Geometry

Introdução e aplicações ($\mathbb{R}^2$)

André Fakhoury *(baseado nos slides de Gabriel Camargo)*

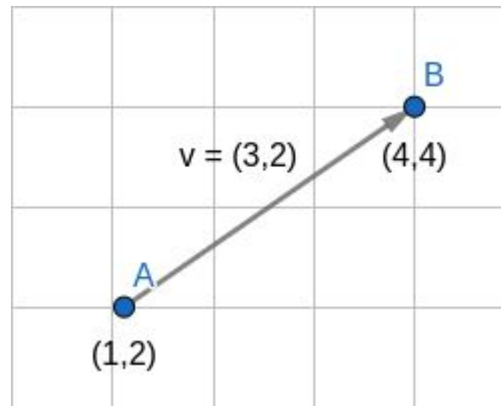
Cronograma

- Pontos e vetores
- Produto escalar
- Produto vetorial
- Distâncias ponto-ponto e ponto-vetor
- Aplicações
  - Verificar se dois segmentos se intersectam
  - Verificar se um ponto está dentro de um polígono convexo
  - Verificar se um ponto está dentro de um polígono qualquer
  - Área de polígonos (shoelace)
  - Convex Hull

Pontos e Vetores

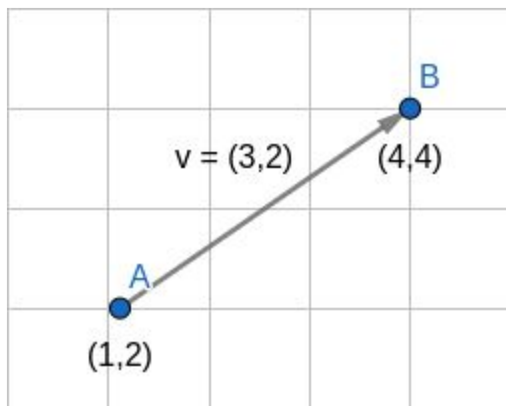
Pontos e Vetores

- **Ponto:** representado por suas coordenadas cartesianas (x, y)
- **Vetor:** classe de equipolência de segmentos orientados
  - Se (A, B) é um segmento orientado, $\vec{v} = \vec{AB}$ é o vetor correspondente



Pontos e Vetores

- **Operações ponto-vetor:** se A e B são pontos e *v* é um vetor:
  - A + v = B
  - Ponto + Vetor = Ponto



A = (1, 2), B = (4, 4), v = (3, 2)

- A + v = B
- B - v = A
- B - A = v

Representação em C++

- Ambos (pontos e vetores) podem ser representados por dois valores
  - A = (x, y)
  - v = (x, y)
- Mesma struct para armazenar os dois!

Representação em C++

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

  Point() : x(), y() {}
  Point(T x, T y) : x(x), y(y) {}
  template<typename Tp> Point(Point<Tp> p) : x(p.x), y(p.y) {}

  // basic operators
  template<typename Tp> Point<T> operator+(Point<Tp> const& p) const {return Point<T>(x + p.x, y + p.y); }
  template<typename Tp> Point<T> operator-(Point<Tp> const& p) const { return Point<T>(x - p.x, y - p.y); }
  template<typename Tp> Point<T> operator*(Tp&& p) const { return Point<T>{x * p, y * p}; }
  template<typename Tp> Point<T> operator/(Tp&& p) const { return Point<T>{x / p, y / p}; }
};
```

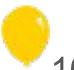
Norma de um vetor

Norma de um vetor

$$|v| = \sqrt{x^2 + y^2}$$

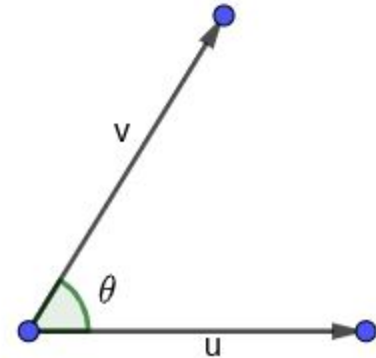
- Sempre não negativa
- Vetor unitário: **||u|| = 1**
- Normalização (para tornar um vetor unitário): $\dfrac{\vec{v}}{\|\vec{v}\|}$

Produto escalar (dot product)



Produto escalar (dot product)

$$\vec{u} \cdot \vec{v} = \begin{cases} 0 & \text{se } \vec{u} = \vec{0} \text{ ou } \vec{v} = \vec{0} \\ \|\vec{u}\| \|\vec{v}\| \cos\theta & \text{se } \vec{u} \neq \vec{0} \text{ e } \vec{v} \neq \vec{0}, \end{cases}$$



- Considera o menor ângulo entre os vetores (centrados na origem)

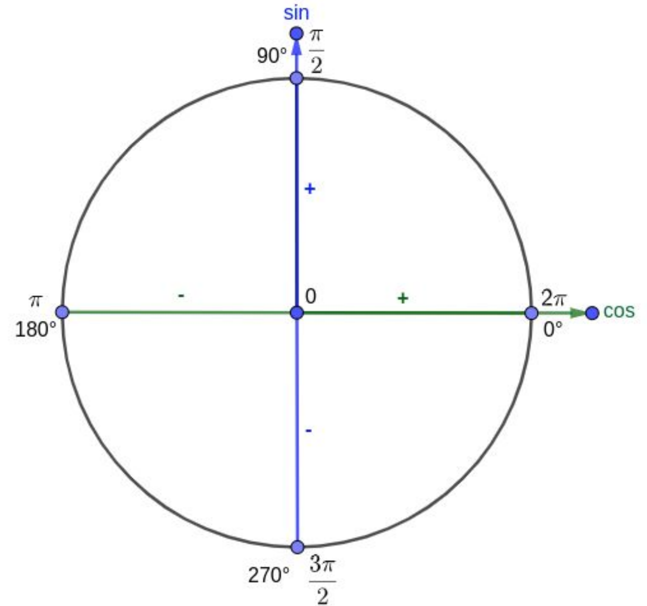
$$\vec{u} \cdot \vec{v} = x_1 x_2 + y_1 y_2$$

Produto escalar (dot product)

Propriedades:

$$u \cdot v = |u||v| \cos \theta$$

- $u \cdot v = 0 \implies \cos \theta = 0 \implies \theta = 90°$
- $u \cdot v > 0 \implies \cos \theta > 0 \implies \theta < 90°$
- $u \cdot v < 0 \implies \cos \theta < 0 \implies \theta > 90°$
- $a \cdot (b + c) = a \cdot b + a \cdot c$
- $u \cdot u = |u|^2 \implies |u| = \sqrt{u \cdot u} = \sqrt{x^2 + y^2}$
- $\cos \theta = \frac{u \cdot v}{|u||v|}$



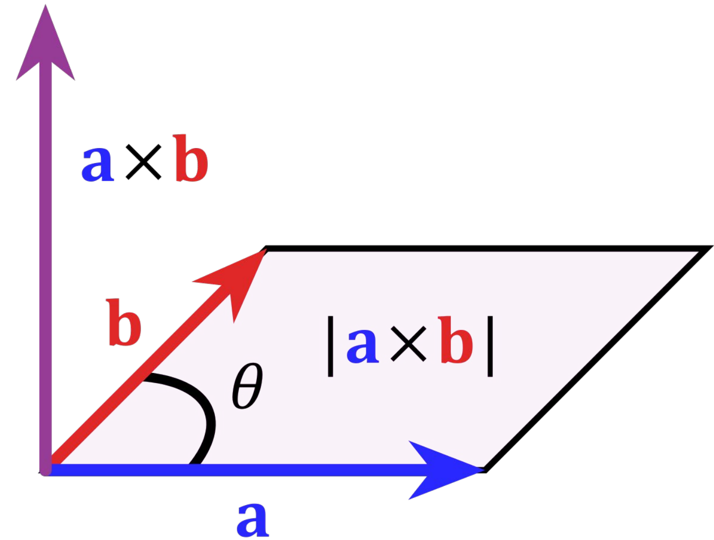
Produto vetorial (cross product)

Produto vetorial (cross product)

- Caso 3D (x, y, z)
- É um **vetor!**

$$\vec{u} \wedge \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

$$\|\vec{u} \wedge \vec{v}\| = \|\vec{u}\| \; \|\vec{v}\| \text{ sen } \theta$$



Produto vetorial (cross product)

- E no $\mathbb{R}^2$?

Produto vetorial (cross product)

- E no $\mathbb{R}^2$?
    - *“Pseudo-scalar product”*

Produto vetorial (cross product)

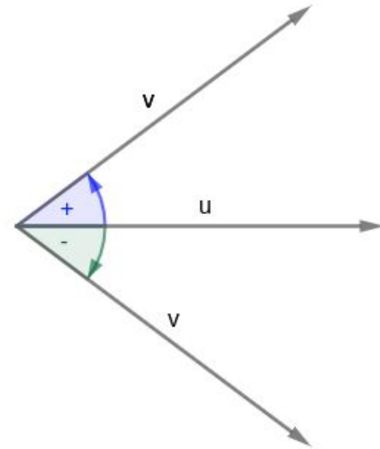
- E no $\mathbb{R}^2$?
  - *“Pseudo-scalar product”*
  - Também iremos considerar o sinal

Produto vetorial (cross product)

- E no $\mathbb{R}^2$?
  - *"Pseudo-scalar product"*
  - Também iremos considerar o sinal

$$u \times v = |u||v| \sin \theta$$

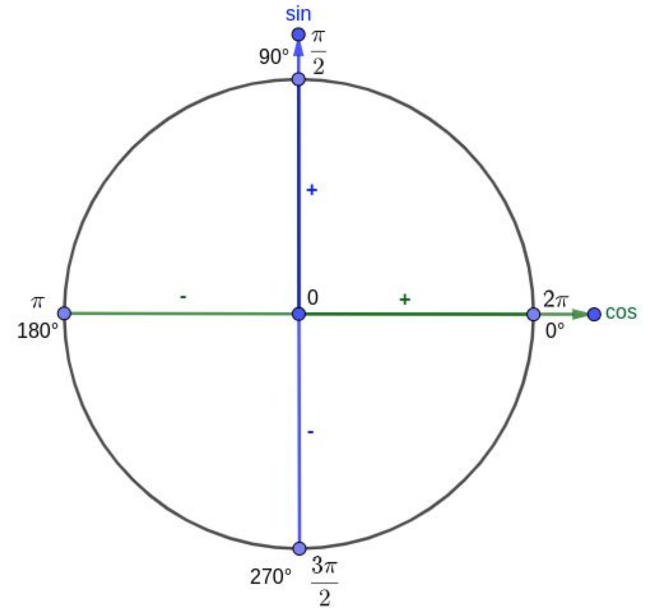
$$\begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix} = u_x v_y - v_x u_y$$



Produto vetorial (cross product)

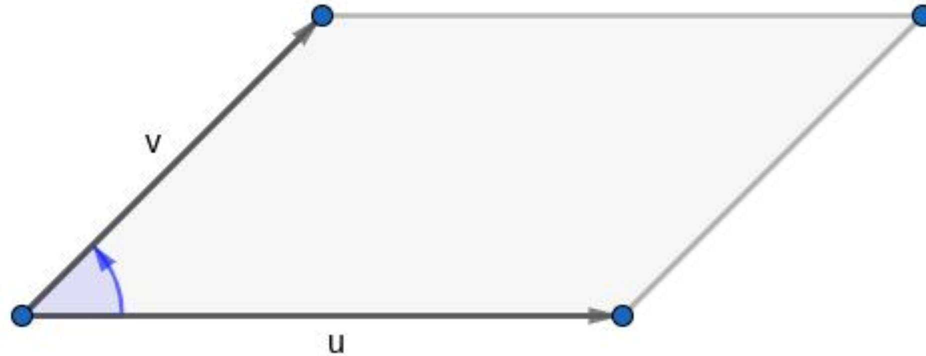
Propriedades:

- $u \times v = 0 \implies \sin\theta = 0 \implies \theta = 0, \pi$
- $u \times v > 0 \implies \sin\theta > 0 \implies \theta \in (0, \pi)$
- $u \times v < 0 \implies \sin\theta < 0 \implies \theta \in (-\pi, 0)$
- $a \times (b + c) = a \times b + a \times c$
- $\sin\theta = \frac{u \times v}{|u||v|}$



Produto vetorial (cross product)

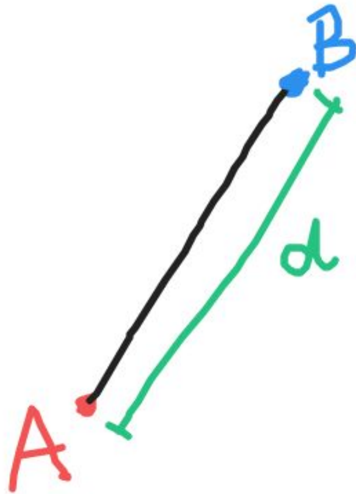
O módulo do produto vetorial nos dá a área do paralelogramo:



Distância ponto-ponto

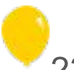
Distância ponto-ponto

- $d(A, B) = ||AB|| = ||(x_b - x_a, y_b - y_a)||$



$$d = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

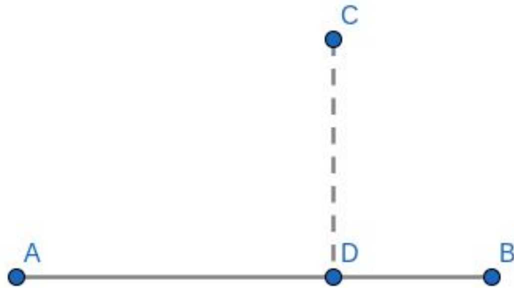
Distância ponto-segmento



Distância ponto-segmento

- Menor distância entre um ponto e um segmento de reta

caso 1

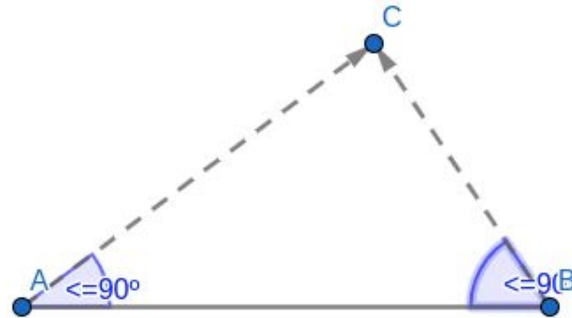


caso 2

Distância ponto-segmento

- Menor distância entre um ponto e um segmento de reta
- Como saber se está no caso 1?

$$\vec{AB} \cdot \vec{AC} >= 0$$
$$\vec{BA} \cdot \vec{BC} >= 0$$



Distância ponto-segmento

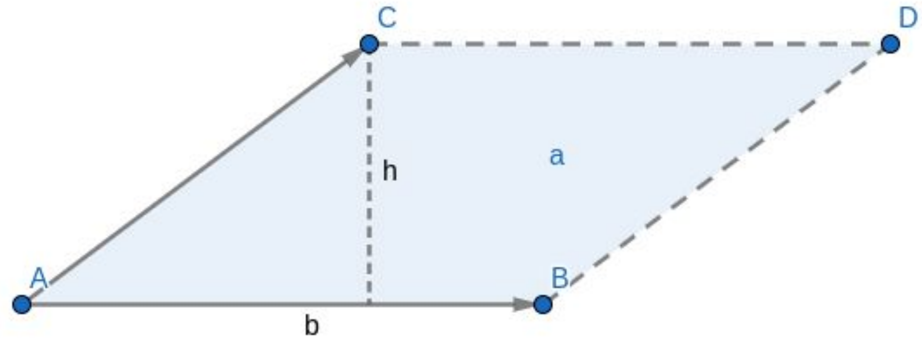
- Menor distância entre um ponto e um segmento de reta
- Caso 1:

$$a = |\vec{AB} \times \vec{AC}|$$

$$a = bh$$

$$h = \frac{a}{b}$$

$$h = \frac{|\vec{AB} \times \vec{BC}|}{||\vec{AB}||}$$



Distância ponto-segmento

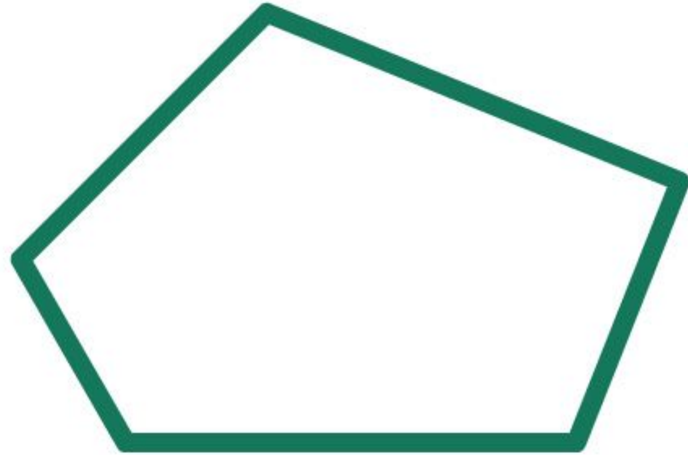
- Menor distância entre um ponto e um segmento de reta
- Caso 2:

$$min(|\overrightarrow{AC}|, |\overrightarrow{BC}|)$$

Distância ponto-segmento

