

# Exercícios de Geometria

---

Exercícios do Contest de 22 a 26/06

<https://vjudge.net/contest/379355>

# A - Polygons

- **Objetivo:** dados um polígono convexo  $A$  e um polígono côncavo  $B$  (ambos não degenerados), determinar se o polígono  $B$  está estritamente dentro de  $A$ .

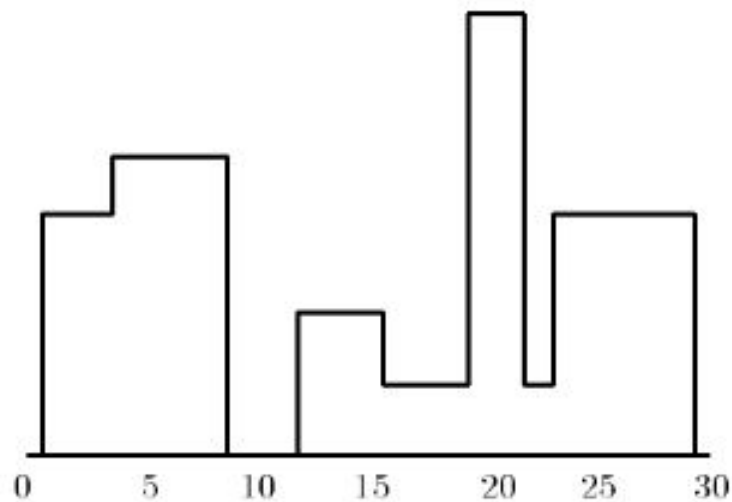
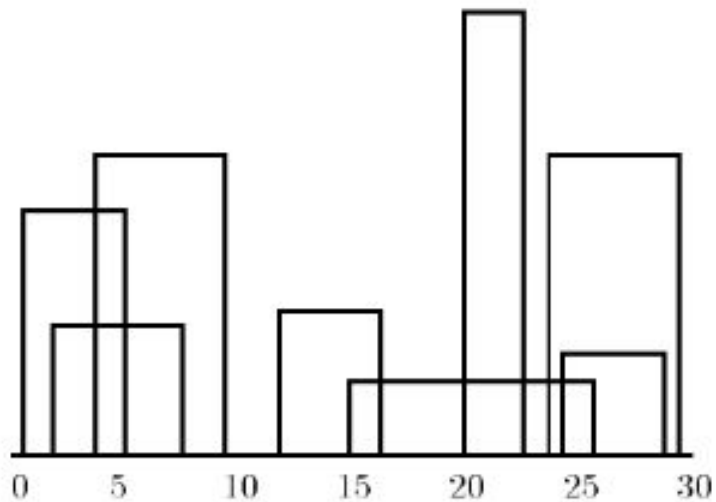
# A - Polygons

- **Objetivo:** dados um polígono convexo A e um polígono côncavo B (ambos não degenerados), determinar se o polígono B está estritamente dentro de A.
- **Possíveis soluções:**
  - Fecho convexo com todos os pontos (se tiver algum ponto de B no fecho, então não ele não está estritamente dentro de A). Pode ter problemas com casos degenerados, quando temos vértices de B exatamente nas arestas de A
  - Triangulação com busca binária pelos ângulos

# B - The Skyline Problem

- **Objetivo:** Determinar o contorno formado por um conjunto de retângulos.

$(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)$



$(1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)$

## B - The Skyline Problem

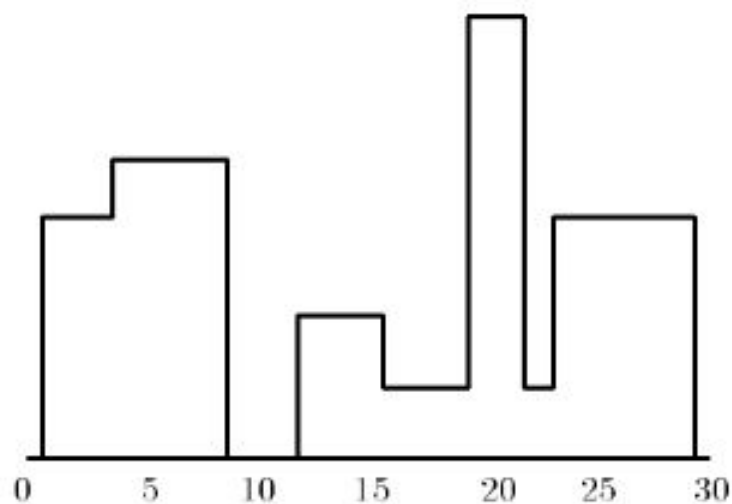
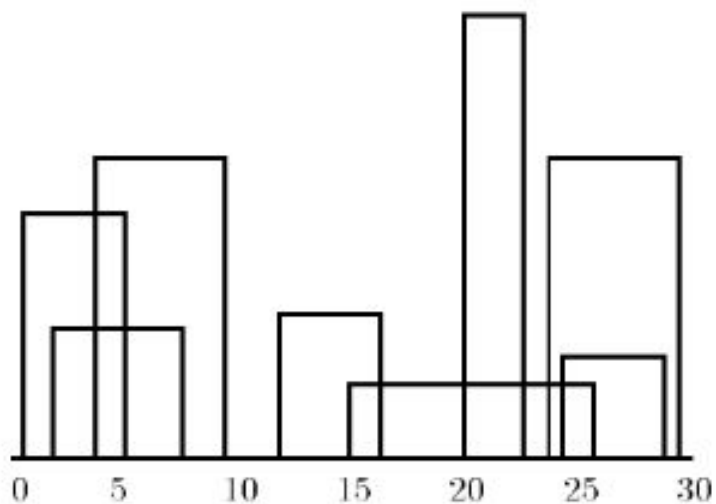
- **Solução 1:** Por força bruta, aproveitando que temos um intervalo “pequeno” de coordenadas (0 - 10000) e poucas construções (5000). Complexidade  $O(n^2)$ 
  - 1º) Instanciar um vetor  $v[]$ , com 10000 posições, que irá armazenar a maior altura em cada posição  $v[x]$
  - 2º) Para cada retângulo  $(l, h, r)$ , vamos percorrer todas as posições de  $v[l]$  a  $v[r]$ , atualizando as posições do vetor  $v$  se for necessário ( $h > v[x]$ )
  - 3º) Percorrer todo o vetor, printando os pontos de “mudança”

## B - The Skyline Problem

- **Solução 2:** Utilizando line sweep. Complexidade  $O(n \log(n))$ 
  - Ideia: vamos fazer uma varredura considerando os pontos de início e fim dos retângulos (ordenados previamente), armazenando as alturas dos retângulos que estamos passando. Sempre que a maior altura armazenada mudar, printamos a alteração.

# B - The Skyline Problem

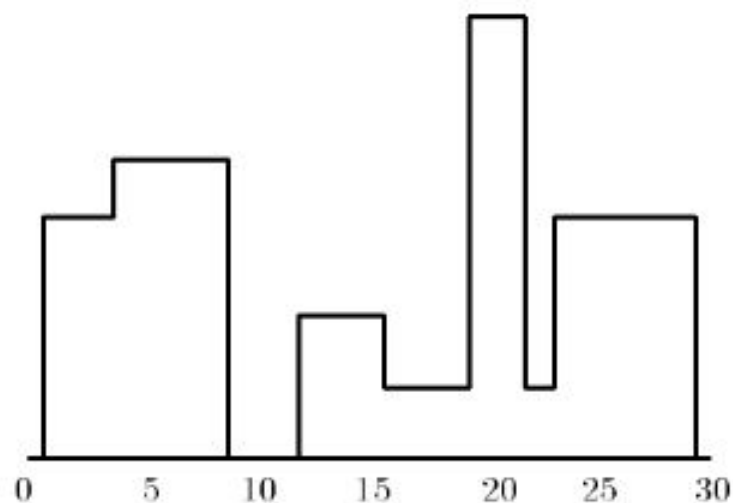
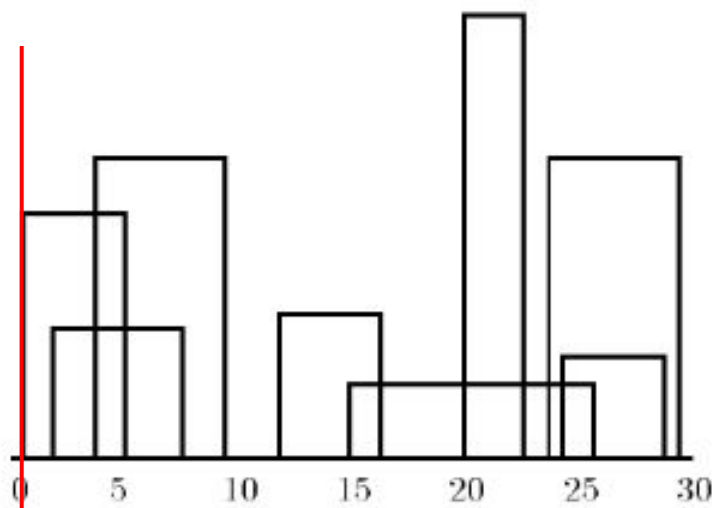
- Multiset:  $\{0\}$
- Resposta:



