

Geometria Computacional

Cristina G. Fernandes

Departamento de Ciência da Computação do IME-USP

<http://www.ime.usp.br/~cris/>

segundo semestre de 2018

Triangulação de Delaunay

P : conjunto de pontos no plano

$\text{Vor}(P)$: diagrama de Voronoi de P

$\mathcal{V}(v)$: célula do ponto v do diagrama de Voronoi de P

Triangulação de Delaunay

P : conjunto de pontos no plano

$\text{Vor}(P)$: diagrama de Voronoi de P

$\mathcal{V}(v)$: célula do ponto v do diagrama de Voronoi de P

Grafo de Delaunay $DG(P)$

- P é o conjunto de vértices de $DG(P)$;
- uv é aresta de $DG(P)$ se $\mathcal{V}(u)$ e $\mathcal{V}(v)$ compartilham aresta de $\text{Vor}(P)$.

Triangulação de Delaunay

P : conjunto de pontos no plano

$\text{Vor}(P)$: diagrama de Voronoi de P

$\mathcal{V}(v)$: célula do ponto v do diagrama de Voronoi de P

Grafo de Delaunay $DG(P)$

- P é o conjunto de vértices de $DG(P)$;
- uv é aresta de $DG(P)$ se $\mathcal{V}(u)$ e $\mathcal{V}(v)$ compartilham aresta de $\text{Vor}(P)$.

Qualquer triangulação de $DG(P)$ é chamada de **triangulação de Delaunay**.

Se P está em posição geral, então $DG(P)$ já é uma triangulação de P : a sua única **triangulação de Delaunay**.

Triangulação de Delaunay

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

Propriedades do grafo de Delaunay:

- (a) p_i, p_j, p_k estão na mesma face de $DG(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;
- (b) $p_i p_j$ é aresta de $DG(P)$ sse existe disco que contém p_i e p_j em sua fronteira e mais nenhum ponto de P .

Triangulação de Delaunay

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

Propriedades do grafo de Delaunay:

- (a) p_i, p_j, p_k estão na mesma face de $DG(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;
- (b) $p_i p_j$ é aresta de $DG(P)$ sse existe disco que contém p_i e p_j em sua fronteira e mais nenhum ponto de P .

Como projetar um algoritmo para obter uma triangulação de Delaunay de P ?

Triangulação de Delaunay

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

Propriedades do grafo de Delaunay:

- (a) p_i, p_j, p_k estão na mesma face de $DG(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;
- (b) $p_i p_j$ é aresta de $DG(P)$ sse existe disco que contém p_i e p_j em sua fronteira e mais nenhum ponto de P .

Como projetar um algoritmo para obter uma triangulação de Delaunay de P ?

Observação:

De $DG(P)$, podemos obter $\text{Vor}(P)$ em tempo linear em n .

Algoritmo quadrático

P : conjunto de pontos no plano

$DT(P)$: triangulação de Delaunay de P

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Algoritmo quadrático

P : conjunto de pontos no plano

$DT(P)$: triangulação de Delaunay de P

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Suponha que já construímos uma parte de $DT(P)$.

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e .

Algoritmo quadrático

P : conjunto de pontos no plano

$DT(P)$: triangulação de Delaunay de P

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Suponha que já construímos uma parte de $DT(P)$.

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e .

Como?

Como encontrar o outro triângulo?

Para cada aresta na borda da parte construída de $DT(P)$, encontra-se o outro triângulo que contém a aresta.

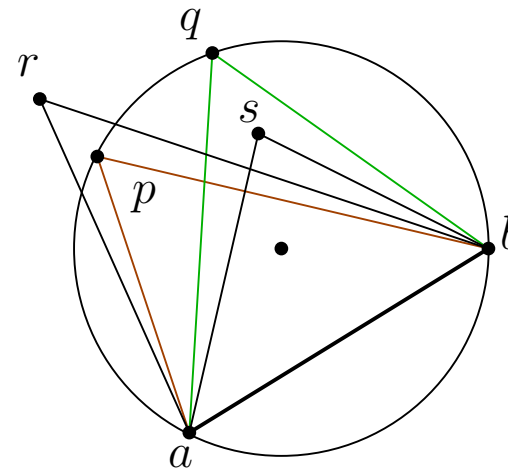
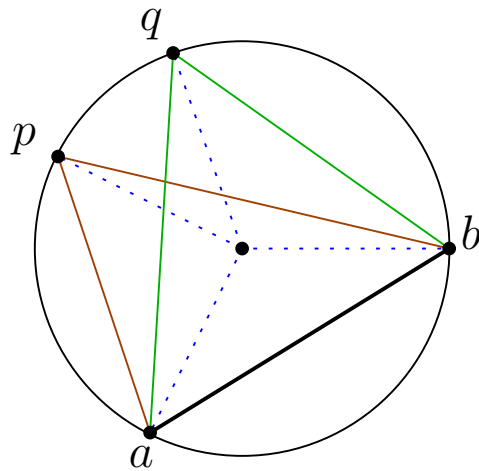
Como?

