

Árvore geradora mínima

CLRS Cap 23

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

Problema: Dado G conexo com peso $w(e)$ para cada aresta e , encontrar árvore geradora em G de peso mínimo.

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

Problema: Dado G conexo com peso $w(e)$ para cada aresta e , encontrar árvore geradora em G de peso mínimo.

O **peso** de uma árvore é a soma dos pesos de suas arestas.

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.
Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

Problema: Dado G conexo com peso $w(e)$ para cada aresta e ,
encontrar árvore geradora em G de peso mínimo.

O **peso** de uma árvore é a soma dos pesos de suas arestas.

Tal árvore é chamada de **árvore geradora mínima** em G .

MST: minimum spanning tree

Arestas seguras

Seja $G = (V, E)$ um grafo conexo e w uma função que atribui um peso $w(e)$ para cada aresta $e \in E$.

Suponha que $A \subseteq E$ está contido em alguma MST de (G, w) .

Arestas seguras

Seja $G = (V, E)$ um grafo conexo e w uma função que atribui um peso $w(e)$ para cada aresta $e \in E$.

Suponha que $A \subseteq E$ está contido em alguma MST de (G, w) .

Aresta $e \in E$ é *segura* para A se
 $A \cup \{e\}$ está contido em alguma MST de (G, w) .

Obs.: Se A não é uma MST, então existe aresta segura para A .

Arestas seguras

Seja $G = (V, E)$ um grafo conexo e w uma função que atribui um peso $w(e)$ para cada aresta $e \in E$.

Suponha que $A \subseteq E$ está contido em alguma MST de (G, w) .

Aresta $e \in E$ é *segura* para A se
 $A \cup \{e\}$ está contido em alguma MST de (G, w) .

Obs.: Se A não é uma MST, então existe aresta segura para A .

GULOSO-GENÉRICO (G, w)

- 1 $A \leftarrow \emptyset$
- 2 **enquanto** A não é geradora **faça**
- 3 encontre aresta segura e para A
- 4 $A \leftarrow A \cup \{e\}$
- 5 **devolva** A

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se
exatamente um de seus extremos está em S .

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S .

Corte $\text{respeita } A \subseteq E$: nenhuma aresta de A o cruza.

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se
exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) ,
então toda e com $w(e)$ mínimo em corte que respeita A
é segura para A .

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então toda e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A . Considere uma tal e e suponha que não está em T .

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então toda e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A . Considere uma tal e e suponha que não está em T .

Alguma aresta f de T cruza o corte.

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então toda e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A . Considere uma tal e e suponha que não está em T .

Alguma aresta f de T cruza o corte.

Então $T' := T - f + e$ é MST e contém $A \cup \{e\}$. ■

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de **Prim**,
que mantém uma árvore T que contém um vértice s ,
acrescentando em cada iteração uma aresta segura a T .

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de **Prim**, que mantém uma árvore T que contém um vértice s , acrescentando em cada iteração uma aresta segura a T .

O segundo é o algoritmo de **Kruskal** que, em cada iteração, escolhe uma aresta segura mais leve possível.

O algoritmo de Kruskal vai aumentando uma **floresta**.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de **Prim**,
que mantém uma árvore T que contém um vértice s ,
acrescentando em cada iteração uma aresta segura a T .

O segundo é o algoritmo de **Kruskal** que, em cada iteração,
escolhe uma aresta segura mais leve possível.

O algoritmo de Kruskal vai aumentando uma **floresta**.

Os dois produzem uma MST de (G, w) .

$n := |V(G)|$ e $m := |E(G)|$.

Algoritmo de Prim

v .key = peso da aresta mais leve ligando v à árvore corrente

Algoritmo de Prim

$v.key$ = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** $v.key \leftarrow \infty$
- 3 $s.key \leftarrow 0$ $s.\pi \leftarrow \text{nil}$
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v.key$
- 9 **então** $v.\pi \leftarrow u$ $v.key \leftarrow w(uv)$
 ▷ DecreaseKey implícito na alteração do $v.key$
- 10 π descreve a MST

Consumo de tempo das operações na fila de prioridade:

Linha 4: $\Theta(n)$ **EXTRACT-MIN** e **DECREASE-KEY** : $O(\lg n)$

Algoritmo de Prim

$v.key$ = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** $v.key \leftarrow \infty$
- 3 $s.key \leftarrow 0$ $s.\pi \leftarrow \text{nil}$
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v.key$
- 9 **então** $v.\pi \leftarrow u$ $v.key \leftarrow w(uv)$
 ▷ DecreaseKey implícito na alteração do $v.key$
- 10 π descreve a MST

Consumo de tempo do Prim: $O(m \lg n)$

Algoritmo de Prim

$v.key$ = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** $v.key \leftarrow \infty$
- 3 $s.key \leftarrow 0$ $s.\pi \leftarrow \text{nil}$
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v.key$
- 9 **então** $v.\pi \leftarrow u$ $v.key \leftarrow w(uv)$
 ▷ DecreaseKey implícito na alteração do $v.key$
- 10 π descreve a MST

Consumo de tempo do Prim: $O(m \lg n)$

Consumo de tempo com Fibonacci heap: $O(m + n \lg n)$

Algoritmo de Kruskal - a idéia

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Basta verificar se suas pontas estão em componentes distintas de $G[A]$.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Basta verificar se suas pontas estão em componentes distintas de $G[A]$.

Como fazer isso eficientemente?

A estrutura de dados

É preciso manter

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome!**

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome!**

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome!**

Melhor:

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome**!

Melhor:

1. Dado um vértice, devolver o nome do seu bloco.

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome**!

Melhor:

1. Dado um vértice, devolver o nome do seu bloco.
2. Dados os nomes de dois blocos, substituí-los por sua união.

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome!**

Melhor:

1. Dado um vértice, devolver o nome do seu bloco. **FINDSET**
2. Dados os nomes de dois blocos, substituí-los por sua união. **UNION**

A estrutura de dados

É preciso manter

uma partição de V ,

e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união

Hmmm, é preciso um jeito de se referir aos blocos...um **nome**!

Melhor:

1. Dado um vértice, devolver o nome do seu bloco. **FINDSET**
2. Dados os nomes de dois blocos, substituí-los por sua união. **UNION**

O problema **UNION-FIND**

O algoritmo de Kruskal

KRUSKAL (G, w)

- 1 $A \leftarrow \emptyset$
- 2 para cada $v \in V(G)$ faça MAKESET(v)
- 3 para $e \in E$ em ordem crescente de $w(e)$ faça
- 4 sejam u e v as pontas de e
- 5 se FINDSET(u) \neq FINDSET(v)
- 6 então $A \leftarrow A \cup \{e\}$
- 7 UNION(FINDSET(u), FINDSET(v))
- 8 devolva A