Projeto e Análise de Algoritmos II (MC558)
Árvores geradoras mínimas

Prof. Dr. Ruben Interian

Resumo

Resumo

Revisão do conteúdo

Seja $G = (V, E)$ um grafo não direcionado.

- O subgrafo $H = (V', E')$ é um **subgrafo gerador** de G se $V' = V$.
- Uma árvore $T = (V', E')$ é chamada de árvore geradora de G, se essa árvore é um **subgrafo gerador** de G.
- Se G é conexo, G **contém** uma árvore geradora.

Objetivo

- **Árvores geradoras** em grafos ponderados;
- **Árvores geradoras mínimas**.

Resumo

Árvore geradora mínima

Problema 1: Queremos conectar vários computadores por meio de cabos de fibra ótica, de forma a usar menor quantidade possível de metros de cabo.

Árvore geradora mínima

Problema 1: Queremos conectar vários computadores por meio de cabos de fibra ótica, de forma a usar menor quantidade possível de metros de cabo.

- O mesmo problema pode ser formulado para uma rede elétrica, para a rede de tratamento de agua/esgoto, ...

Árvore geradora mínima

Problema 1: Queremos conectar vários computadores por meio de cabos de fibra ótica, de forma a usar menor quantidade possível de metros de cabo.

- O mesmo problema pode ser formulado para uma rede elétrica, para a rede de tratamento de agua/esgoto, . . .

Problema 2: Já temos uma rede de computadores, e queremos encaminhar pacotes de dados, de modo que a perda total de pacotes (e.g., por ruido/falhas) seja a menor.

Árvore geradora mínima

Problema 1: Queremos conectar vários computadores por meio de cabos de fibra ótica, de forma a usar menor quantidade possível de metros de cabo.

- O mesmo problema pode ser formulado para uma rede elétrica, para a rede de tratamento de agua/esgoto, ...

Problema 2: Já temos uma rede de computadores, e queremos encaminhar pacotes de dados, de modo que a perda total de pacotes (e.g., por ruido/falhas) seja a menor.
$\rightarrow$ **Spanning Tree Protocol**

Árvore geradora mínima

Estes problemas podem ser modelados usando **um grafo** (não direcionado), onde os **vértices** representam os computadores (ou domicílios, endereços, cidades), e as **arestas** representam as possíveis conexões entre eles.

Árvore geradora mínima

Estes problemas podem ser modelados usando **um grafo** (não direcionado), onde os **vértices** representam os computadores (ou domicílios, endereços, cidades), e as **arestas** representam as possíveis conexões entre eles.

Veja que o **peso** de uma aresta já não é binário (aresta existe ou não), mas representa o custo daquela conexão.

- Em metros de cabo, quilômetros de tubulação, pacotes perdidos, ...

Árvore geradora mínima

O problema computacional que queremos resolver é **encontrar um subgrafo**:

- gerador (que contém todos os vértices),

Árvore geradora mínima

O problema computacional que queremos resolver é **encontrar um subgrafo**:

- gerador (que contém todos os vértices),
- conexo (para garantir a interligação de todas os computadores),

Árvore geradora mínima

O problema computacional que queremos resolver é **encontrar um subgrafo**:

- gerador (que contém todos os vértices),
- conexo (para garantir a interligação de todas os computadores),
- no qual a soma dos custos das arestas seja a menor possível.

Árvore geradora mínima

O problema computacional que queremos resolver é **encontrar um subgrafo**:

- gerador (que contém todos os vértices),
- conexo (para garantir a interligação de todas os computadores),
- no qual a soma dos custos das arestas seja a menor possível.

Esta é a parte difícil!

Árvore geradora mínima

O problema computacional que queremos resolver é **encontrar um subgrafo**:

- gerador (que contém todos os vértices),
- conexo (para garantir a interligação de todas os computadores),
- no qual a soma dos custos das arestas seja a menor possível.

Esta é a parte difícil!

Veja que o problema só tem solução se o grafo for conexo. Daqui pra frente vamos supor que o grafo de entrada é conexo.

Árvore geradora mínima

O subgrafo gerador de menor custo pode ter **mais de** $n - 1$ **arestas**?

Árvore geradora mínima

O subgrafo gerador de menor custo pode ter **mais de** $n - 1$ **arestas**? – Não.

- O resultado será uma árvore, porém . . .

Árvore geradora mínima

O subgrafo gerador de menor custo pode ter **mais de** $n - 1$ **arestas**? – Não.

- O resultado será uma árvore, porém . . .
- Esse raciocínio funciona apenas quando **os pesos são positivos!**

Árvore geradora mínima

O subgrafo gerador de menor custo pode ter **mais de** $n - 1$ **arestas**? – Não.

- O resultado será uma árvore, porém ...
- Esse raciocínio funciona apenas quando **os pesos são positivos!**
- Veja que diferentes árvores terão pesos diferentes.

Árvore geradora mínima

Resumo: Problema da Árvore Geradora Mínima

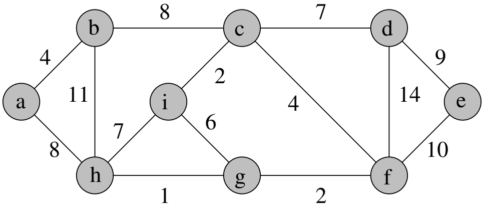
Entrada: grafo conexo $G = (V, E)$, função $\omega(e) > 0$ definida $\forall$ aresta $e = (u, v)$.