Como encontrar o outro triângulo?

Para cada aresta na borda da parte construída de $DT(P)$, encontra-se o outro triângulo que contém a aresta.

Como?

(a) p_i, p_j, p_k estão na mesma face de $DT(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;

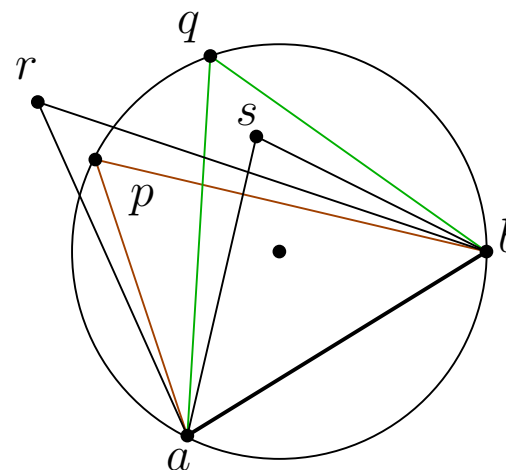
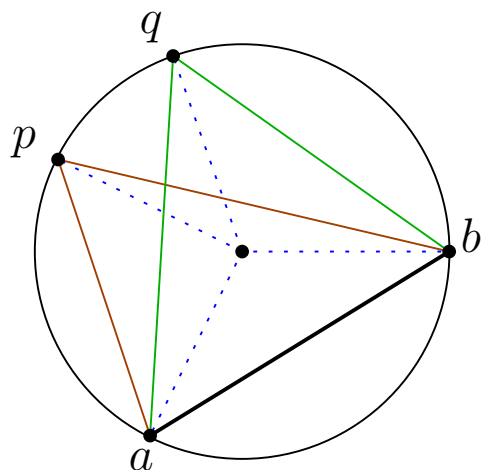


Como encontrar o outro triângulo?

Para cada aresta na borda da parte construída de $DT(P)$, encontra-se o outro triângulo que contém a aresta.

Como?

(a) p_i, p_j, p_k estão na mesma face de $DT(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;



ache p em P do lado certo da aresta
que forma ângulo máximo com a aresta.

Algoritmo quadrático

P : conjunto de pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Suponha que já construímos uma parte de $DT(P)$.

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e assim:

ache p em P do lado certo de e
que forma ângulo máximo com e .

Algoritmo quadrático

P : conjunto de pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Suponha que já construímos uma parte de $DT(P)$.

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e assim:

ache p em P do lado certo de e
que forma ângulo máximo com e .

Quanto tempo consome este algoritmo?

Algoritmo quadrático

P : conjunto de n pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Algoritmo quadrático

P : conjunto de n pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Inicialização: encontre aresta do fecho convexo de P .

Consumo de tempo: $\Theta(n)$

Algoritmo quadrático

P : conjunto de n pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Inicialização: encontre aresta do fecho convexo de P .

Consumo de tempo: $\Theta(n)$

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e assim:

ache p em P do lado certo de e
que forma ângulo máximo com e .

Algoritmo quadrático

P : conjunto de n pontos no plano

Ideia do algoritmo: construir $DT(P)$ iterativamente, encontrando mais um triângulo de cada vez.

Inicialização: encontre aresta do fecho convexo de P .

Consumo de tempo: $\Theta(n)$

Para cada aresta e na borda da parte construída, encontra-se o outro triângulo que contém e assim:

ache p em P do lado certo de e
que forma ângulo máximo com e .

Consumo de tempo: $\Theta(n^2)$

Cada aresta é processada uma vez, e são $\Theta(n)$ arestas.

Para cada aresta, verifica cada ponto de P .

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

legalize recursivamente eventuais arestas ilegais
até obter $DT(P_i)$.

