Aula 12

Algoritmo para triangulação de Delaunay

Secs 9.3 e 9.4 do livro de de Berg e outros

Triangulação de Delaunay

P: conjunto de pontos no plano

$\mathrm{Vor}(P)$: 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

$\mathrm{Vor}(P)$: 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 $\mathrm{Vor}(P)$.

Triangulação de Delaunay

P: conjunto de pontos no plano

$\mathrm{Vor}(P)$: 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 $\mathrm{Vor}(P)$.

Triangulação de Delaunay é qualquer triangulação de $DG(P)$.

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

Triangulação de Delaunay

$P = \{p_1, \ldots, p_n\}$: 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

$P = \{p_1, \ldots, p_n\}$: 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

$P = \{p_1, \ldots, p_n\}$: 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 $\mathrm{Vor}(P)$ em tempo linear em n.

Primeiro algoritmo

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.

Primeiro algoritmo

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.

Primeiro algoritmo

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.

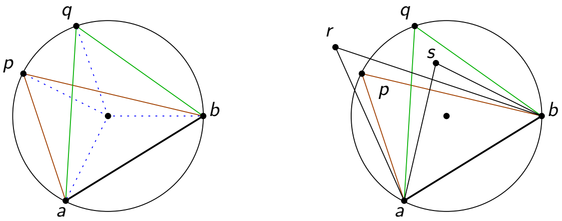
Como?

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 $DG(P)$ sse existe círculo que passa pelos três e não contém nenhum ponto de P em seu interior;

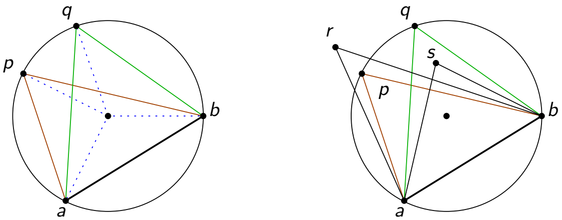


Como encontrar o outro triângulo?

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 $DG(P)$ 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.

Primeiro algoritmo

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.

Primeiro algoritmo

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?

Consumo de tempo

P: conjunto de n pontos no plano

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

Consumo de tempo

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: $\Theta(n)$

Consumo de tempo

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

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.

Segundo algoritmo: incremental

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

Segundo algoritmo: incremental

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

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

Segundo algoritmo: incremental

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

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

A cada iteração,

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

Segundo algoritmo: incremental

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

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

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.

Segundo algoritmo: incremental

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

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

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 $DT(P_i)$.

Segundo algoritmo: incremental

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

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

A cada iteração,

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

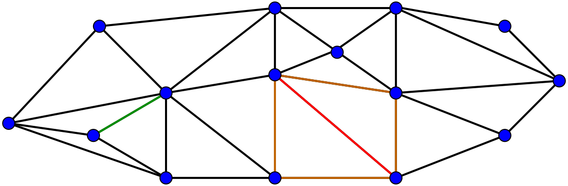
inclua p_i e três novas arestas
entre p_i e os vértices deste triângulo.

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

Dois slides de recordação...

Triangulação legal

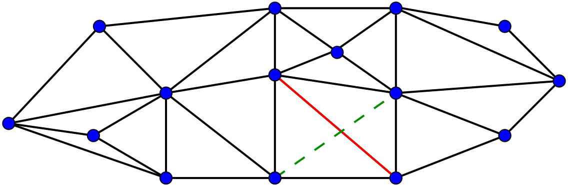
T: triangulação da coleção P de pontos do plano.



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

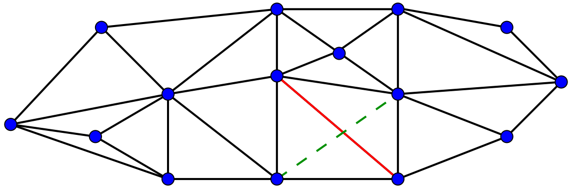
f: outra diagonal do quadrilátero de e

$\{\alpha_1, \dots, \alpha_6\}$: ângulos dos Δs de e
$\{\beta_1, \dots, \beta_6\}$: ângulos dos Δs 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

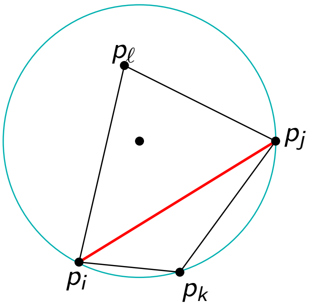
$\{\alpha_1, \ldots, \alpha_6\}$: ângulos dos Δs de e
$\{\beta_1, \ldots, \beta_6\}$: ângulos dos Δs de f

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 $p_i p_j p_k$.