```
template <typename T>
struct Point {
  ...

  inline double len() const { return hypot(x, y); }
  template<typename Tp> inline T dot(Point<Tp> const& p) const { return x * p.x + y * p.y; }
  template<typename Tp> inline T cross(Point<Tp> const& p) const { return x * p.y - y * p.x; }

  inline double dist_to_segment(Point const& a, Point const& b) const {
     pt c = *this;

     pt ac = c - a;
     pt bc = c - b;
     pt ab = b - a;
     pt ba = a - b;

     if (ab.dot(ac) >= 0 && ba.dot(bc) >= 0) {
        return abs(ab.cross(bc)) / ab.len();
     }

     return min({ac.len(), bc.len()});
  }
};
```

Concavidade de polígonos



Aplicações

Aplicações

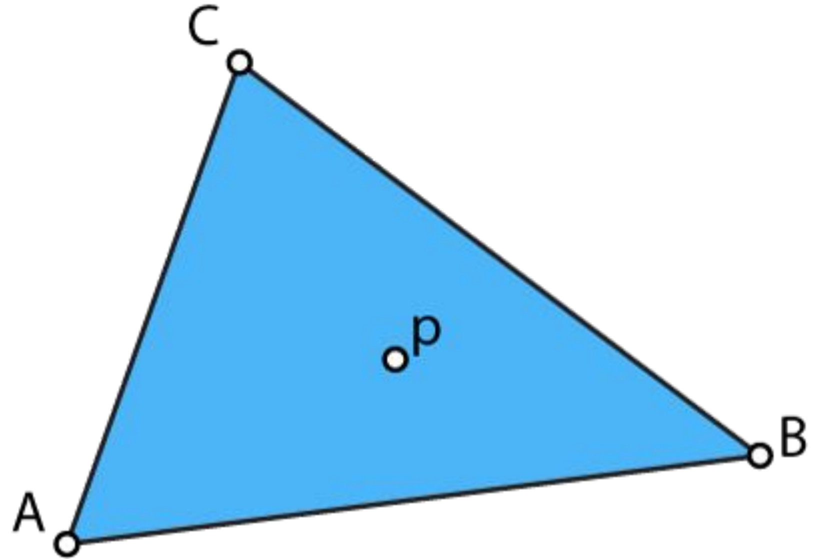
Verificar se um ponto está dentro de um triângulo

Aplicações

Verificar se um ponto está dentro de um triângulo

Problema

Dado os vértices A, B e C de um triângulo e um ponto p, verificar se *p* está dentro ou não deste triângulo.

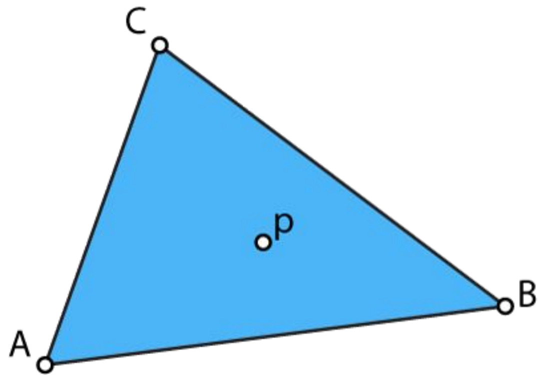


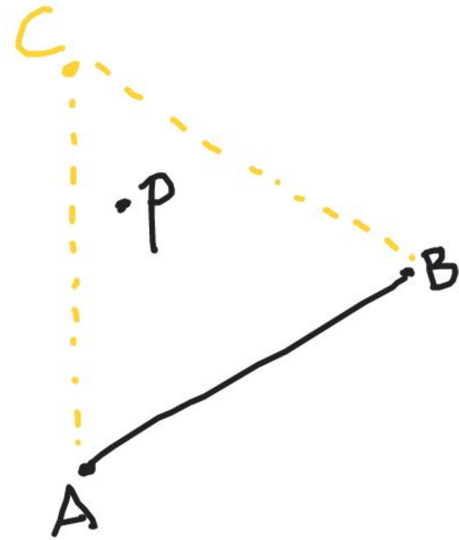
Aplicações

Verificar se um ponto está dentro de um triângulo

Solução

Vamos analisar inicialmente a aresta AB:



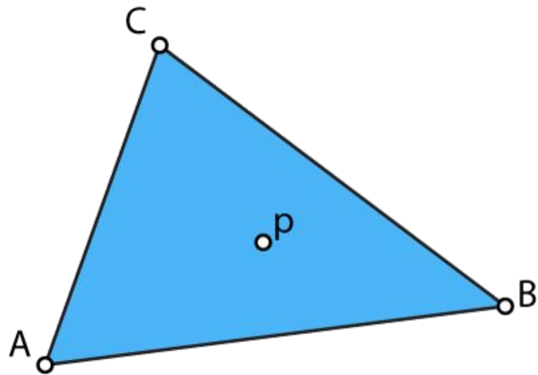


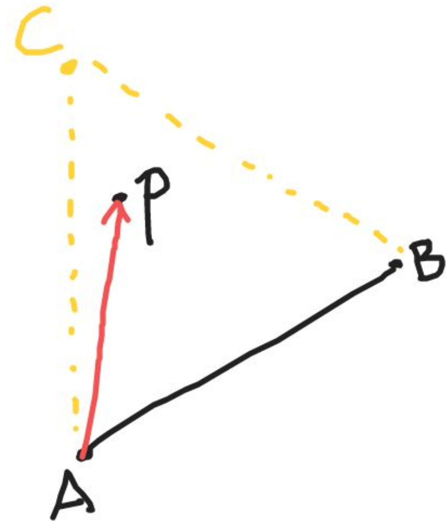
Aplicações

Verificar se um ponto está dentro de um triângulo

Solução

Traçando um vetor de A para *p*, temos:



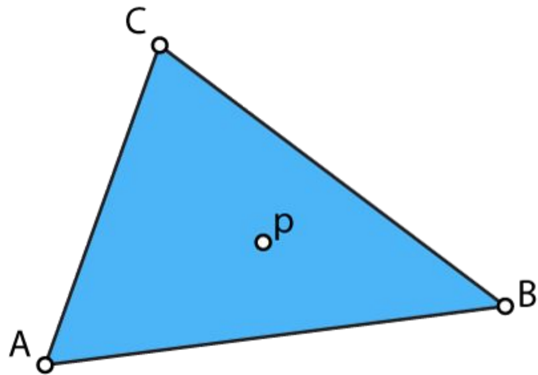


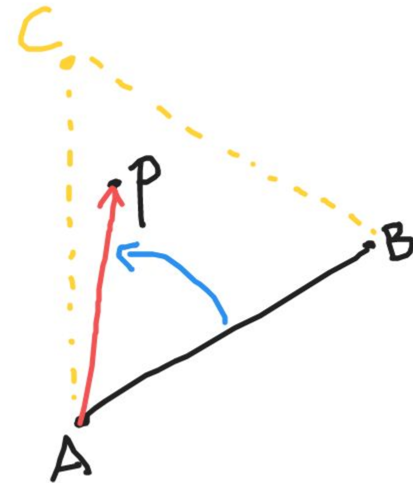
Aplicações

Verificar se um ponto está dentro de um triângulo

Solução

Podemos verificar o *Produto Vetorial* para conferir se AP está à esquerda do vetor AB!



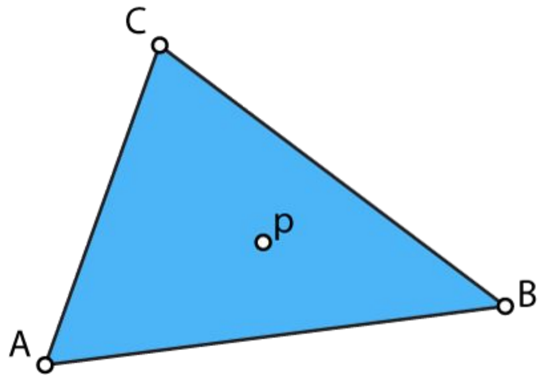


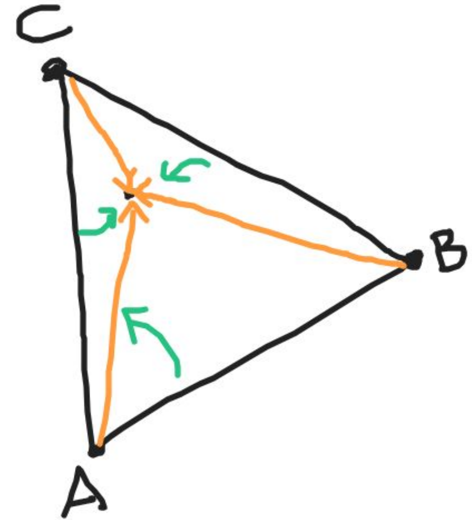
Aplicações

Verificar se um ponto está dentro de um triângulo

Solução

E isso deve ser válido para todas as outras arestas BC e CA





Aplicações

Verificar se um ponto está dentro de um triângulo

```
bool is inside triangle(pt a, pt b, pt c, pt p) {
  if((b-a).cross(p-a) < 0) return false;
  if((c-b).cross(p-b) < 0) return false;
  if((a-c).cross(p-c) < 0) return false;

  return true;
}
```

Aplicações

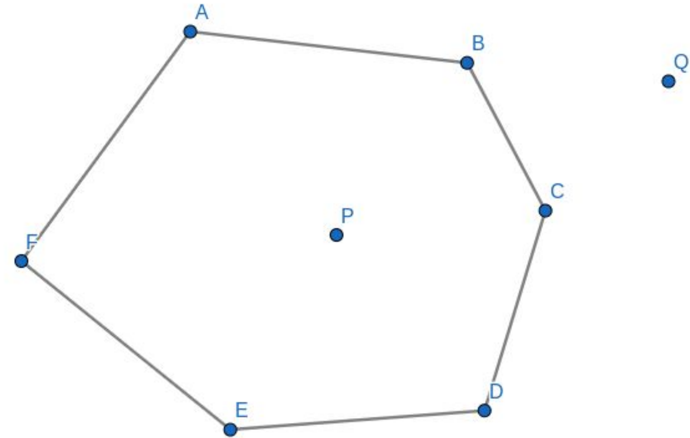
Verificar se um ponto está dentro de um polígono convexo

Aplicações

Verificar se um ponto está dentro de um polígono convexo

Problema

Dado um vetor de pontos ordenados em sentido anti-horário e um ponto, dizer se o ponto está dentro ou fora do polígono.

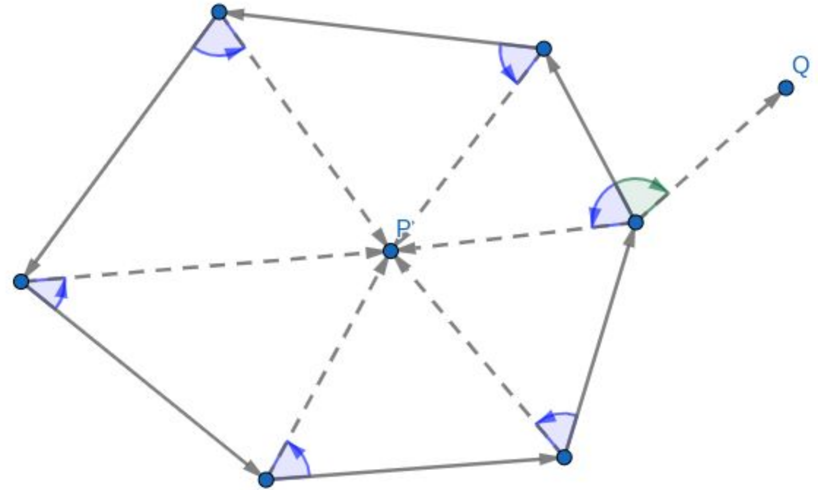


Aplicações

Verificar se um ponto está dentro de um polígono convexo

Iterar sobre todas as arestas verificando se o cross entre a aresta e o vetor do vértice a P é maior ou igual a zero

- Complexidade O(n)

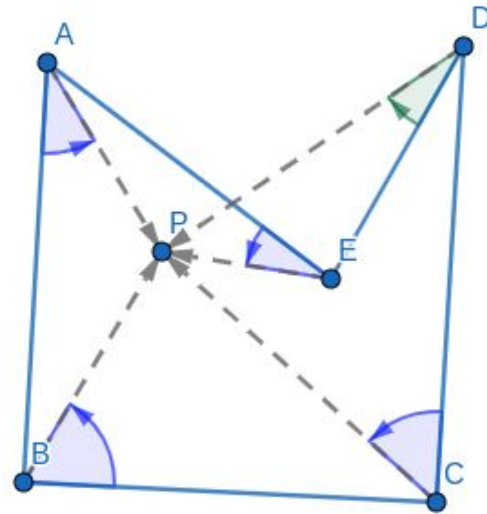


Aplicações

Verificar se um ponto está dentro de um polígono convexo

Observação

- Não funciona para polígonos côncavos



Aplicações

Verificar se um ponto está dentro de um polígono convexo