Saída: subgrafo gerador conexo T de G cujo peso total

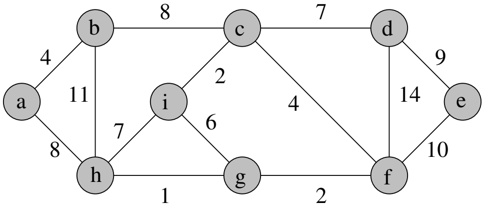
$$\omega(T) = \sum_{(u,v) \in T} \omega(u, v)$$

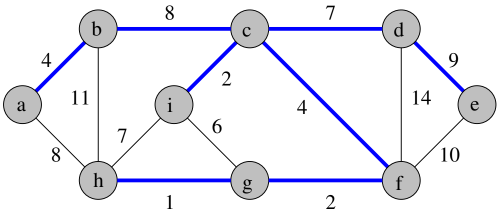
seja o menor possível.

Árvore geradora mínima: exemplo



Árvore geradora mínima: exemplo





Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

- Podemos tentar criar um conjunto A de arestas, e ir adicionando arestas ao conjunto até chegar a todas as arestas de uma árvore geradora mínima.

Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

- Podemos tentar criar um conjunto A de arestas, e ir adicionando arestas ao conjunto até chegar a todas as arestas de uma árvore geradora mínima.
- Essa ideia assume que conseguiremos criar a árvore geradora mínima sem verificar todas as combinações de arestas que podem ser escolhidas.

Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

- Podemos tentar criar um conjunto A de arestas, e ir adicionando arestas ao conjunto até chegar a todas as arestas de uma árvore geradora mínima.
- Essa ideia assume que conseguiremos criar a árvore geradora mínima sem verificar todas as combinações de arestas que podem ser escolhidas.
- Essa estratégia **gulosa** nem sempre funciona para todos os problemas. **Exemplo**: problema da mochila.

Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

- Podemos tentar criar um conjunto A de arestas, e ir adicionando arestas ao conjunto até chegar a todas as arestas de uma árvore geradora mínima.
- Essa ideia assume que conseguiremos criar a árvore geradora mínima sem verificar todas as combinações de arestas que podem ser escolhidas.
- Essa estratégia **gulosa** nem sempre funciona para todos os problemas. **Exemplo**: problema da mochila.
- Para que essa estratégia funcione, precisamos que a todo momento, A esteja contido em uma AGM.

Árvore geradora mínima

Como podemos construir uma árvore geradora mínima (AGM)?

- Podemos tentar criar um conjunto A de arestas, e ir adicionando arestas ao conjunto até chegar a todas as arestas de uma árvore geradora mínima.
- Essa ideia assume que conseguiremos criar a árvore geradora mínima sem verificar todas as combinações de arestas que podem ser escolhidas.
- Essa estratégia **gulosa** nem sempre funciona para todos os problemas. **Exemplo**: problema da mochila.
- Para que essa estratégia funcione, precisamos que a todo momento, A esteja contido em uma AGM. $\rightarrow$ **Invariante**

Árvore geradora mínima

Em cada iteração, dado um conjunto A de arestas contido em uma AGM, precisamos encontrar uma aresta (u, v) tal que $A \cup \{(u, v)\}$ esteja contido em uma AGM.

- Essa aresta (u, v) é chamada aresta segura.

Árvore geradora mínima

Em cada iteração, dado um conjunto A de arestas contido em uma AGM, precisamos encontrar uma aresta (u, v) tal que $A \cup \{(u, v)\}$ esteja contido em uma AGM.

- Essa aresta (u, v) é chamada aresta segura.

Definição

Seja $G = (V, E)$ um grafo, e seja $A \subseteq E$ contido em alguma AGM. Uma aresta (u, v) é uma aresta segura para A, se $A \cup \{(u, v)\}$ também está contido em uma AGM.

Árvore geradora mínima: "algoritmo" genérico

AGM(G, ω)

1: $A \leftarrow \emptyset$
2: **enquanto** A não é árvore geradora **faça**
3: $\quad (u, v) \leftarrow$ Encontre aresta segura para A
4: $\quad A \leftarrow A \cup \{(u, v)\}$
devolva A

Árvore geradora mínima: "algoritmo" genérico

AGM(G, ω)

1: $A \leftarrow \emptyset$
2: **enquanto** A não é árvore geradora **faça**
3: $\quad (u, v) \leftarrow$ Encontre aresta segura para A
4: $\quad A \leftarrow A \cup \{(u, v)\}$
devolva A

Complexidade: Desconhecida.

Correção: Sem dúvida, se a linha 3 funciona, o algoritmo está correto (Por quê?).

Árvore geradora mínima

Se a linha 3 funciona, o algoritmo está correto.

- O algoritmo devolve uma árvore geradora.
- Ela é mínima, pela definição de aresta segura obtida na última iteração.

Árvore geradora mínima

Se a linha 3 funciona, o algoritmo está correto.

- O algoritmo devolve uma árvore geradora.
- Ela é mínima, pela definição de aresta segura obtida na última iteração.

Se a linha 3 está bem definida, em cada iteração **existe** uma aresta segura para ser escolhida:

- Se entramos no *loop*, A não contém todas arestas de uma AGM T;
- Portanto, sempre existe uma aresta segura (u, v) de T que não esteja em A.

Árvore geradora mínima

Se a linha 3 funciona, o algoritmo está correto.

- O algoritmo devolve uma árvore geradora.
- Ela é mínima, pela definição de aresta segura obtida na última iteração.