Algoritmo incremental

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

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

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

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

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

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 incluímos p_i, quantas arestas são legalizadas?

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 $DT(P_i)$.

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 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?

  Resposta: usa-se um DAG descrito à frente.

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?

  Resposta: usa-se um DAG descrito à frente.

- Como inicializar o processo?

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?

  Resposta: usa-se um DAG descrito à 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.

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.

Vamos ver uma animação do algoritmo...

Consumo de tempo

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 $2(3(i + 3) - 6 - 3)/i = 6$.
(O -3 é pelas arestas do triângulo $p_{-1}p_{-2}p_{-3}$.)

Ou seja, o número esperado de arestas legalizadas
em cada iteração é 3 (o que resulta em grau 6).

Consumo de tempo

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 $2(3(i + 3) - 6 - 3)/i = 6$.
(O -3 é pelas arestas do triângulo $p_{-1}p_{-2}p_{-3}$.)

Ou seja, o número esperado de arestas legalizadas
em cada iteração é 3 (o que resulta em grau 6).

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

Consumo esperado de tempo sem a busca no DAG: $\mathrm{O}(n)$

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Ao inserir mais um ponto p, são criados três triângulos novos.

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Ao inserir mais um ponto p, são criados três triângulos novos.

Se o grau final de p é k,
foram legalizadas $k - 3$ arestas na sua inserção.

Cada aresta legalizada cria mais dois triângulos,
num total de $3 + 2(k - 3) = 2k - 3$ triângulos novos por inserção.

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Ao inserir mais um ponto p, são criados três triângulos novos.

Se o grau final de p é k,
foram legalizadas $k - 3$ arestas na sua inserção.

Cada aresta legalizada cria mais dois triângulos,
num total de $3 + 2(k - 3) = 2k - 3$ triângulos novos por inserção.

Como o grau esperado de p é 6,
o número esperado de triângulos novos por inserção é $2.6 - 3 = 9$,
para um total esperado de $9n - 1$ triângulos criados
(incluindo o triângulo $p_{-1}p_{-2}p_{-3}$).

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Ao inserir mais um ponto p, são criados três triângulos novos.

Se o grau final de p é k,
foram legalizadas $k - 3$ arestas na sua inserção.

Cada aresta legalizada cria mais dois triângulos,
num total de $3 + 2(k - 3) = 2k - 3$ triângulos novos por inserção.

Como o grau esperado de p é 6,
o número esperado de triângulos novos por inserção é $2.6 - 3 = 9$,
para um total esperado de $9n - 1$ triângulos criados
(incluindo o triângulo $p_{-1}p_{-2}p_{-3}$).

Com isso, o tamanho esperado do DAG é $\mathrm{O}(n)$.

Tamanho esperado do DAG

O número esperado de triângulos criados no total é $9n + 1$.

Ao inserir mais um ponto p, são criados três triângulos novos.

Se o grau final de p é k,
foram legalizadas $k - 3$ arestas na sua inserção.

Cada aresta legalizada cria mais dois triângulos,
num total de $3 + 2(k - 3) = 2k - 3$ triângulos novos por inserção.

Como o grau esperado de p é 6,
o número esperado de triângulos novos por inserção é $2.6 - 3 = 9$,
para um total esperado de $9n - 1$ triângulos criados
(incluindo o triângulo $p_{-1}p_{-2}p_{-3}$).

Com isso, o tamanho esperado do DAG é $\mathrm{O}(n)$.

Consumo esperado de espaço: $\mathrm{O}(n)$

Consumo de tempo

Quanto tempo se leva para buscar p_i no DAG?

Consumo de tempo

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

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.

Cada triângulo t destes corresponde a um triângulo Δ de $DT(P_j)$
para $j < i$ que foi destruído com t e cujo circuncírculo contém p_i.

Consumo de tempo

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.

Cada triângulo t destes corresponde a um triângulo Δ de $DT(P_j)$ para $j < i$ que foi destruído com t e cujo circuncírculo contém p_i.

