

# Interseção de segmentos

**Problema:** Dados  $n$  segmentos, determinar se dois deles se intersectam.

# Interseção de segmentos

**Problema:** Dados  $n$  segmentos, determinar se dois deles se intersectam.

**Aula passada:** algoritmo  $O(n \lg n)$  para esse problema.

# Interseção de segmentos

**Problema:** Dados  $n$  segmentos, determinar se dois deles se intersectam.

**Aula passada:** algoritmo  $O(n \lg n)$  para esse problema.

**Estrutura de dados:**

- ▶ árvore de busca binária balanceada (ABBB) ou
- ▶ treap ou skip list (tempo esperado  $O(n \lg n)$ )

# Interseção de segmentos

**Problema:** Dados  $n$  segmentos, determinar se dois deles se intersectam.

**Aula passada:** algoritmo  $O(n \lg n)$  para esse problema.

**Estrutura de dados:**

- ▶ árvore de busca binária balanceada (ABBB) ou
- ▶ treap ou skip list (tempo esperado  $O(n \lg n)$ )

**Hipótese simplificadora:**

Não há dois pontos extremos com mesma  $x$ -coordenada.

Em particular, não há segmentos verticais,  
nem dois segmentos com extremos coincidentes.

# Interseção de segmentos

**Problema:** Dados  $n$  segmentos, determinar se dois deles se intersectam.

**Aula passada:** algoritmo  $O(n \lg n)$  para esse problema.

**Estrutura de dados:**

- ▶ árvore de busca binária balanceada (ABBB) ou
- ▶ treap ou skip list (tempo esperado  $O(n \lg n)$ )

**Hipótese simplificadora:**

Não há dois pontos extremos com mesma  $x$ -coordenada.

Em particular, não há segmentos verticais,  
nem dois segmentos com extremos coincidentes.

**Pré-processamento:**

ordene os extremos dos segmentos por  $x$ -coordenada.

## Como evitar a hipótese simplificadora?

Pontos extremos com mesma  $x$ -coordenada:

Se existir um segmento vertical,  
deixe o extremo inferior no vetor  $e$  e o superior no vetor  $d$ .

## Como evitar a hipótese simplificadora?

Pontos extremos com mesma  $x$ -coordenada:

Se existir um segmento vertical,  
deixe o extremo inferior no vetor  $e$  e o superior no vetor  $d$ .

Se houver extremos repetidos, há interseção.

# Como evitar a hipótese simplificadora?

Pontos extremos com mesma  $x$ -coordenada:

Se existir um segmento vertical,  
deixe o extremo inferior no vetor  $e$  e o superior no vetor  $d$ .

Se houver extremos repetidos, há interseção.

Extremo **esquerdo** de um segmento:

- ▶ extremo cuja  $x$ -coordenada é menor.
- ▶ caso o segmento seja **vertical**,  
chame de esquerdo o extremo  
com  $y$ -coordenada menor.



# Como evitar a hipótese simplificadora?

Pontos extremos com mesma  $x$ -coordenada:

Se existir um segmento vertical,  
deixe o extremo inferior no vetor  $e$  e o superior no vetor  $d$ .

Se houver extremos repetidos, há interseção.

Extremo **esquerdo** de um segmento:

- ▶ extremo cuja  $x$ -coordenada é menor.
- ▶ caso o segmento seja **vertical**,  
chame de esquerdo o extremo  
com  $y$ -coordenada menor.

O outro extremo é o **direito**.

# Como evitar a hipótese simplificadora?

Pontos extremos com mesma  $x$ -coordenada:

Se existir um segmento vertical,  
deixe o extremo inferior no vetor  $e$  e o superior no vetor  $d$ .

Se houver extremos repetidos, há interseção.

Extremo **esquerdo** de um segmento:

- ▶ extremo cuja  $x$ -coordenada é menor.
- ▶ caso o segmento seja **vertical**,  
chame de esquerdo o extremo  
com  $y$ -coordenada menor.

