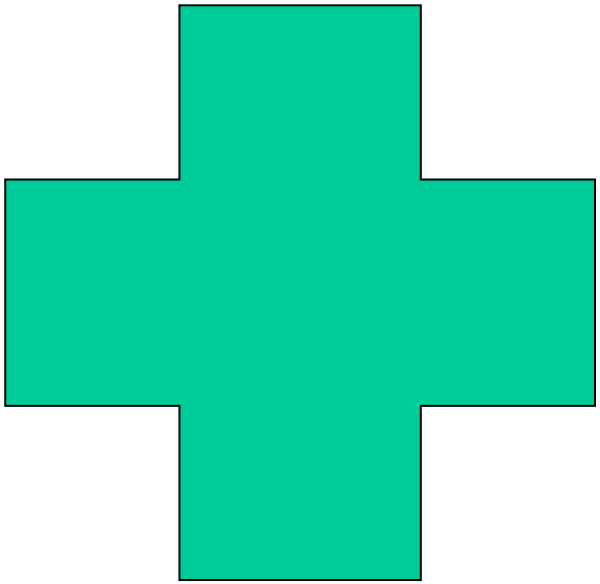
- O polígono deve ser convexo;
- Os pontos devem estar ordenados radialmente.

Área do polígono

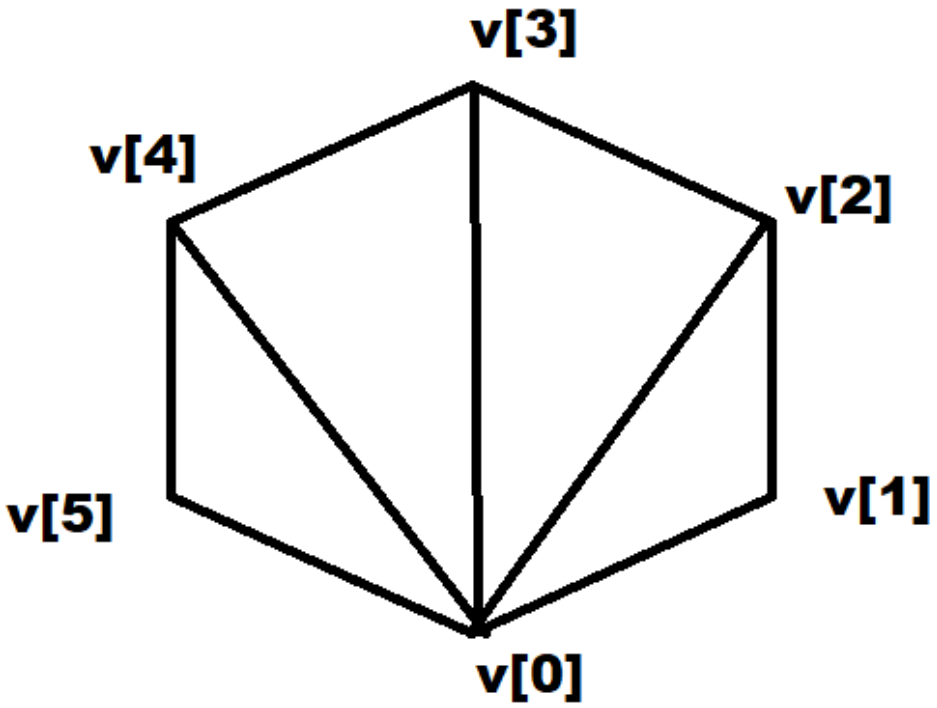
Polígono convexo



Polígono não convexo



Área do polígono



Área do polígono

```
float area_poligono(ponto v[],int n)
{
        float area = 0;
        for(int i = 0;i < n-2;i++)
        {
                area = area +
                        area_triangulo(v[0],v[i+1],v[i+2]);
        }
        return area;
}
```

Reta e segmento

```
struct reta
{
        ponto A;
        ponto B;
};
typedef reta segmento;
```

Ponto e segmento

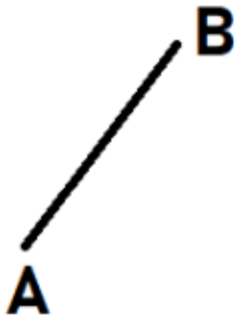
Verifica se um ponto pertence ou não à um segmento de reta;

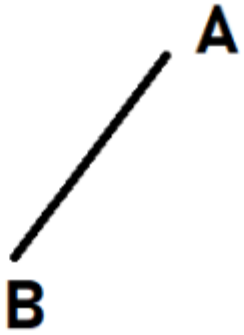
1º) O ponto deve ser colinear ao segmento.

Se não for colinear, ele não pertence ao segmento.

Ponto e segmento

2º) Se for colinear, temos 4 situações:





Ponto e segmento

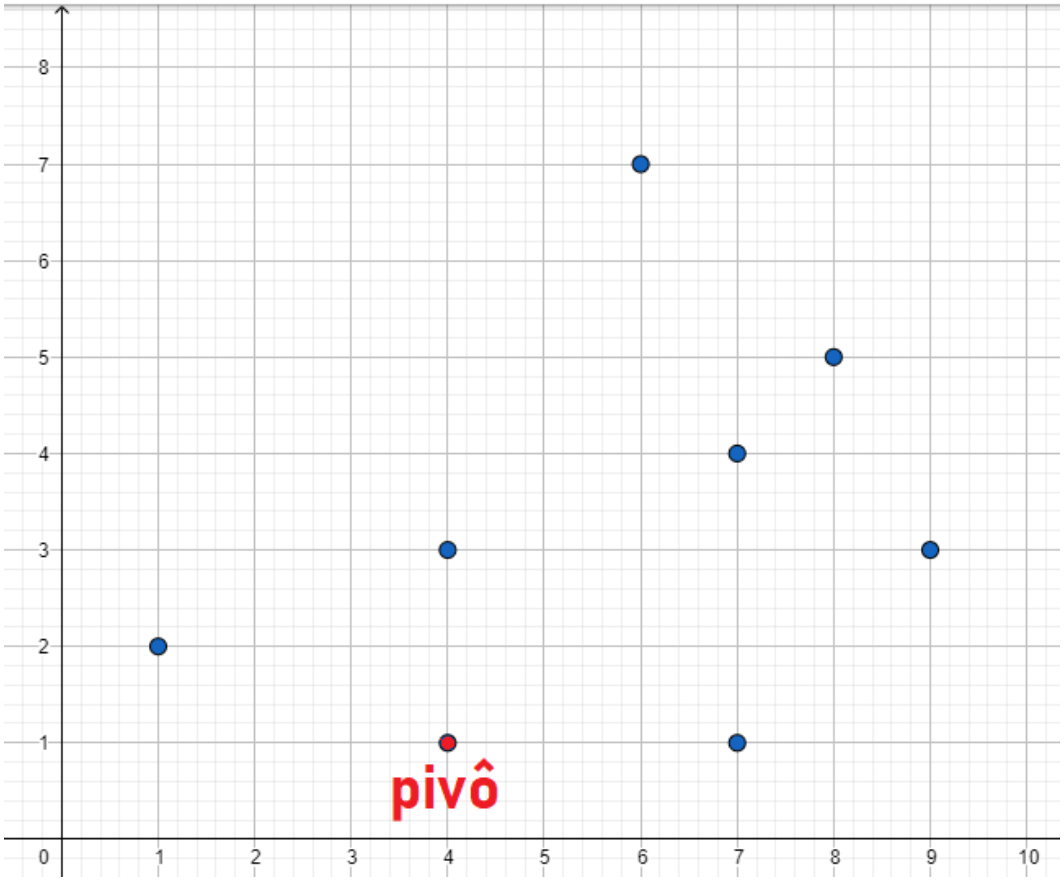
```
bool ponto_segmento(segmento S, ponto P)
{
        if(sentido(S.A,S.B,P) != 0)
                return false;
        else if(P.x > S.A.x && P.x < S.B.x)
                return true;
        else if(P.x < S.A.x && P.x > S.B.x)
                return true;
        else if(P.y < S.A.y && P.y > S.B.y)
                return true;
        else if(P.y > S.A.y && P.y < S.B.y)
                return true;
        else
                return false;
}
```

Ordenação pelo ângulo

- Fixamos um pivô;
- Ordenamos em relação ao ângulo que cada ponto faz com o eixo horizontal.