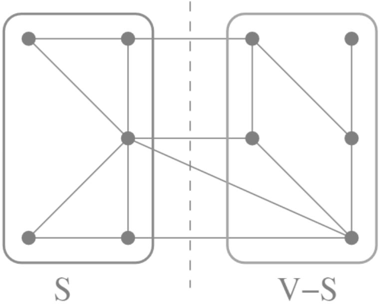
Se a linha 3 está bem definida, em cada iteração **existe** uma aresta segura para ser escolhida:

- Se entramos no *loop*, A não contém todas arestas de uma AGM T;
- Portanto, sempre existe uma aresta segura (u, v) de T que não esteja em A.

Para que este algoritmo funcionar, falta especificar como **encontrar** uma aresta segura.

Árvore geradora mínima: corte (retrospectiva)

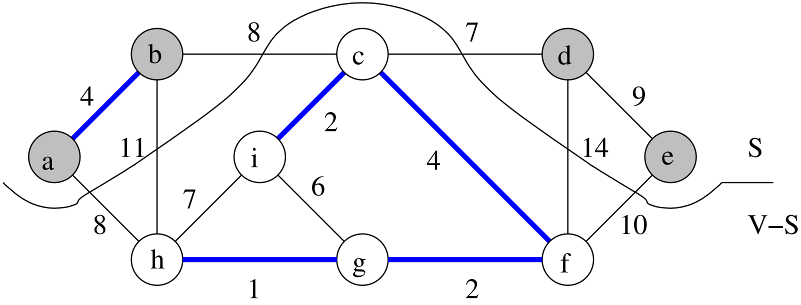
Seja $G = (V, E)$ um grafo e $S \subset V$, $S \neq \emptyset$. Denotamos por $\delta(S)$ o conjunto das arestas de G com um extremo em S e outro em $V - S$.



Quando S consiste de um único vértice v, escrevemos $\delta(v)$ em vez de $\delta(\{v\})$.
Dizemos que $\delta(S)$ é um corte de G **induzido** por S.

Árvore geradora mínima

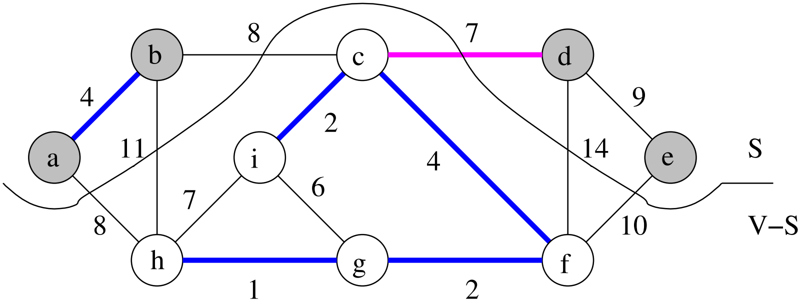
Corte $\delta(S)$: conjunto de arestas de com um extremo em S e outro em $V - S$.



Vamos dizer que um corte $\delta(S)$ **respeita** um conjunto A de arestas se o corte $\delta(S)$ não contém nenhuma aresta de A.

Árvore geradora mínima

Corte $\delta(S)$: conjunto de arestas de com um extremo em S e outro em $V - S$.

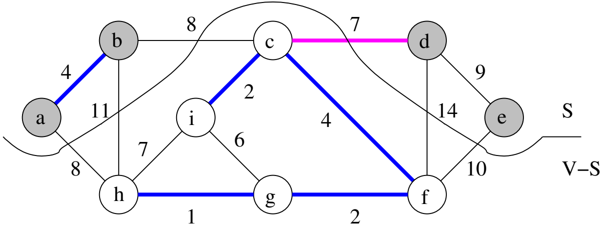


Uma aresta de um corte $\delta(S)$ é leve
se tem o menor peso entre as arestas do corte.

Árvore geradora mínima

Teorema

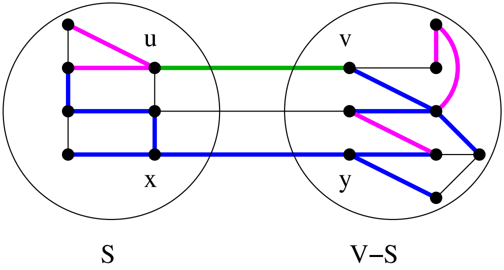
Seja G um grafo com uma função w de pesos das arestas, e seja A um subconjunto de arestas de uma AGM de G. Se o corte $\delta(S)$ respeita A, e (u, v) é uma aresta leve do corte, então (u, v) é uma aresta segura.



Árvore geradora mínima

Seja:

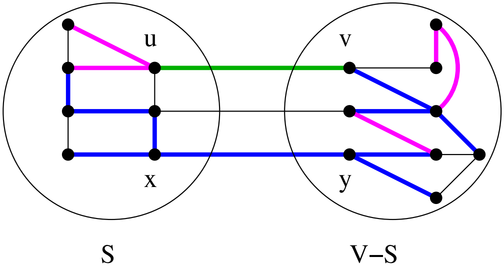
- T uma AGM que contém A,
- $\delta(S)$ um corte que respeita A,
- (u, v) uma **aresta leve** deste corte.



Árvore geradora mínima

Seja:

- T uma AGM que contém A,
- $\delta(S)$ um corte que respeita A,
- (u, v) uma **aresta leve** deste corte.