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

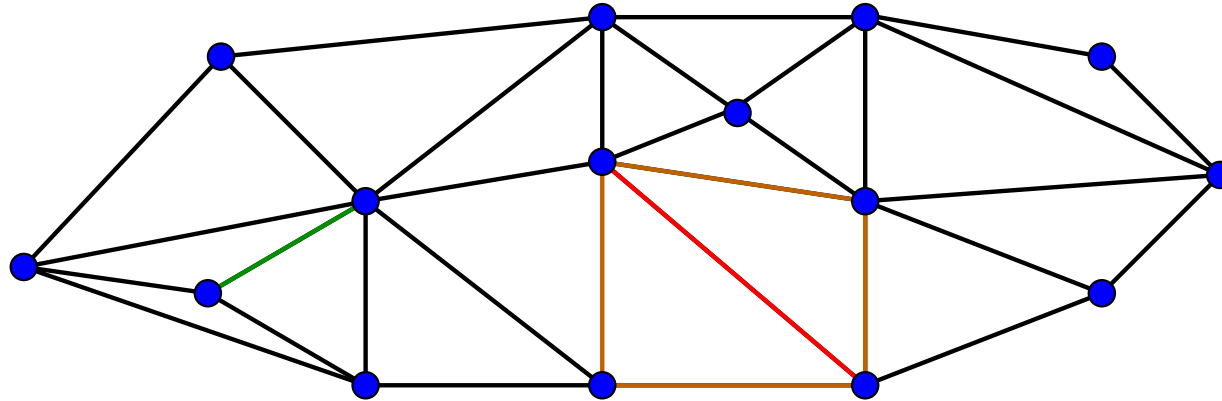
inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

legalize recursivamente eventuais arestas ilegais
até obter $DT(P_i)$.

Dois slides de recordação...

Triangulação legal

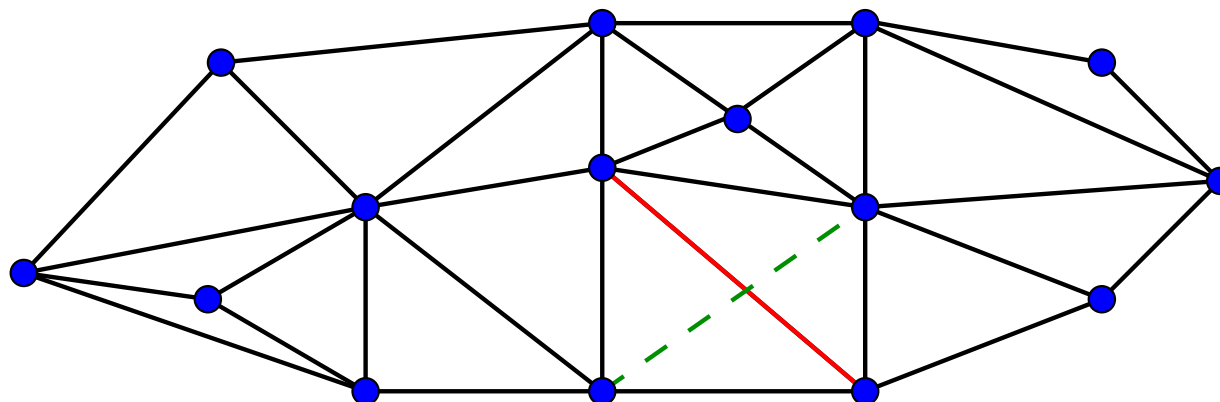
T : triangulação da coleção P de pontos do plano.



e : aresta interna de T cujos triângulos de T que a compartilham formam um **quadrilátero convexo** (a aresta verde não satisfaz esta condição)

Triangulação legal

T : triangulação da coleção P de pontos do plano.



e : aresta interna de T cujos triângulos de T que a compartilham formam um **quadrilátero convexo**