```
// estamos considerando que os pontos do polígono já estão ordenados em sentido anti-horário
bool isInsidePolygon (vector<Point>& polygon, Point p){
  bool isInside = true;

  int n = polygon.size();
  for(int i = 0; i < polygon.size(); i++){
     // quando i=n-1, estaremos olhando para o vetor entre o último e o primeiro ponto
     Point u = polygon[(i+ 1)%n]-polygon[i];
     Point v = p-polygon[i];

     // se está à direita está fora do polígono
     if((u.cross(v)) <  0){
        isInside =  false;
     }
  }

  return isInside;
}
```

Aplicações

Verificar se um ponto está dentro de um polígono convexo

- E se fossem várias queries? $O(n * q)$!!!

Aplicações

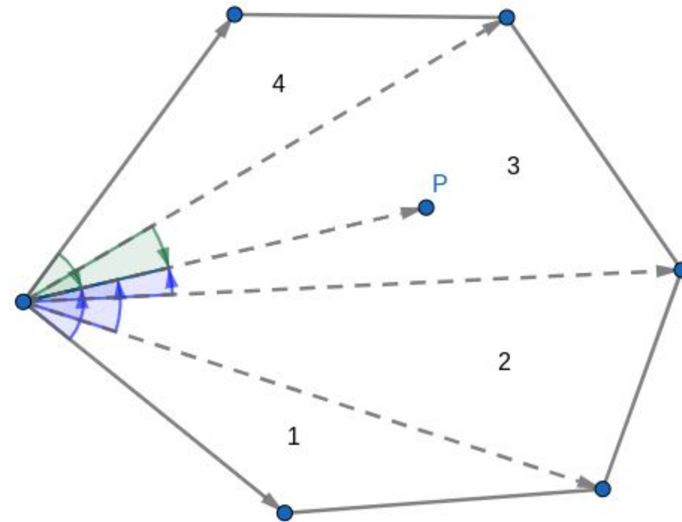
Verificar se um ponto está dentro de um polígono convexo

- E se fossem várias queries? O(n * q)!!!
- Precisamos achar um jeito de fazer cada query em O(logn):

Aplicações

Verificar se um ponto está dentro de um polígono convexo

- Fixar um vértice e separar o polígono em seções triangulares

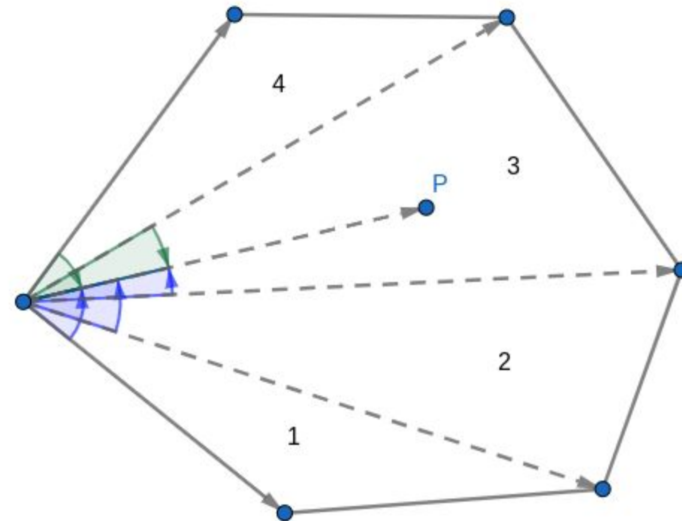


Aplicações

Verificar se um ponto está dentro de um polígono convexo

- Fixar um vértice e separar o polígono em seções triangulares

- Observações:
  - No triângulo em que o ponto está contido, ele está do lado direito de um lado e do lado esquerdo de outro
  - Nos outros, ele estará ou de um ou de outro em ambos os lados

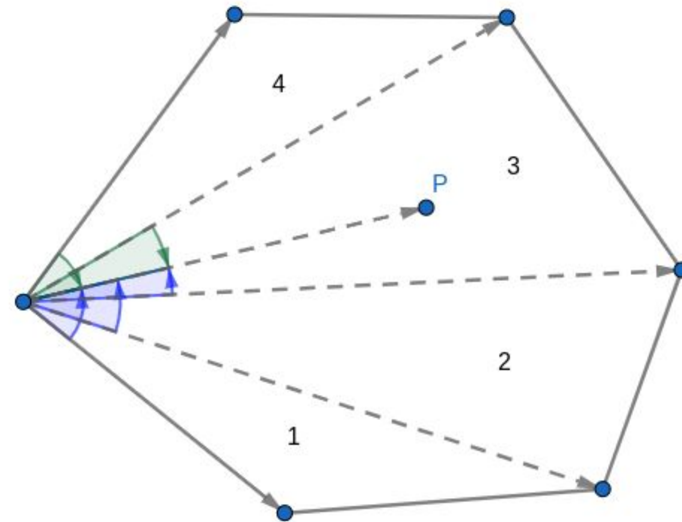


Aplicações

Verificar se um ponto está dentro de um polígono convexo

- Fixar um vértice e separar o polígono em seções triangulares

- Observações:
  - Nos primeiros vértices, ele estará do lado direito
  - Nos últimos vértices, ele estará do lado esquerdo
  - Busca binária para achar onde separa!!

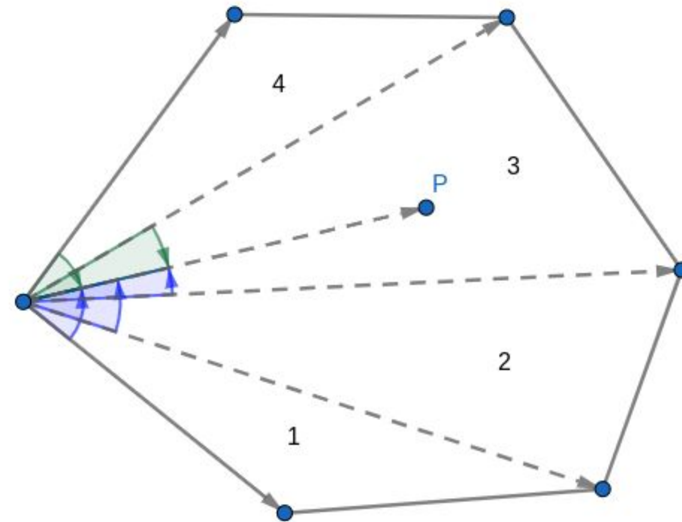


Aplicações

Verificar se um ponto está dentro de um polígono convexo

- Fixar um vértice e separar o polígono em seções triangulares

- Observações:
    - Após a busca binária, basta checar se este ponto pertence ao triângulo escolhido!



Aplicações

Verificar se um ponto está dentro de um polígono convexo

```
bool is_inside_triangle(pt a, pt b, pt c, pt p) {
  if ((b-a).cross(p-a) < 0) return false;
  if ((c-b).cross(p-b) < 0) return false;
  if ((a-c).cross(p-c) < 0) return false;
  return true;
}

bool is_inside(vector<pt> const& a, pt const& p) {
  const int n = a.size();

  int lo = 1, hi = n - 2;
  while(lo < hi) {
     int mi = (lo + hi + 1) / 2;

     pt u = a[mi] - a[0];
     pt v = p - a[0];

     if (u.cross(v) >= 0) lo = mi;
     else hi = mi - 1;
  }

  return is_inside_triangle(a[0], a[lo], a[(lo+1)%n], p);
}
```

Aplicações

Verificar se um ponto está dentro de um polígono qualquer

Aplicações

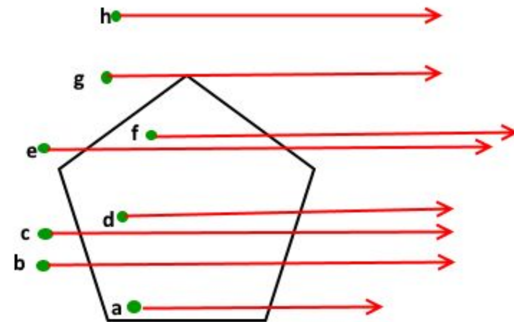
Verificar se um ponto está dentro de um polígono qualquer

- Agora, o polígono pode ser côncavo!

Aplicações

Verificar se um ponto está dentro de um polígono qualquer

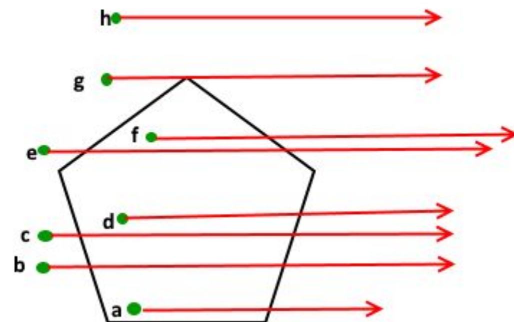
- Agora, o polígono pode ser côncavo!
- Desenha um segmento de reta entre o ponto e algum “infinito”



Aplicações

Verificar se um ponto está dentro de um polígono qualquer

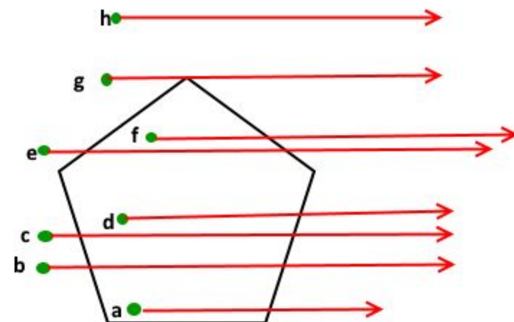
- Agora, o polígono pode ser côncavo!
- Desenha um segmento de reta entre o ponto e algum "infinito"
  - Conta o número de intersecções entre este segmento e as arestas do polígono



Aplicações

Verificar se um ponto está dentro de um polígono qualquer

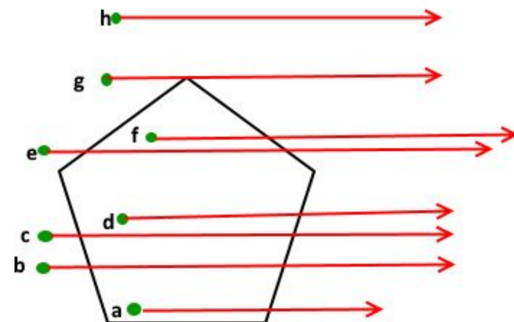
- Agora, o polígono pode ser côncavo!
- Desenha um segmento de reta entre o ponto e algum “infinito”
  - Conta o número de intersecções entre este segmento e as arestas do polígono
  - Se este número for ímpar: dentro
  - Senão, fora



Aplicações

Verificar se um ponto está dentro de um polígono qualquer

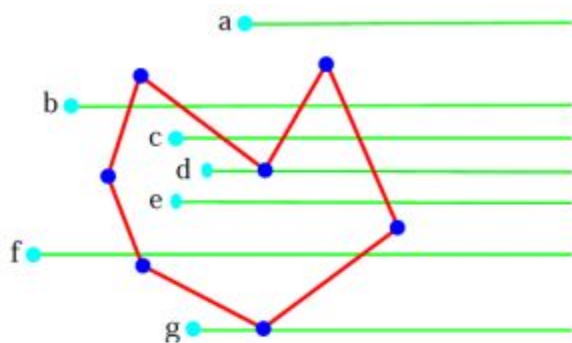
- Agora, o polígono pode ser côncavo!
- Desenha um segmento de reta entre o ponto e algum “infinito”
  - Conta o número de intersecções entre este segmento e as arestas do polígono
  - Se este número for ímpar: dentro
  - Senão, fora
  - “Para cada vez que ele sai, tem que entrar de novo”

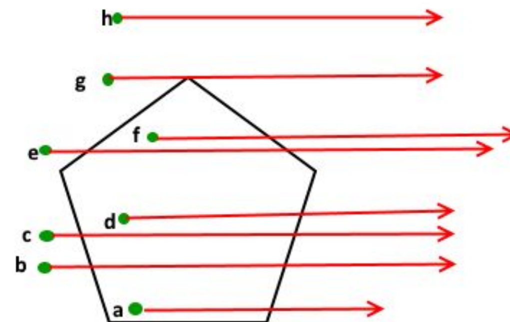


Aplicações

Verificar se um ponto está dentro de um polígono qualquer

- Solução:
  - Criar um segmento (ortogonal às arestas do polígono) com um dos endpoints sendo o “infinito”
  - Verificar se o número de intersecções entre este segmento e as arestas é ímpar





Aplicações

Verificar se um ponto está dentro de um polígono qualquer

```
/** Checks whether 1-dimensional segments [a, b] and [c, d] intersect */
template<typename T> bool segments_intersect_1d (T a, T b, T c, T d) {
  if (a > b) swap(a, b);
  if (c > d) swap(c, d);
  return max(a, c) <= min(b, d);
}

/** Checks whether segments AB and CD intersect */
template <typename T> bool segments intersect (Point<T> const& a, Point<T> const& b, Point<T> const& c, Point<T> const& d) {
  if (c.cross(a, d) == 0 && c.cross(b, d) == 0)
     return segments intersect 1d(a. x, b.x, c.x, d.x) && segments_intersect_1d(a. y, b.y, c.y, d.y);
  return sign(a.cross(b, c)) != sign(a. cross(b, d)) &&
        sign(c. cross(d, a)) != sign(c. cross(d, b));
}

