AULA 13 – Árvores Geradoras Mínimas (MST – Minimum Spanning Trees) [Parte II]

Prof. Daniel Kikuti

Universidade Estadual de Maringá

13 de julho de 2015

Retrospectiva do último episódio

Árvore geradora mínima

Uma **árvore geradora mínima** T de um grafo G com pesos em suas arestas é uma árvore geradora cujo peso total (a soma dos pesos de suas arestas, $w(T) = \sum_{u,v \in T} w(u,v)$) não é maior que o peso total de qualquer outra árvore geradora.

Retrospectiva do último episódio

Árvore geradora mínima

Uma **árvore geradora mínima** T de um grafo G com pesos em suas arestas é uma árvore geradora cujo peso total (a soma dos pesos de suas arestas, $w(T) = \sum_{u,v \in T} w(u,v)$) não é maior que o peso total de qualquer outra árvore geradora.

Propriedade do corte

Dado qualquer corte em um grafo com pesos nas arestas, a aresta de menor peso (**aresta leve**) que cruza este corte está na árvore geradora mínima deste grafo.

Retrospectiva do último episódio

Árvore geradora mínima

Uma **árvore geradora mínima** T de um grafo G com pesos em suas arestas é uma árvore geradora cujo peso total (a soma dos pesos de suas arestas, $w(T) = \sum_{u,v \in T} w(u,v)$) não é maior que o peso total de qualquer outra árvore geradora.

Propriedade do corte

Dado qualquer corte em um grafo com pesos nas arestas, a aresta de menor peso (**aresta leve**) que cruza este corte está na árvore geradora mínima deste grafo.

Algoritmo genérico

```
GENERIC-MST(W)
A ← ∅
Enquanto A não é uma árvore geradora mínima faça
    encontre uma aresta (u,v) que seja segura para A
    A ← A ∪ (u,v)
devolva A
```

Sumário

- Algoritmo de Prim
- Algoritmo de Kruskal

Definições

Invariante

O conjunto de arestas A é subconjunto de alguma árvore geradora mínima.

Aresta segura

Uma aresta (u, v) é **segura** para A se e somente se $A \cup \{(u, v)\}$ também é subconjunto de uma árvore geradora mínima.

Respeita

Um corte **respeita** um conjunto A se e somente se nenhuma aresta em A cruza o corte.

Ideia do algoritmo de Prim

Ideia principal

Anexar uma nova aresta a uma única árvore que cresce a cada iteração.

- Comece com qualquer vértice r como único vértice da árvore.
- Adicione $|V| - 1$ arestas a ele, sempre escolhendo uma aresta leve que conecta um vértice na árvore a um vértice que ainda não está na árvore.

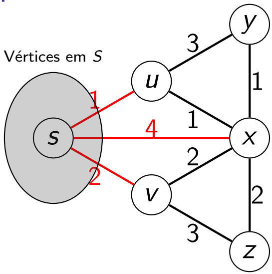
Ideia do algoritmo de Prim

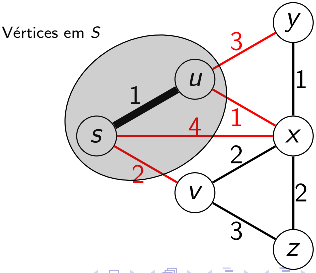
Ideia principal

Anexar uma nova aresta a uma única árvore que cresce a cada iteração.

- Comece com qualquer vértice r como único vértice da árvore.
- Adicione $|V| - 1$ arestas a ele, sempre escolhendo uma aresta leve que conecta um vértice na árvore a um vértice que ainda não está na árvore.

Exemplo





Algoritmo de Prim

```
mst-prim(G, w, r)
para cada u em G.V
  u.chave = ∞
  u.pai = NIL
r.chave = 0
Q = G.V
enquanto Q ≠ ∅
  u = extract-min(Q)
  para cada v em u.adj
    se v em Q e w(u, v) < v.chave
      v.pai = u
      v.chave = w(u, v)
```

Análise do algoritmo

Correção

A correção segue imediatamente da propriedade do corte demonstrada na aula anterior.

Análise do algoritmo

Correção

A correção segue imediatamente da propriedade do corte demonstrada na aula anterior.