De fato, ou t já é um triângulo de $DT(P_j)$ para $j < i$,
ou ele foi criado e destruído durante uma inserção.

Consumo de tempo

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.

Cada triângulo t destes corresponde a um triângulo Δ de $DT(P_j)$
para $j < i$ que foi destruído com t e cujo circuncírculo contém p_i.

De fato, ou t já é um triângulo de $DT(P_j)$ para $j < i$,
ou ele foi criado e destruído durante uma inserção.

Para cada triângulo Δ de alguma $DT(P_j)$, seja
$K(\Delta)$ o conjunto de pontos de P no interior do circuncírculo de Δ.

Consumo de tempo

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.

Cada triângulo t destes corresponde a um triângulo Δ de $DT(P_j)$
para $j < i$ que foi destruído com t e cujo circuncírculo contém p_i.

De fato, ou t já é um triângulo de $DT(P_j)$ para $j < i$,
ou ele foi criado e destruído durante uma inserção.

Para cada triângulo Δ de alguma $DT(P_j)$, seja
$K(\Delta)$ o conjunto de pontos de P no interior do circuncírculo de Δ.

O custo total dos percursos é $\mathrm{O}(n + \sum_\Delta |K(\Delta)|)$.

Consumo de tempo

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

Consumo de tempo

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

Veja que

$$\sum_\Delta |K(\Delta)| \;=\; n + \sum_{r=1}^{n} \sum_{\Delta \in DT(P_r) \setminus DT(P_{r-1})} |K(\Delta)|.$$

Consumo de tempo

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

Veja que

$$\sum_{\Delta} |K(\Delta)| \;=\; n + \sum_{r=1}^{n} \sum_{\Delta \in DT(P_r) \setminus DT(P_{r-1})} |K(\Delta)|.$$

Para adquirir intuição sobre isso, veja que

o triângulo Δ de $DT(P_0)$ tem $|K(\Delta)| = n$ e

um triângulo Δ de $DT(P_n)$ tem $|K(\Delta)| = 0$.

Consumo de tempo

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

Veja que

$$\sum_{\Delta} |K(\Delta)| \;=\; n + \sum_{r=1}^{n} \sum_{\Delta \in DT(P_r) \setminus DT(P_{r-1})} |K(\Delta)|.$$

Para adquirir intuição sobre isso, veja que

o triângulo Δ de $DT(P_0)$ tem $|K(\Delta)| = n$ e

um triângulo Δ de $DT(P_n)$ tem $|K(\Delta)| = 0$.

Intuitivamente, por causa da permutação aleatória de P no início, espera-se que um triângulo Δ de $DT(P_r)$ tenha $|K(\Delta)| = \mathrm{O}(n/r)$.

Consumo de tempo

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

Veja que

$$\sum_\Delta |K(\Delta)| \;=\; n + \sum_{r=1}^{n} \sum_{\Delta \in DT(P_r) \setminus DT(P_{r-1})} |K(\Delta)|.$$

Para adquirir intuição sobre isso, veja que

o triângulo Δ de $DT(P_0)$ tem $|K(\Delta)| = n$ e

um triângulo Δ de $DT(P_n)$ tem $|K(\Delta)| = 0$.

Intuitivamente, por causa da permutação aleatória de P no início, espera-se que um triângulo Δ de $DT(P_r)$ tenha $|K(\Delta)| = \mathrm{O}(n/r)$.

Se $|K(\Delta)| \leq n/r$ para $\Delta \in DT(P_r) \setminus DT(P_{r-1})$ com $r \geq 1$,

$$\mathrm{E}[\sum_\Delta |K(\Delta)|] \;\leq\; n + \sum_{r=1}^{n} 6\,\frac{n}{r} \;=\; n + 6n \sum_{r=1}^{n} \frac{1}{r} \;=\; \mathrm{O}(n \lg n).$$

Consumo de tempo: resumo

Consumo esperado de tempo sem a busca no DAG: $\mathrm{O}(n)$

Quanto tempo se leva para buscar p_i no DAG?

O custo total dos percursos no DAG é $\mathrm{O}(n + \sum_\Delta |K(\Delta)|)$.

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

Logo o consumo esperado desta etapa do algoritmo é $\mathrm{O}(n \lg n)$.

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

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