O outro extremo é o **direito**.

**Extremos-Ordenados**( $n, S$ ): ordena os extremos  
dos  $n$  segmentos em  $S$  e já dá a resposta se houver repetição.

# Detecção de interseção

Detecta-Interseção( $n, S$ )

```
1   $E \leftarrow$  Extremos-Ordenados( $n, S$ )
2   $T \leftarrow \emptyset$       ▷ ABBB ou treap ou skip list
3  para cada  $p \in E$  faça
4     $s \leftarrow$  segmento( $p$ )
5     $pred \leftarrow$  Predecessor( $T, s$ )     $suc \leftarrow$  Sucessor( $T, s$ )
6    se  $p$  é extremo esquerdo de  $s$ 
7      então Insere( $T, s$ )
8          se ( $pred \neq \text{NIL}$  e Intersecta( $s, pred$ ))
9              ou ( $suc \neq \text{NIL}$  e Intersecta( $s, suc$ ))
10             então devolva verdade
11    senão Remove( $T, s$ )
12          se  $pred$  e  $suc \neq \text{NIL}$  e Intersecta( $pred, suc$ )
13             então devolva verdade
14    devolva falso
```

## Inserção em ABB (rubro-negra)

**InsiraRec** ( $T, x$ )

- 1 se  $T = \text{NIL}$
- 2     então  $q \leftarrow \text{NovaCélula}(x, \text{NIL}, \text{NIL}, \text{rubro})$
- 3     devolva  $q$
- 4 se  $x < \text{info}(T)$      ▷ Vamos alterar aqui!
- 5     então  $\text{esq}(T) \leftarrow \text{InsiraRec}(\text{esq}(T), x)$
- 6     senão  $\text{dir}(T) \leftarrow \text{InsiraRec}(\text{dir}(T), x)$
- 7 se **Rubro**( $\text{dir}(T)$ ) e **Negro**( $\text{esq}(T)$ )
- 8     então  $T \leftarrow \text{GireEsq}(T)$
- 9 se **Rubro**( $\text{esq}(T)$ ) e **Rubro**( $\text{esq}(\text{esq}(T))$ )
- 10     então  $T \leftarrow \text{GireDir}(T)$
- 11 se **Rubro**( $\text{esq}(T)$ ) e **Rubro**( $\text{dir}(T)$ )
- 12     então **TroqueCores**( $T$ )
- 13 devolva  $T$

## Inserção em ABB rubro-negra

**InsiraRec** ( $T, e, d, i$ )

- 1 se  $T = \text{NIL}$
- 2     então  $q \leftarrow \text{NovaCélula}(i, \text{NIL}, \text{NIL}, \text{rubro})$
- 3     devolva  $q$
- 4 se  $\text{Esquerda}(e[\text{segmento}(T)], d[\text{segmento}(T)])$ ,  $e[i]$
- 5     então  $\text{esq}(T) \leftarrow \text{InsiraRec}(\text{esq}(T), i)$
- 6     senão  $\text{dir}(T) \leftarrow \text{InsiraRec}(\text{dir}(T), i)$
- 7 se  $\text{Rubro}(\text{dir}(T))$  e  $\text{Negro}(\text{esq}(T))$
- 8     então  $T \leftarrow \text{GireEsq}(T)$
- 9 se  $\text{Rubro}(\text{esq}(T))$  e  $\text{Rubro}(\text{esq}(\text{esq}(T)))$
- 10    então  $T \leftarrow \text{GireDir}(T)$
- 11 se  $\text{Rubro}(\text{esq}(T))$  e  $\text{Rubro}(\text{dir}(T))$
- 12    então  $\text{TroqueCores}(T)$
- 13 devolva  $T$

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Você consegue projetar um algoritmo que consuma tempo  $O(n \lg n)$  para este problema?

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Você consegue projetar um algoritmo que consuma tempo  $O(n \lg n)$  para este problema?