$(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)$

# B - The Skyline Problem

- Multiset: {11, 0}
- Resposta: 1 11

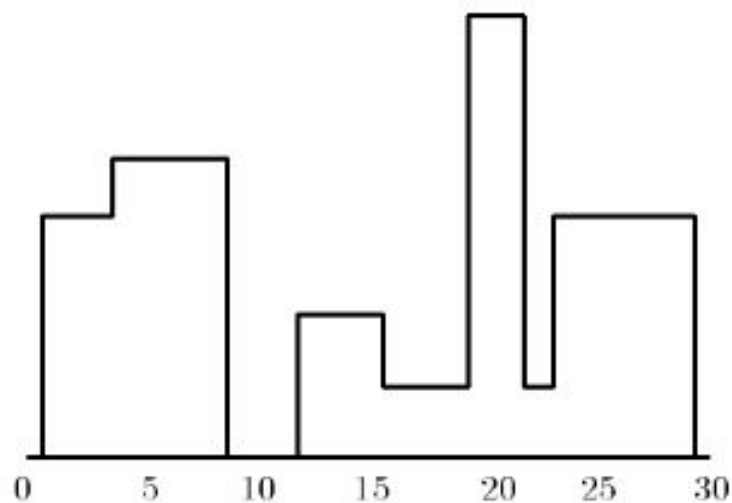
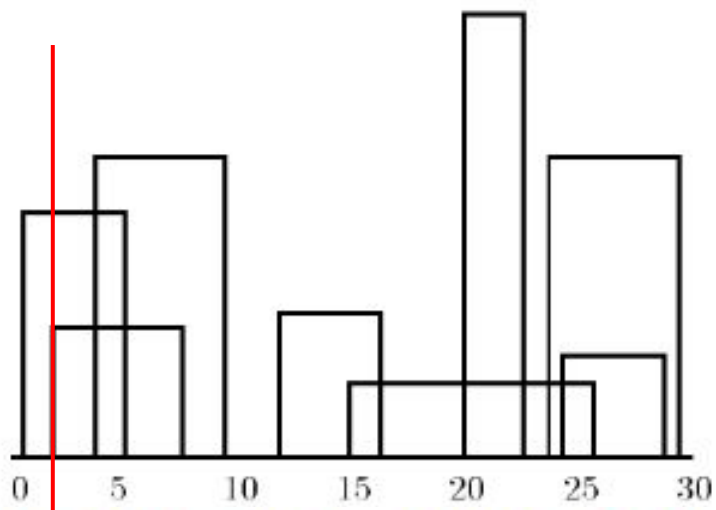


(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)



# B - The Skyline Problem

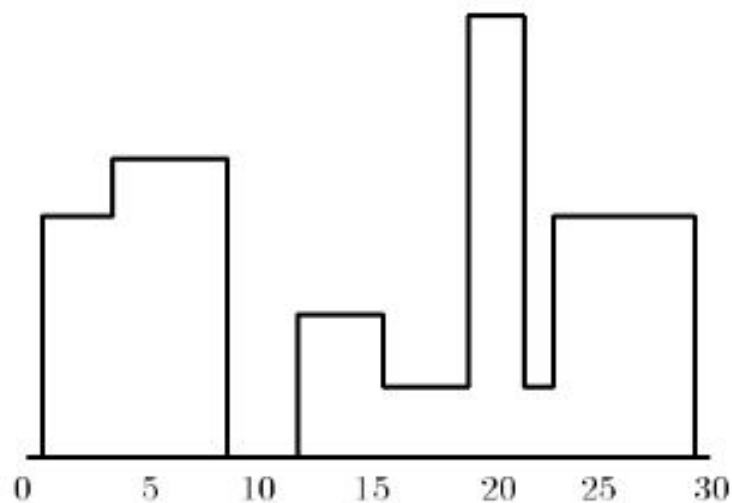
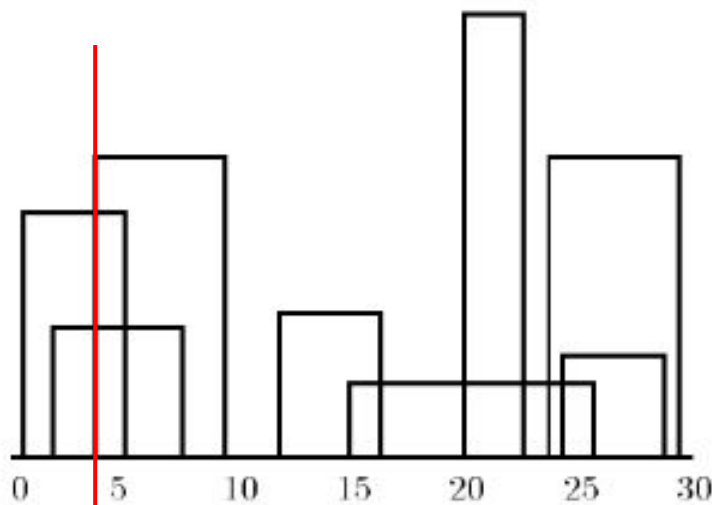
- Multiset: {11, 6, 0}
- Resposta: 1 11



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

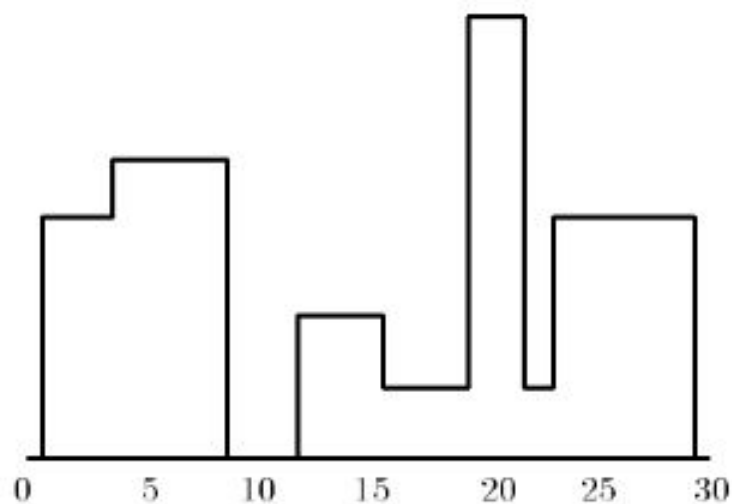
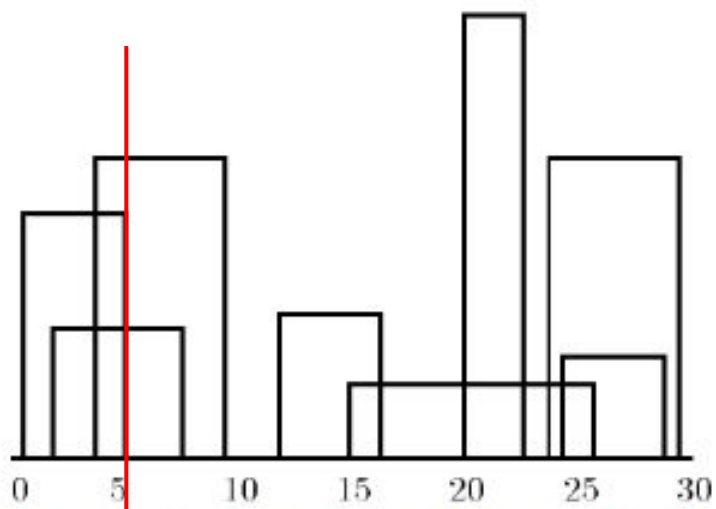
- Multiset: {13, 11, 6, 0}
- Resposta: 1 11 3 13