f : outra diagonal do **quadrilátero** de e

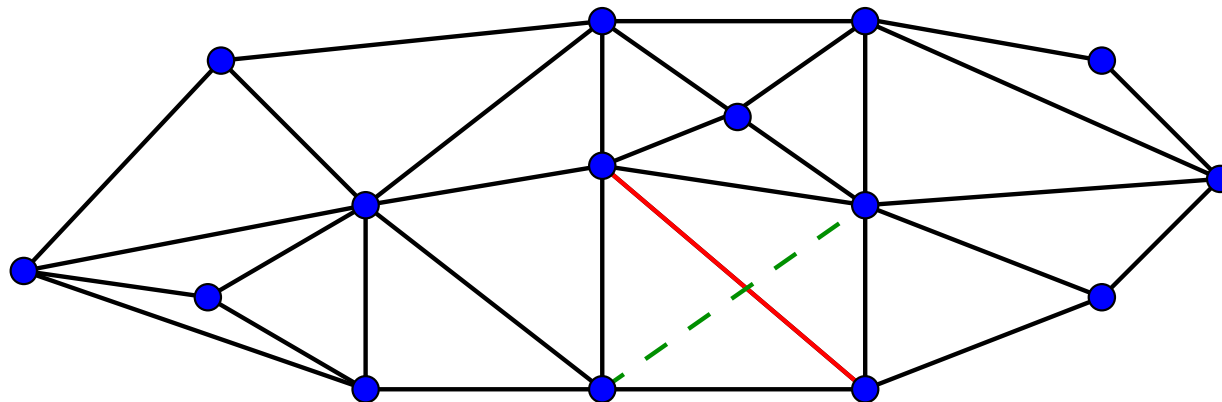
$\{\alpha_1, \dots, \alpha_6\}$: ângulos dos Δ s de e

$\{\beta_1, \dots, \beta_6\}$: ângulos dos Δ s de f

e é **ilegal** se $\min \alpha_i < \min \beta_j$

Triangulação legal

T : triangulação da coleção P de pontos do plano.



e : aresta interna de T cujos triângulos de T que a compartilham formam um **quadrilátero convexo**

f : outra diagonal do **quadrilátero** de e

$\{\alpha_1, \dots, \alpha_6\}$: ângulos dos Δ s de e

$\{\beta_1, \dots, \beta_6\}$: ângulos dos Δ s de f

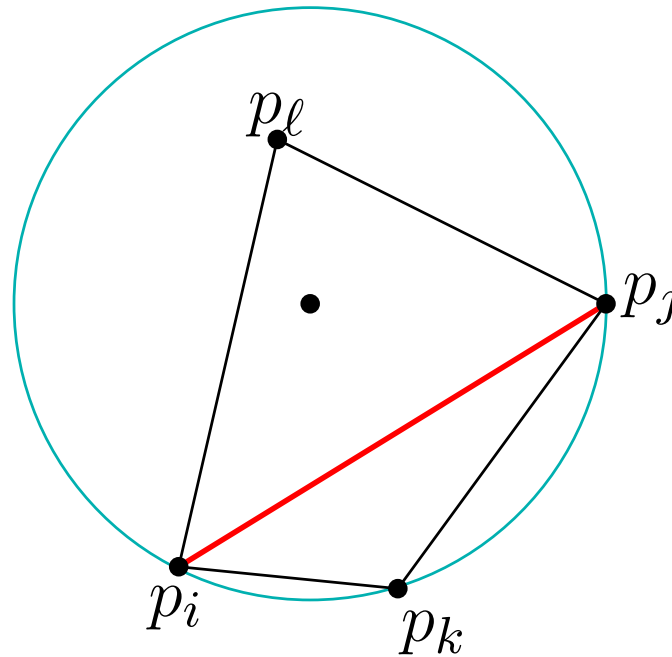
e é **ilegal** se $\min \alpha_i < \min \beta_j$

T é **legal** se não tem arestas ilegais

Aresta ilegal

Aresta interna $e = p_i p_j$ e

p_k e p_ℓ pontas dos triângulos que compartilham e .



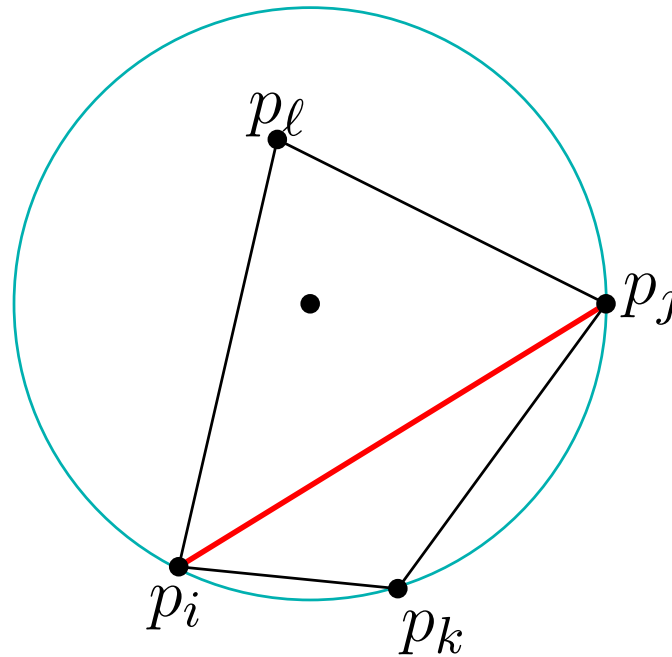
e é ilegal sse

p_ℓ está no interior do círculo determinado por $p_i p_j p_k$.