No máximo, quantos pares teremos que imprimir?



# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Você consegue projetar um algoritmo que consuma tempo  $O(n \lg n)$  para este problema?

No máximo, quantos pares teremos que imprimir?

Algoritmos sensíveis à saída (*output sensitive*).

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Como adaptar o algoritmo de Shamos e Hoey?

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Como adaptar o algoritmo de Shamos e Hoey?

**Novo tipo de ponto evento:** as interseções.

Como tratá-las?

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Como adaptar o algoritmo de Shamos e Hoey?

**Novo tipo de ponto evento:** as interseções.

**Como tratá-las?**

Ao detectar cada uma, além de imprimi-la, a colocamos na fila de eventos (que é agora **dinâmica**).

# Todas as interseções de segmentos

**Problema:** Dada uma coleção de  $n$  segmentos no plano, encontrar todos os pares de segmentos da coleção que se intersectam.

Como adaptar o algoritmo de Shamos e Hoey?

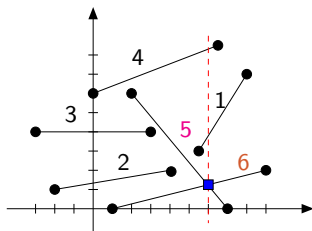
**Novo tipo de ponto evento:** as interseções.

**Como tratá-las?**

Ao detectar cada uma, além de imprimi-la, a colocamos na fila de eventos (que é agora **dinâmica**).

Ao processar um ponto evento que é uma interseção, deve-se inverter a ordem dos segmentos que se intersectam neste ponto.

## Ponto evento: interseção



Antes do ponto evento:  $4 \prec 1 \prec 5 \prec 6$

Depois do ponto evento:  $4 \prec 1 \prec 6 \prec 5$

# Algoritmo de Bentley e Ottmann

**Entrada:** coleção  $e[1..n]$ ,  $d[1..n]$  de segmentos.



# Algoritmo de Bentley e Ottmann

**Entrada:** coleção  $e[1..n]$ ,  $d[1..n]$  de segmentos.

**Saída:** todos os pares de segmentos da coleção que se intersectam.

# Algoritmo de Bentley e Ottmann

**Entrada:** coleção  $e[1..n]$ ,  $d[1..n]$  de segmentos.

**Saída:** todos os pares de segmentos da coleção que se intersectam.

**Hipótese simplificadora:**

Não há dois pontos eventos com a mesma  $x$ -coordenada.

Em particular, não há interseção com mesma  $x$ -coordenada que outra, ou que algum extremo de segmento.

# Algoritmo de Bentley e Ottmann

**Entrada:** coleção  $e[1..n]$ ,  $d[1..n]$  de segmentos.

**Saída:** todos os pares de segmentos da coleção que se intersectam.

**Hipótese simplificadora:**

Não há dois pontos eventos com a mesma  $x$ -coordenada.

Em particular, não há interseção com mesma  $x$ -coordenada que outra, ou que algum extremo de segmento.

Não há interseções múltiplas, ou seja, não há um ponto em mais do que dois segmentos da coleção.

## Fila de eventos

Agora ela é **dinâmica**: sofre **inserções** (e, como antes, **remoções**).

Que ED usar para a fila de eventos?

## Fila de eventos

Agora ela é **dinâmica**: sofre **inserções** (e, como antes, **remoções**).

Que ED usar para a fila de eventos?

**ABBB** ou **heap** com ordem dada pelas  $x$ -coordenadas dos pontos.

## Fila de eventos

Agora ela é **dinâmica**: sofre **inserções** (e, como antes, **remoções**).

Que ED usar para a fila de eventos?

**ABBB** ou **heap** com ordem dada pelas  $x$ -coordenadas dos pontos.

A fila começa com os extremos dos intervalos.

A cada iteração,  
removemos um evento da fila para processá-lo.