Ordenação pelo ângulo

Pivô: ponto mais abaixo (desempata pelo mais a esquerda)

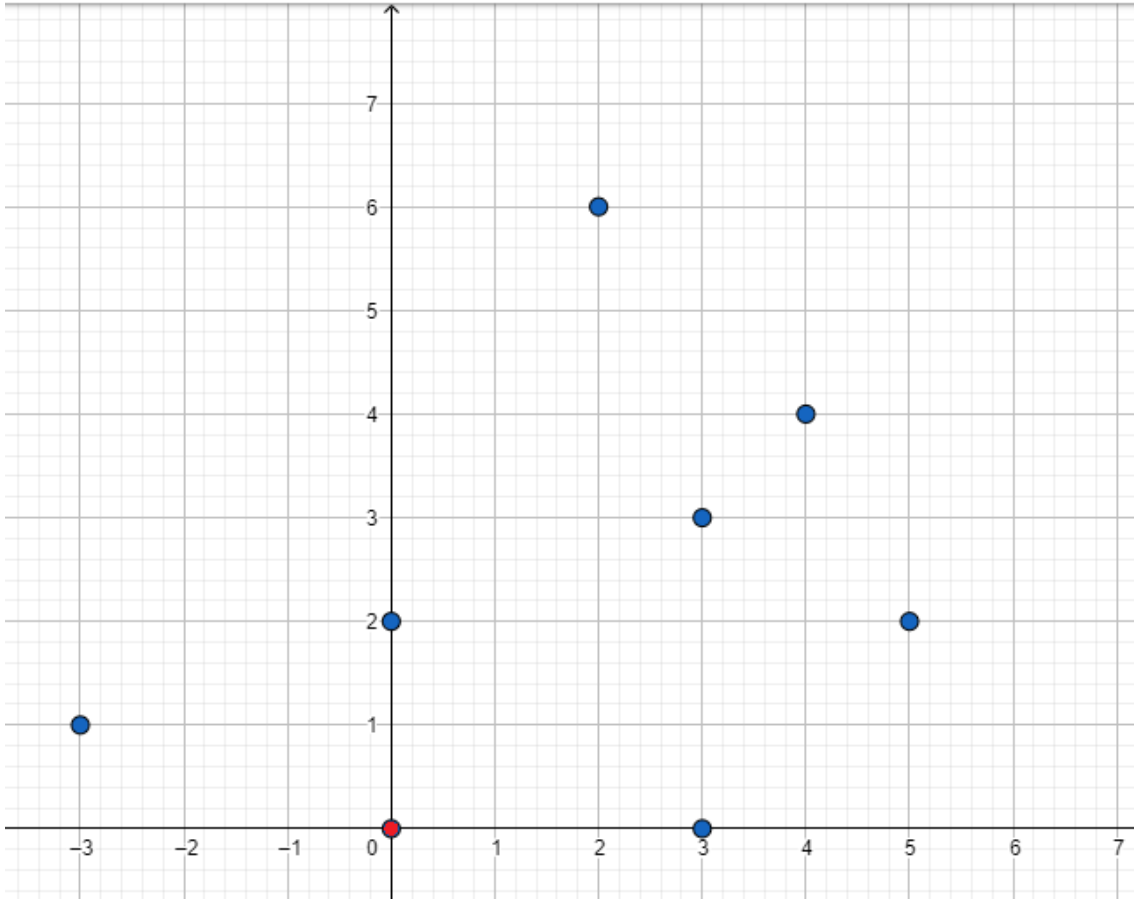


Ordenação pelo ângulo

```
pivot = p[0];
pos_pivot = 0;
for(int i = 1;i < n;i++)
{
        if(p[i].y < pivot.y)
        {
                pivot = p[i];
                pos_pivot = i;
        }
        else if(p[i].y == pivot.y && p[i].x < pivot.x)
        {
                pivot = p[i];
                pos_pivot = i;
        }
}
```

Ordenação pelo ângulo

Deslocamos o pivô para a origem, e ajustamos os demais:



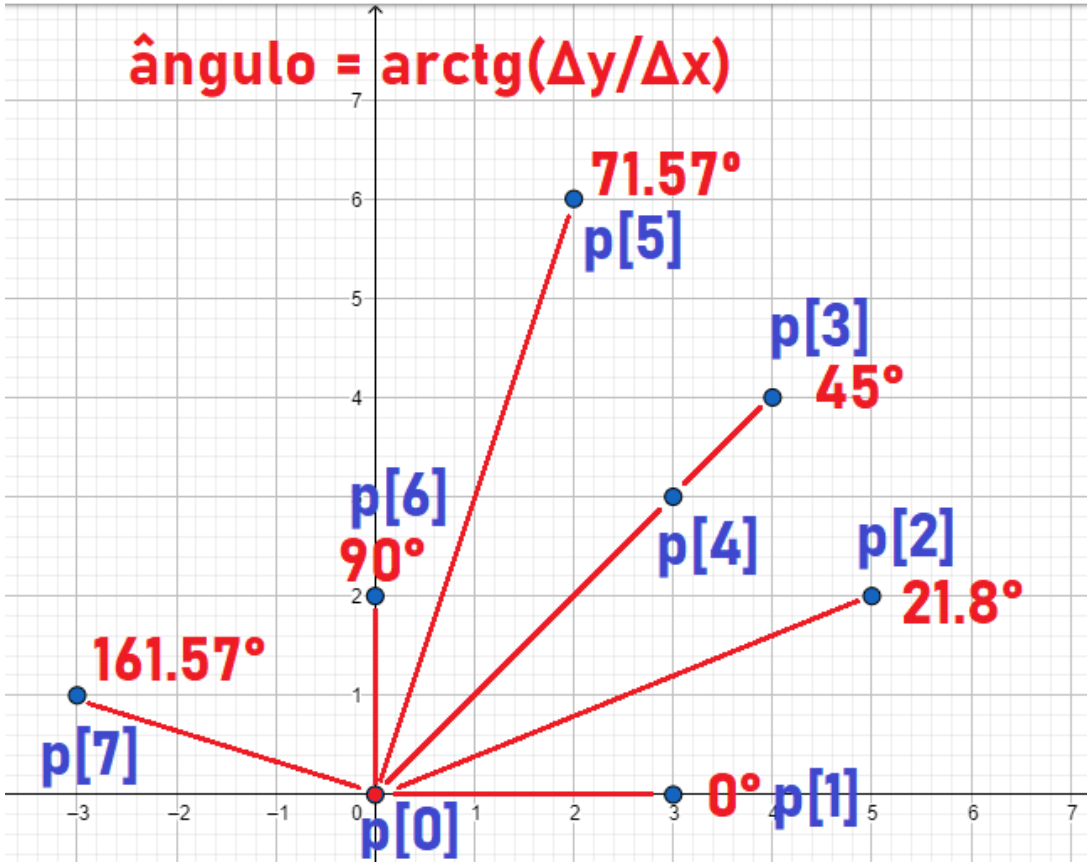
Ordenação pelo ângulo

```
p[pos_pivot] = p[0];
p[0] = pivot;

for(int i = 0;i < n; i++)
{
        p[i].x -= pivot.x;
        p[i].y -= pivot.y;
}
```