/** Check if point p is inside polygon. Return: 0: outside, 1: inside, 2: boundary.
* BE CAREFUL: inf has to be greater than any other point, to make sure it isn't collinear with any edge */
template<typename T> int in polygon(Point<T> p, vector<Point<T>> const& v) {
  int n = v.size(), count = 0;

  const T inf = 2e9;
  Point<T> p2{p.x + 1, inf};
  for (int i = 0, j = n-1; i < n; j = i, i++) {
     if (p.in segment(v[i], v[j])) return 2;
     count += segments_intersect(p,  p2, v[i], v[j]);
  }

  return count % 2;
}
```

Aplicações

Área de um polígono (shoelace)

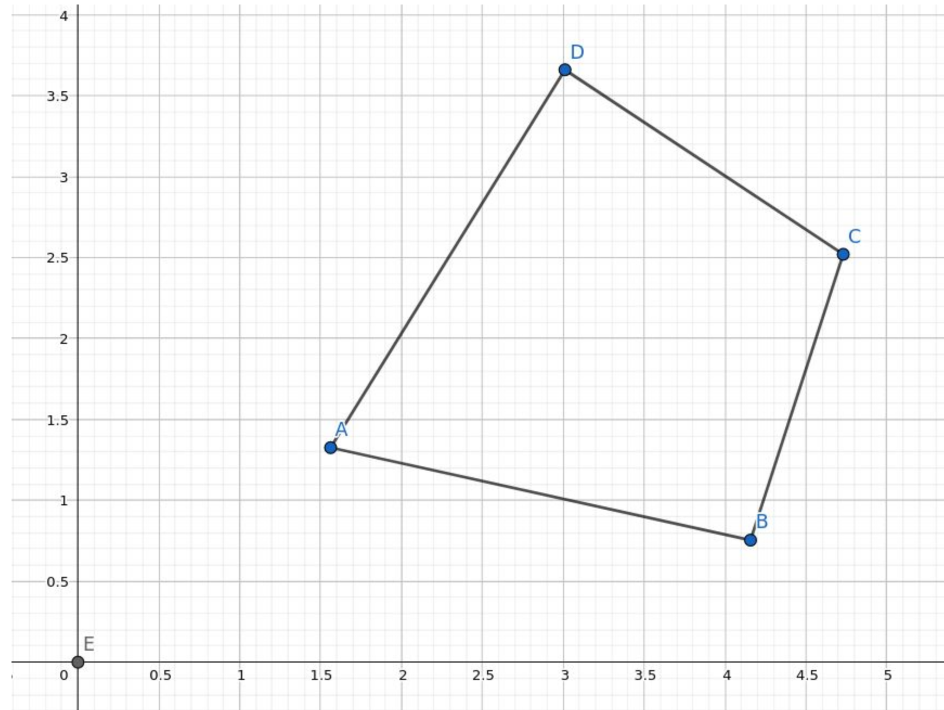
Aplicações

Área de um polígono (shoelace)

- Ideia base:
  - “Separar o polígono em triângulos”
  - Pegar cada aresta AB e calcular a área (com sinal) do triângulo ABO (com um vértice na origem O)
  - Os triângulos com área positiva e negativa irão se sobrepor, e apenas o triângulo interno sobrará

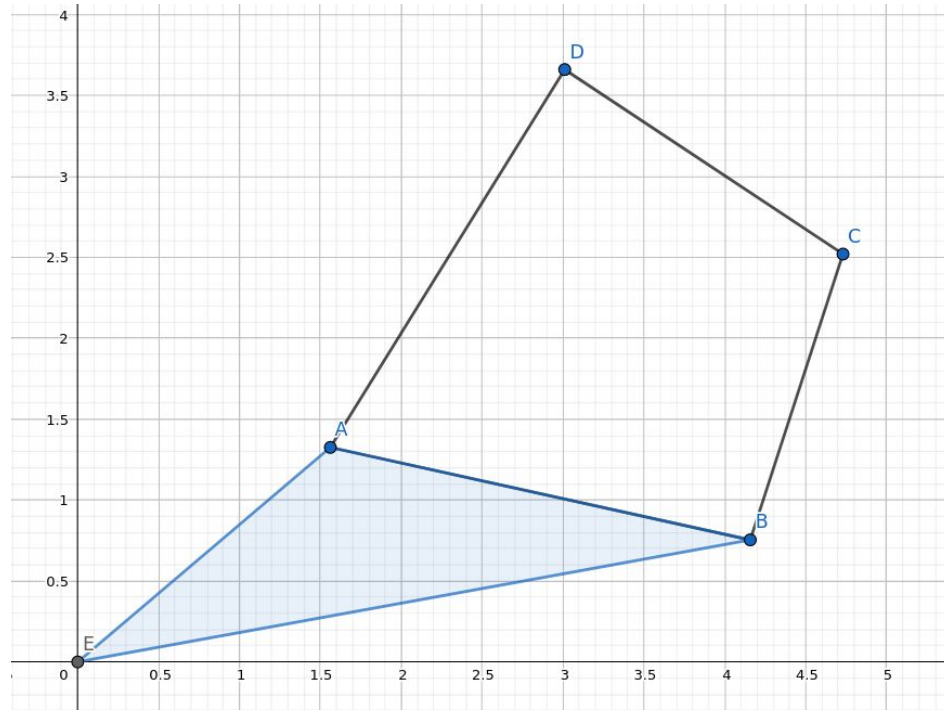
Aplicações

Área de um polígono (shoelace)



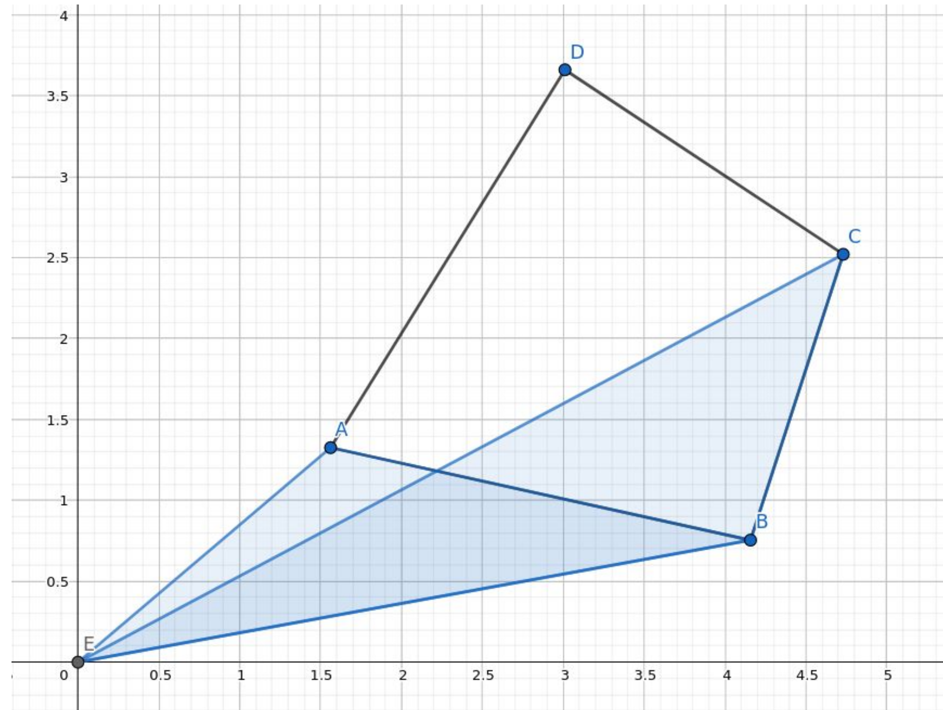
Aplicações

Área de um polígono (shoelace)



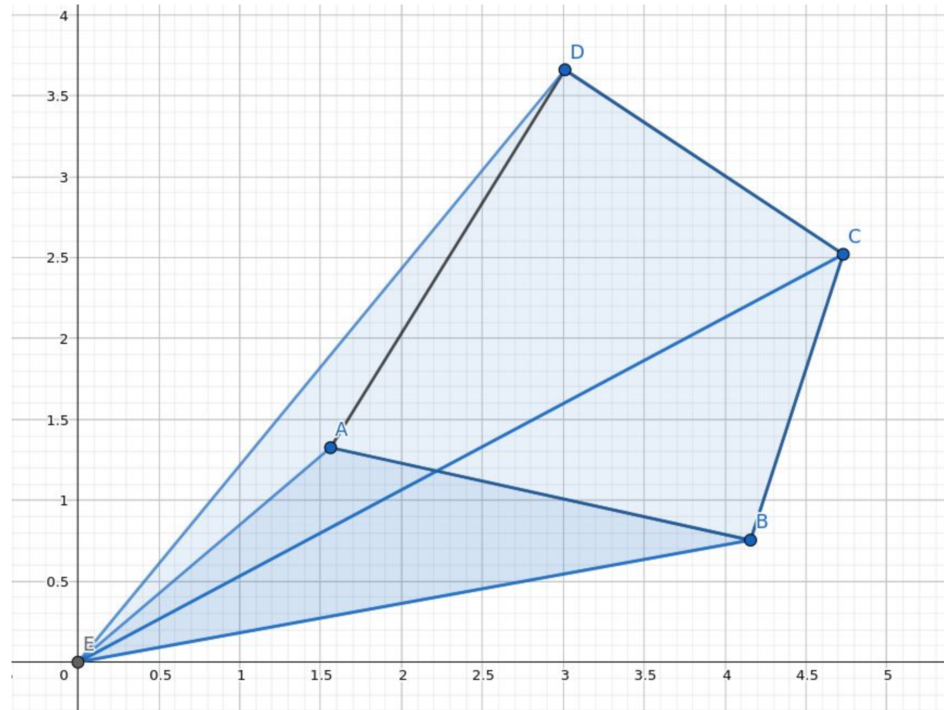
Aplicações

Área de um polígono (shoelace)



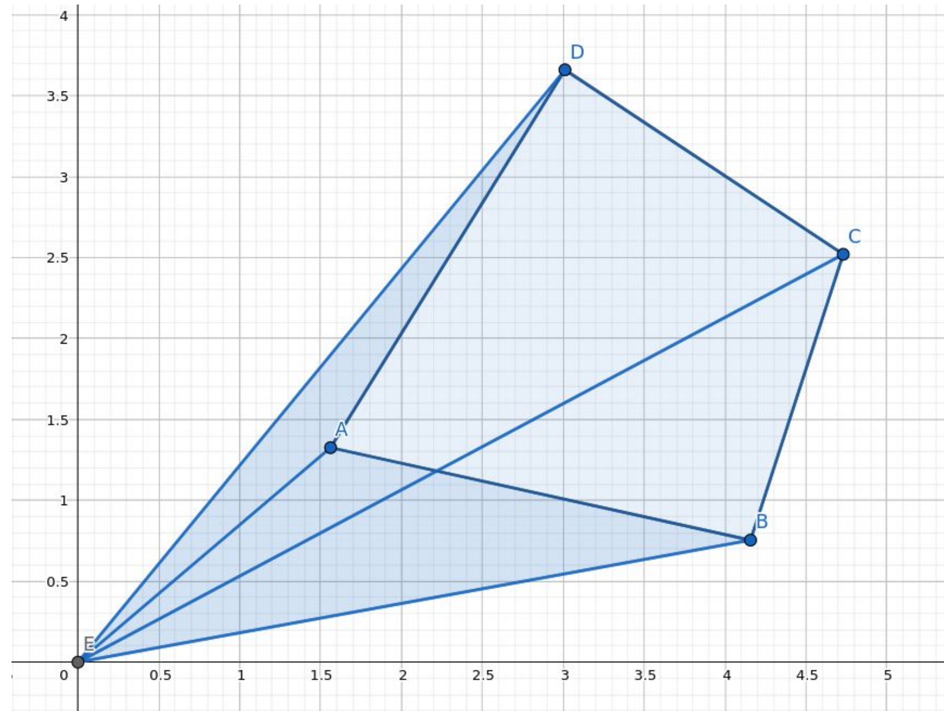
Aplicações

Área de um polígono (shoelace)



Aplicações

Área de um polígono (shoelace)



Aplicações

Área de um polígono (shoelace)

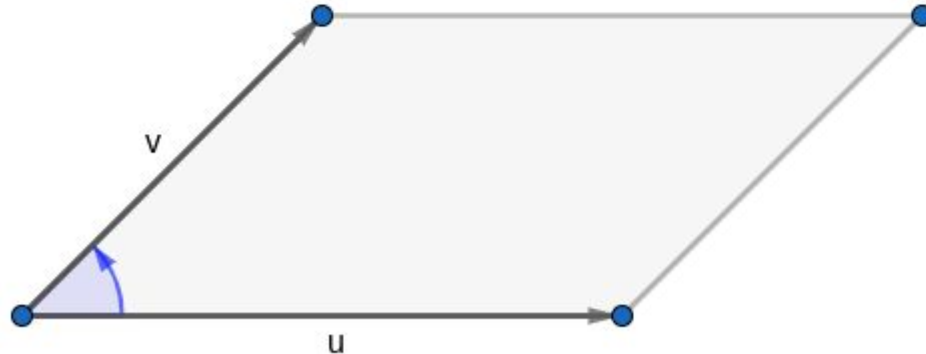
- Como pegar a área dos triângulos?

Aplicações

Área de um polígono (shoelace)

- Como pegar a área dos triângulos?
- Lembrando:

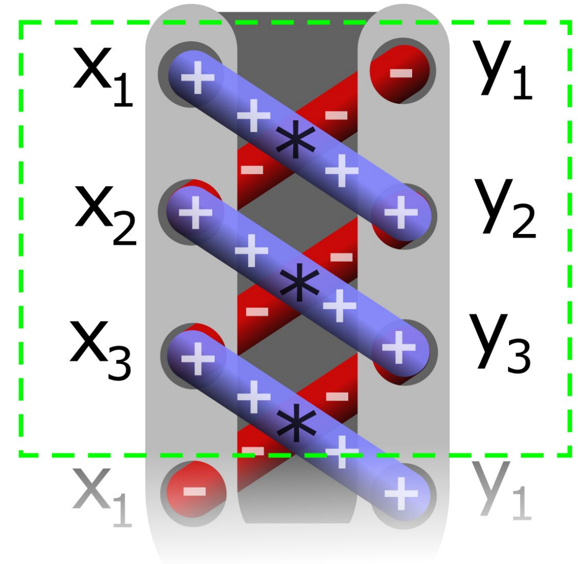
O módulo do produto vetorial nos dá a área do paralelogramo:



Aplicações

Área de um polígono (shoelace)

- Como pegar a área dos triângulos?
  - Produto vetorial (e divide por 2)!
- Por isso chamado shoelace (cadarço)
  - Vai calculando entre as arestas (i, i+1)
- Importante pegar o valor absoluto depois
  - Depende se está em sentido horário ou anti
- Também funciona com polígonos côncavos



Aplicações

Área de um polígono (shoelace)