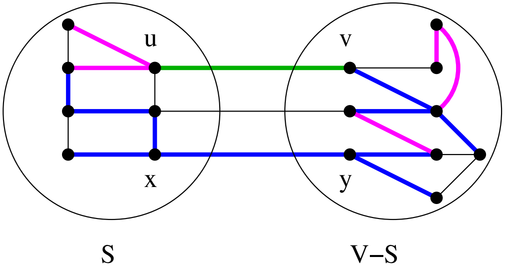


Se $(u, v) \in T$, não há nada a provar: (u, v) é segura por definição.

Árvore geradora mínima

Seja:

- T uma AGM que contém A,
- $\delta(S)$ um corte que respeita A,
- (u, v) uma **aresta leve** deste corte.



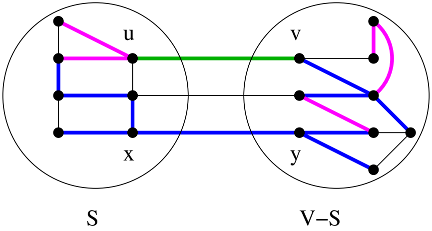
Se $(u, v) \in T$, não há nada a provar: (u, v) é segura por definição.

Vamos supor então que (u, v) **não** é uma aresta de T.

Construiremos uma AGM T' que contém $A \cup \{(u, v)\}$, mostrando que (u, v) é segura.

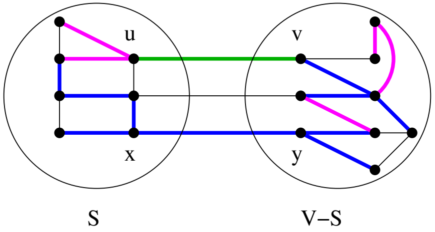
Árvore geradora mínima

Existe um único caminho P de u a v em T. Como u a v estão em lados opostos do corte $\delta(S)$, há uma aresta de P que pertence a $\delta(S)$.



Árvore geradora mínima

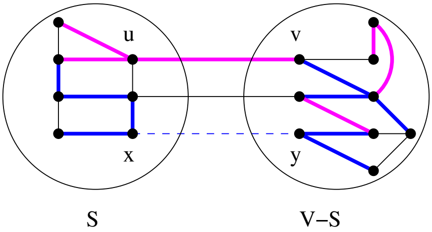
Existe um único caminho P de u a v em T. Como u a v estão em lados opostos do corte $\delta(S)$, há uma aresta de P que pertence a $\delta(S)$.



Seja (x, y) essa aresta. Note que (x, y) não pertence a A pois o corte $\delta(S)$ respeita A.

Árvore geradora mínima

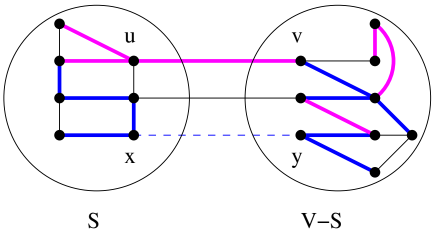
$T' = (T \cup \{(u,v)\}) - \{(x,y)\}$ é uma árvore geradora.



Mostraremos que T' é uma AGM.

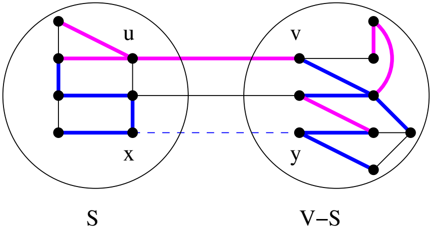
Árvore geradora mínima

Como (u, v) é uma aresta leve do corte $\delta(S)$ e (x, y) pertence ao corte, temos que $\omega(u, v) \leq \omega(x, y)$.



Árvore geradora mínima

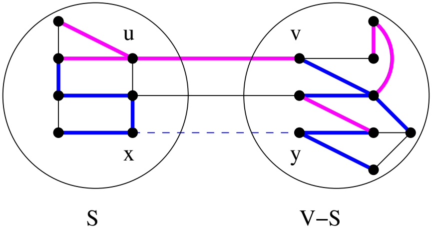
Como (u, v) é uma aresta leve do corte $\delta(S)$ e (x, y) pertence ao corte, temos que $\omega(u, v) \leq \omega(x, y)$.



Portanto, $\omega(T') = \omega(T) - \omega(x, y) + \omega(u, v) \leq \omega(T)$.

Árvore geradora mínima

Como (u, v) é uma aresta leve do corte $\delta(S)$ e (x, y) pertence ao corte, temos que $\omega(u, v) \leq \omega(x, y)$.



Portanto, $\omega(T') = \omega(T) - \omega(x, y) + \omega(u, v) \leq \omega(T)$.
Como T é uma AGM, então $\omega(T) \leq \omega(T')$. Logo, T' é uma AGM.
Veja que T' contém $A \cup \{(u, v)\}$ e portanto, (u, v) é uma aresta segura. ■

Árvore geradora mínima

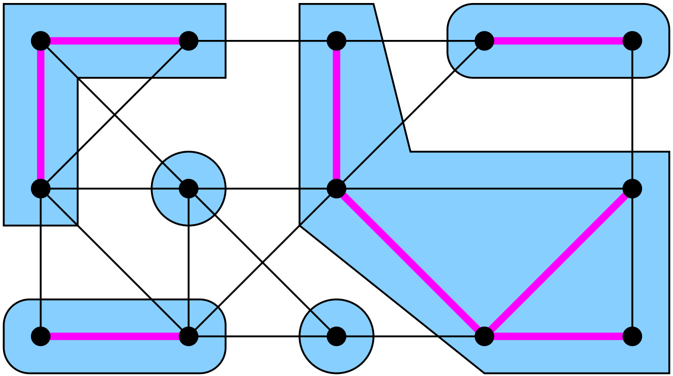
Teorema

Seja G um grafo com uma função w de pesos das arestas, e seja A um subconjunto de arestas de uma AGM de G. Se o corte $\delta(S)$ respeita A, e (u, v) é uma aresta leve do corte, então (u, v) é uma aresta segura.