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

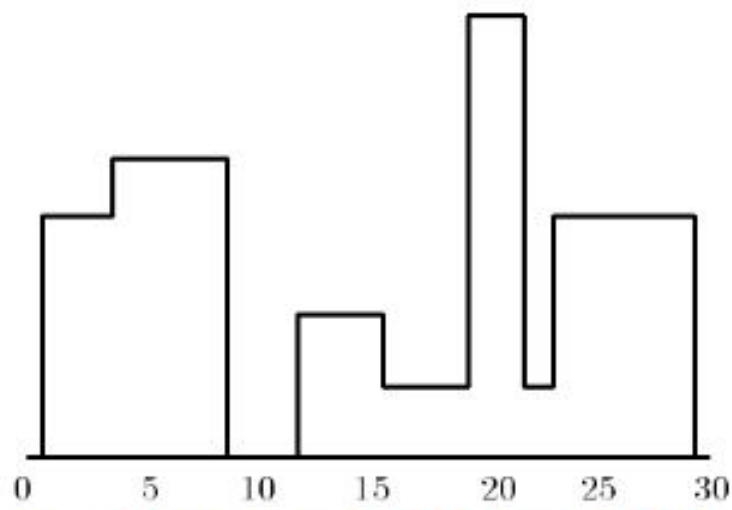
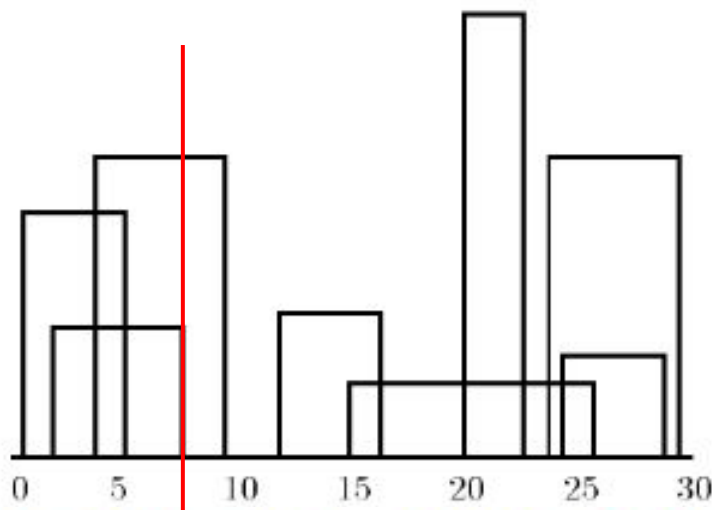
- Multiset: {13, **11**, 6, 0}
- Resposta: 1 11 3 13



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

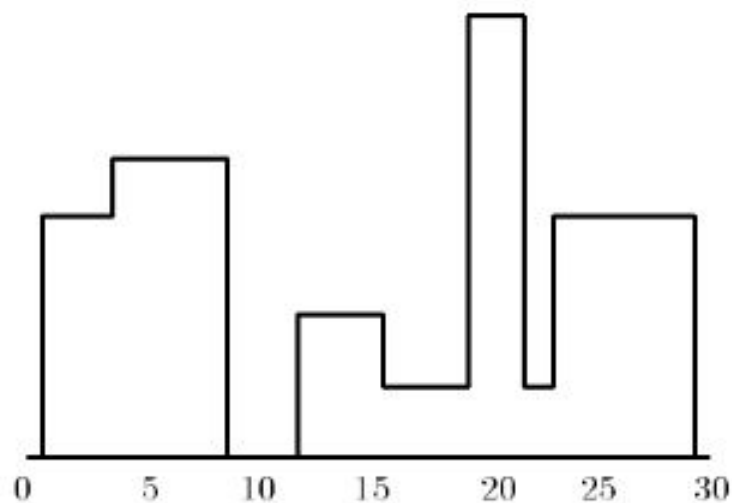
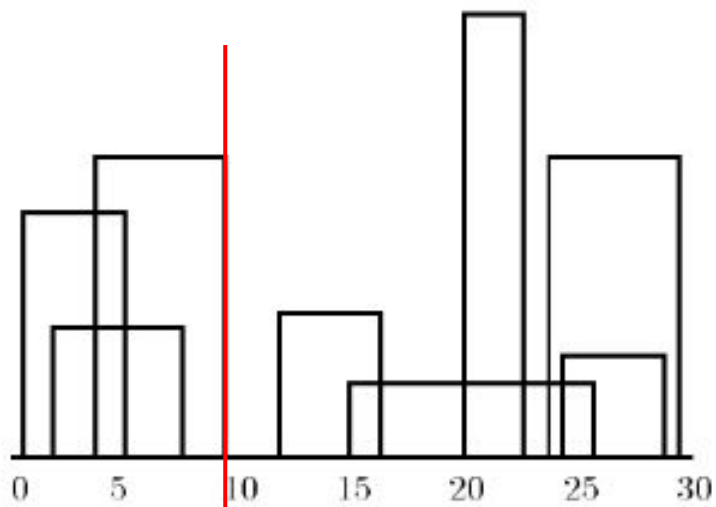
- Multiset: {13, **6**, 0}
- Resposta: 1 11 3 13



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

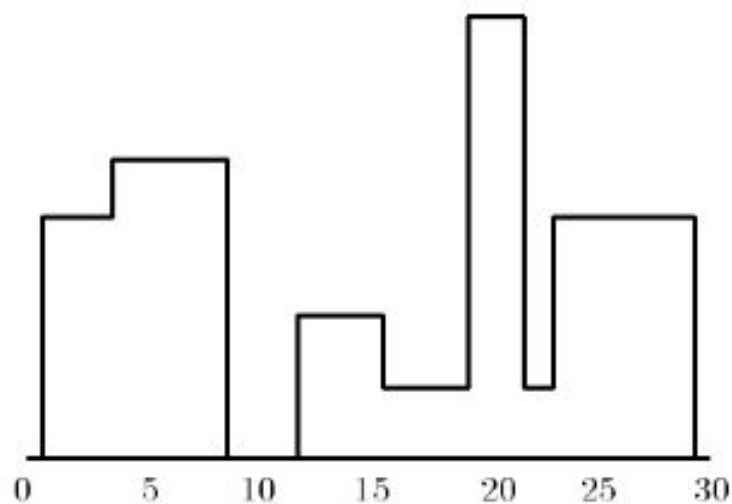
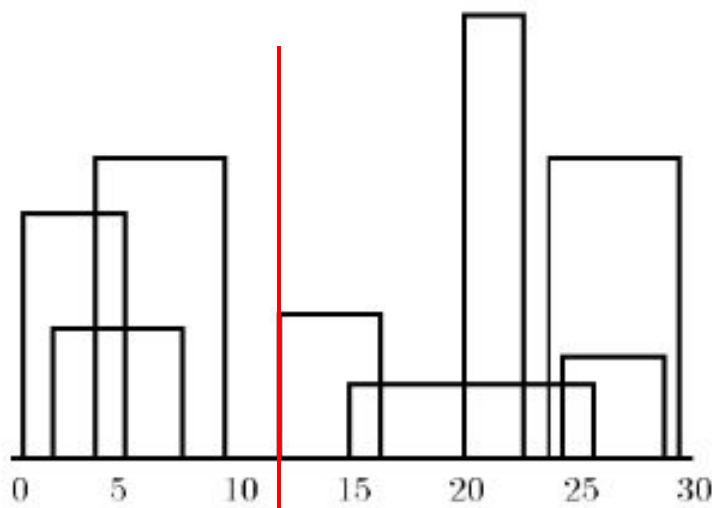
- Multiset: {**13**, 0}
- Resposta: 1 11 3 13 **9 0**