Aresta ilegal

Aresta interna $e = p_i p_j$ e

p_k e p_ℓ pontas dos triângulos que compartilham e .



e é ilegal sse

p_ℓ está no interior do círculo determinado por $p_i p_j p_k$.

Teorema. Uma triangulação de P é legal se e somente se é uma triangulação de Delaunay de P .

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

legalize recursivamente eventuais arestas ilegais
até obter $DT(P_i)$.

Algoritmo incremental

$P = \{p_1, \dots, p_n\}$: conjunto de pontos no plano

O algoritmo constrói $DT(P_i)$ para $i = 1, \dots, n$,
onde $P_i = \{p_1, \dots, p_i\}$.

A cada iteração,

ache o triângulo onde p_i está em $DT(P_{i-1})$.

inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

legalize recursivamente eventuais arestas ilegais
até obter $DT(P_i)$.

Qual o consumo de tempo?

Como implementar cada passo?

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluimos p_i , quantas arestas são legalizadas?

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluimos p_i , quantas arestas são legalizadas?

Resposta: $k - 3$, onde k é o grau de p_i em $DT(P_i)$.

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluímos p_i , quantas arestas são legalizadas?

Resposta: $k - 3$, onde k é o grau de p_i em $DT(P_i)$.

- Como localizar o triângulo de $DT(P_{i-1})$ em que p_i cai?

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluimos p_i , quantas arestas são legalizadas?

Resposta: $k - 3$, onde k é o grau de p_i em $DT(P_i)$.

- Como localizar o triângulo de $DT(P_{i-1})$ em que p_i cai?

Resposta: usa-se um DAG descrito a frente.

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluimos p_i , quantas arestas são legalizadas?
Resposta: $k - 3$, onde k é o grau de p_i em $DT(P_i)$.
- Como localizar o triângulo de $DT(P_{i-1})$ em que p_i cai?
Resposta: usa-se um DAG descrito a frente.
- Como inicializar o processo?

Algoritmo incremental

Pontos a serem discutidos:

- Quando incluimos p_i , quantas arestas são legalizadas?

Resposta: $k - 3$, onde k é o grau de p_i em $DT(P_i)$.

- Como localizar o triângulo de $DT(P_{i-1})$ em que p_i cai?

Resposta: usa-se um DAG descrito a frente.

- Como inicializar o processo?

Resposta: acrescentamos pontos fictícios p_{-3} , p_{-2} e p_{-1} que determinam um triângulo que contém todos os pontos de P e que estão distantes o suficiente para não influenciarem em $DT(P)$.

Estrutura de dados

Objetivo: determinação do triângulo que contém um ponto.

Estrutura de dados

Objetivo: determinação do triângulo que contém um ponto.

Usa-se um **DAG** com um nó para cada triângulo de alguma das triangulações.

Folhas do DAG: triângulos de $DT(P_i)$.

Nós internos: triângulos que foram destruídos.

Estrutura de dados

Objetivo: determinação do triângulo que contém um ponto.

Usa-se um **DAG** com um nó para cada triângulo de alguma das triangulações.

Folhas do DAG: triângulos de $DT(P_i)$.

Nós internos: triângulos que foram destruídos.

Quando p_i é inserido, uma folha vira nó interno, pai de três novas folhas.

Estrutura de dados

Objetivo: determinação do triângulo que contém um ponto.

Usa-se um **DAG** com um nó para cada triângulo de alguma das triangulações.

Folhas do DAG: triângulos de $DT(P_i)$.

Nós internos: triângulos que foram destruídos.

Quando p_i é inserido,
uma folha vira nó interno, pai de três novas folhas.

Quando uma aresta é legalizada,
dois triângulos viram nós internos,
e viram pais de duas novas folhas.

Estrutura de dados

Objetivo: determinação do triângulo que contém um ponto.

Usa-se um **DAG** com um nó para cada triângulo de alguma das triangulações.

Folhas do DAG: triângulos de $DT(P_i)$.

Nós internos: triângulos que foram destruídos.

Quando p_i é inserido,
uma folha vira nó interno, pai de três novas folhas.