## Fila de eventos

Agora ela é **dinâmica**: sofre **inserções** (e, como antes, **remoções**).

Que ED usar para a fila de eventos?

**ABBB** ou **heap** com ordem dada pelas  $x$ -coordenadas dos pontos.

A fila começa com os extremos dos intervalos.

A cada iteração,  
removemos um evento da fila para processá-lo.

Ao detectar uma interseção,  
inserimos tal ponto na fila de eventos. (**Sempre?**)

**Quantos elementos estão na fila no pior caso?**

## Versão simplificada

**Hipótese simplificadora:** não há pontos extremos repetidos, e as interseções são em pontos internos dos segmentos.



## Versão simplificada

**Hipótese simplificadora:** não há pontos extremos repetidos, e as interseções são em pontos internos dos segmentos.

**Extremos-Ordenados**( $n, S$ ):

ordena os extremos dos  $n$  segmentos em  $S$ .

## Versão simplificada

**Hipótese simplificadora:** não há pontos extremos repetidos, e as interseções são em pontos internos dos segmentos.

**Extremos-Ordenados**( $n, S$ ):

ordena os extremos dos  $n$  segmentos em  $S$ .

**Acha-Interseções**( $n, S$ )

- 1  $Q \leftarrow$  **Extremos**( $n, S$ ) ▷ inicializa a ABBB ou heap  $Q$  com os extremos
- 2  $T \leftarrow \emptyset$
- 3 **enquanto** não **Vazia**( $Q$ ) **faça**
- 4      $p \leftarrow$  **Extrai-Min**( $Q$ )
- 5     **Trata-Evento**( $p$ )

# Versão simplificada

**Hipótese simplificadora:** não há pontos extremos repetidos, e as interseções são em pontos internos dos segmentos.

**Extremos-Ordenados**( $n, S$ ):

ordena os extremos dos  $n$  segmentos em  $S$ .

**Acha-Interseções**( $n, S$ )

- 1  $Q \leftarrow \text{Extremos}(n, S)$  ▷ inicializa a ABBB ou heap  $Q$  com os extremos
- 2  $T \leftarrow \emptyset$
- 3 **enquanto** não **Vazia**( $Q$ ) **faça**
- 4      $p \leftarrow \text{Extrai-Min}(Q)$
- 5     **Trata-Evento**( $p$ )

**Notação:** Para dois pontos evento  $p$  e  $q$ , escrevemos  $p \prec q$  se  $p_x < q_x$  ou ( $p_x = q_x$  e  $p_y < q_y$ )

## Versão simplificada

Trata-Evento( $p$ )

- 1 se  $p$  é extremo esquerdo de um segmento  $s$
- 2     então **Insera**( $T, s$ )
- 3          $pred \leftarrow$  Predecessor( $T, s$ )
- 4          $suc \leftarrow$  Sucessor( $T, s$ )
- 5         se  $pred \neq \text{NIL}$  e **Intersecta**( $s, pred$ )
- 6             então **Verifica-Novo-Evento**( $p, Q, s, pred$ )
- 7         se  $suc \neq \text{NIL}$  e **Intersecta**( $s, suc$ )
- 8             então **Verifica-Novo-Evento**( $p, Q, s, suc$ )

## Versão simplificada

### Trata-Evento( $p$ )

- 1 se  $p$  é extremo esquerdo de um segmento  $s$
- 2     então  $\text{Insere}(T, s)$
- 3          $pred \leftarrow \text{Predecessor}(T, s)$
- 4          $suc \leftarrow \text{Sucessor}(T, s)$
- 5         se  $pred \neq \text{NIL}$  e  $\text{Intersecta}(s, pred)$
- 6             então  $\text{Verifica-Novo-Evento}(p, Q, s, pred)$
- 7         se  $suc \neq \text{NIL}$  e  $\text{Intersecta}(s, suc)$
- 8             então  $\text{Verifica-Novo-Evento}(p, Q, s, suc)$