Ordenação pelo ângulo

Ordenamos pelo ângulo com o eixo horizontal:

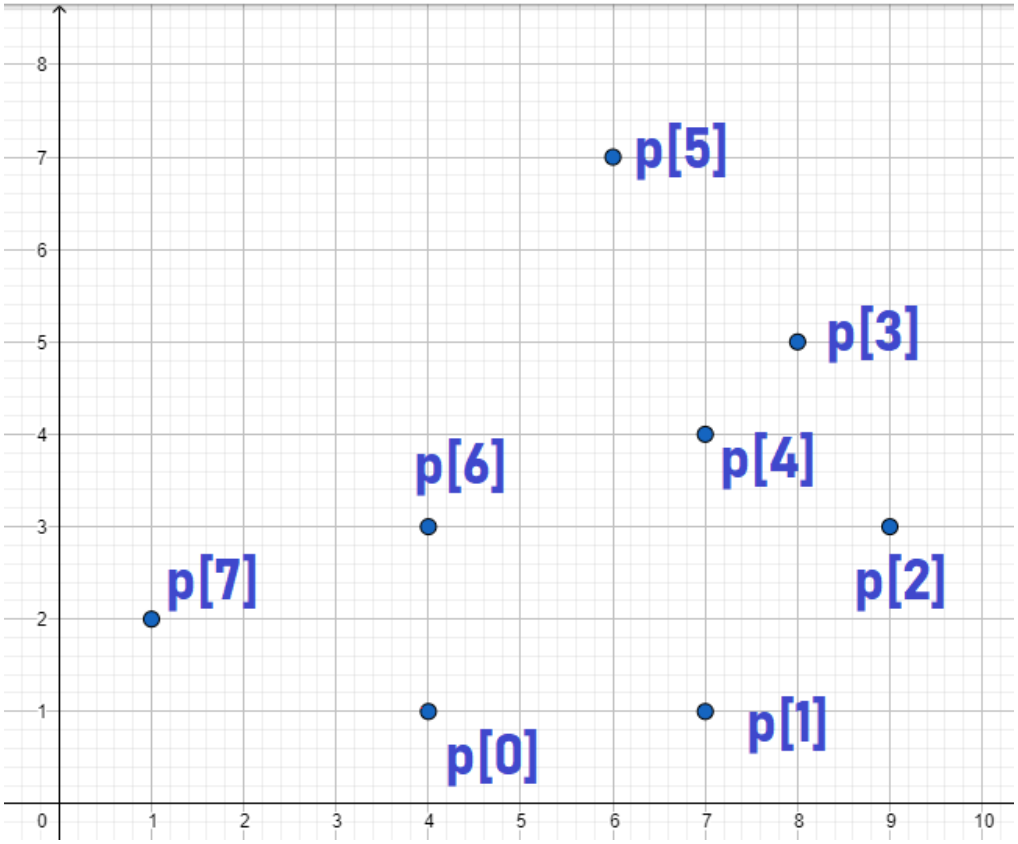


Ordenação pelo ângulo

```
// função da biblioteca algorithm
sort(p + 1, p + n, cmp);
// função de comparação
bool cmp(ponto a, ponto b){
        float angA =atan2(a.y,a.x);
        float angB = atan2(b.y,b.x);
        if(angA == angB)
        {
                float distA = sqrt(pow(a.x,2)+pow(a.y,2));
                float distB = sqrt(pow(b.x,2)+pow(b.y,2));
                return distA > distB; // primeiro o de maior distancia
        }
        else
                return angA < angB; // primeiro o de menor angulo
}
```

Ordenação pelo ângulo

Retornamos para as posições originais (somando x e y do pivô)



Ordenação pelo ângulo

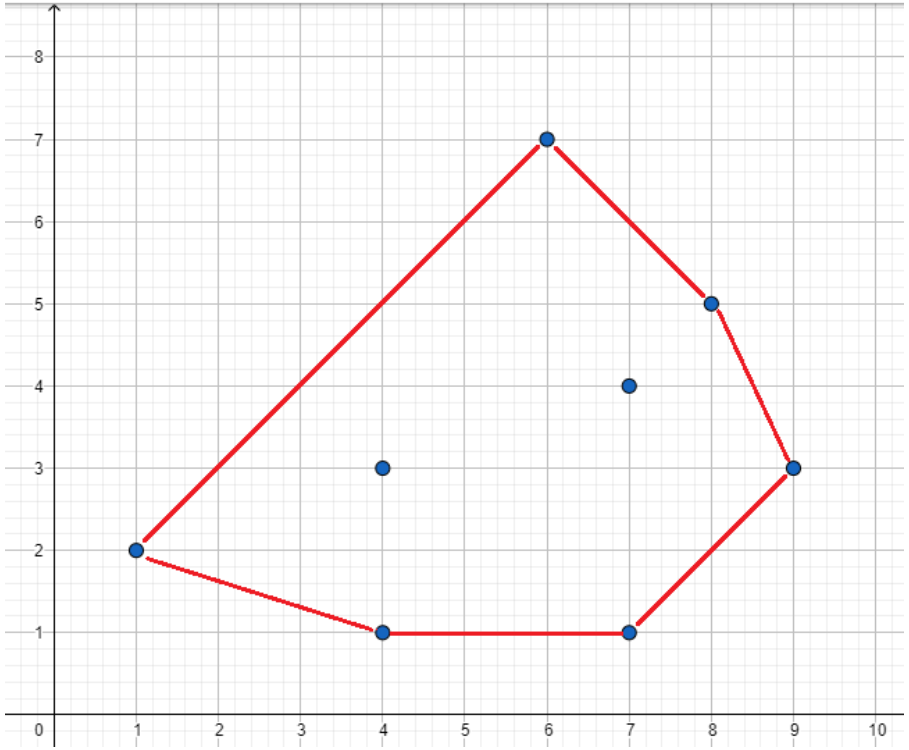
```
for(int i = 0;i < n;i++)
{
        p[i].x += pivot.x;
        p[i].y += pivot.y;
}
```

Fecho convexo

Problema:
“Encontrar os pontos que são vértices do menor polígono (convexo) capaz de envolver todos os pontos listados”;