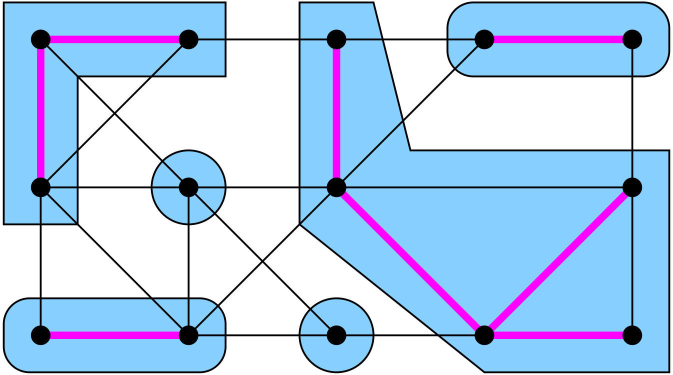
Corolário

Seja G um grafo com uma função w de pesos das arestas, e seja A um subconjunto de arestas de uma AGM de G. Seja C uma componente (árvore) de $G_A = (V, A)$. Se (u, v) é uma aresta leve do corte $\delta(C)$, então (u, v) é uma aresta segura.

Árvore geradora mínima



Árvore geradora mínima



Algoritmos de Prim e Kruskal: são versões do algoritmo genérico, e usam a ideia do Corolário.

Resumo

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

- Inicialmente, o conjunto de arestas A é vazio.

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

- Inicialmente, o conjunto de arestas A é vazio.
- As arestas de A sempre formam uma **única árvore**.

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

- Inicialmente, o conjunto de arestas A é vazio.
- As arestas de A sempre formam uma **única árvore**.
- Se $A = \emptyset$, um vértice r é escolhido para analisar o corte $\delta(r)$ na 1^{ra} iteração.

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

- Inicialmente, o conjunto de arestas A é vazio.
- As arestas de A sempre formam uma **única árvore**.
- Se $A = \emptyset$, um vértice r é escolhido para analisar o corte $\delta(r)$ na 1^{ra} iteração.
- Em cada iteração posterior, o algoritmo considera o corte $\delta(S)$ onde S é o conjunto de vértices que são extremos de A.

Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

- Inicialmente, o conjunto de arestas A é vazio.
- As arestas de A sempre formam uma **única árvore**.
- Se $A = \emptyset$, um vértice r é escolhido para analisar o corte $\delta(r)$ na 1^{ra} iteração.
- Em cada iteração posterior, o algoritmo considera o corte $\delta(S)$ onde S é o conjunto de vértices que são extremos de A.
- Neste corte $\delta(S)$, o algoritmo encontra uma aresta leve (u, v), adiciona (u, v) ao conjunto A. Isto é repetido até que A seja uma árvore geradora.

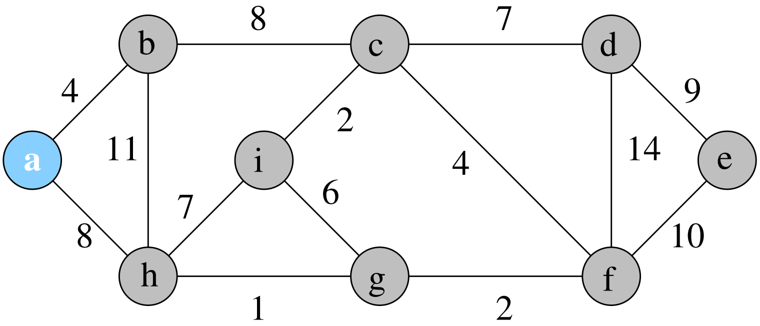
Árvore geradora mínima: algoritmo de Prim

Algoritmo de Prim: como encontrar uma aresta segura para A.

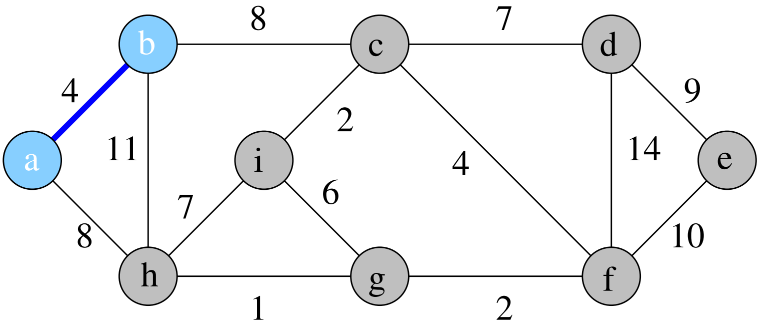
- Inicialmente, o conjunto de arestas A é vazio.
- As arestas de A sempre formam uma **única árvore**.
- Se $A = \emptyset$, um vértice r é escolhido para analisar o corte $\delta(r)$ na 1^{ra} iteração.
- Em cada iteração posterior, o algoritmo considera o corte $\delta(S)$ onde S é o conjunto de vértices que são extremos de A.
- Neste corte $\delta(S)$, o algoritmo encontra uma aresta leve (u, v), adiciona (u, v) ao conjunto A. Isto é repetido até que A seja uma árvore geradora.

Como encontrar **eficientemente** uma aresta leve **nesse corte**?