### Verifica-Novo-Evento( $p, Q, s_1, s_2$ )

- 1  $q \leftarrow \text{Ponto-de-Interseção}(s_1, s_2)$
- 2 se  $q \succ p$  e não  $\text{Pertence}(Q, q)$
- 3     então  $\text{Insere}(Q, q)$
- 4     imprima  $q$

## Versão simplificada

Trata-Evento( $p$ )

- 1 se  $p$  é extremo esquerdo de um segmento  $s$
- 2   então **Insera**( $T, s$ )
- 3        $pred \leftarrow$  Predecessor( $T, s$ )
- 4        $suc \leftarrow$  Sucessor( $T, s$ )
- 5       se  $pred \neq \text{NIL}$  e **Intersecta**( $s, pred$ )
- 6           então **Verifica-Novo-Evento**( $p, Q, s, pred$ )
- 7       se  $suc \neq \text{NIL}$  e **Intersecta**( $s, suc$ )
- 8           então **Verifica-Novo-Evento**( $p, Q, s, suc$ )
- 9 se  $p$  é extremo direito de um segmento  $s$
- 10   então **Remove**( $T, s$ )
- 11        $pred \leftarrow$  Predecessor( $T, s$ )
- 12        $suc \leftarrow$  Sucessor( $T, s$ )
- 13       se  $pred$  e  $suc \neq \text{NIL}$  e **Intersecta**( $pred, suc$ )
- 14           então **Verifica-Novo-Evento**( $p, Q, suc, pred$ )

# Versão simplificada

Trata-Evento( $p$ )

...

15 se  $p$  é ponto de interseção

16 então sejam  $s$  e  $s'$  os segmentos em  $T$  que contém  $p$

17  $pred \leftarrow$  Predecessor( $T, s$ )

18  $suc \leftarrow$  Sucessor( $T, s'$ )

19 Remove( $T, s$ ) Remove( $T, s'$ )

▷ insere  $s$  e  $s'$  na ordem inversa

20 Inse(re( $T, s'$ ) Inse(re( $T, s$ )

21 se  $pred \neq \text{NIL}$  e Intersecta( $pred, s'$ )

22 então Verifica-Novo-Evento( $p, Q, pred, s'$ )

23 se  $suc \neq \text{NIL}$  e Intersecta( $s, suc$ )

24 então Verifica-Novo-Evento( $p, Q, s, suc$ )

## Consumo de tempo

Seja  $i$  o número de interseções.

O algoritmo executa  $2n + i$  iterações.



## Consumo de tempo

Seja  $i$  o número de interseções.

O algoritmo executa  $2n + i$  iterações.

Cada iteração faz uma chamada a **Predecessor**, **Sucessor**, e uma ou duas a **Insere** e/ou **Remove**, na ABBB  $T$ .

Na ABBB  $T$ , em qualquer momento, há  $O(n)$  segmentos.

Assim, cada operação destas consome tempo  $O(\lg n)$ .

## Consumo de tempo

Seja  $i$  o número de interseções.

O algoritmo executa  $2n + i$  iterações.

Cada iteração faz uma chamada a **Predecessor**, **Sucessor**, e uma ou duas a **Insere** e/ou **Remove**, na ABBB  $T$ .

Na ABBB  $T$ , em qualquer momento, há  $O(n)$  segmentos.

Assim, cada operação destas consome tempo  $O(\lg n)$ .

Cada iteração faz uma chamada a **Extrai-Min** e, eventualmente, uma ou duas a **Insere** na ABBB ou heap  $Q$ .

Na ABBB ou heap  $Q$ , em qq momento, há  $O(n + i) = O(n^2)$  pontos.

Assim, cada operação consome tempo  $O(\lg n^2) = O(\lg n)$ .

## Consumo de tempo

Seja  $i$  o número de interseções.

O algoritmo executa  $2n + i$  iterações.