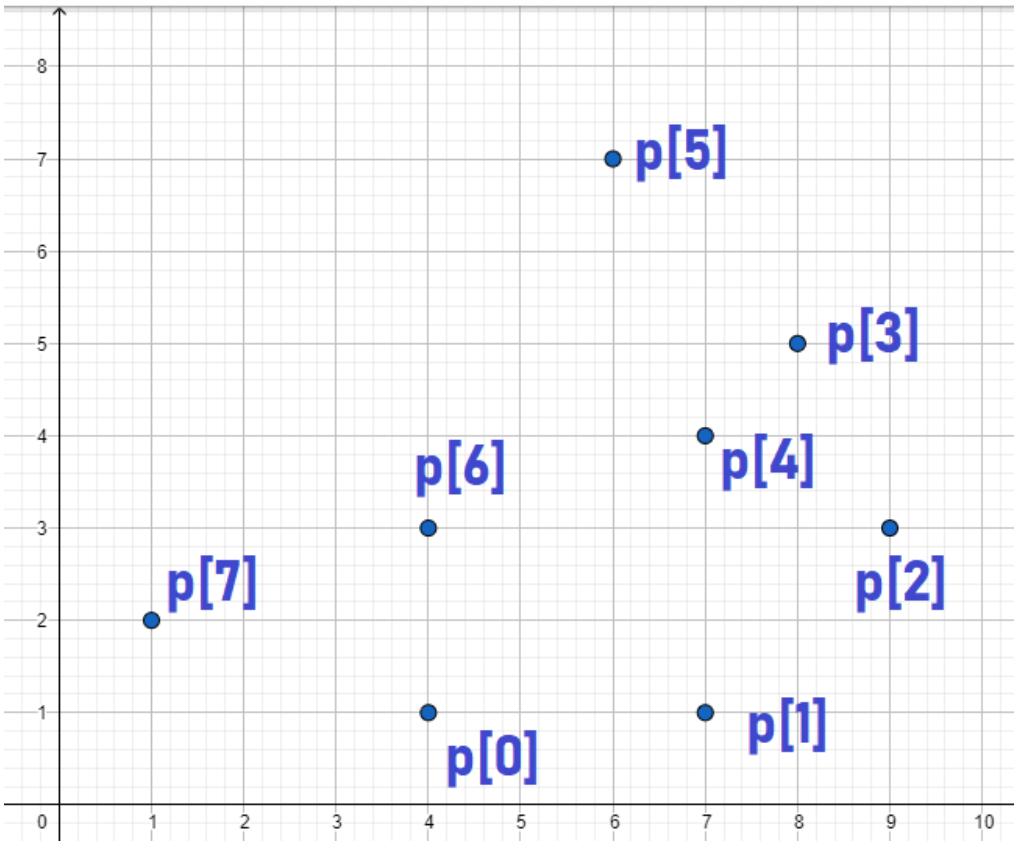
Também visto como
“Envoltória convexa” ou
“Convex Hull”;

Algoritmo utilizado:
“Graham’s Scan”



Fecho convexo

Ordenação pelo ângulo:



Fecho convexo

Crio uma pilha de pontos;

No final do algoritmo, os pontos na pilha farão parte do fecho convexo;

Inicialmente, apenas os pontos p[0], p[1] e p[2] estarão na pilha.

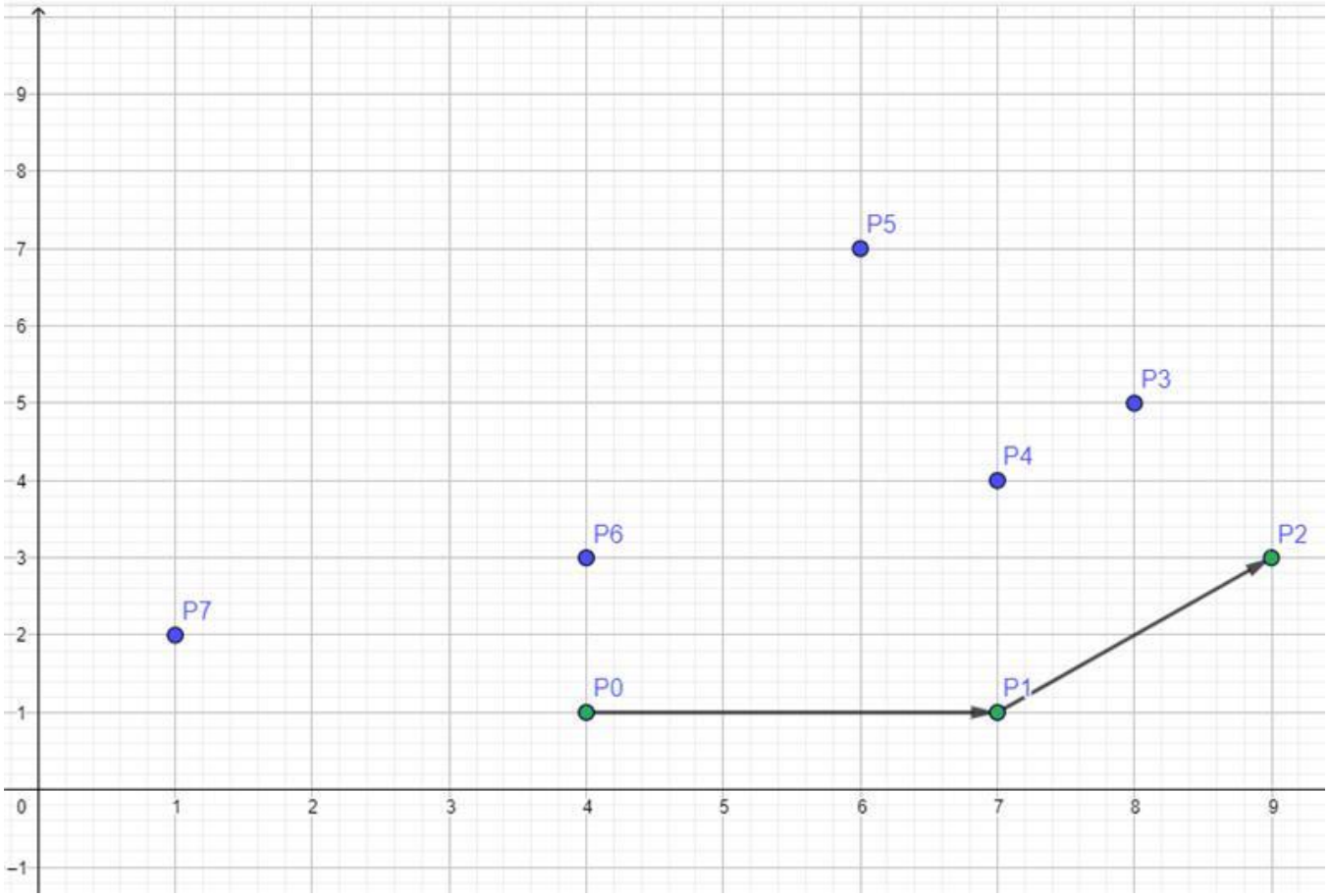
p[2]
p[1]
p[0]
Pilha

Fecho convexo

Se eu tiver uma curva à esquerda, eu adiciono o ponto de análise à pilha e vou para o próximo ponto;

Se eu tiver uma curva à direita, eu elimino o último elemento da pilha;