```
/** returns 2 * area(polygon) */
template<typename T> T shoelace2(vector<Point<T>> const& p) {
  T ans = 0; int n = p.size();
  for (int i = 0, j = n-1; i < n; j = i, i++) ans += p[i].cross(p[j]);
  return abs(ans);
}
```

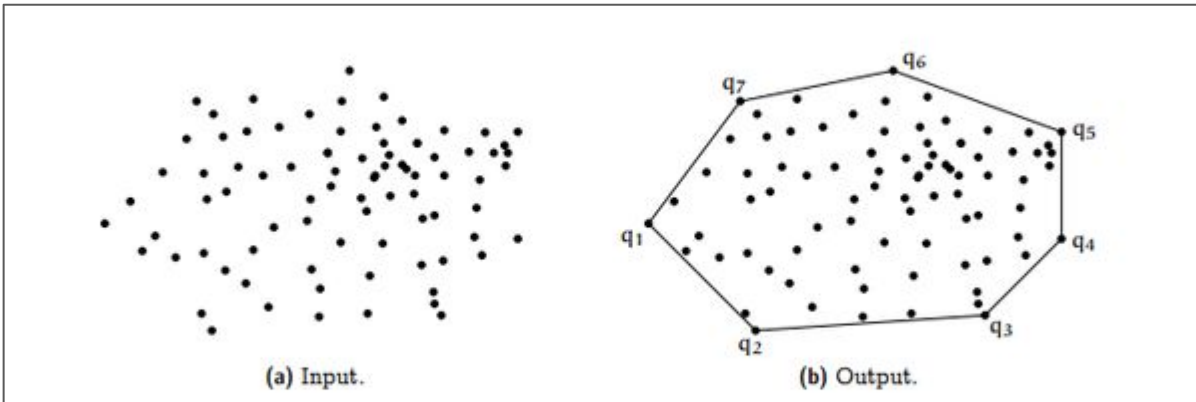
Aplicações

Convex Hull

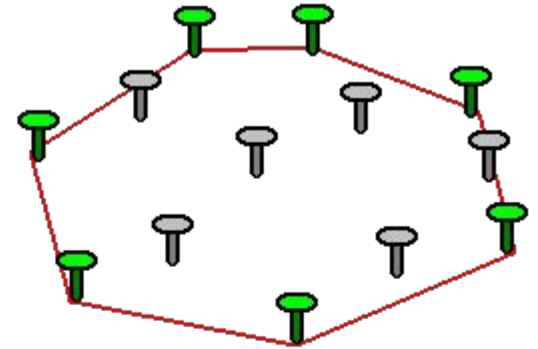
Aplicações

Convex Hull

- Menor polígono convexo que contém todos os pontos



(a) Input. (b) Output.



Aplicações

Convex Hull

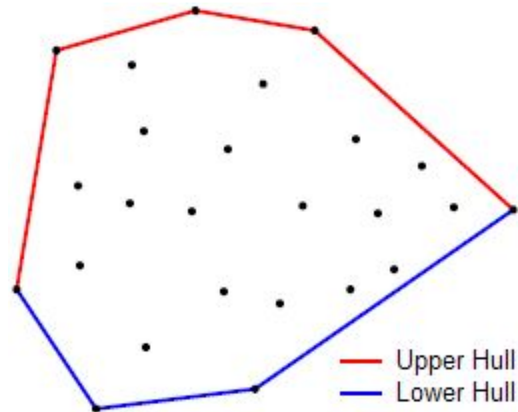
- Algoritmo *Monotone Chain*

Aplicações

Convex Hull

- Algoritmo *Monotone Chain*
- Etapas:

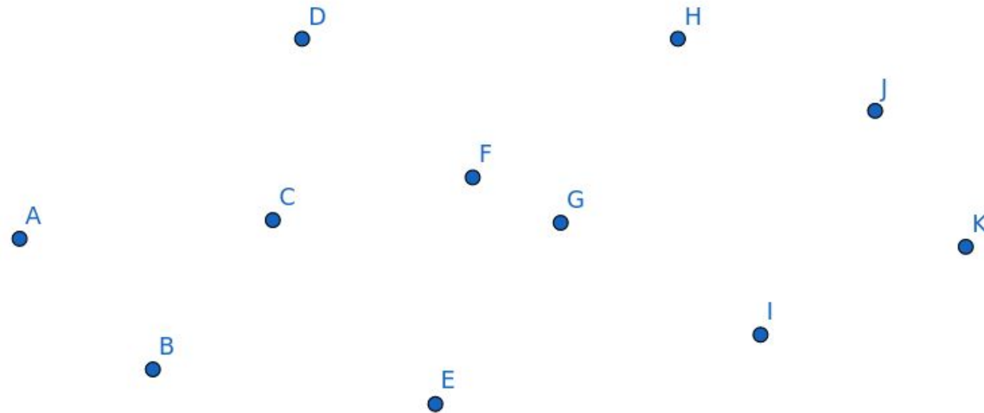
1. Calcular o *lower hull*
2. Calcular o *upper hull*



Aplicações

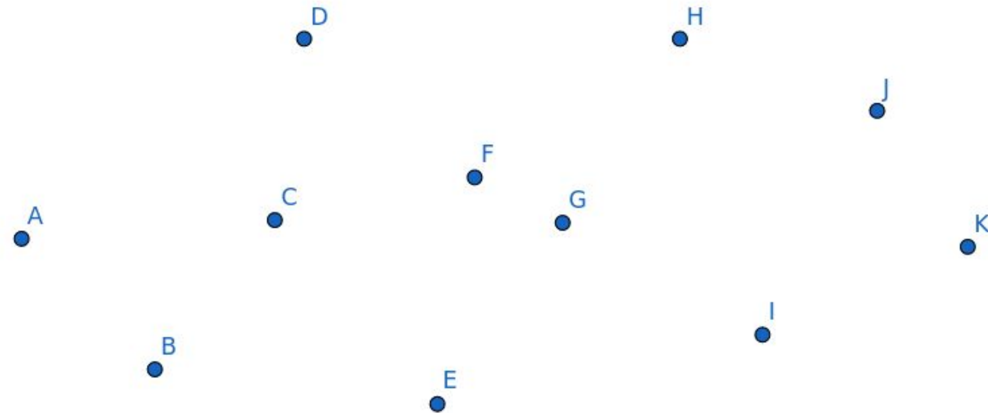
Convex Hull

- Primeiro passo: ordenar os pontos por x e desempatar por y



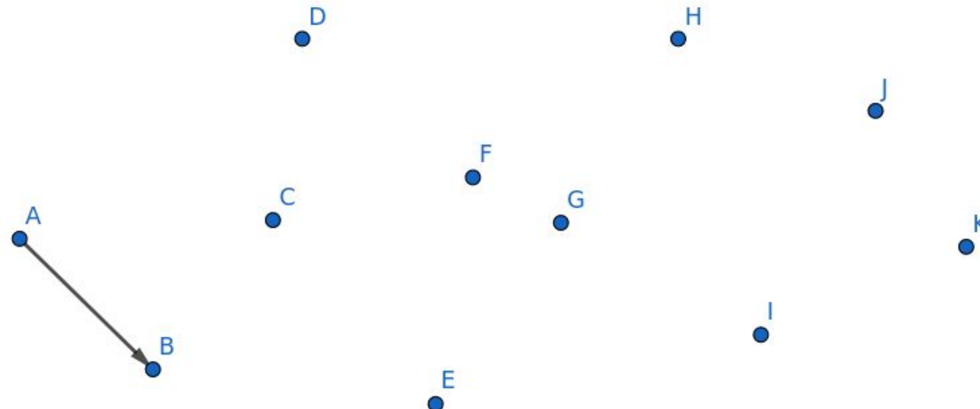
Aplicações

Convex Hull - Lower Hull



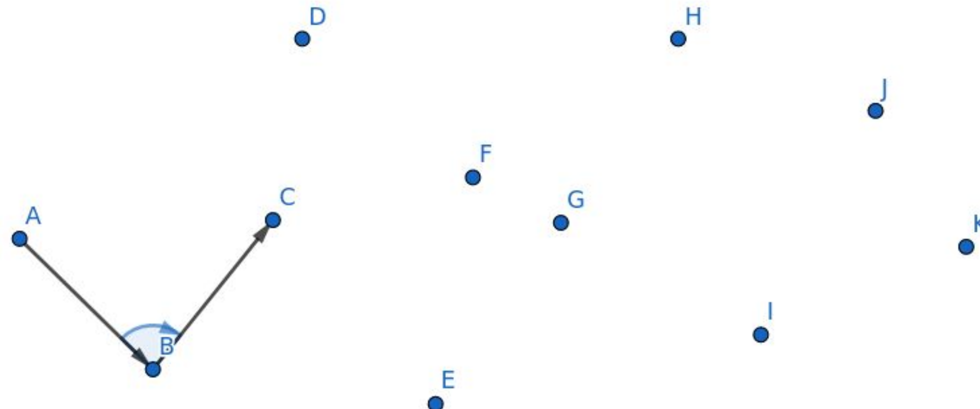
Aplicações

Convex Hull - Lower Hull



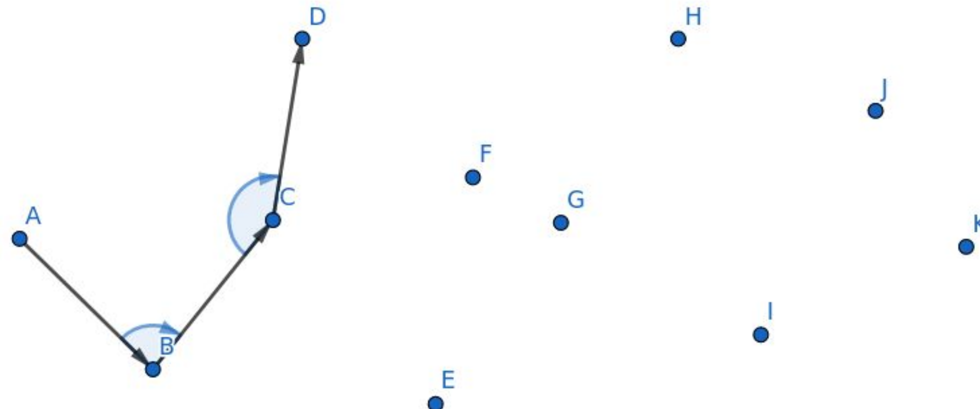
Aplicações

Convex Hull - Lower Hull



Aplicações

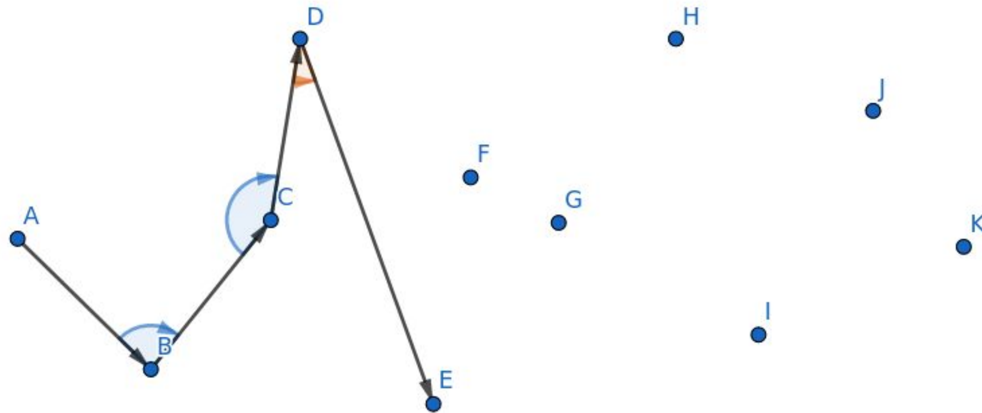
Convex Hull - Lower Hull



Aplicações

Convex Hull - Lower Hull

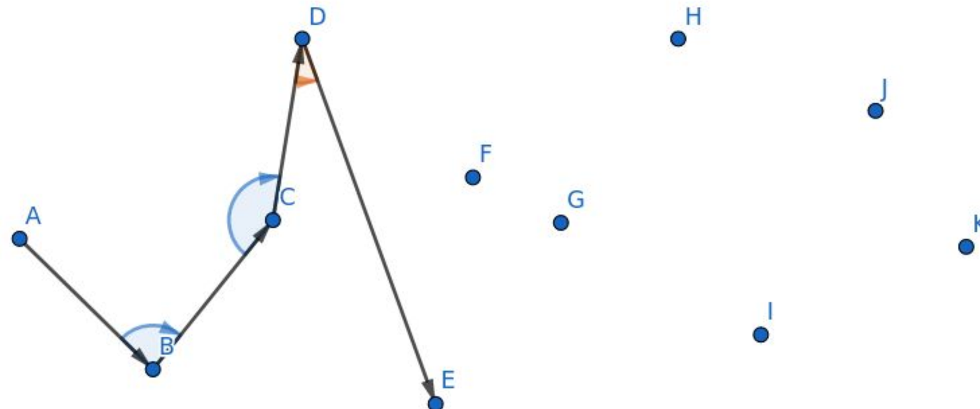
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

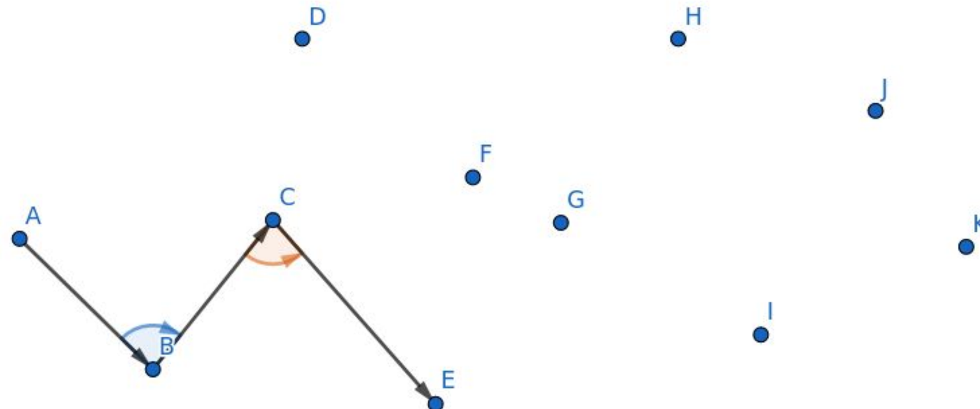
- Adicionar os pontos até encontrar uma ***concavidade***
  - Ou seja, o novo vetor estar **à direita** do vetor anterior



Aplicações

Convex Hull - Lower Hull

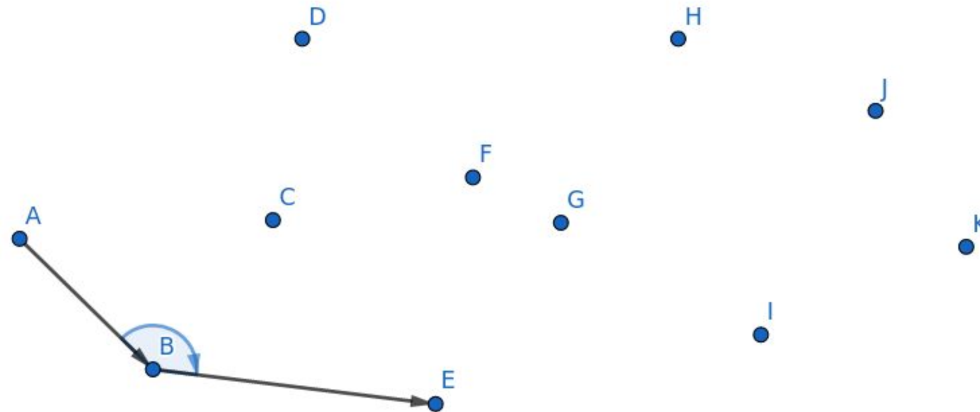
- Adicionar os pontos até encontrar uma ***concavidade***
  - Remover os pontos anteriores até que não haja mais isso



Aplicações

Convex Hull - Lower Hull

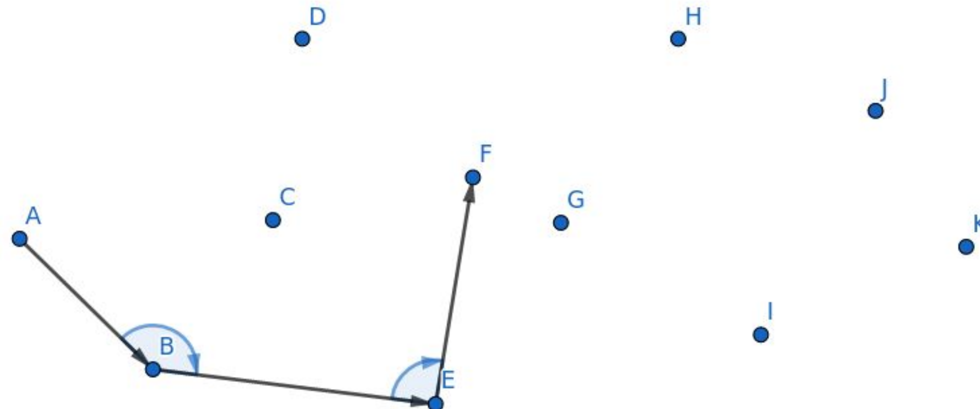
- Adicionar os pontos até encontrar uma ***concavidade***
  - Ideia de uma *pilha* para ir removendo os pontos anteriores!



Aplicações

Convex Hull - Lower Hull

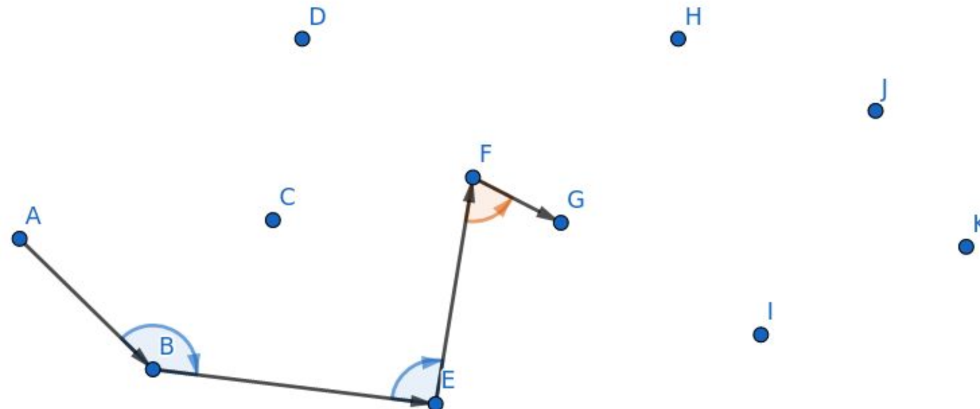