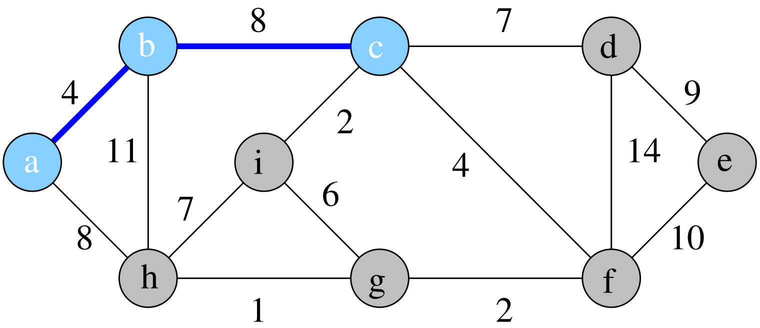
O algoritmo de Prim



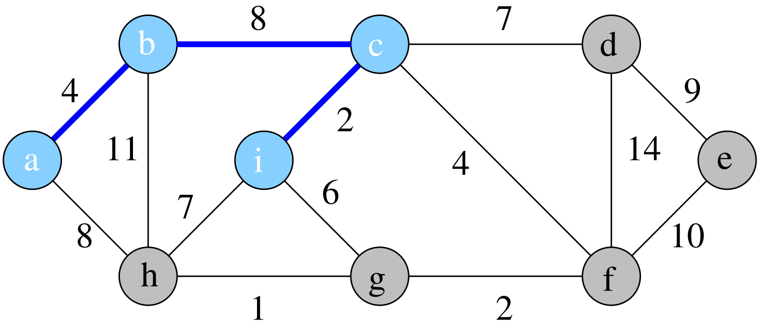
O algoritmo de Prim



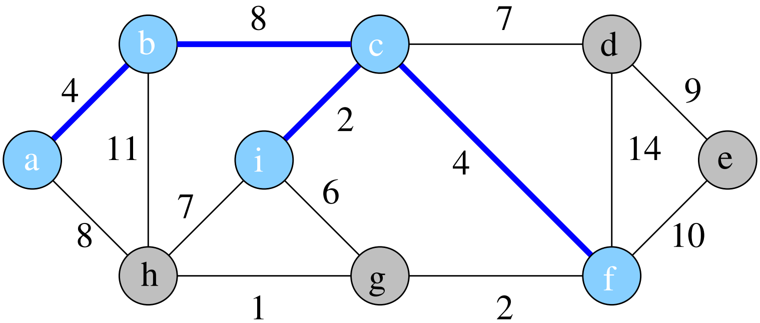
O algoritmo de Prim



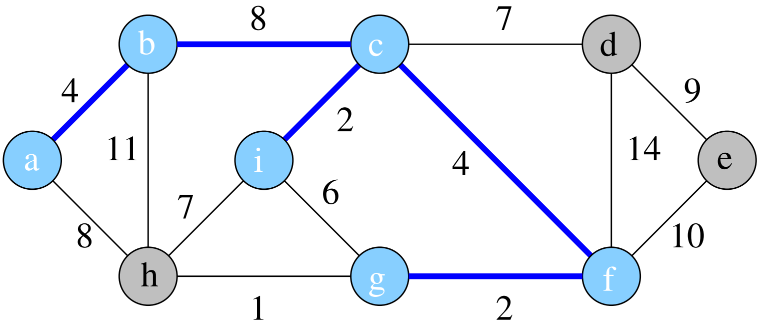
O algoritmo de Prim



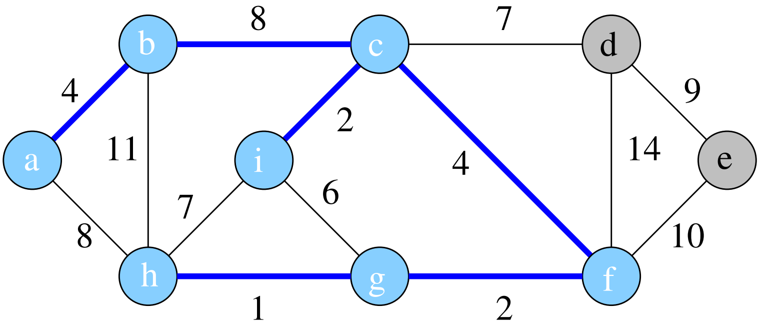
O algoritmo de Prim



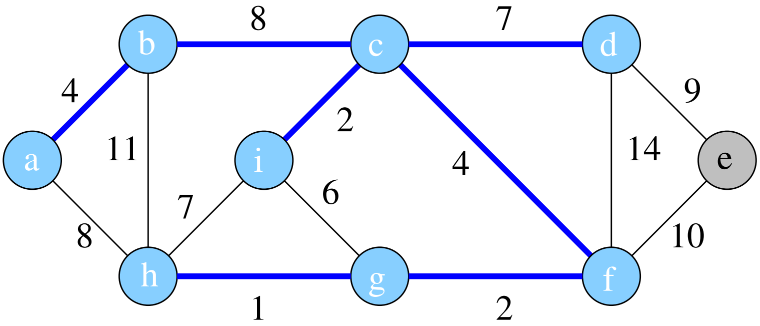
O algoritmo de Prim



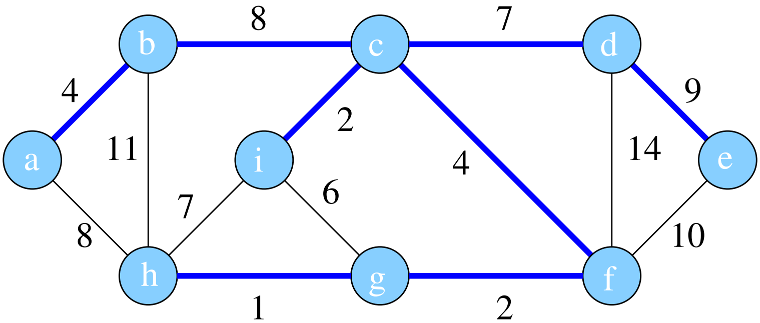
O algoritmo de Prim



O algoritmo de Prim



O algoritmo de Prim



Algoritmo de Prim

Problema: Como encontrar **eficientemente** uma aresta leve?

