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 $\mathrm{O}(n \lg n)$ para esse problema.

Interseção de segmentos

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

Aula passada: algoritmo $\mathrm{O}(n \lg n)$ para esse problema.

Estrutura de dados:

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

Detecção de interseção

```
Detecta-Interseção(n, S)
E ← Extremos-Ordenados(n, S)
T ← ∅          ▷ ABBB ou treap ou skip list
para cada p ∈ E faça
    s ← segmento(p)
    pred ← Predecessor(T, s)        suc ← Sucessor(T, s)
    se p é extremo esquerdo de s
        então Insere(T, s)
                  se (pred ≠ NIL e Intersecta(s, pred))
                  ou (suc ≠ NIL e Intersecta(s, suc))
                      então devolva verdade
        senão Remove(T, s)
                  se pred e suc ≠ NIL e Intersecta(pred, suc)
                      então devolva verdade
devolva falso
```

Inserção em ABB (rubro-negra)

```
InsiraRec (T, x)
se T = NIL
    então q ← NovaCélula(x, NIL, NIL, rubro)
            devolva q
se x < info(T)          ▷ Vamos alterar aqui!
    então esq(T) ← InsiraRec(esq(T), x)
    senão dir(T) ← InsiraRec(dir(T), x)
se Rubro(dir(T)) e Negro(esq(T))
    então T ← GireEsq(T)
se Rubro(esq(T)) e Rubro(esq(esq(T)))
    então T ← GireDir(T)
se Rubro(esq(T)) e Rubro(dir(T))
    então TroqueCores(T)
devolva T
```

Inserção em ABB (rubro-negra)

```
InsiraRec (T, e, d, i)
se T = NIL
    então q ← NovaCélula(i, NIL, NIL, rubro)
            devolva q
se Esquerda(e[segmento(T)], d[segmento(T)]), e[i])
    então esq(T) ← InsiraRec(esq(T), i)
    senão dir(T) ← InsiraRec(dir(T), i)
se Rubro(dir(T)) e Negro(esq(T))
    então T ← GireEsq(T)
se Rubro(esq(T)) e Rubro(esq(esq(T)))
    então T ← GireDir(T)
se Rubro(esq(T)) e Rubro(dir(T))
    então TroqueCores(T)
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 $\mathrm{O}(n \lg n)$ 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 $\mathrm{O}(n \lg n)$ 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 $\mathrm{O}(n \lg n)$ 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?

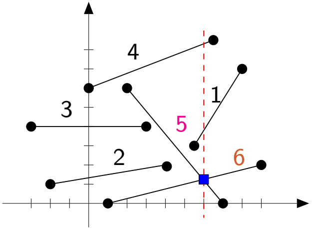
Novo tipo de ponto evento: as interseções.

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 com 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 com 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 por x-coordenada.

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 por x-coordenada.

```
Acha-Interseções(n, S)
    Q ← Extremos(n, S)        ▷ inicializa a ABBB ou heap Q com os extremos
    T ← ∅
    enquanto não Vazia(Q) faça
        p ← Extrai-Min(Q)
        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 por x-coordenada.

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 $\quad p \leftarrow$ Extrai-Min(Q)
5 $\quad$ Trata-Evento(p)

Notação: Para dois pontos evento p e q, escrevemos $p \prec q$ se $p_x < q_x$.

Versão simplificada

```
Trata-Evento(p)
se p é extremo esquerdo de um segmento s
   então Insere(T, s)
         pred ← Predecessor(T, s)
         suc ← Sucessor(T, s)
         se pred ≠ NIL e Intersecta(s, pred)
            então Verifica-Novo-Evento(p, Q, s, pred)
         se suc ≠ NIL e Intersecta(s, suc)
            então Verifica-Novo-Evento(p, Q, s, suc)
```