Quando uma aresta é legalizada,
dois triângulos viram nós internos,
e viram pais de duas novas folhas.

A busca é natural nesta estrutura.

Consumo de tempo e espaço

Para maior eficiência, o algoritmo é aleatorizado:
seu primeiro passo é embaralhar P .

Consumo de tempo e espaço

Para maior eficiência, o algoritmo é aleatorizado: seu primeiro passo é embaralhar P .

Lembre-se que $DT(P_i)$ é um grafo planar com $i + 3$ vértices e $3(i + 3) - 6$ arestas.

O grau esperado de p_i é portanto $2(3(i + 3) - 6 - 3)/i = 6$.

Ou seja, o número esperado de arestas legalizadas em cada iteração é 3.

Consumo de tempo e espaço

Para maior eficiência, **o algoritmo é aleatorizado**: seu primeiro passo é embaralhar P .

Lembre-se que $DT(P_i)$ é um grafo planar com $i + 3$ vértices e $3(i + 3) - 6$ arestas.

O grau esperado de p_i é portanto $2(3(i + 3) - 6 - 3)/i = 6$.

Ou seja, o número esperado de arestas legalizadas em cada iteração é 3.

Uma análise semelhante mostra que o número esperado de triângulos criados no total é $9n + 1$.

Com isso, o tamanho esperado do DAG é $O(n)$.

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo esperado de espaço: $O(n)$

Consumo esperado de tempo sem a busca no DAG: $O(n)$

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo esperado de espaço: $O(n)$

Consumo esperado de tempo sem a busca no DAG: $O(n)$

Quanto tempo se leva para buscar p_i no DAG?

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo esperado de espaço: $O(n)$

Consumo esperado de tempo sem a busca no DAG: $O(n)$

Quanto tempo se leva para buscar p_i no DAG?

O percurso de p_i no DAG passa por todos os triângulos que existiram contendo p_i .

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo esperado de espaço: $O(n)$

Consumo esperado de tempo sem a busca no DAG: $O(n)$

Quanto tempo se leva para buscar p_i no DAG?

O percurso de p_i no DAG passa por todos os triângulos que existiram contendo p_i .

Somando-se por triângulo Δ , se $K(\Delta)$ é o número de pontos de P no interior de Δ , o custo total dos percursos é $O(n + \sum_{\Delta} |K(\Delta)|)$.

Consumo de tempo e espaço

Descontado o tempo gasto na busca feita no DAG, temos:

Consumo esperado de espaço: $O(n)$

Consumo esperado de tempo sem a busca no DAG: $O(n)$

Quanto tempo se leva para buscar p_i no DAG?

O percurso de p_i no DAG passa por todos os triângulos que existiram contendo p_i .

Somando-se por triângulo Δ , se $K(\Delta)$ é o número de pontos de P no interior de Δ , o custo total dos percursos é $O(n + \sum_{\Delta} |K(\Delta)|)$.

É possível mostrar que $E[\sum_{\Delta} |K(\Delta)|] = O(n \lg n)$, logo

Consumo esperado de tempo do algoritmo: $O(n \lg n)$

Comentários finais

Há um caso a mais na inserção de p_i ,
quando p_i cai sobre uma aresta de $DT(P_{i-1})$.

Comentários finais

Há um caso a mais na inserção de p_i ,
quando p_i cai sobre uma aresta de $DT(P_{i-1})$.

Legalização de arestas incidentes aos pontos fictícios
é tratada de modo a não influenciar em $DT(P)$.

Comentários finais

Há um caso a mais na inserção de p_i ,
quando p_i cai sobre uma aresta de $DT(P_{i-1})$.

Legalização de arestas incidentes aos pontos fictícios
é tratada de modo a não influenciar em $DT(P)$.

Começamos com a triangulação $p_{-3}p_{-2}p_{-1}$ e,
ao final, removemos estes três pontos de $DT(P)$.

Comentários finais

Há um caso a mais na inserção de p_i ,
quando p_i cai sobre uma aresta de $DT(P_{i-1})$.

Legalização de arestas incidentes aos pontos fictícios
é tratada de modo a não influenciar em $DT(P)$.

Começamos com a triangulação $p_{-3}p_{-2}p_{-1}$ e,
ao final, removemos estes três pontos de $DT(P)$.

A prova de que $E[\sum_{\Delta} |K(\Delta)|] = O(n \lg n)$ está feita no livro
dos holandeses para o caso em que P está em posição
geral (ou seja, em que $DT(P)$ é uma triangulação).