Cada iteração faz uma chamada a **Predecessor**, **Sucessor**, e uma ou duas a **Insere** e/ou **Remove**, na ABBB  $T$ .

Na ABBB  $T$ , em qualquer momento, há  $O(n)$  segmentos.

Assim, cada operação destas consome tempo  $O(\lg n)$ .

Cada iteração faz uma chamada a **Extrai-Min** e, eventualmente, uma ou duas a **Insere** na ABBB ou heap  $Q$ .

Na ABBB ou heap  $Q$ , em qq momento, há  $O(n + i) = O(n^2)$  pontos.

Assim, cada operação consome tempo  $O(\lg n^2) = O(\lg n)$ .

As demais operações efetuadas em uma iteração consomem tempo  $O(1)$  (mesmo as chamadas a **Inter**).

## Consumo de tempo

Seja  $i$  o número de interseções.

O algoritmo executa  $2n + i$  iterações.

Cada iteração faz uma chamada a **Predecessor**, **Sucessor**, e uma ou duas a **Insere** e/ou **Remove**, na ABBB  $T$ .

Na ABBB  $T$ , em qualquer momento, há  $O(n)$  segmentos.

Assim, cada operação destas consome tempo  $O(\lg n)$ .

Cada iteração faz uma chamada a **Extrai-Min** e, eventualmente, uma ou duas a **Insere** na ABBB ou heap  $Q$ .

Na ABBB ou heap  $Q$ , em qq momento, há  $O(n + i) = O(n^2)$  pontos.

Assim, cada operação consome tempo  $O(\lg n^2) = O(\lg n)$ .

O consumo de tempo por iteração é  $O(\lg n)$ , e o algoritmo de Bentley e Ottmann consome tempo  $O((n + i) \lg n)$ .

## Versão completa

O que fazer com os casos que excluimos?

## Versão completa

O que fazer com os casos que excluimos?

### Alterações:

- ▶  $Q$  conterà os **pontos evento**, sem repetições.
- ▶ **Ponto evento extremo**:  
tem a lista dos segmentos que têm esse ponto como extremo.
- ▶ impressão apenas no momento de processamento do ponto.

## Versão completa

O que fazer com os casos que excluimos?

### Alterações:

- ▶  $Q$  conterá os **pontos evento**, sem repetições.
- ▶ **Ponto evento extremo**:  
tem a lista dos segmentos que têm esse ponto como extremo.
- ▶ impressão apenas no momento de processamento do ponto.

Ao processar um ponto evento,  
determinam-se todos os segmentos que o contém  
(pela lista do ponto e/ou pelos segmentos em  $T$ ).

## Versão completa

O que fazer com os casos que excluimos?

### Alterações:

- ▶  $Q$  conterà os **pontos evento**, sem repetições.
- ▶ **Ponto evento extremo**:  
tem a lista dos segmentos que têm esse ponto como extremo.
- ▶ impressão apenas no momento de processamento do ponto.

Ao processar um ponto evento,  
determinam-se todos os segmentos que o contém  
(**pela lista do ponto e/ou pelos segmentos em  $T$** ).

Se mais de um segmento o contém, imprimimos o ponto.



## Versão completa

O que fazer com os casos que excluimos?

### Alterações:

- ▶  $Q$  conterà os **pontos evento**, sem repetições.
- ▶ **Ponto evento extremo**:  
tem a lista dos segmentos que têm esse ponto como extremo.
- ▶ impressão apenas no momento de processamento do ponto.

Ao processar um ponto evento,  
determinam-se todos os segmentos que o contém  
(**pela lista do ponto e/ou pelos segmentos em  $T$** ).

Se mais de um segmento o contém, imprimimos o ponto.

Atualiza-se  $T$ .

## Atualização de $T$

Se o ponto evento é um extremo, faz-se como antes:

## Atualização de $T$

Se o **ponto evento é um extremo**, faz-se como antes:

- ▶ extremos esquerdos causam inclusões em  $T$ .
- ▶ extremos direitos causam remoções.