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

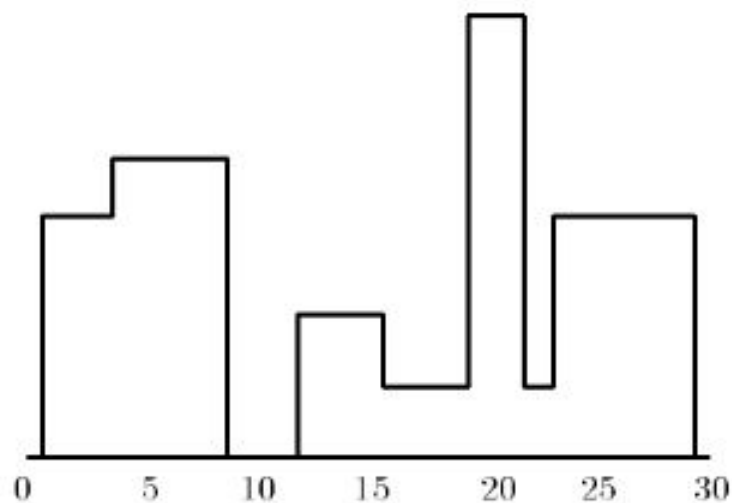
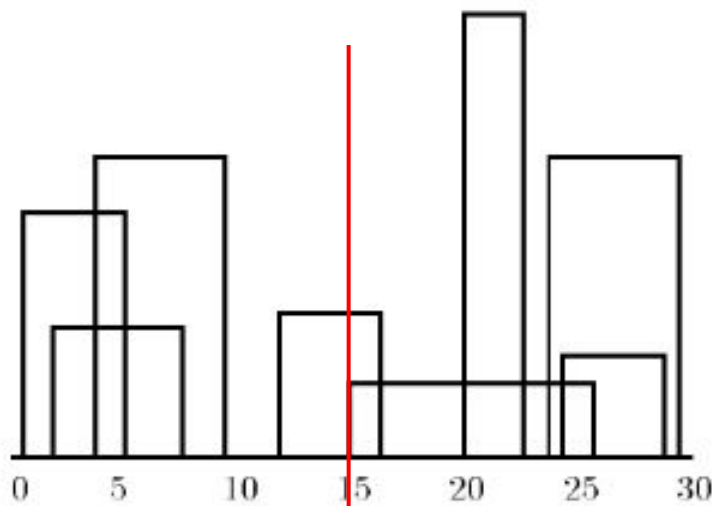
- Multiset: {7, 0}
- Resposta: 1 11 3 13 9 0 **12 7**



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

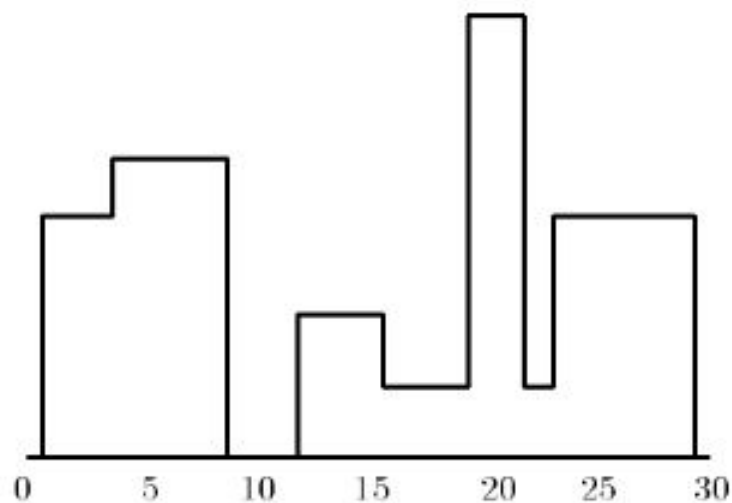
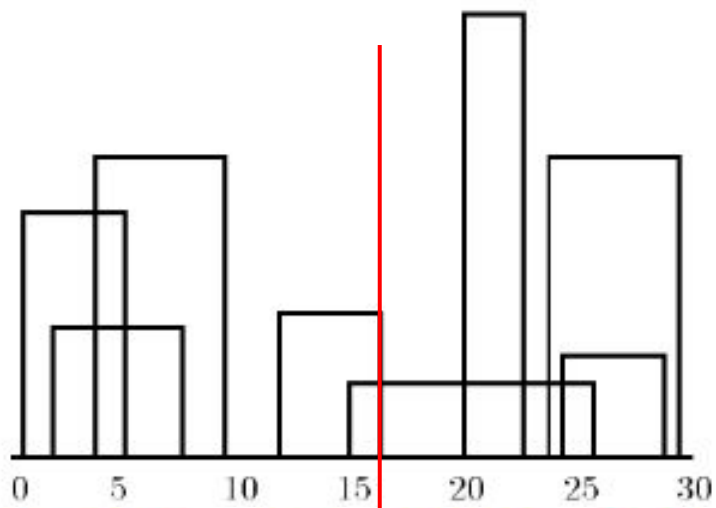
- Multiset: {7, 3, 0}
- Resposta: 1 11 3 13 9 0 12 7



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

# B - The Skyline Problem

- Multiset: {**7**, 3, 0}
- Resposta: 1 11 3 13 9 0 12 7 **16 3**



(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)

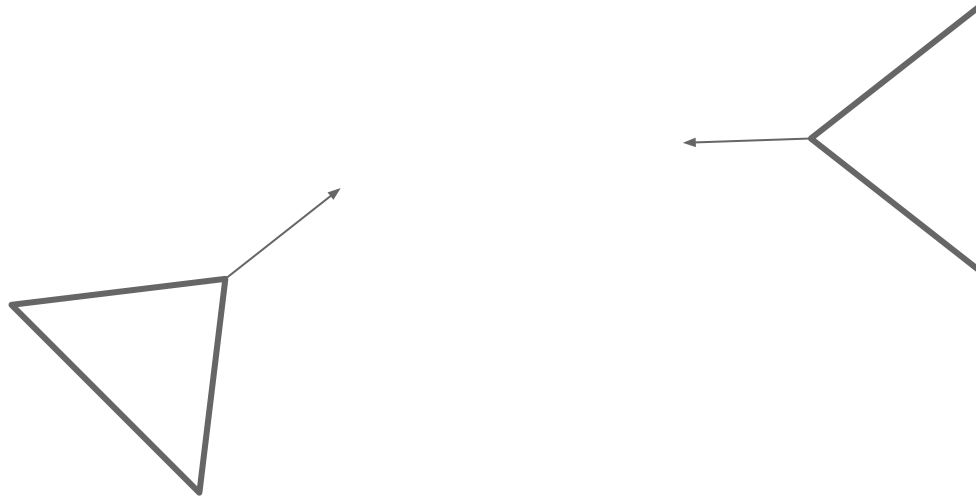


## C - Cover Points

- **Objetivo:** dado um conjunto de pontos  $(x,y)$ , determinar o menor triângulo que contenha todos esses pontos (o ponto pode estar na aresta). Com as restrições adicionais que este triângulo é isósceles e dois dos seus lados estão nos eixos coordenados.

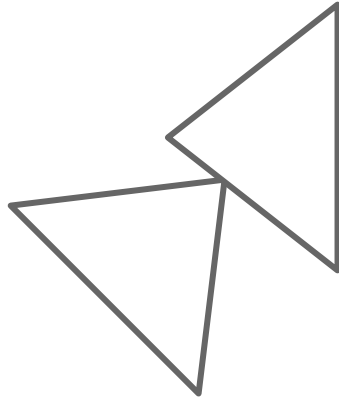
## D - Geometry Problem

- **Objetivo:** dado dois projéteis (representados por triângulos) determinar se eles se intersectam e em qual momento. Se a intersecção ocorrer mais de uma vez, a resposta é o primeiro momento de encontro.



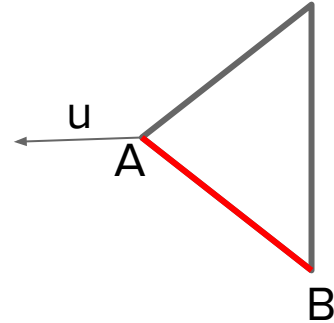
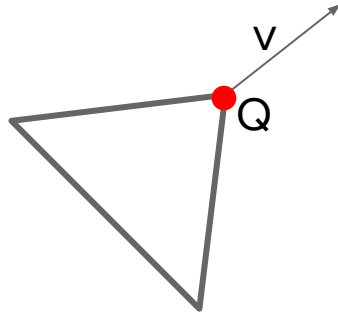
## D - Geometry Problem

- **Observação:** Analisando o problema, podemos perceber que quando houver o encontro entre os triângulos, ele ocorrerá entre UM ponto de um triângulo com UMA aresta do outro.



## D - Geometry Problem

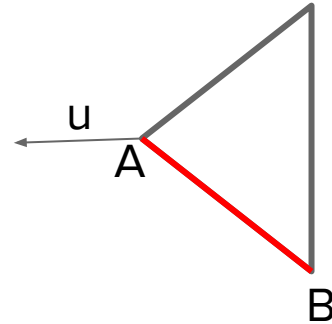
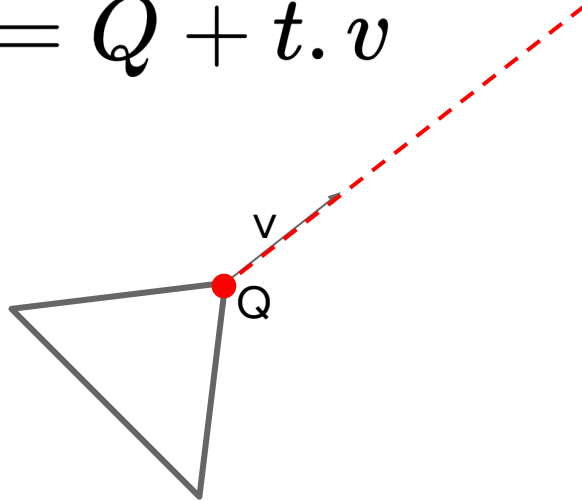
- Isso simplifica um pouco o problema, agora temos que lidar com um ponto e um segmento por vez.