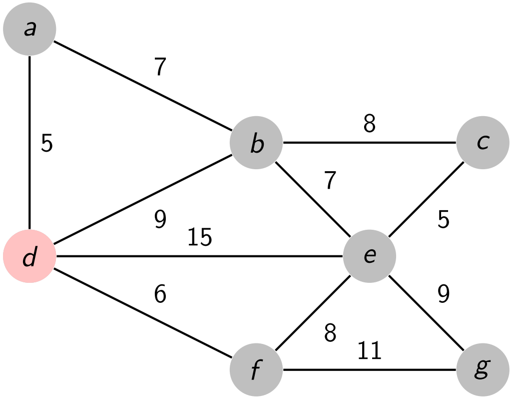
Complexidade

- Depende de como a fila de prioridades é implementada
- Linhas 1–5 usando `build-min-heap` tem custo $O(V)$.
- Laço enquanto (linhas 6–11) é executado $|V|$ vezes.
- `extract-min` consome tempo $O(\lg V)$.
- Laço para (linhas 8–11) é executado $O(E)$ vezes no total.
- Teste de pertinência de $v \in Q$ pode ser feito em $O(1)$.
- Atribuição na linha 11 é uma operação de `decrease-key`, que pode ser feita em $O(\lg V)$ (tempo para todas as chamadas de `decrease-key` é $O(E \lg V)$).
- **Total** $O(V \lg V + E \lg V) = O(E \lg V)$.

Fixação do algoritmo

Exercício[1]

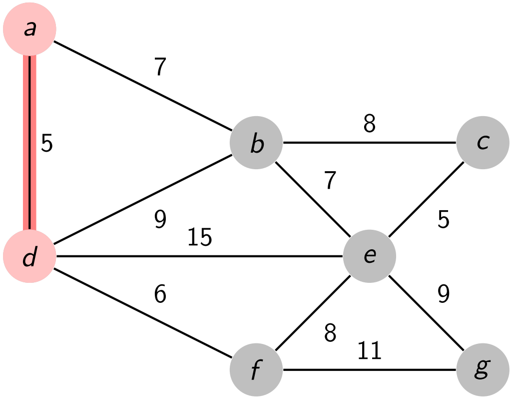
Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.



Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.

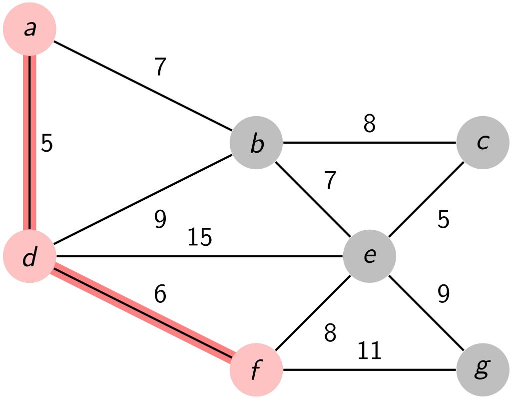


[1] Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice d como inicial.

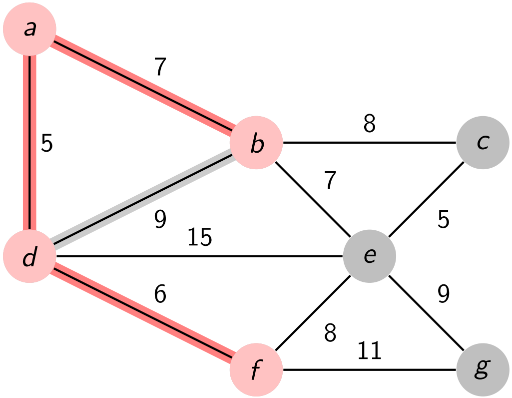


[1] Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.

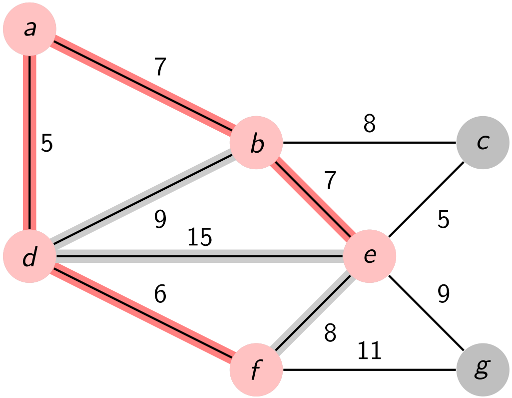


[1] Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.

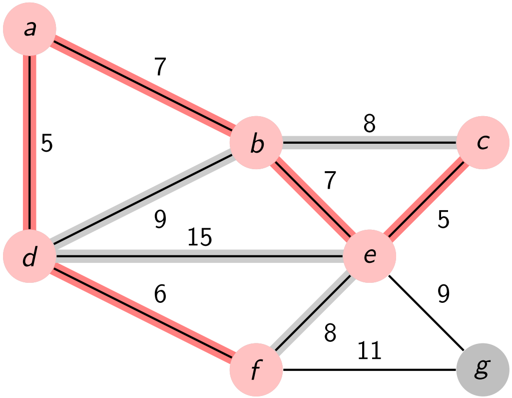


[1] Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.