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

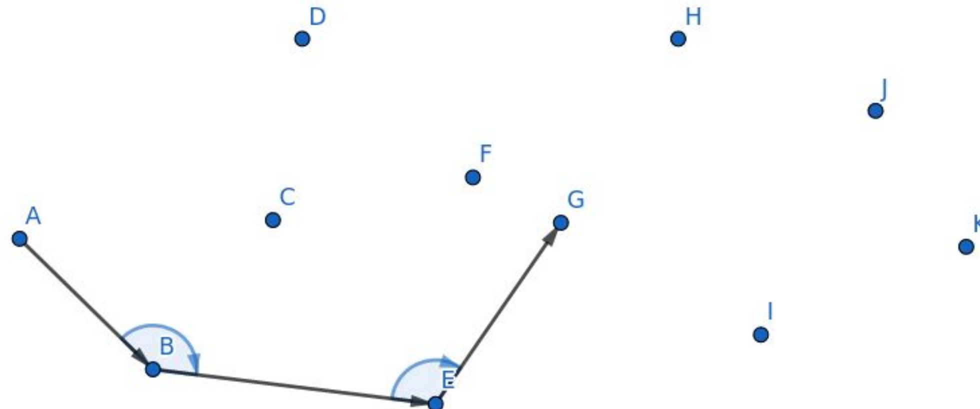
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

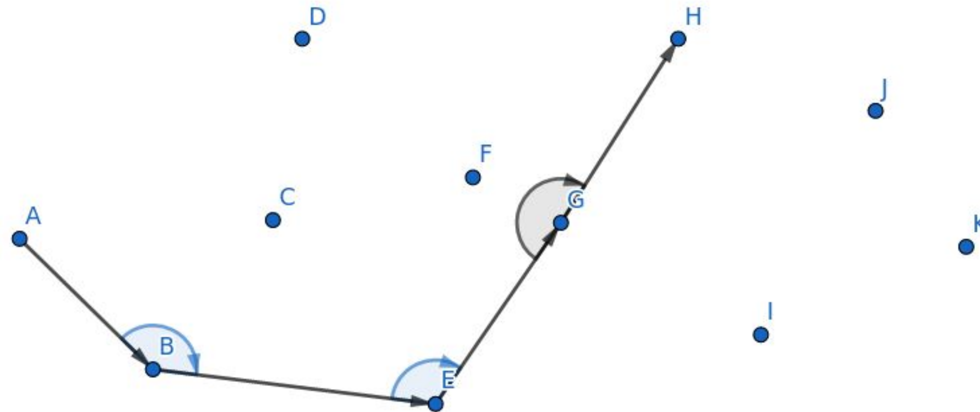
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

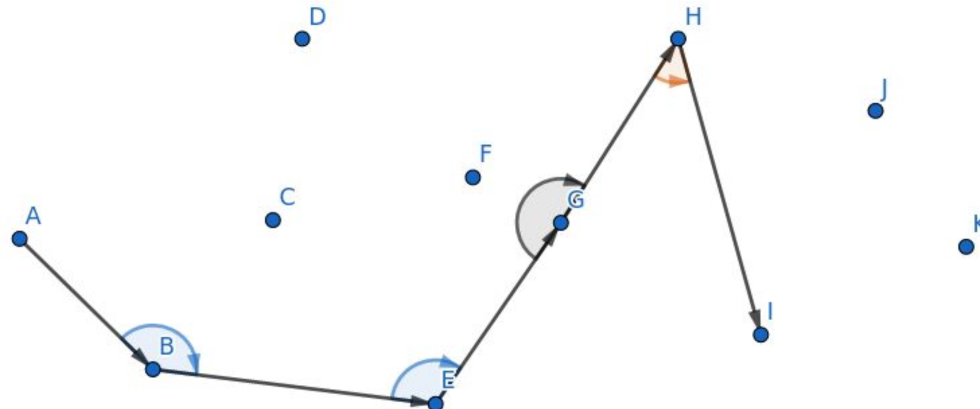
- Adicionar os pontos até encontrar uma ***concavidade***
  - Caso especial: pontos colineares (depende do problema)



Aplicações

Convex Hull - Lower Hull

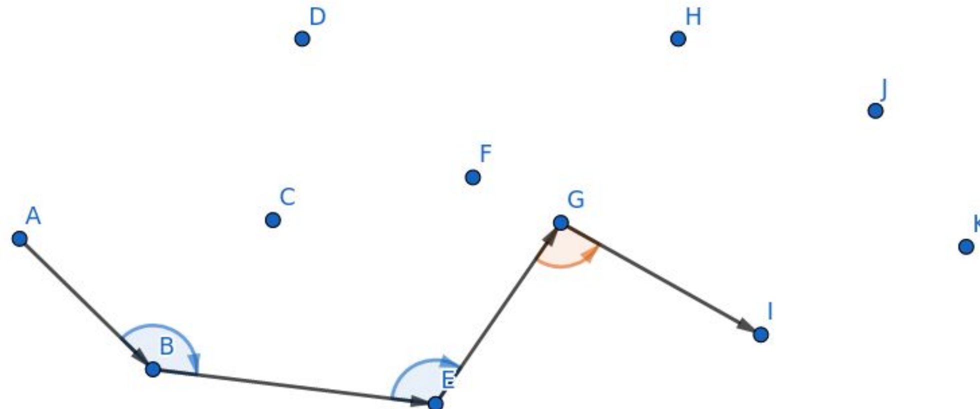
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

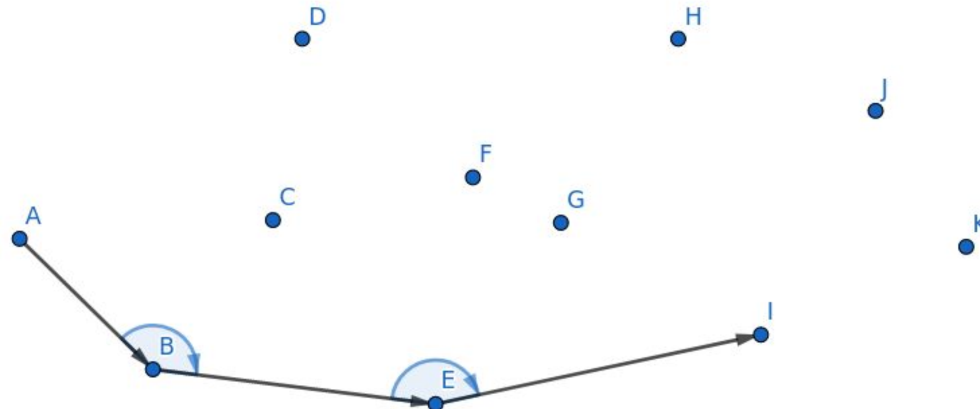
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

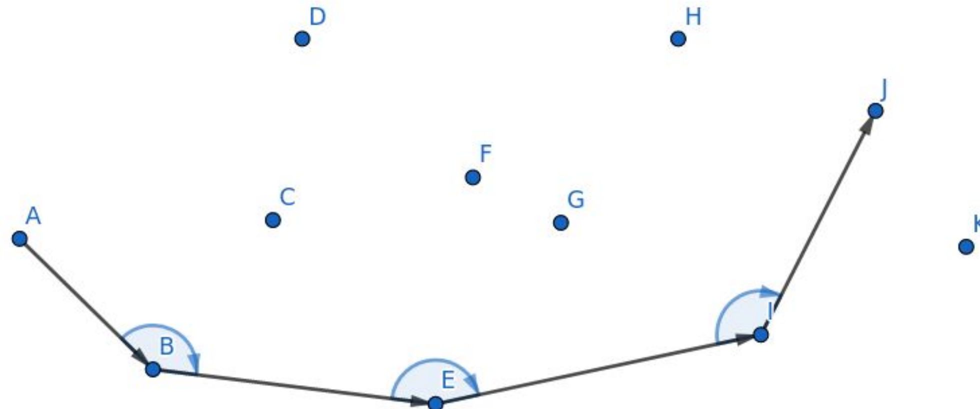
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

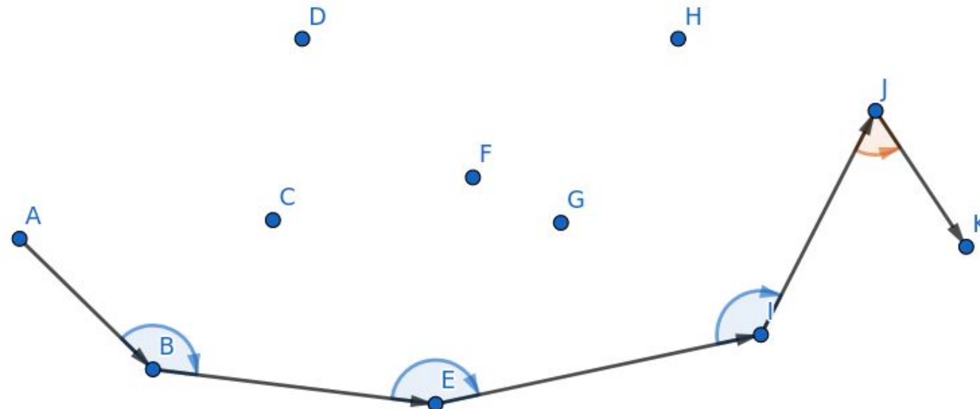
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

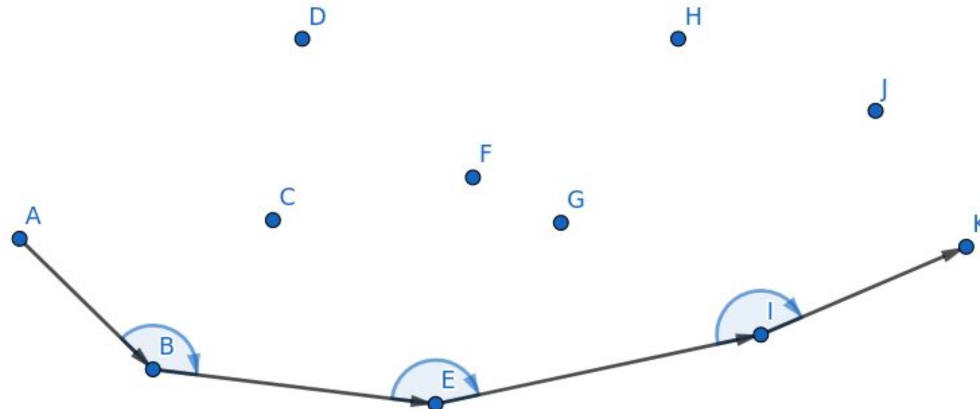
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

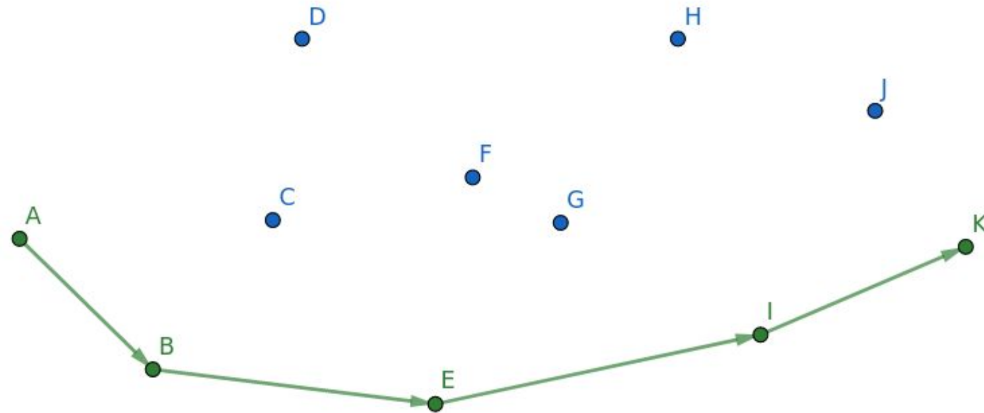
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Lower Hull

- Adicionar os pontos até encontrar uma ***concavidade***
  - Ao chegar no último ponto, o lower hull está pronto!



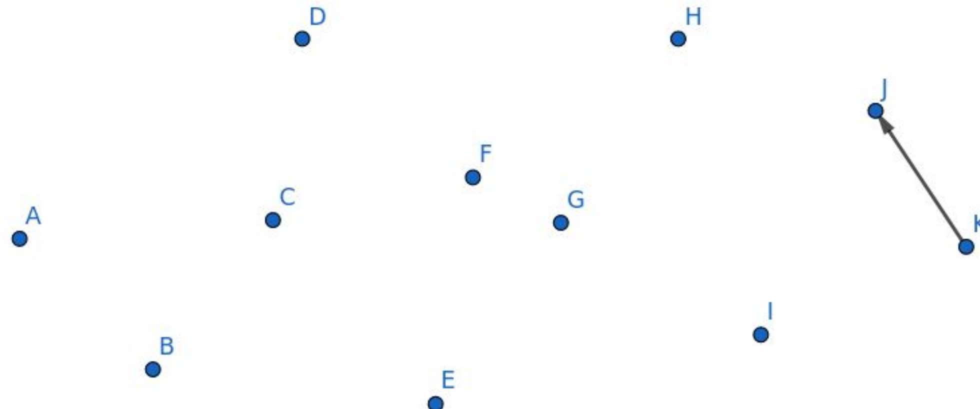
Aplicações

Convex Hull - Upper Hull

Aplicações

Convex Hull - Upper Hull

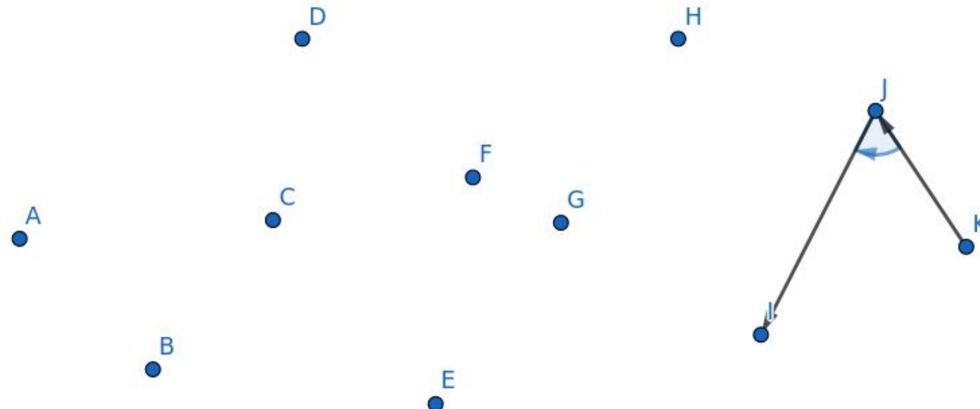
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

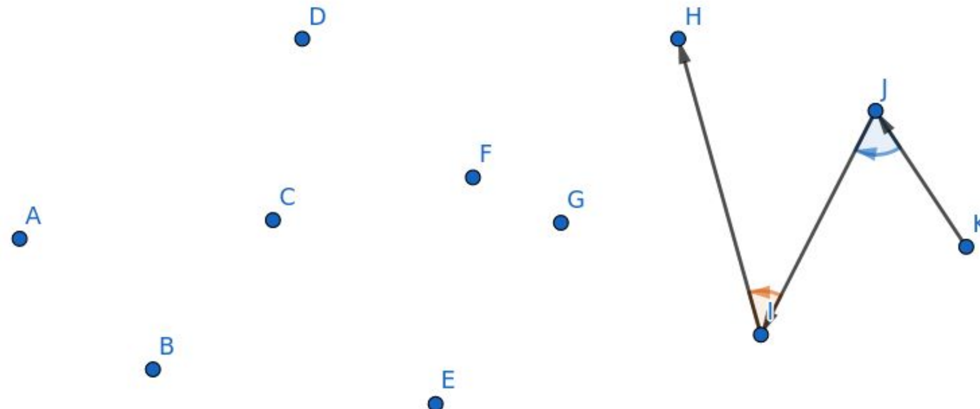
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

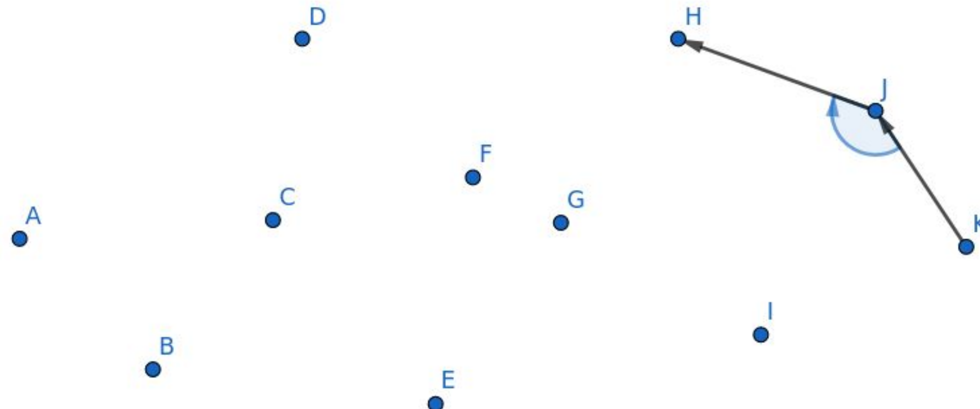
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

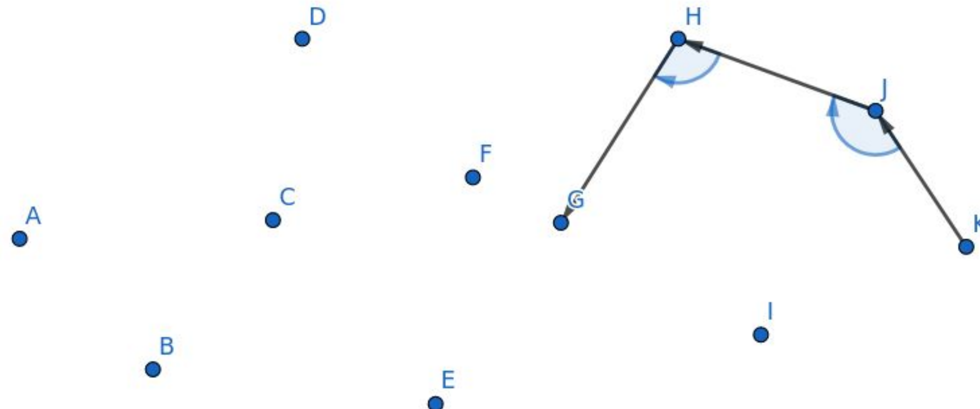
- Adicionar os pontos até encontrar uma ***concavidade***
  - Mesma coisa do lower - vai removendo até não ter mais!



Aplicações

Convex Hull - Upper Hull

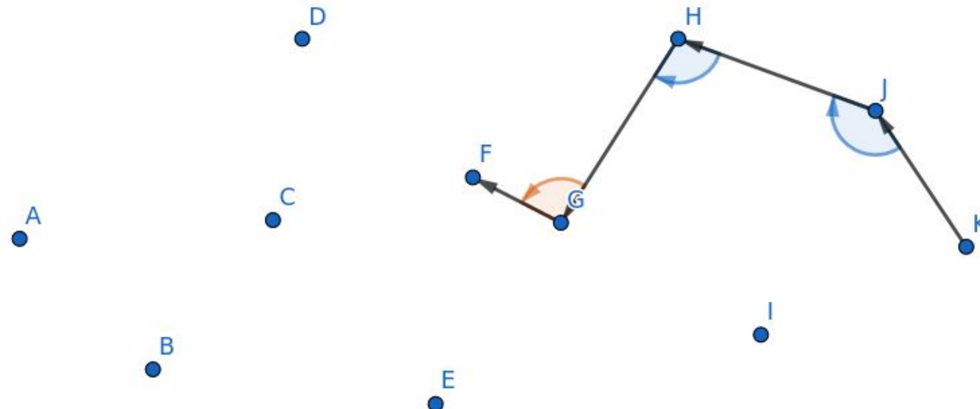
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

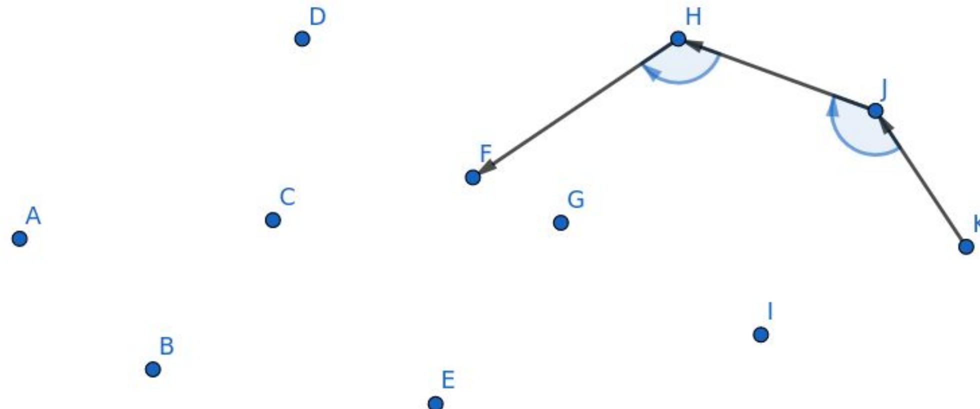