## D - Geometry Problem

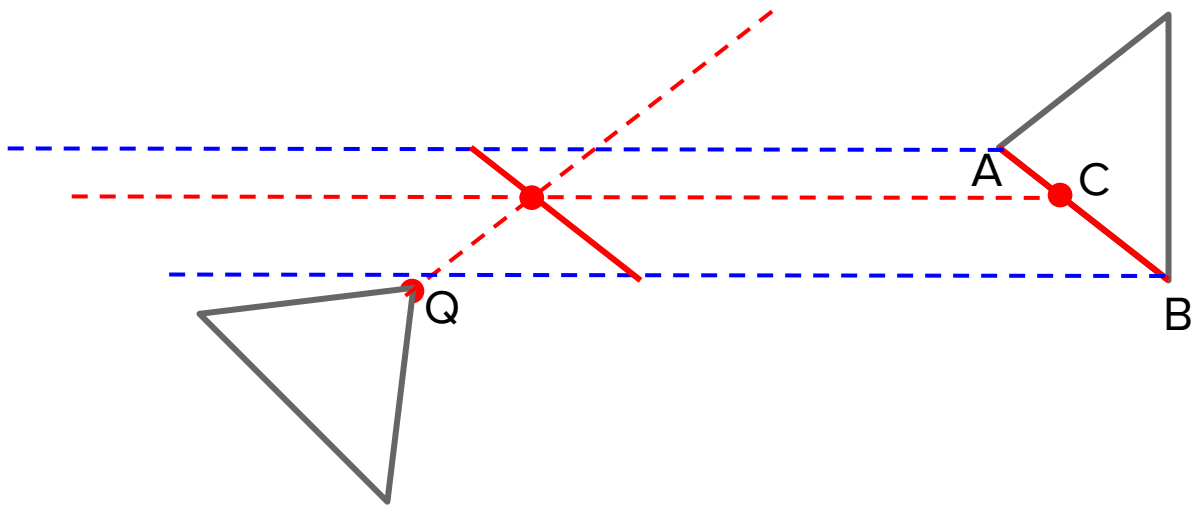
- A partir do ponto Q e do vetor v, podemos determinar uma equação vetorial da reta, onde t pode ser interpretado justamente como o tempo (considerando o problema:

$$P = Q + t.v$$



## D - Geometry Problem

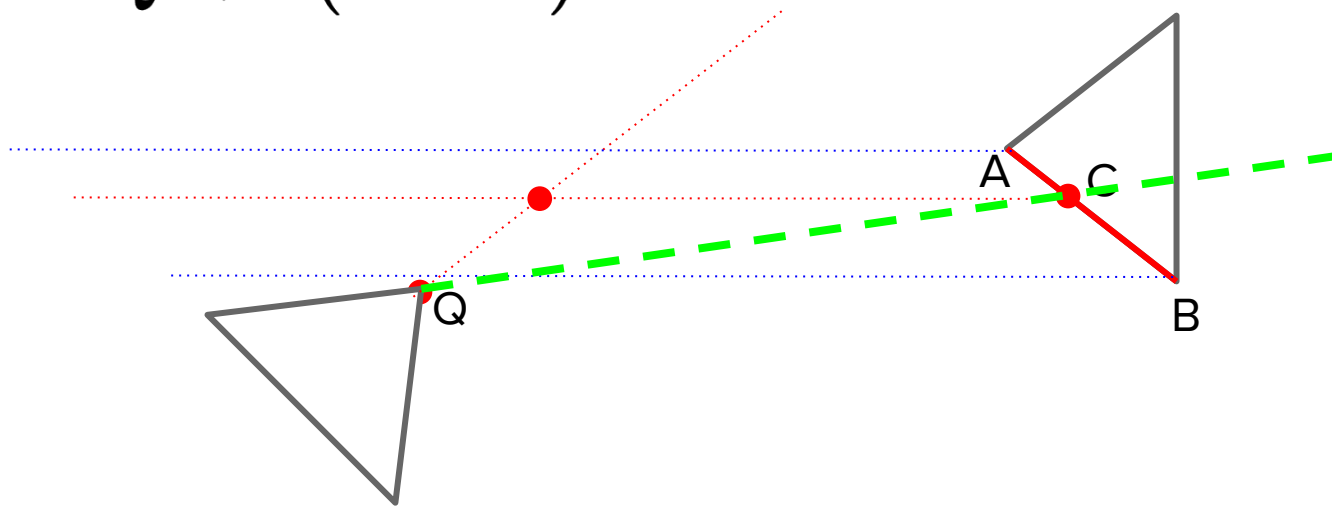
- Se existe alguma intersecção de  $Q$  com  $AB$ , então existe algum tempo  $t$  de forma que  $P = Q + t.v$  pertence ao segmento  $(A + t.u)(B + t.u)$ , que podemos chamar de  $C + t.u$



## D - Geometry Problem

$$Q + t \cdot v = C + t \cdot u$$

$$C = Q + t(v - u)$$



## D - Geometry Problem

- Agora temos uma nova equação vetorial da reta, em que conhecemos um ponto inicial ( $Q$ ) e o vetor diretor ( $v - u$ ), com isso podemos descobrir a equação geral da reta, calcular a intersecção com a reta subjacente ao segmento  $AB$ , verificar se esse ponto está no segmento. Se tudo der certo, no final calcula-se o  $t$  para este ponto de encontro



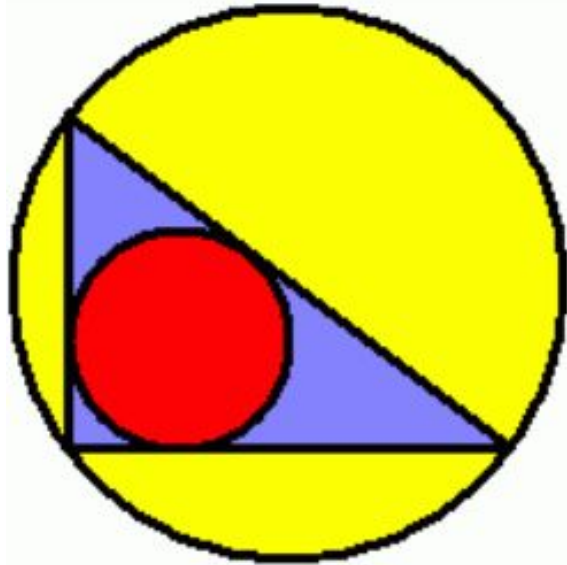
# E - Doors and Penguins



- **Objetivo:** dado dois conjuntos de pontos, determinar se eles são linearmente separáveis, ou seja, se é possível construir uma reta no plano de forma que cada conjunto fique de um lado diferente da reta.

# E - Doors and Penguins

- **Objetivo:** dado dois conjuntos de pontos, determinar se eles são linearmente separáveis, ou seja, se é possível construir uma reta no plano de forma que cada conjunto fique de um lado diferente da reta.
- **Solução:** construir o fecho convexo para cada um dos conjuntos e verificar se não há intersecção entre eles.
  - Para verificar se há intersecção, basta conferir para cada ponto de um fecho convexo se ele está no interior do outro.

# F - Coloufur Flowers



-  sunflowers
-  violets
-  roses

## G - Hide and seek

- Neste problema, temos um conjunto de pontos, representando as crianças, e segmentos (representando as paredes) espalhados no plano.
- **Objetivo:** Para cada *seeker*, determinar quantas crianças ele é capaz de ver, sendo que ele é capaz de ver uma criança quando não há uma parede entre o segmento que liga as duas crianças.