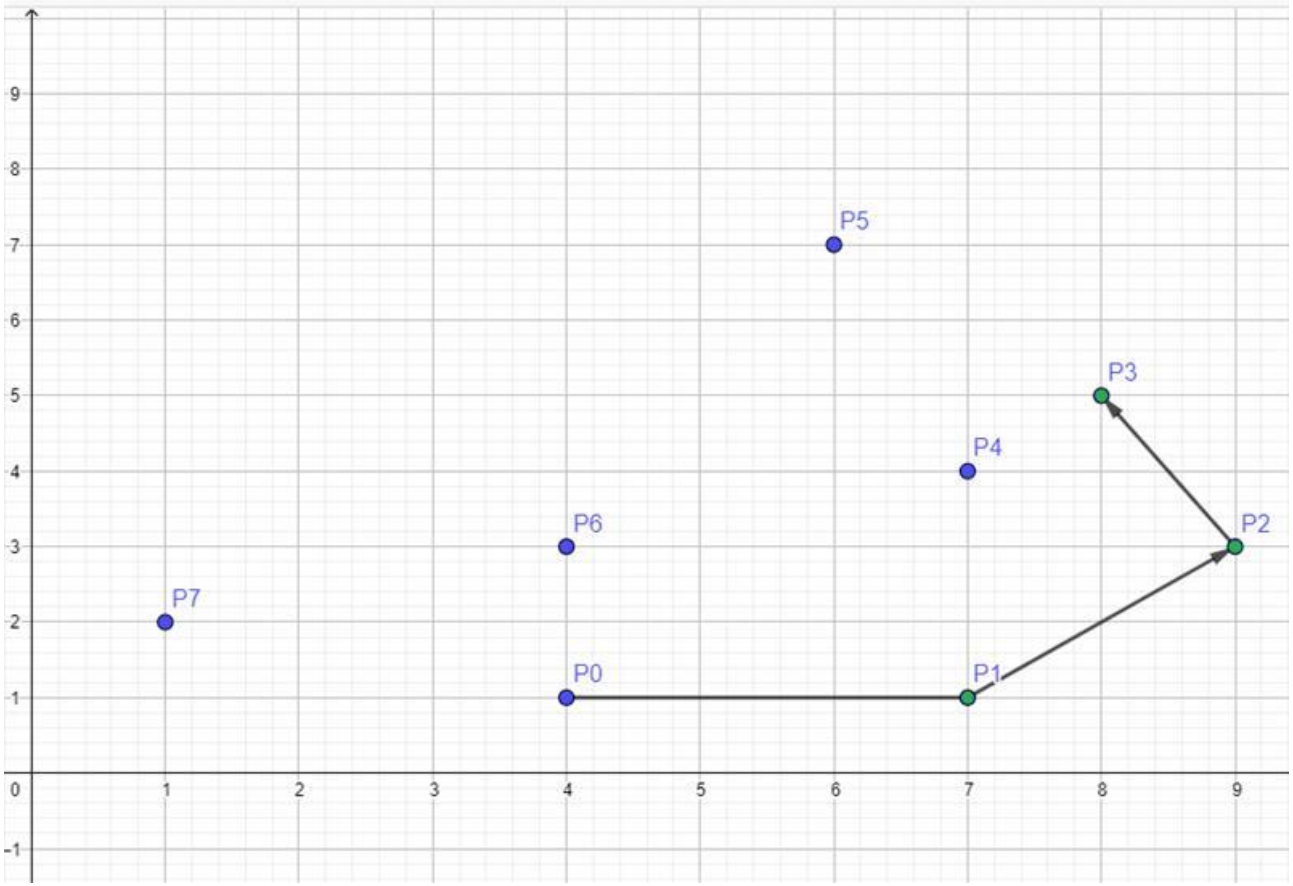
Fecho convexo



p[2]
p[1]
p[0]
Pilha

Fecho convexo

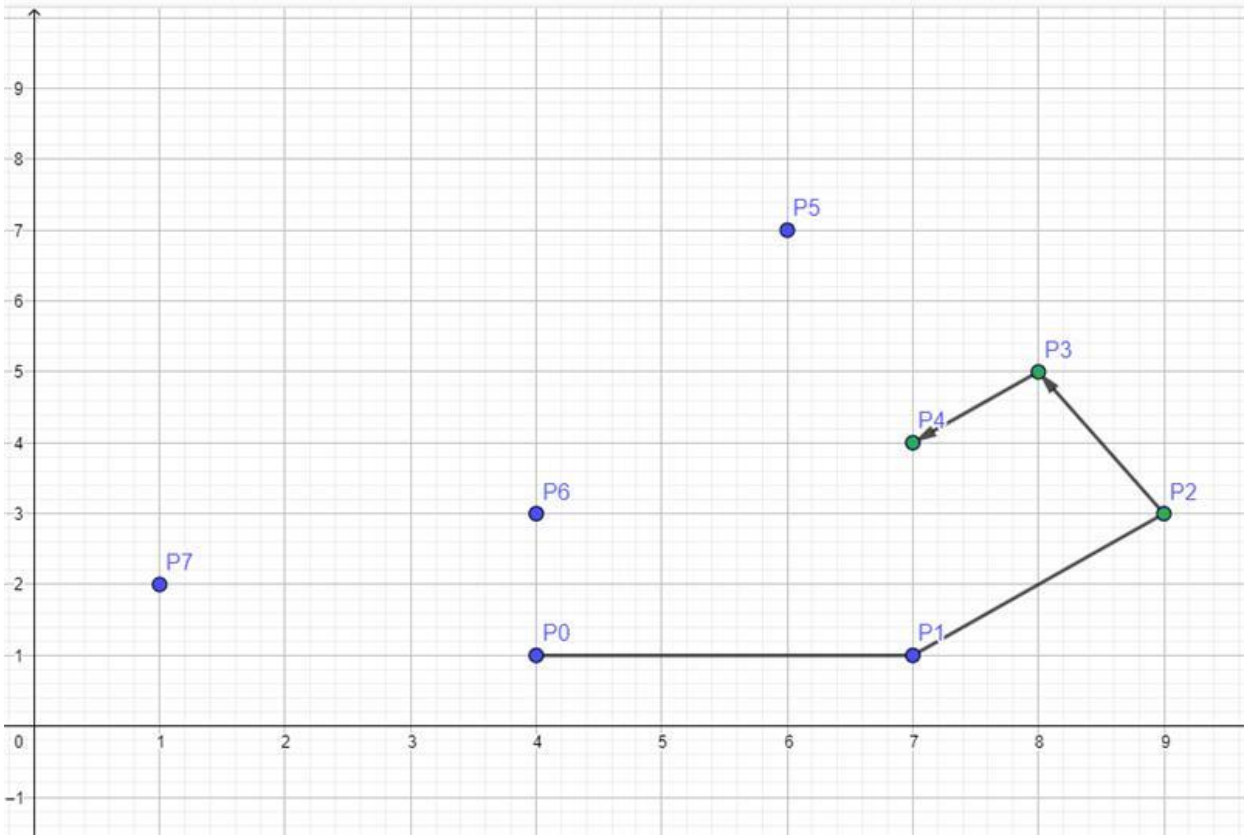
Curva à esquerda, OK



p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