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

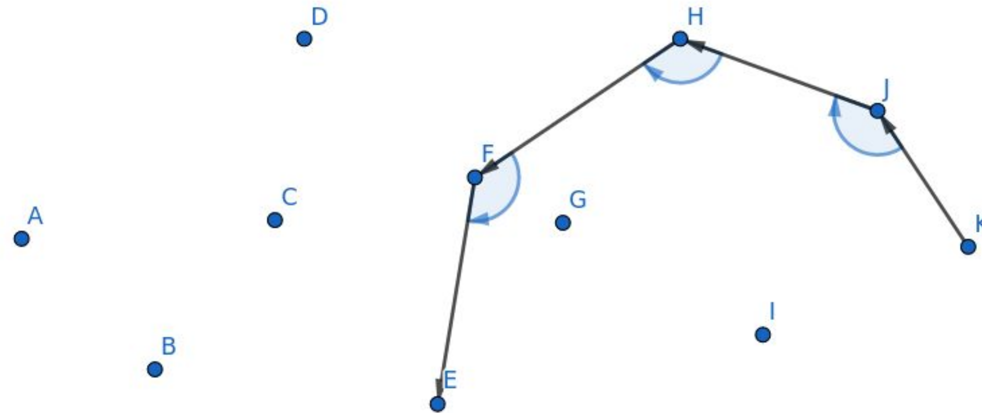
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

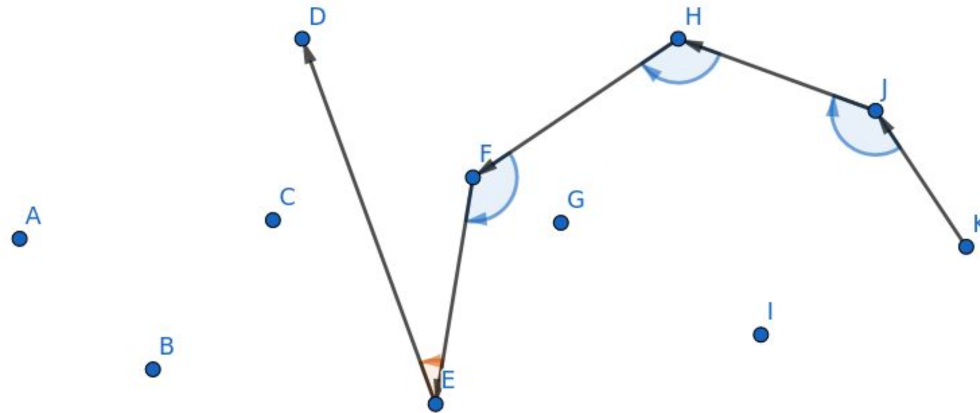
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

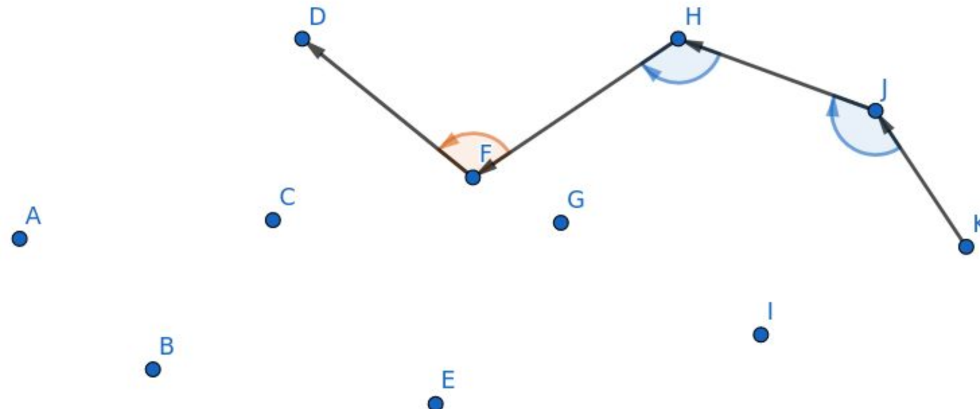
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

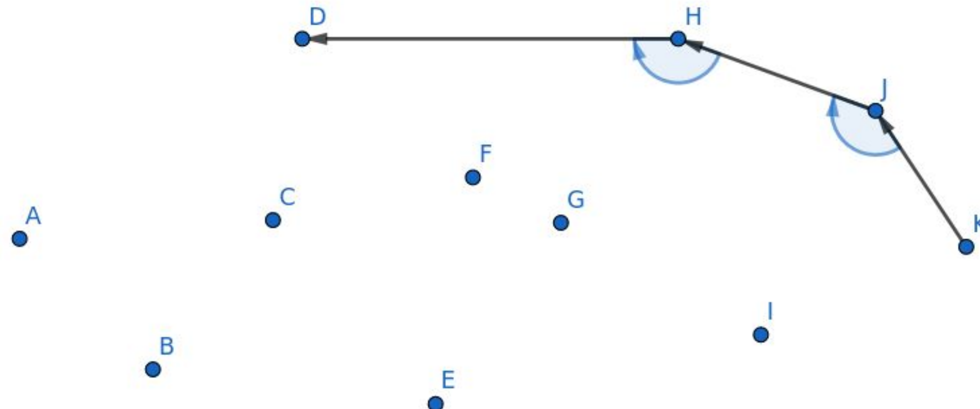
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

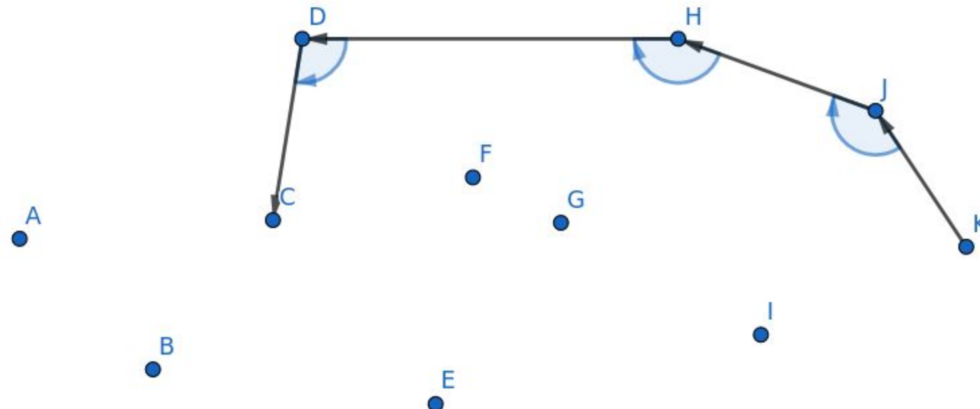
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

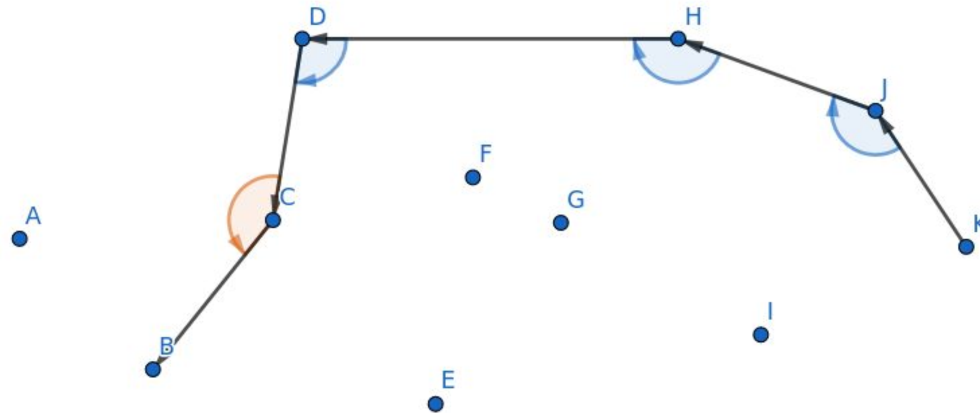
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

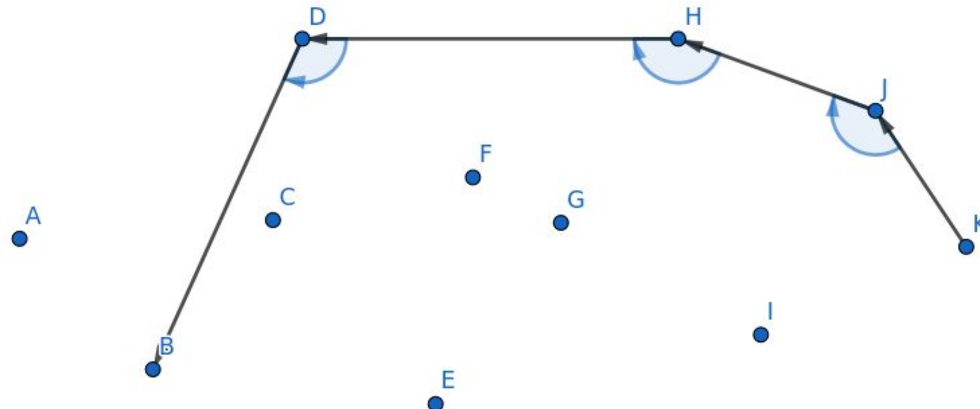
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

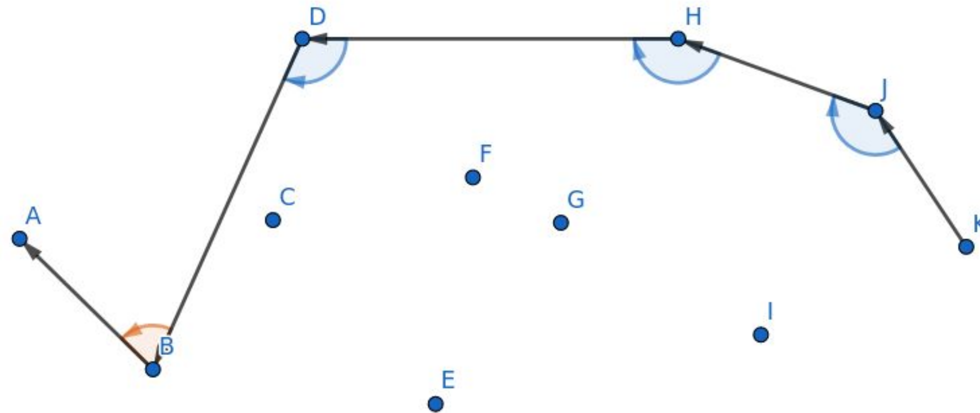
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

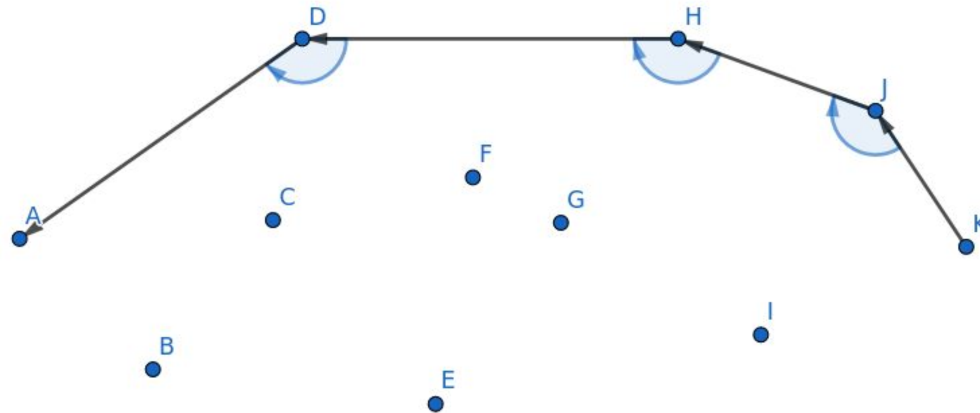
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

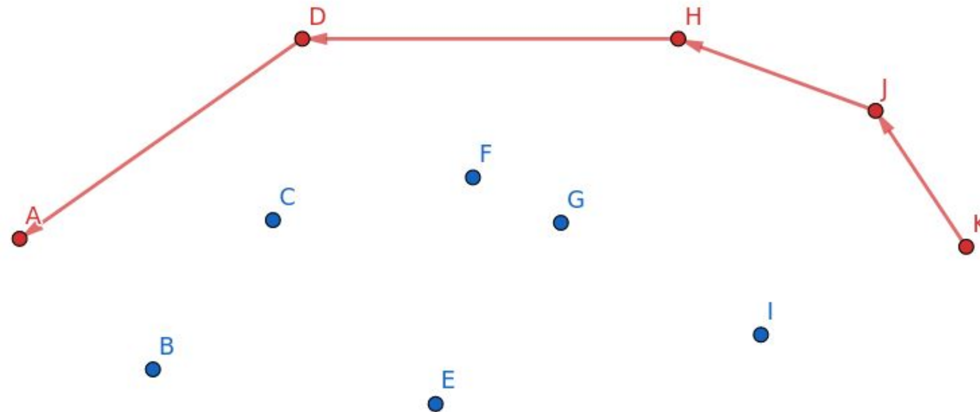
- Adicionar os pontos até encontrar uma ***concavidade***



Aplicações

Convex Hull - Upper Hull

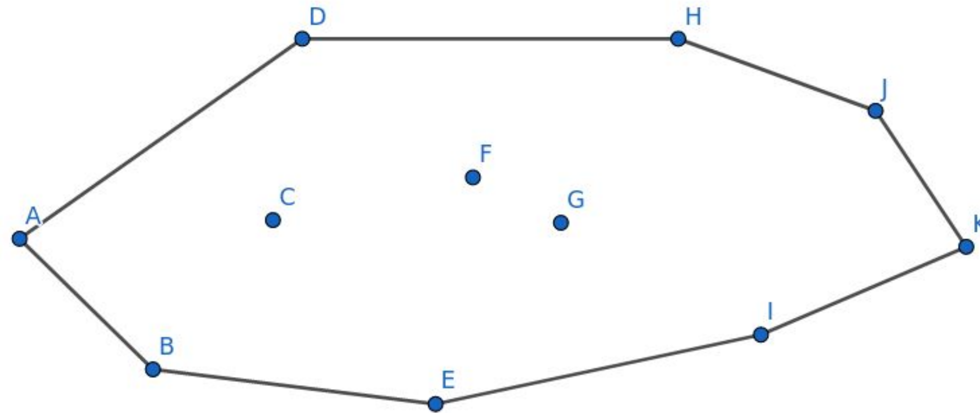
- Adicionar os pontos até encontrar uma ***concavidade***
  - Ao chegar no primeiro ponto, está pronto!



Aplicações

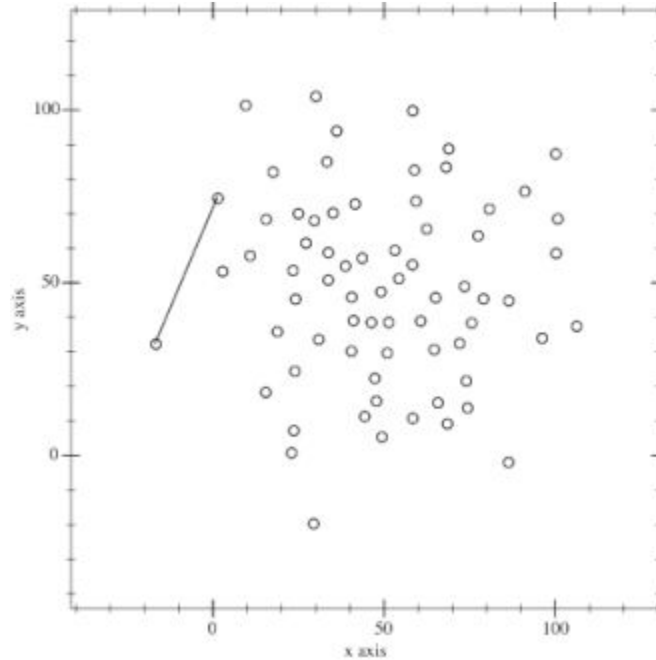
Convex Hull

- Basta juntar os dois para ter o convex hull



Aplicações

Convex Hull



Aplicações

Convex Hull

- Como codificar essa “concavidade”?

Aplicações

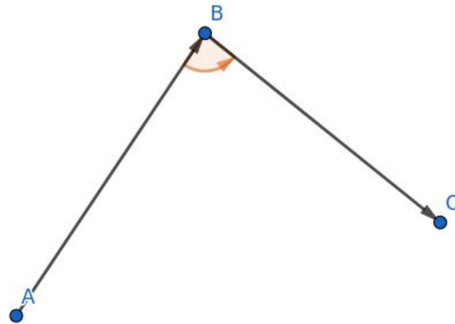
Convex Hull

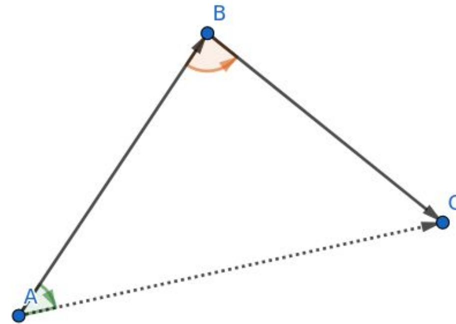
- Como codificar essa “concavidade”?

O novo ponto C estará à direita do vetor AB:

cross(AB, AC) <= 0

(Se aceitar colinear: <)





Aplicações

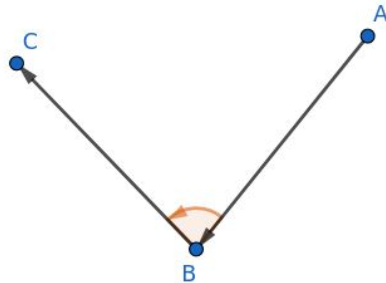
Convex Hull

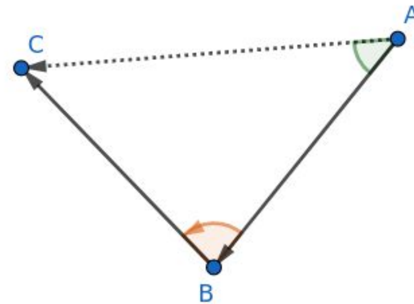
- Como codificar essa “concavidade”?

O novo ponto C estará à direita do vetor AB:

cross(AB, AC) <= 0

(Se aceitar colinear: <)





Aplicações

Convex Hull - Lower Hull