# G - Hide and seek

- **Solução ingênua:**

```
Para cada seeker s
```

```
    ans = 0
```

```
    Para cada criança c (!= s)
```

```
        temInterseccao = 0
```

```
        Para cada parede p
```

```
            Se tem intersecção entre p e Segmento(s,c)
```

```
                temInterseccao = 1
```

```
        Se !temInterseccao
```

```
            ans++
```

# G - Hide and seek

- Solução ingênua:

```
Para cada seeker s
```

```
    ans = 0
```

```
    Para cada criança c (!= s)
```

```
        temInterseccao = 0
```

```
        Para cada parede p
```

```
            Se tem intersecção entre p e Segmento(s,c)
```

```
                temInterseccao = 1
```

```
        Se !temInterseccao
```

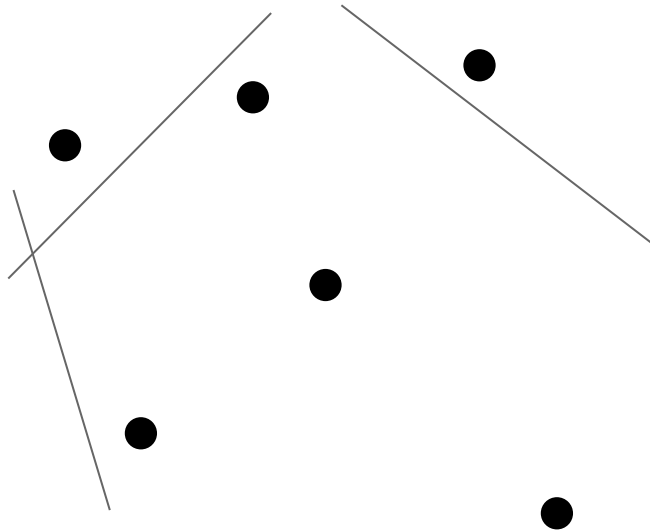
```
            ans++
```

**TLE!**

## G - Hide and seek

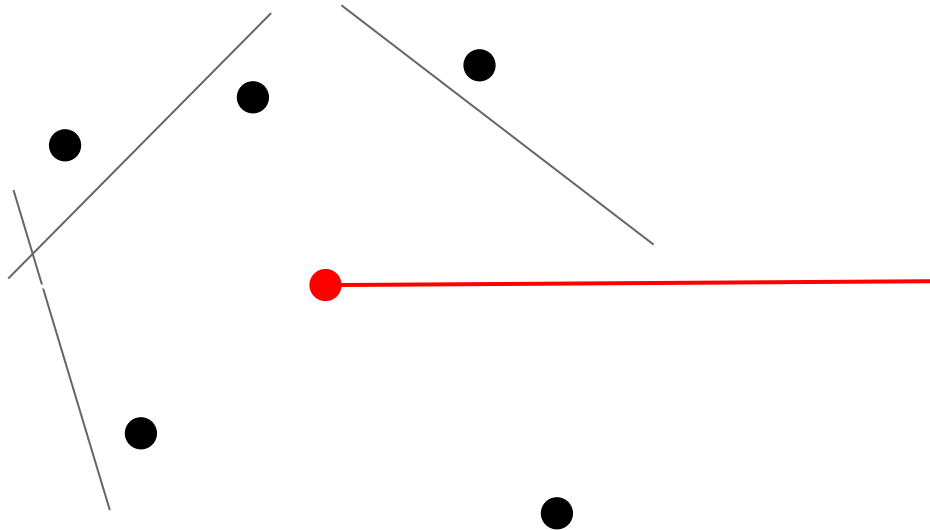
- **Solução:** a ideia ainda é parecida, mas vamos reduzir muito o nível de paredes(segmentos) que precisam ser testados. Para isso, para cada seeker vamos realizar um line sweep variando a inclinação da reta, e assim, quando a reta passar por uma criança, vamos saber por quais paredes estamos passando, e só vamos testar elas, ignorando as outras.