Versão simplificada

```
Trata-Evento(p)
se p é extremo esquerdo de um segmento s
   então Insere(T, s)
         pred ← Predecessor(T, s)
         suc ← Sucessor(T, s)
         se pred ≠ NIL e Intersecta(s, pred)
            então Verifica-Novo-Evento(p, Q, s, pred)
         se suc ≠ NIL e Intersecta(s, suc)
            então Verifica-Novo-Evento(p, Q, s, suc)

Verifica-Novo-Evento(p, Q, s1, s2)
q ← Ponto-de-Interseção(s1, s2)
se q ≻ p e não Pertence(Q, q)
   então Insere(Q, q)
         imprima q
```

Versão simplificada

```
Trata-Evento(p)
  se p é extremo esquerdo de um segmento s
     então Insere(T, s)
           pred ← Predecessor(T, s)
           suc ← Sucessor(T, s)
           se pred ≠ NIL e Intersecta(s, pred)
              então Verifica-Novo-Evento(p, Q, s, pred)
           se suc ≠ NIL e Intersecta(s, suc)
              então Verifica-Novo-Evento(p, Q, s, suc)
  se p é extremo direito de um segmento s
     então Remove(T, s)
           pred ← Predecessor(T, s)
           suc ← Sucessor(T, s)
           se pred e suc ≠ NIL e Intersecta(pred, suc)
              então Verifica-Novo-Evento(p, Q, suc, pred)
```

Versão simplificada

```
Trata-Evento(p)
. . .
se p é ponto de interseção
   então sejam s e s′ os segmentos em T que contém p
         pred ← Predecessor(T, s)
         suc  ← Sucessor(T, s′)
         Remove(T, s)   Remove(T, s′)
         ▷ insere s e s′ na ordem inversa
         Insere(T, s′)     Insere(T, s)
         se pred ≠ NIL e Intersecta(pred, s′)
            então Verifica-Novo-Evento(p, Q, pred, s′)
         se suc ≠ NIL e Intersecta(s, suc)
            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 $\mathrm{O}(n)$ segmentos.

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

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 $\mathrm{O}(n)$ segmentos.

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

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 $\mathrm{O}(n + i) = \mathrm{O}(n^2)$ pontos.

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

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 $\mathrm{O}(n)$ segmentos.

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

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 $\mathrm{O}(n + i) = \mathrm{O}(n^2)$ pontos.

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

As demais operações efetuadas em uma iteração consomem tempo $\mathrm{O}(1)$ (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 $\mathrm{O}(n)$ segmentos.

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

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 $\mathrm{O}(n + i) = \mathrm{O}(n^2)$ pontos.

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

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

Versão completa

O que fazer com os casos que excluímos?

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Para dois pontos eventos p e q,
  $p \prec q$ se $p_x < q_x$ ou $(p_x = q_x$ e $p_y < q_y)$.
- 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.

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Para dois pontos eventos p e q,
  $p \prec q$ se $p_x < q_x$ ou $(p_x = q_x$ e $p_y < q_y)$.
- 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 excluímos?

Alterações:

- Para dois pontos eventos p e q,
  $p \prec q$ se $p_x < q_x$ ou $(p_x = q_x$ e $p_y < q_y)$.
- 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).

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

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Para dois pontos eventos p e q,
  $p \prec q$ se $p_x < q_x$ ou $(p_x = q_x$ e $p_y < q_y)$.
- 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).

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: $\mathrm{O}((n+i)\lg n)$
(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: $\mathrm{O}((n+i)\lg n)$
(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: $\mathrm{O}(n)$ para T e $\mathrm{O}(n+i)$ 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: $\mathrm{O}((n+i)\lg n)$
(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: $\mathrm{O}(n)$ para T e $\mathrm{O}(n+i)$ para Q.

Melhora: Guarde em Q apenas
os pontos de interseção de segmentos que estão consecutivos em T.
Espaço cai para $\mathrm{O}(n)$.

Comentários finais

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

Consumo de tempo: $\mathrm{O}((n + i) \lg n)$
(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: $\mathrm{O}(n)$ para T e $\mathrm{O}(n + i)$ para Q.

Melhora: Guarde em Q apenas
os pontos de interseção de segmentos que estão consecutivos em T.
Espaço cai para $\mathrm{O}(n)$.

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