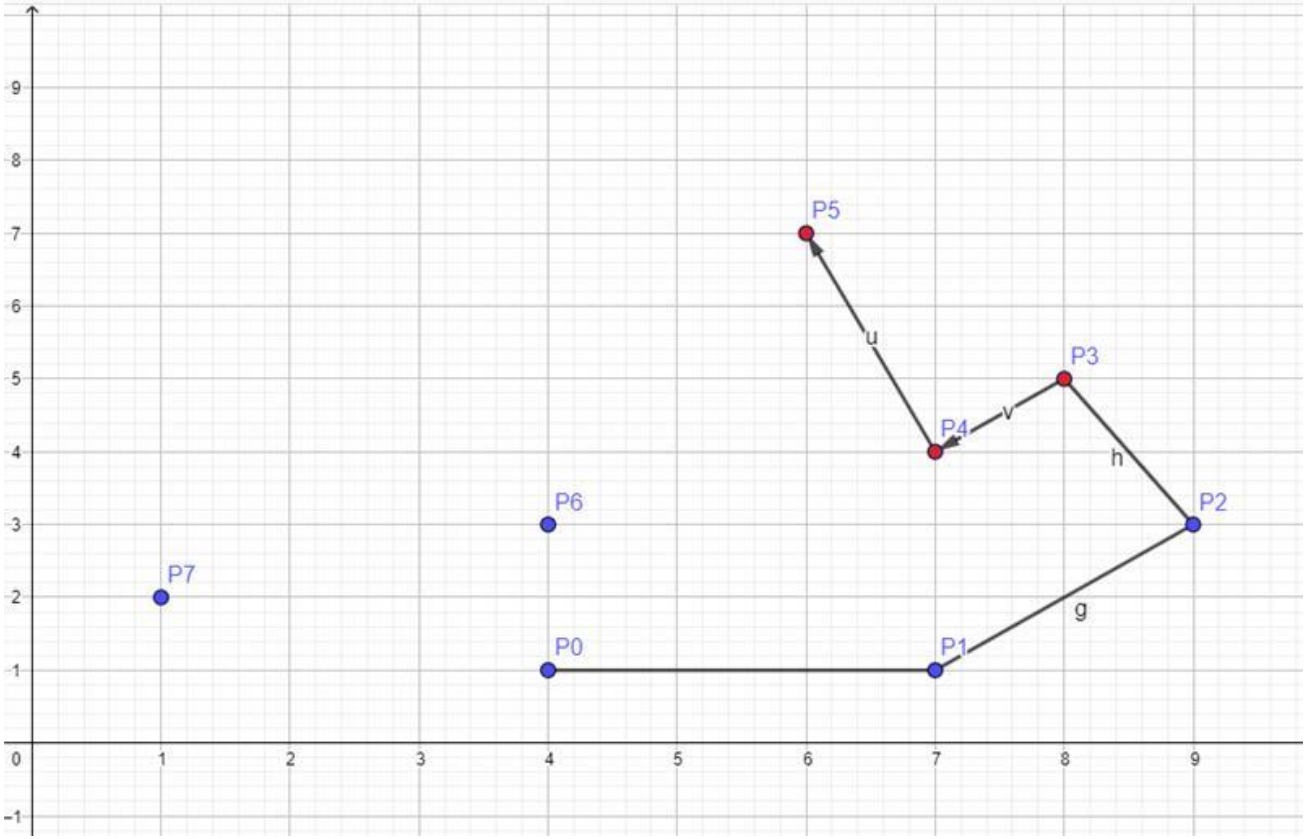
Curva à esquerda, OK



p[4]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

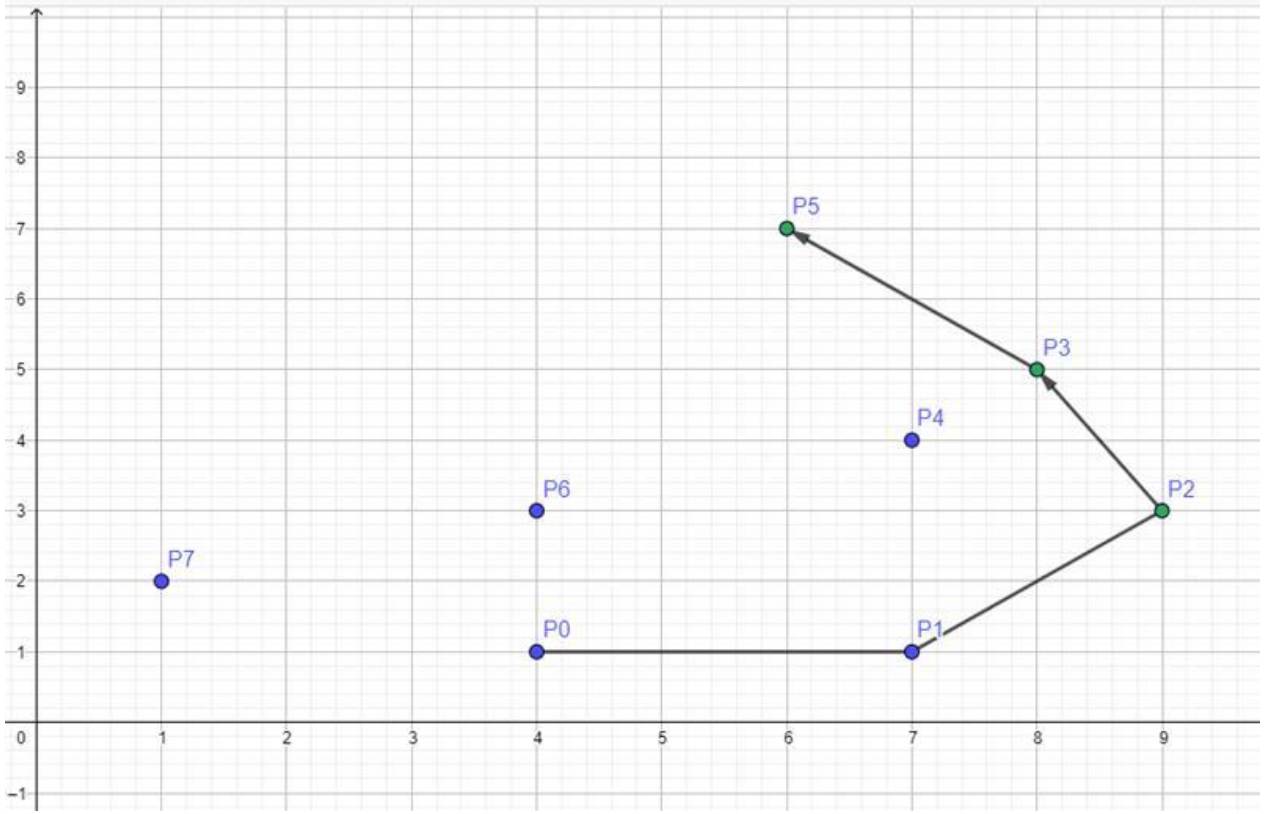
Curva à direita, NÃO OK -> removo o último da pilha