Durante a execução, vamos manter as seguintes informações:

- Todos os vértices do grafo, **que não estão na árvore**, estão em uma fila de prioridade (de mínimo) Q.

Algoritmo de Prim

Problema: Como encontrar **eficientemente** uma aresta leve?

Durante a execução, vamos manter as seguintes informações:

- Todos os vértices do grafo, **que não estão na árvore**, estão em uma fila de prioridade (de mínimo) Q.
- Cada vértice v em Q tem uma chave key$[v]$ que indica o menor peso de qualquer aresta ligando v a algum vértice u da árvore – neste caso, $\pi[v] = u$.

Algoritmo de Prim

Problema: Como encontrar **eficientemente** uma aresta leve?

Durante a execução, vamos manter as seguintes informações:

- Todos os vértices do grafo, **que não estão na árvore**, estão em uma fila de prioridade (de mínimo) Q.
- Cada vértice v em Q tem uma chave $\text{key}[v]$ que indica o menor peso de qualquer aresta ligando v a algum vértice u da árvore – neste caso, $\pi[v] = u$.
- Se não existir nenhuma aresta ligando v à árvore, então $\text{key}[v] = \infty$ e $\pi[v] = \text{NIL}$.

Algoritmo de Prim

Problema: Como encontrar **eficientemente** uma aresta leve?

Durante a execução, vamos manter as seguintes informações:

- Todos os vértices do grafo, **que não estão na árvore**, estão em uma fila de prioridade (de mínimo) Q.
- Cada vértice v em Q tem uma chave $\text{key}[v]$ que indica o menor peso de qualquer aresta ligando v a algum vértice u da árvore – neste caso, $\pi[v] = u$.
- Se não existir nenhuma aresta ligando v à árvore, então $\text{key}[v] = \infty$ e $\pi[v] = \text{NIL}$.

Uma **aresta leve** do corte $\delta(S)$ é uma aresta $(\pi[v], v)$ com chave $\text{key}[v]$ mínima.

Algoritmo de Prim

Problema: Como encontrar **eficientemente** uma aresta leve?

Durante a execução, vamos manter as seguintes informações:

- Todos os vértices do grafo, **que não estão na árvore**, estão em uma fila de prioridade (de mínimo) Q.
- Cada vértice v em Q tem uma chave $\text{key}[v]$ que indica o menor peso de qualquer aresta ligando v a algum vértice u da árvore – neste caso, $\pi[v] = u$.
- Se não existir nenhuma aresta ligando v à árvore, então $\text{key}[v] = \infty$ e $\pi[v] = \text{NIL}$.

Uma **aresta leve** do corte $\delta(S)$ é uma aresta $(\pi[v], v)$ com chave $\text{key}[v]$ mínima.

Durante a execução, as arestas são: $A = \{(u, \pi[u]) : u \in V - Q - \{r\}\}$.

Algoritmo de Prim

AGM-Prim(G, ω, r)

```
para cada u ∈ V faça
    key[u] ← ∞
    π[u] ← NIL
key[r] ← 0
Q ← Build(V)    ▷ Criar a fila de prioridades com todos os vértices
enquanto Q ≠ ∅ faça
    u ← ExtractMin(Q)    ▷ Extrair o vértice com a menor chave "key"
    para cada v ∈ Adj[u] faça
        se v ∈ Q e ω(u, v) < key[v] então
            π[v] ← u
            key[v] ← ω(u, v)    ▷ Operação DecreaseKey
```

Algoritmo de Prim

Durante a execução do algoritmo, no inicio de cada iteração **enquanto**:

- Em A temos arestas de uma árvore T com raiz r:

$$A = \{(u, \pi[u]) : u \in V - Q - \{r\}\}.$$

- Para cada $v \in Q$, com $\pi[v] \neq \mathrm{NIL}$, no vetor $\mathrm{key}[v]$ temos **o peso da aresta de menor peso que liga v a um vértice $\pi[v]$ na árvore.**

Algoritmo de Prim

Durante a execução do algoritmo, no inicio de cada iteração **enquanto**:

- Em A temos arestas de uma árvore T com raiz r:

$$A = \{(u, \pi[u]) \,:\, u \in V - Q - \{r\}\}.$$

- Para cada $v \in Q$, com $\pi[v] \neq \text{NIL}$, no vetor $\text{key}[v]$ temos **o peso da aresta de menor peso que liga v a um vértice $\pi[v]$ na árvore.**

Essa invariante garante que o algoritmo sempre escolhe uma **arestra leve**, e portanto uma **aresta segura** para adicionar a A. Pelo Corolário, o algoritmo está correto.

Algoritmo de Prim: complexidade de tempo

Complexidade de tempo:

- A complexidade de **AGM-Prim** depende de como a fila de prioridade Q é implementada.
- Operações que estamos usando:
  - **Build**, que é uma seqência de n operações **Insert**,
  - **ExtractMin**,
  - **DecreaseKey**.