# G - Hide and seek



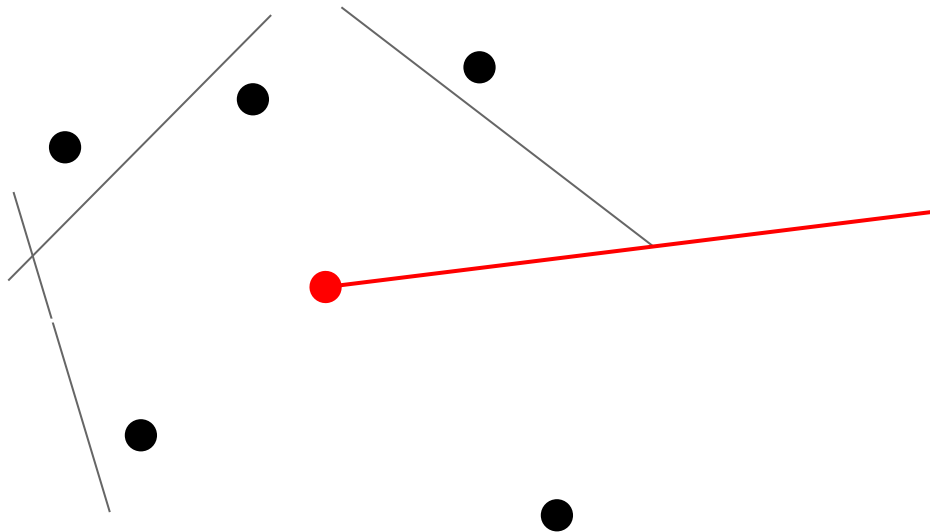


# G - Hide and seek



# G - Hide and seek

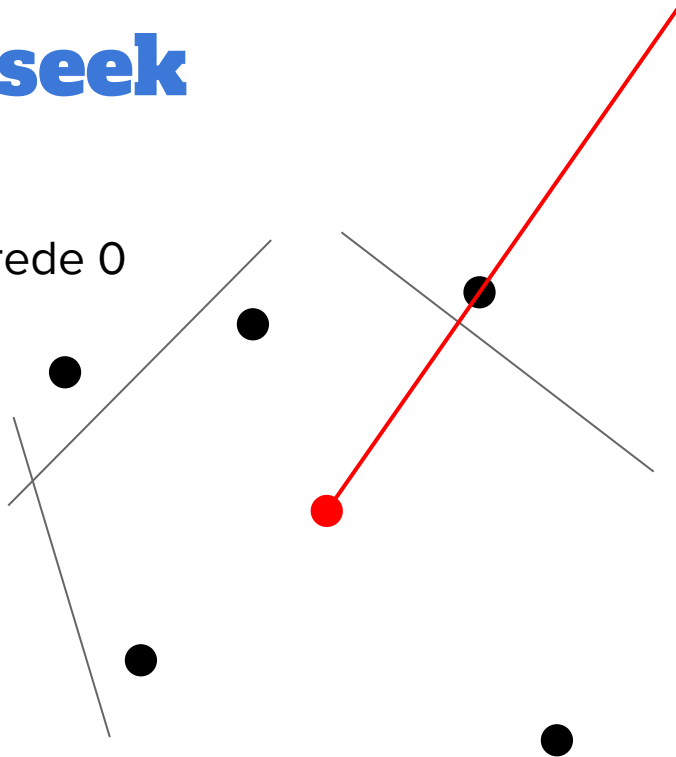
Paredes “ativas”: 0



# G - Hide and seek

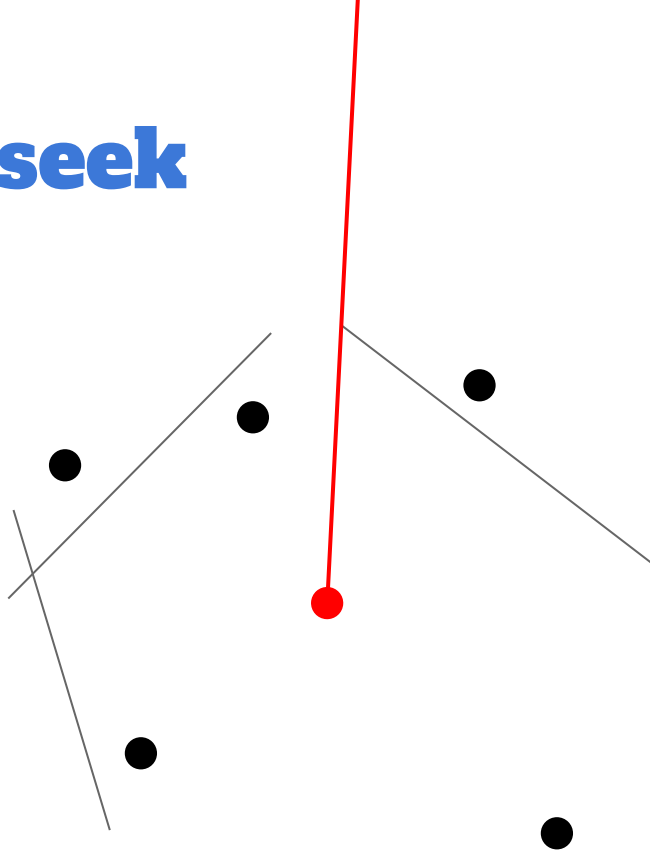
Paredes “ativas”: 0

Tem intersecção com a parede 0



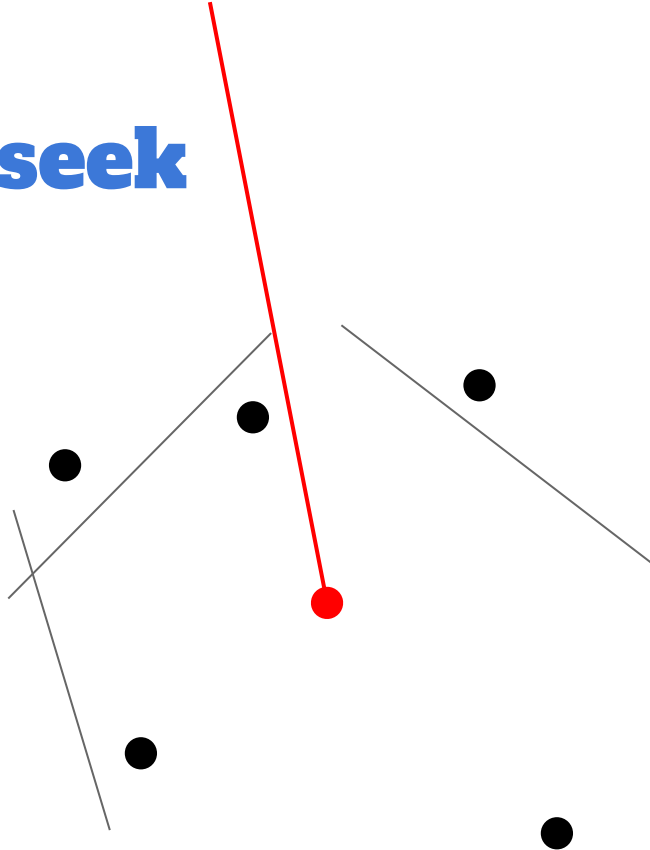
# G - Hide and seek

Paredes “ativas”: 0



# G - Hide and seek

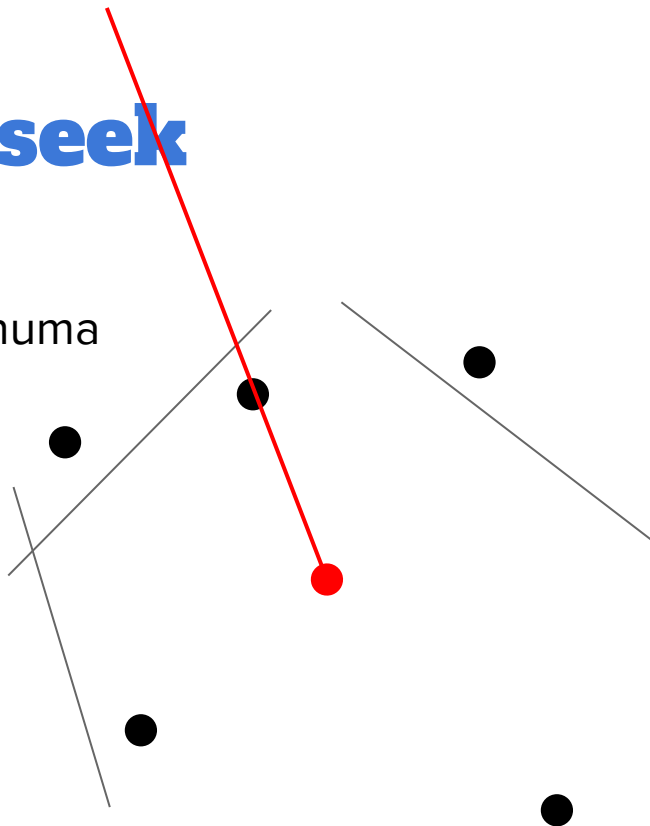
Paredes “ativas”: 1



# G - Hide and seek

Paredes “ativas”: 1

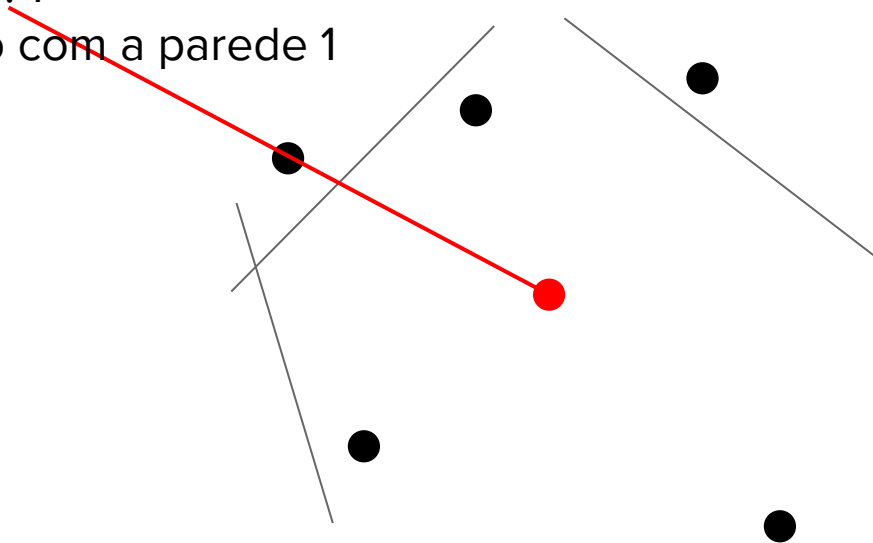
Sem intersecção com nenhuma  
parede ativa