p[4]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

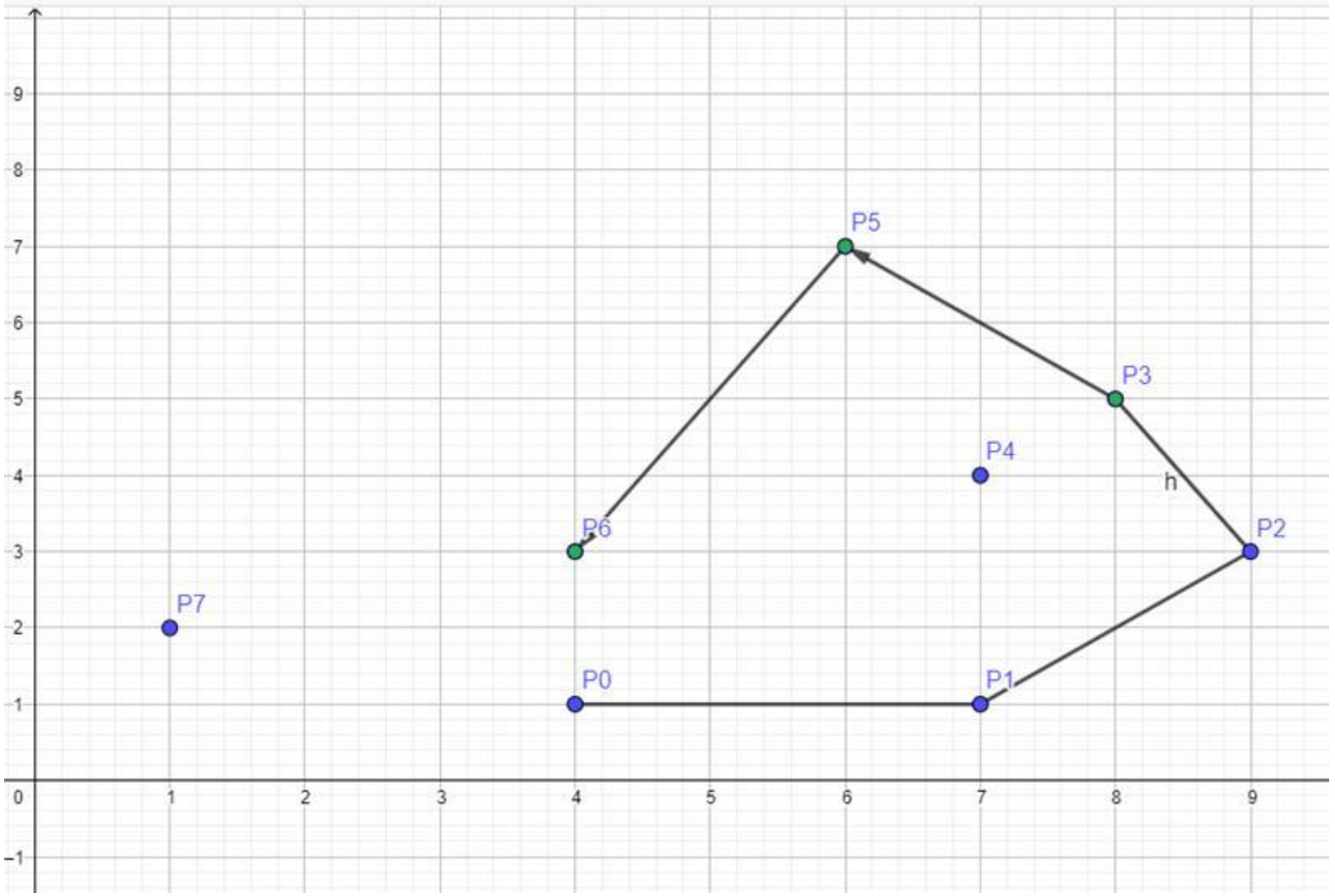
Curva à esquerda, OK



p[5]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

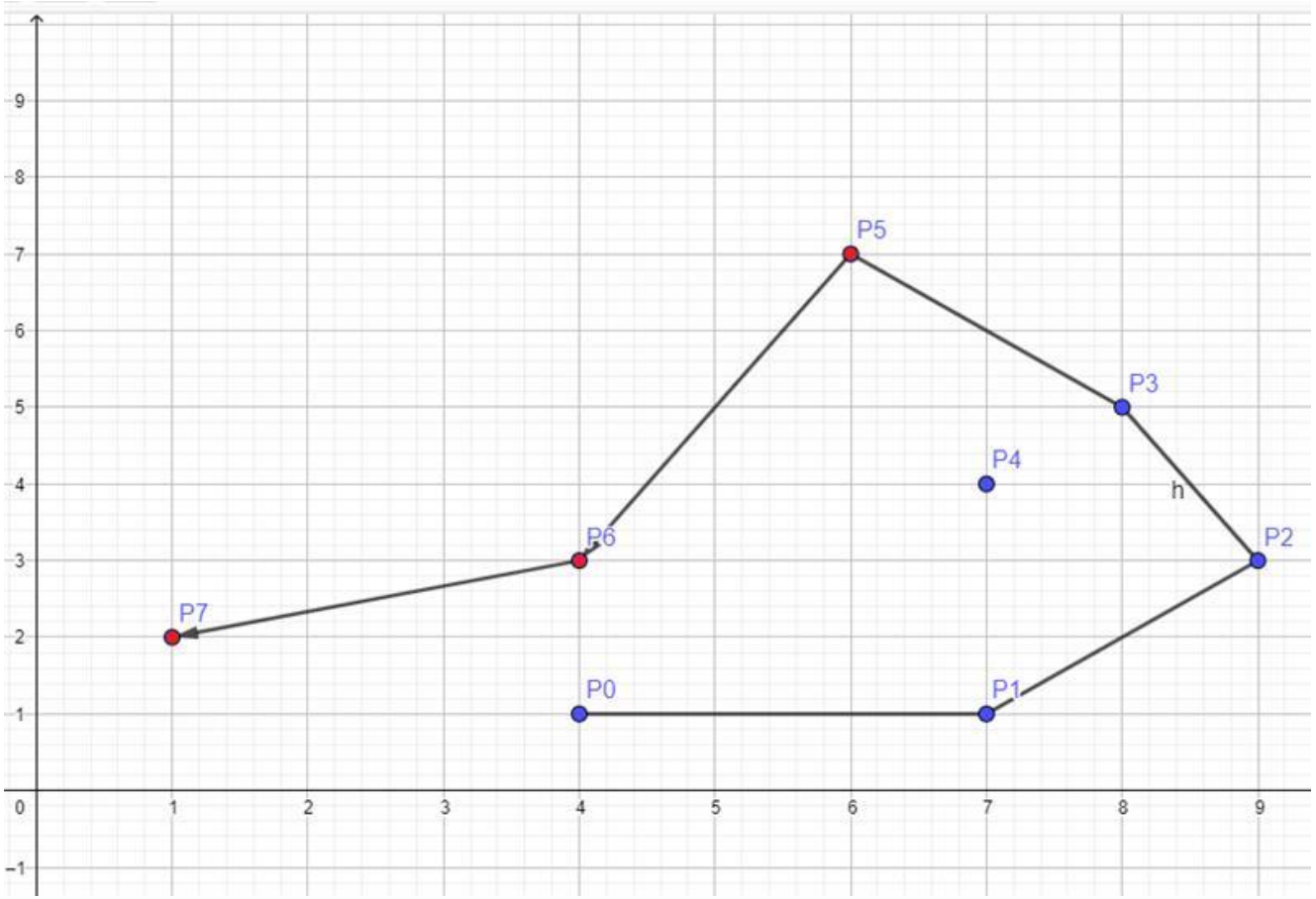
Curva à esquerda, OK



p[6]
p[5]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

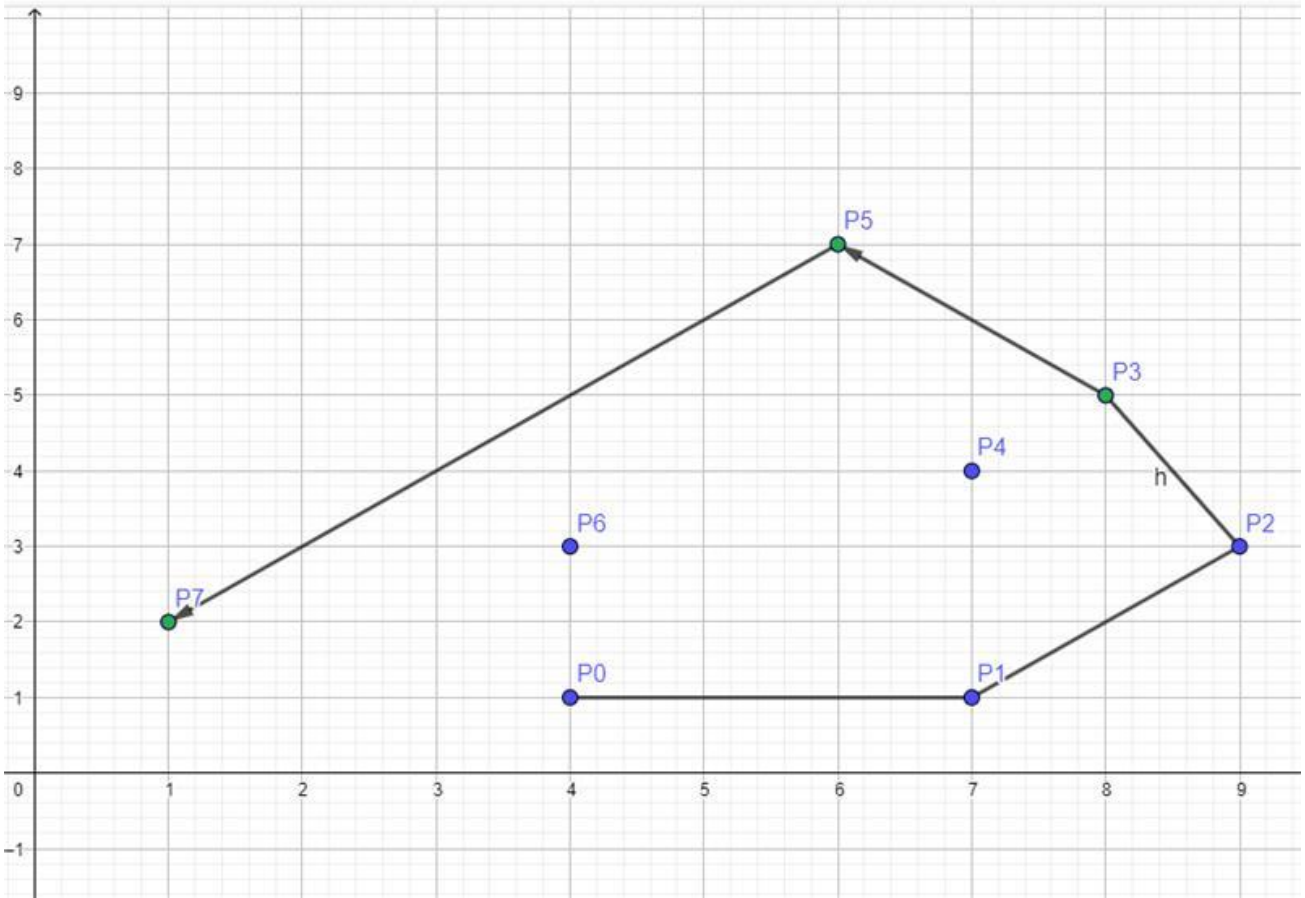