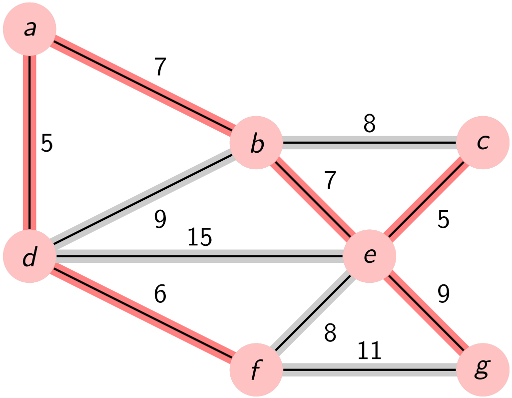


[1]Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

Fixação do algoritmo

Exercício[1]

Apresente a árvore geradora mínima para o seguinte grafo. Para isto, use o agoritmo de Prim e considere o vértice *d* como inicial.



[1] Copiado de http://www.texample.net/tikz/examples/prims-algorithm/

O algoritmo de Kruskal

Ideia principal

- Comece com nenhuma aresta e construa a árvore geradora mínima pela inserção sucessiva de arestas de E, em ordem crescente de custo.
- A medida em que analisamos as arestas nesta ordem, se a aresta $(u, v) \in E$ não gera um ciclo com as demais que foram adicionadas a A, então adicione (u, v) em A.
- Caso contrário (a aresta (u, v) forma um ciclo com arestas em A), descarte a aresta (u, v).

O algoritmo de Kruskal

Corolário

Seja $G = (V, E)$ um grafo conexo não orientado com uma função peso de valor real w definido em E. Seja A um subconjunto de E que está incluído em alguma árvore geradora mínima correspondente a G, e seja $C = (V_C, E_C)$ um componente conexo (árvore) na floresta $G_A = (V, A)$. Se (u, v) é uma aresta leve conectando C a algum outro componente em G_A, então (u, v) é segura para A.

Demonstração

O corte $(V_C, V - V_C)$ respeita A e, (u, v) é uma aresta leve para este corte. Portanto, (u, v) é segura para A.

Union Find data structure

O problema

Precisamos de uma estrutura de dados para manter um conjunto de componentes conexos do grafo que permita **busca** e **atualização** de componentes de maneira eficiente.

A estrutura `Union-find` suporta três operações

- `Make-set(s)`: cria um conjunto para o elemento `s`.
- `Find(u)`: devolve o nome do conjunto contendo `u`.
- `Union(A,B)`: devolve um único conjunto contendo a união dos conjuntos `A` e `B`.

Opções de projeto para construção destas operações???

O algoritmo de Kruskal

- Baseia-se diretamente no algoritmo genérico apresentado.
- Inicialmente cada vértice está em sua própria componente (árvore).
- De todas as arestas que conectam duas árvores quaisquer na floresta, uma aresta (u, v) de peso mínimo é escolhida. A aresta (u, v) é segura para alguma das duas árvores.
- Utiliza uma estrutura de dados de conjuntos disjuntos (ver capítulo 21 do Cormen).
- Cada conjunto contém os vértices de uma árvore da floresta atual.

Algoritmo de Kruskal

```
mst-kruskal(G, w)
A = ∅
para cada vértice v em G.V
  make-set(v)
ordenar as arestas de G.E por peso w
para cada aresta (u, v) em G.E, em ordem
crescente de peso w
  se find-set(u) != find-set(v)
    A = A U (u, v)
    union(u, v)
return A
```

Análise do algoritmo de Kruskal

Complexidade

Depende da implementação das operações com conjuntos disjuntos (supondo implementação da seção 21.3)

- Ordenação das arestas (linha 4): $O(E \lg E)$.
- Laço das linhas 5 a 8 executa $O(E)$ `find-set` e `union`. Juntamente com as $|V|$ operações `make-set`, elas demoram $O((V+E)\alpha(V))$, onde α é uma função de crescimento muito lento.
- Pelo fato de G ser supostamente conexo, temos que $|E| \geq |V| - 1$, portanto o tempo com operações com conjuntos disjuntos é $O(E\alpha(V))$.
- $\alpha(|V|) = O(\lg V) = O(\lg E)$, e portanto o tempo total das operações com conjuntos disjuntos é $O(E \lg E)$.
- **Total** $O(E \lg E)$.