## Atualização de $T$

Se o **ponto evento é um extremo**, faz-se como antes:

- ▶ extremos esquerdos causam inclusões em  $T$ .
- ▶ extremos direitos causam remoções.

Primeiro trata-se de extremos esquerdos.

## Atualização de $T$

Se o **ponto evento é um extremo**, faz-se como antes:

- ▶ extremos esquerdos causam inclusões em  $T$ .
- ▶ extremos direitos causam remoções.

Primeiro trata-se de extremos esquerdos.

Se o **ponto evento é uma interseção**

# Atualização de $T$

Se o **ponto evento é um extremo**, faz-se como antes:

- ▶ extremos esquerdos causam inclusões em  $T$ .
- ▶ extremos direitos causam remoções.

Primeiro trata-se de extremos esquerdos.

Se o **ponto evento é uma interseção**

- ▶ remove-se de  $T$  todos os segmentos que o contém no interior.
- ▶ estes são incluídos novamente **na ordem inversa**.

## Atualização de $T$

Se o **ponto evento é um extremo**, faz-se como antes:

- ▶ extremos esquerdos causam inclusões em  $T$ .
- ▶ extremos direitos causam remoções.

Primeiro trata-se de extremos esquerdos.

Se o **ponto evento é uma interseção**

- ▶ remove-se de  $T$  todos os segmentos que o contém no interior.
- ▶ estes são incluídos novamente **na ordem inversa**.

Os dois casos podem acontecer ao mesmo tempo...

Isso está detalhado no livro de de Berg e outros, capítulo 2.

## Comentários finais

O algoritmo pode ser ajustado para imprimir,  
para cada ponto de interseção, a lista de segmentos que o contém.



## Comentários finais

O algoritmo pode ser ajustado para imprimir, para cada ponto de interseção, a lista de segmentos que o contém.

**Consumo de tempo:**  $O((n + i) \lg n)$

(esperado, no caso de uso de treaps ou skip lists)

onde  $i$  agora é o número de segmentos

impressos junto com as interseções.

## Comentários finais

O algoritmo pode ser ajustado para imprimir, para cada ponto de interseção, a lista de segmentos que o contém.

**Consumo de tempo:**  $O((n + i) \lg n)$

(esperado, no caso de uso de treaps ou skip lists)

onde  $i$  agora é o número de segmentos

impressos junto com as interseções.

**Consumo de espaço:**  $O(n)$  para  $T$  e  $O(n + i)$  para  $Q$ .

## Comentários finais

O algoritmo pode ser ajustado para imprimir, para cada ponto de interseção, a lista de segmentos que o contém.

**Consumo de tempo:**  $O((n + i) \lg n)$

(esperado, no caso de uso de treaps ou skip lists)

onde  $i$  agora é o número de segmentos

impressos junto com as interseções.

**Consumo de espaço:**  $O(n)$  para  $T$  e  $O(n + i)$  para  $Q$ .

**Melhora:** Guarde em  $Q$  apenas

os pontos de interseção de segmentos que estão consecutivos em  $T$ .

Espaço cai para  $O(n)$ .

## Comentários finais

O algoritmo pode ser ajustado para imprimir, para cada ponto de interseção, a lista de segmentos que o contém.

**Consumo de tempo:**  $O((n + i) \lg n)$

(esperado, no caso de uso de treaps ou skip lists)

onde  $i$  agora é o número de segmentos

impressos junto com as interseções.

**Consumo de espaço:**  $O(n)$  para  $T$  e  $O(n + i)$  para  $Q$ .

**Melhora:** Guarde em  $Q$  apenas

os pontos de interseção de segmentos que estão consecutivos em  $T$ .

Espaço cai para  $O(n)$ .

**Algoritmo de Balaban:** tempo  $O(n \lg n + i)$  e espaço  $O(n)$ .