Curva à direita, NÃO OK -> removo o último da pilha



p[6]
p[5]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

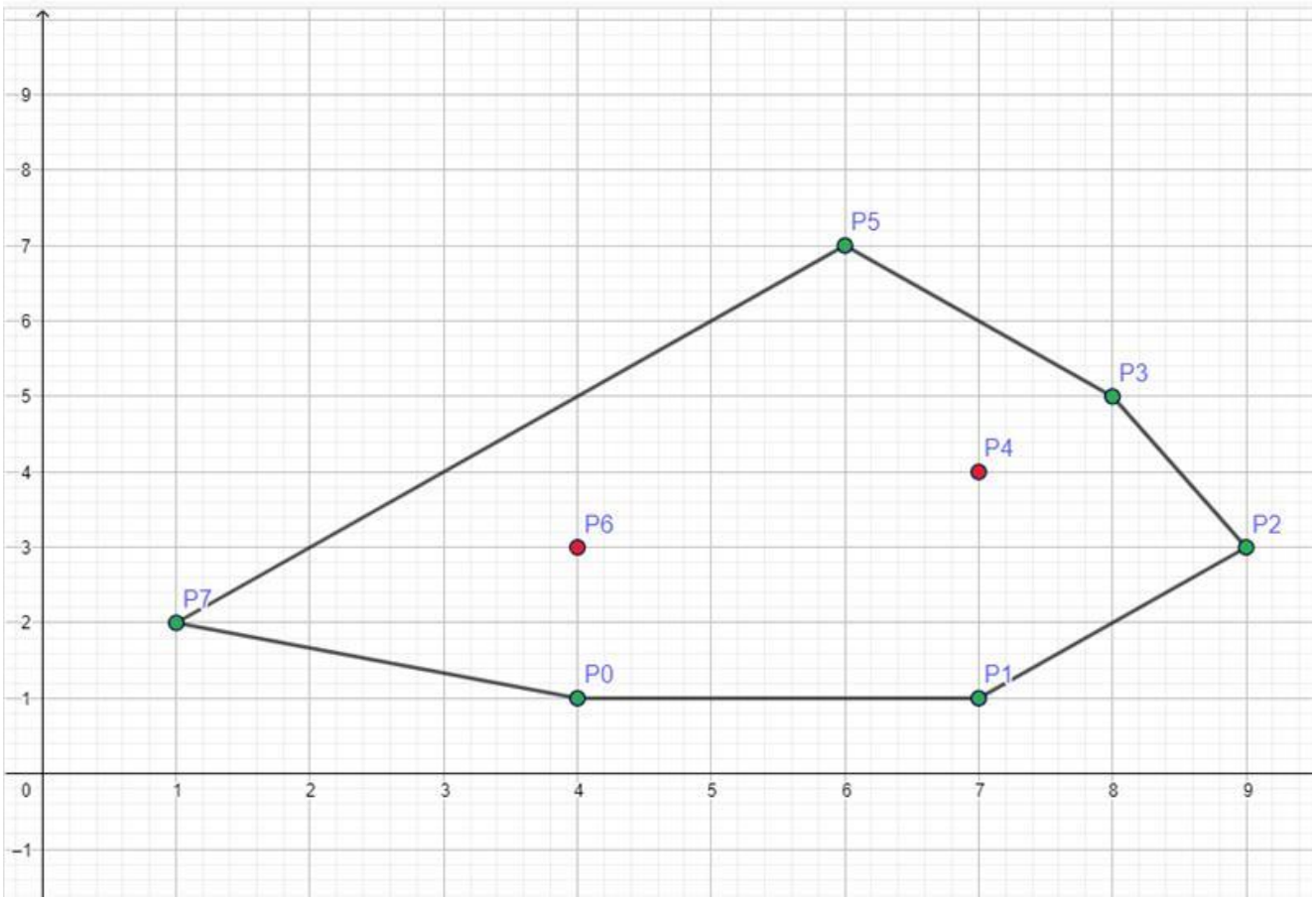
Curva à esquerda, OK



p[7]
p[5]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

PRONTO!



p[7]
p[5]
p[3]
p[2]
p[1]
p[0]
Pilha

Fecho convexo

- Vídeo: https://youtu.be/Ps1idzOx6LA