```
sort(a.begin(), a.end());

vector<pt> lw{a[0], a[1]};
for (int i = 2; i < n; i++) {
  while(lw.size() >= 2u) {
     pt A = lw[lw.size() - 2];
     pt B = lw[lw.size() - 1];
     pt C = a[i];

     if ((B-A).cross(C-A) < 0) lw.pop_back();
     else break;
  }

  lw.pb(a[i]);
}
```

Aplicações

Convex Hull - Upper Hull

```
vector<pt> up{a[n-1], a[n-2]};
for (int i = n - 3; i >= 0; i--) {
  while(up.size() >= 2u) {
     pt A = up[up.size() - 2];
     pt B = up[up.size() - 1];
     pt C = a[i];

     if ((B-A).cross(C-A) < 0) up.pop_back();
     else break;
  }

  up.pb(a[i]);
}

vector<pt> chull = lw;
chull.insert(chull.end(), up.begin() +1, up.begin() + up.size() - 1);
```

Problemas

- https://cses.fi/problemset/task/2189
- https://cses.fi/problemset/task/2191
- https://cses.fi/problemset/task/2192
- https://cses.fi/problemset/task/2195
- https://codeforces.com/problemsets/acmsguru/problem/99999/253
- https://www.spoj.com/problems/INOROUT/
- https://vjudge.net/contest/345084
- https://vjudge.net/contest/66208

Referências

- https://cp-algorithms.com/geometry/basic-geometry.html
- https://cp-algorithms.com/geometry/convex-hull.html
- https://github.com/icmcgema/gema/blob/master/13-Geometria_Computacional.ipynb
- https://github.com/icmcgema/icpc-notebook/tree/main/Geometry
- BOULOS, Paulo; CAMARGO, Ivan. Geometria analítica - um tratamento vetorial.

Agradecimentos especiais ao Gabriel Camargo (grande Artista), peguei muita coisa dos slides que ele preparou antes :)

Obrigado!