Algoritmo de Prim: complexidade de tempo

Complexidade de tempo:

- A complexidade de **AGM-Prim** depende de como a fila de prioridade Q é implementada.
- Operações que estamos usando:
  - **Build**, que é uma seqência de n operações **Insert**,
  - **ExtractMin**,
  - **DecreaseKey**.

Observação: como G é conexo, temos que $V = O(E)$. Portanto, $O(V + E) = O(E)$.

Algoritmo de Prim: complexidade de tempo

Quantas vezes vamos executar cada operação de fila:

- **Insert**: $|V| = O(V)$ chamadas.
- **ExtractMin**: $|V| = O(V)$ chamadas.
- **DecreaseKey**: $O(E)$ chamadas.
- * $v \in Q$ – pode ser feito em tempo constante com um **vetor booleano**.

Algoritmo de Prim: complexidade de tempo

Quantas vezes vamos executar cada operação de fila:

- **Insert**: $|V| = O(V)$ chamadas.
- **ExtractMin**: $|V| = O(V)$ chamadas.
- **DecreaseKey**: $O(E)$ chamadas.
- * $v \in Q$ – pode ser feito em tempo constante com um **vetor booleano**.

Tempo total: $O(V)\cdot$**Insert** + $O(V)\cdot$**ExtractMin** + $O(E)\cdot$**DecreaseKey**.

Algoritmo de Prim: complexidade de tempo

Tempo total: $O(V) \cdot$**Insert** $+$ $O(V) \cdot$**ExtractMin** $+$ $O(E) \cdot$**DecreaseKey**.

Se implementarmos Q como um **heap de mínimo** (min-heap):

- **Insert**: $O(\log V)$.
- **ExtractMin**: $O(\log V)$.
- **DecreaseKey**: $O(\log V)$.

Algoritmo de Prim: complexidade de tempo

Tempo total: $O(V) \cdot$**Insert** $+$ $O(V) \cdot$**ExtractMin** $+$ $O(E) \cdot$**DecreaseKey**.

Se implementarmos Q como um **heap de mínimo** (min-heap):

- **Insert**: $O(\log V)$.
- **ExtractMin**: $O(\log V)$.
- **DecreaseKey**: $O(\log V)$.

Tempo total: $O(V \log V + V \log V + E \log V) = O(E \log V)$.

*Podemos inicializar o min-heap em $O(V)$, mas isso não muda nada.

Algoritmo de Prim: complexidade de tempo

Tempo total usando **heap de mínimo**: $O(E \log V)$

Existe uma implementação mais eficiente?

Algoritmo de Prim: complexidade de tempo

Tempo total usando **heap de mínimo**: $O(E \log V)$

Existe uma implementação mais eficiente? – SIM.

- Heap de Fibonacci é uma estrutura de dados usada para implementar uma fila de prioridade com as operações **Insert**, **ExtractMin**, **DecreaseKey**.
- No Heap de Fibonacci, **ExtractMin** é $O(\log V)$, mas **Insert** e **DecreaseKey** têm **custo amortizado** $O(1)$.
- Melhoramos o tempo $O(V)$·**Insert** + $O(V)$·**ExtractMin** + $O(E)$·**DecreaseKey**:

$$O(V + V \log V + E) = O(E + V \log V).$$

Algoritmo de Prim: complexidade de tempo

Tempo total usando **heap de mínimo**: $O(E \log V)$

Existe uma implementação mais eficiente? – SIM.

- Heap de Fibonacci é uma estrutura de dados usada para implementar uma fila de prioridade com as operações **Insert**, **ExtractMin**, **DecreaseKey**.
- No Heap de Fibonacci, **ExtractMin** é $O(\log V)$, mas **Insert** e **DecreaseKey** têm **custo amortizado** $O(1)$.
- Melhoramos o tempo $O(V)\cdot$**Insert** $+\ O(V)\cdot$**ExtractMin** $+\ O(E)\cdot$**DecreaseKey**:

$$O(V + V \log V + E) = O(E + V \log V).$$

Mas o que é **custo amortizado**?

Custo amortizado

Custo amortizado:

- Seja S uma estrutura de dados, e $\mathbf{P}(S)$ uma operação sobre S. Por exemplo, inserir ou remover um elemento de S.
- Se durante a execução de um algoritmo são feitas k chamadas a $\mathbf{P}(S)$, seja T o **tempo total** das k operações **P** durante a execução do algoritmo. Então o custo (ou tempo) amortizado de **P** é T/k.
- Por exemplo, se $T = 4n$ e $k = 2n$ então o custo amortizado é 2 (constante). Isto não significa que cada operação gasta tempo constante!
- O custo amortizado nos indica **a média de tempo** de cada operação **P**, em uma sequência de k operações.
- O custo amortizado não é usado na análise de uma operação, mas para analisar sequências de k operações (como **Insert**, **ExtractMin**, **DecreaseKey** no Prim).

Resumo

Síntese

- Analisamos o problema de obter uma **árvore geradora mínima** em **grafos ponderados**.
- Estudamos o **Algoritmo de Prim**, que é um dos dois algoritmos que resolvem esse problema.

Material bibliográfico e exercícios

T. Cormen et al. Algoritmos - Teoria e Prática (3a ed.). – **Cap. 23**

Exercícios: ver exercícios no final dos (sub)capítulos 23.1 e 23.2.

Dúvidas

Dúvidas?