# G - Hide and seek

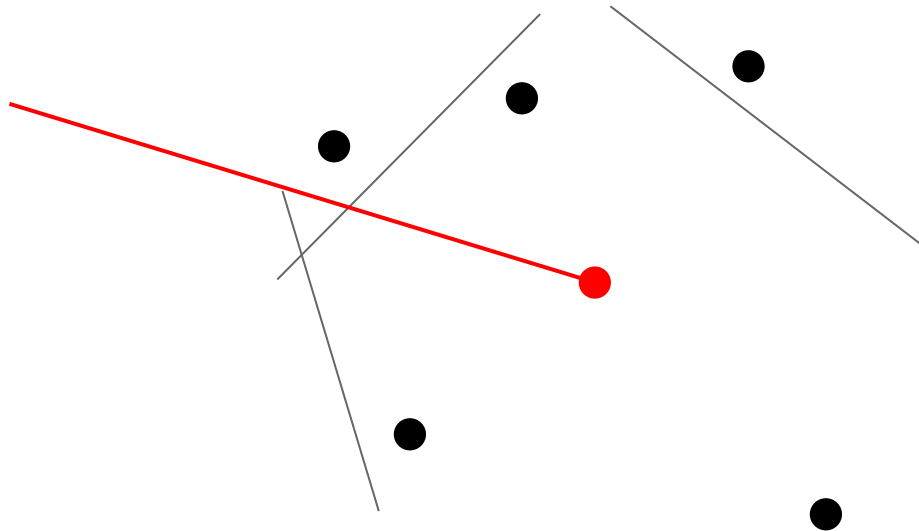
Paredes “ativas”: 1

Tem intersecção com a parede 1



# G - Hide and seek

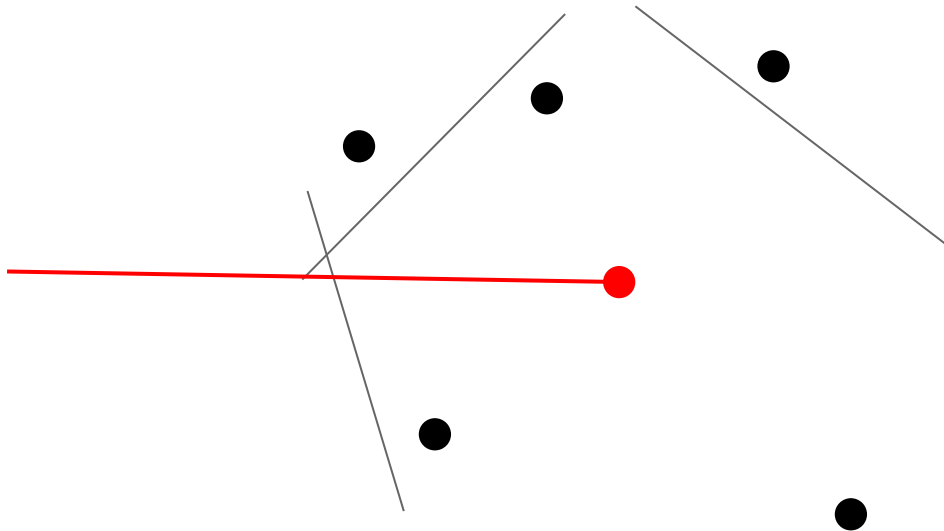
Paredes “ativas”: 1 **2**





# G - Hide and seek

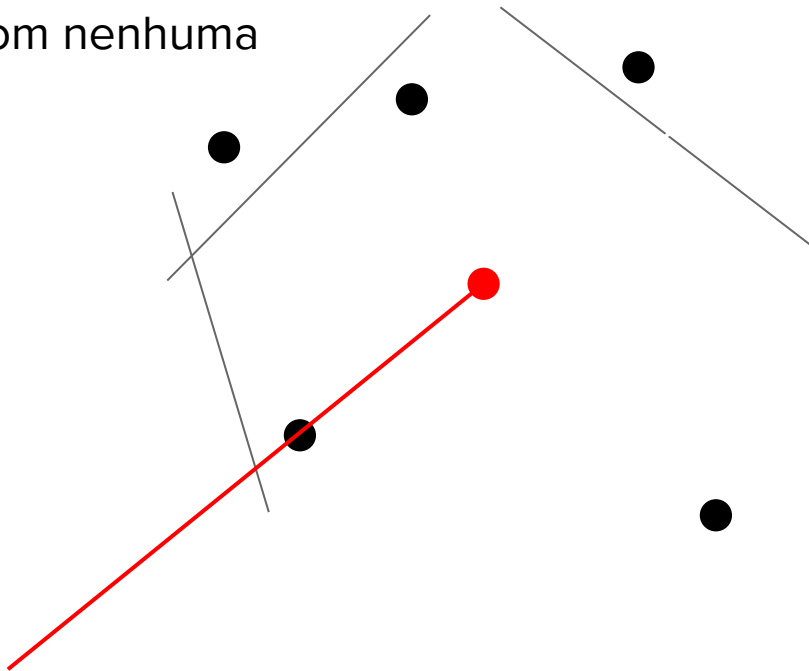
Paredes “ativas”: **1** 2



# G - Hide and seek

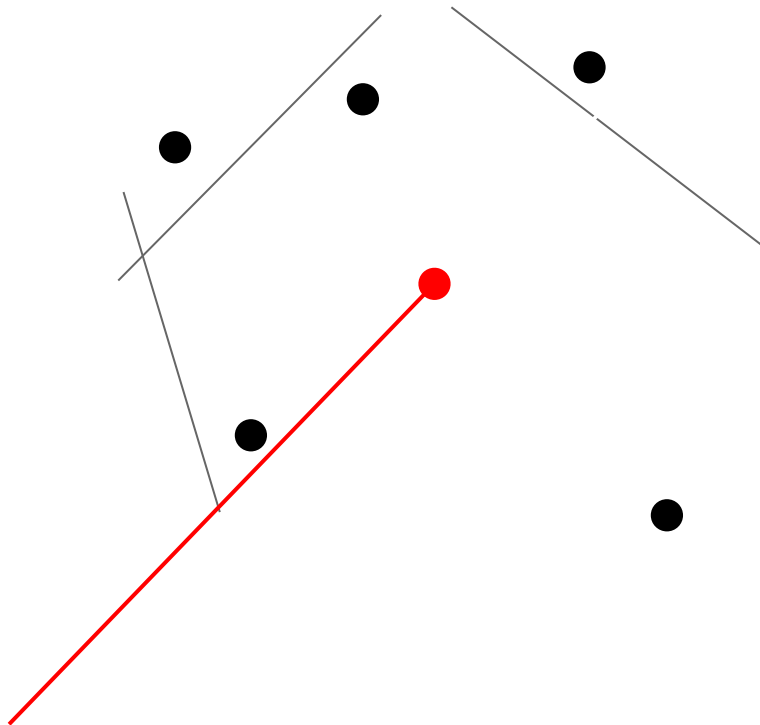
Paredes “ativas”: 2

Sem intersecção com nenhuma  
parede ativa



# G - Hide and seek

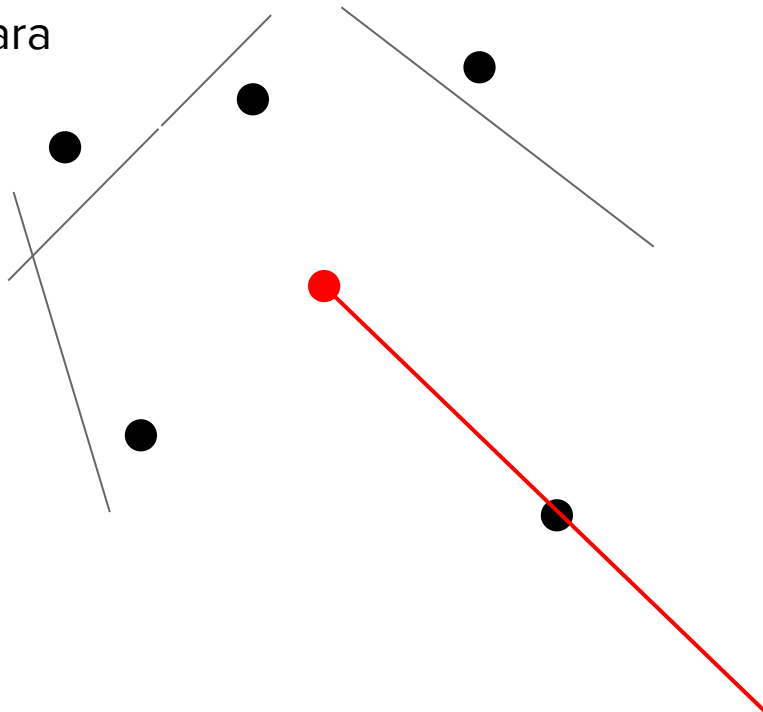
Paredes “ativas”: **2**



# G - Hide and seek

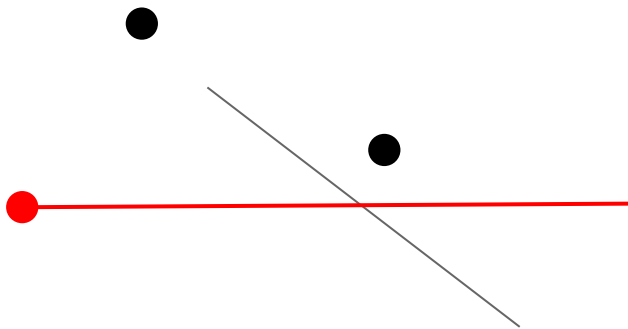
Paredes “ativas”:

Nem sequer há paredes para  
testar



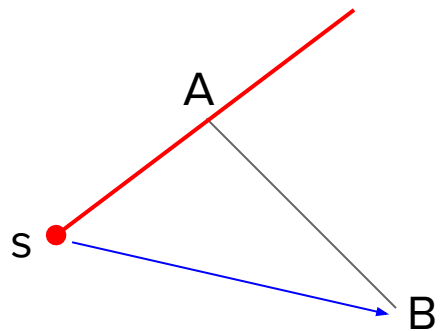
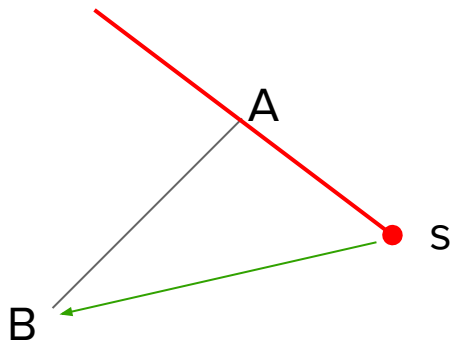
# G - Hide and seek

- **Problemas:** Como todo bom problema de geometria, temos casos degenerados ou específicos que nos causam dor de cabeça. Neste caso pode acontecer de passarmos pelo fim de um segmento ANTES de passar pelo início:



# G - Hide and seek

- **Como resolver?**
  - 1º) Vamos fazer um teste de orientação: considerando o seeker  $s$  e que estamos passando pela extremidade  $A$  da reta, se a extremidade  $B$  está a **esquerda**, então vamos começar a passar pela reta agora. Agora se  $B$  estiver a **direita**, então na verdade estamos saindo da reta



# G - Hide and seek

- **Como resolver?**
  - 2º) Para garantir que não tenhamos retas ativas que ainda não enxergamos (exemplo abaixo), vamos fazer o line sweep duas vezes, na primeira olhando apenas para as retas, de forma que na segunda passada saberemos quais retas estão ativas antes do início da varredura.

