Intersecção 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
- skip list (tempo esperado $O(n \lg n)$)

Intersecção de segmentos

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

Intersecção 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
- skip list (tempo esperado $O(n \lg n)$)

Pré-processamento: ordene os extremos dos segmentos por x-coordenada (em caso de empate, por y-coordenada)

Se houver extremos repetidos, há intersecção

Intersecção de segmentos

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

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

Extremos

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.

Extremos

Extremo esquerdo de um segmento:

Detecção de intersecção

Detecta-Intersecção(n, S)

```
 1  E ← Extremos-Ordenados(n, S)
 2  T ← ∅        ▷ ABBB ou skip list
 3  para cada p ∈ E faça
 4     s ← segmento(p)
 5     pred ← Predecessor(T, s)      suc ← Sucessor(T, s)
 6     se p é extremo esquerdo de s
 7        então Insere(T, s)
 8              se (pred ≠ NIL e Intersect(s, pred))
                ou (suc ≠ NIL e Intersect(s, suc))
 9                 então devolva VERDADE
10        senão Remove(T, s)
11              se pred e suc ≠ NIL e Intersect(pred, suc)
12                 então devolva VERDADE
13  devolva FALSO
```


Extremos

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

Intersecção de segmentos

Problema: Dados n segmentos, determinar todas as intersecções entre eles.

Objetivo: não mais um algoritmo $O(n \lg n)$, mas um algoritmo $O((n+i) \lg n)$, onde i é o número de intersecções

Idéia: além dos extremos dos segmentos, incluir em E as intersecções a medida que elas vão sendo descobertas

Intersecção de segmentos

Problema: Dados n segmentos, determinar todas as intersecções entre eles.

Intersecção de segmentos

Problema: Dados n segmentos, determinar todas as intersecções entre eles.

Objetivo: não mais um algoritmo $O(n \lg n)$, mas um algoritmo $O((n+i) \lg n)$, onde i é o número de intersecções

Idéia: além dos extremos dos segmentos, incluir em E as intersecções a medida que elas vão sendo descobertas

Estruturas de dados:

- ABBB ou skip list, tanto para T como para E

Intersecção de segmentos

Problema: Dados n segmentos, determinar todas as intersecções entre eles.

Objetivo: não mais um algoritmo $O(n \lg n)$, mas um algoritmo $O((n+i) \lg n)$, onde i é o número de intersecções

Versão simplificada

Hipótese simplificadora: apenas intersecções de dois segmentos, e em pontos interiores dos segmentos.

Acha-Intersecções(n, S)
1 $Q \leftarrow$ Extremos(n, S) ▷ inicializa a ABBB Q com os extremos
2 $T \leftarrow \emptyset$
3 **enquanto não** Vazia(Q) **faça**
4 $p \leftarrow$ 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 $p_y < q_y$)

Intersecção de segmentos

Problema: Dados n segmentos, determinar todas as intersecções entre eles.

Objetivo: não mais um algoritmo $O(n \lg n)$, mas um algoritmo $O((n + i) \lg n)$, onde i é o número de intersecções

Idéia: além dos extremos dos segmentos, incluir em E as intersecções a medida que elas vão sendo descobertas

Estruturas de dados:

- ABBB ou skip list, tanto para T como para E

Hipótese simplificadora: apenas intersecções de dois segmentos, e em pontos interiores dos segmentos.

Versão simplificada

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

Versão simplificada

Hipótese simplificadora: apenas intersecções de dois segmentos, e em pontos interiores dos segmentos.

Acha-Intersecções(n, S)
1 $Q \leftarrow$ Extremos(n, S) ▷ inicializa a ABBB 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

```
Trata-Evento(p)
...
15  se p é ponto de intersecção
16    então sejam s e s' os segmentos em T que contém p
17          pred ← Predecessor(T, s)
18          suc ← Sucessor(T, s')
19          Remove(T, s)   Remove(T, s')
            ▷ insere s e s' na ordem inversa
20          Insere(T, s')   Insere(T, s)
21          se pred ≠ NIL e Intersect(pred, s')
22            então Verifica-Novo-Evento(p, Q, pred, s')
23          se suc ≠ NIL e Intersect(s, suc)
24            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 Insere(T, s)
 3          pred ← Predecessor(T, s)
 4          suc ← Sucessor(T, s)
 5          se pred ≠ NIL e Intersect(s, pred)
 6            então Verifica-Novo-Evento(p, Q, s, pred)
 7          se suc ≠ NIL e Intersect(s, suc)
 8            então Verifica-Novo-Evento(p, Q, s, suc)

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


Versão completa

O que fazer com os casos que excluímos?

Versão simplificada

```
Trata-Evento(p)
 1  se p é extremo esquerdo de um segmento s
 2    então Insere(T, s)
 3          pred ← Predecessor(T, s)
 4          suc ← Sucessor(T, s)
 5          se pred ≠ NIL e Intersect(s, pred)
 6            então Verifica-Novo-Evento(p, Q, s, pred)
 7          se suc ≠ NIL e Intersect(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 ← Predecessor(T, s)
12          suc ← Sucessor(T, s)
13          se pred e suc ≠ NIL e Intersect(pred, suc)
14            então Verifica-Novo-Evento(p, Q, suc, pred)
```

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Q conterá os pontos-evento, sem repetições
- Ponto-evento extremo: tem a lista dos segmentos que têm esse ponto como extremo

Ao processar um ponto-evento,
determina-se todos os segmentos em T que o contém
(eles estão todos consecutivos em T)

Verifica-se se este ponto-evento é uma intersecção,
imprimindo-o, se for o caso

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Q conterá os pontos-evento, sem repetições
- Ponto-evento extremo: tem a lista dos segmentos que têm esse ponto como extremo

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Q conterá os pontos-evento, sem repetições
- Ponto-evento extremo: tem a lista dos segmentos que têm esse ponto como extremo

Ao processar um ponto-evento,
determina-se todos os segmentos em T que o contém
(eles estão todos consecutivos em T)

Verifica-se se este ponto-evento é uma intersecção,
imprimindo-o, se for o caso

Atualiza-se T

Versão completa

O que fazer com os casos que excluímos?

Alterações:

- Q conterá os pontos-evento, sem repetições
- Ponto-evento extremo: tem a lista dos segmentos que têm esse ponto como extremo

Ao processar um ponto-evento,
determina-se todos os segmentos em T que o contém
(eles estão todos consecutivos em T)

Atualização de T

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

- extremos esquerdos causam inclusões em T
- extremos direitos causam remoções

Se o ponto-evento é uma intersecção

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

Se o ponto-evento é uma intersecção

- 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

Comentários finais

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

Consumo de tempo: $O((n+i)\lg n)$
(esperado, no caso de uso de skip lists)
onde i agora é o número de segmentos impressos junto com as intersecções.

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 intersecção, a lista de segmentos que o contém.

Comentários finais

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

Consumo de tempo: $O((n+i)\lg n)$
(esperado, no caso de uso de skip lists)
onde i agora é o número de segmentos impressos junto com as intersecções.

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 intersecção, a lista de segmentos que o contém.

Consumo de tempo: $O((n+i)\lg n)$
(esperado, no caso de uso de skip lists)
onde i agora é o número de segmentos impressos junto com as intersecções.

Comentários finais

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

Consumo de tempo: $O((n+i)\lg n)$
(esperado, no caso de uso de skip lists)
onde i agora é o número de segmentos impressos junto com as intersecções.

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

Melhora: Guarde em Q apenas os pontos de intersecção de segmentos que estão consecutivos em T.
Espaço cai para $O(n)$.

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