

O estudo utilizando apenas este material **não é suficiente** para o entendimento do conteúdo. Recomendamos a leitura das referências no final deste material e a resolução (por parte do aluno) de todos os exercícios indicados.

Grafos

Árvores geradores mínimas

Conteúdo

Introdução

Como construir uma árvore geradora mínima

Algoritmos

- Algoritmo de Kruskal

- Algoritmo de Prim

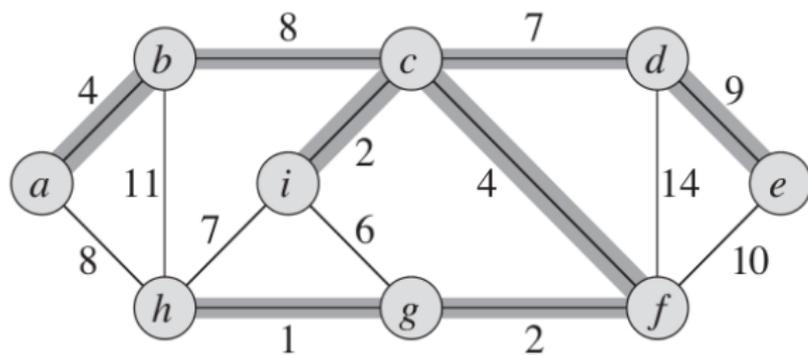
Exercícios

Referências

Introdução

- ▶ Dado um grafo conexo não orientado $G = (V, E)$ e uma função peso $w : E \rightarrow R$, queremos encontrar um subconjunto acíclico $T \subseteq E$ que conecte todos os vértices de G e cujo peso total $w(T) = \sum_{(u,v) \in T} w(u, v)$ seja minimizado
- ▶ Como T é acíclico e conecta todos os vértices, T forma uma árvore, que chamamos de **árvore geradora mínima** (MST)
- ▶ Chamamos o problema de determinar T de **problema da árvore geradora mínima**
- ▶ Veremos dois algoritmos gulosos para resolver este problema
 - ▶ Algoritmo de Kruskal
 - ▶ Algoritmo de Prim

Exemplo de árvore geradora mínima



- ▶ Propriedades de uma MST
 - ▶ Tem $|V| - 1$ arestas
 - ▶ Não tem ciclos
 - ▶ Pode não ser única

Como construir uma árvore geradora mínima

- ▶ Como construir uma árvore geradora mínima? Uma aresta de cada vez!
- ▶ Começamos com um conjunto vazio A
- ▶ Em cada etapa, determinamos um aresta (u, v) que pode ser adicionada a A , de forma a manter a seguinte invariante
 - ▶ Antes de cada iteração, A é um subconjunto de alguma árvore geradora mínima
- ▶ A aresta (u, v) é chamada de **aresta segura** para A

`generic-mst(G, w)`

1 $A = \emptyset$

2 **while** A não forma uma árvore geradora

3 encontre uma aresta (u, v) que seja segura para A

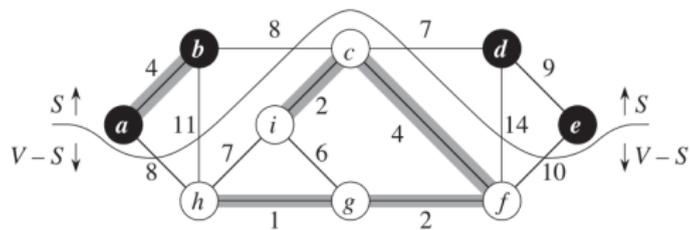
4 $A = A \cup \{(u, v)\}$

5 **return** A

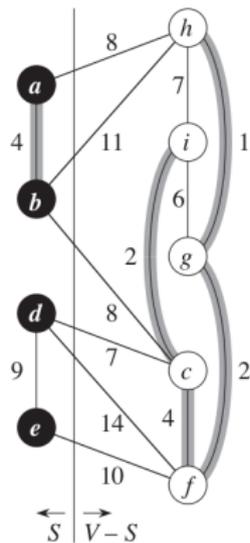
Como construir uma árvore geradora mínima

- ▶ Vamos fornecer uma regra para reconhecer arestas seguras, mas antes precisamos de algumas definições
- ▶ Seja $S \subset V$ e $A \subseteq E$
- ▶ Um **corte** $(S, V - S)$ de um grafo não orientado $G = (V, E)$ é uma partição de V
- ▶ Uma aresta $(u, v) \in E$ **cruza** o corte $(S, V - S)$ se um de seus extremos está em S e o outro em $V - S$
- ▶ Um corte **respeita** o conjunto A de arestas se nenhuma aresta em A cruza o corte
- ▶ Uma aresta é uma **aresta leve** cruzando um corte se seu peso é o mínimo de qualquer aresta que cruza o corte

Como construir uma árvore geradora mínima



(a)



(b)

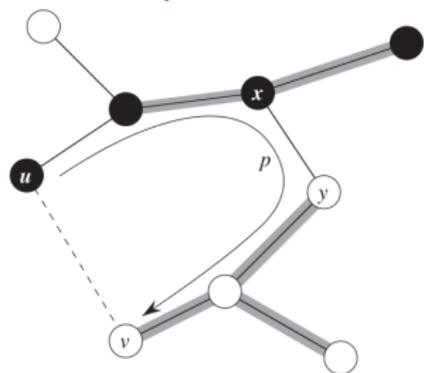
Como construir uma árvore geradora mínima

Teorema 23.1

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 , seja $(S, V - S)$ qualquer corte de G que respeita A e seja (u, v) uma aresta leve cruzando $(S, V - S)$. Então a aresta (u, v) é segura para A .

Como construir uma árvore geradora mínima

Ideia da prova



- ▶ Seja T uma MST que inclui A
 - ▶ Se T contém (u, v) , é claro que (u, v) é segura para A
 - ▶ Se T não contém (u, v) , construímos outra MST T' que inclui $A \cup \{(u, v)\}$ (veja o livro)

Como construir uma árvore geradora mínima

Corolário 23.2

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 .

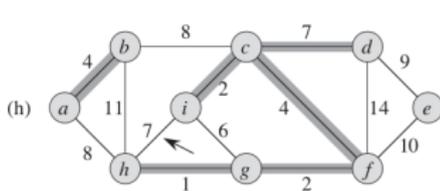
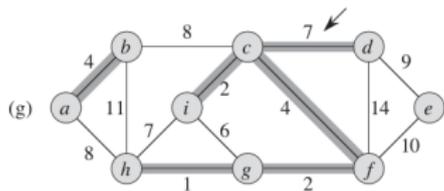
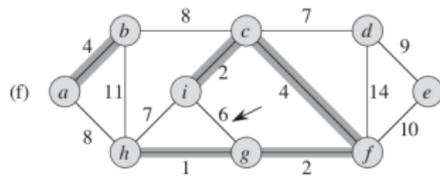
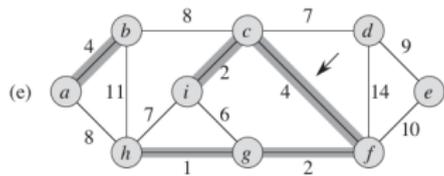
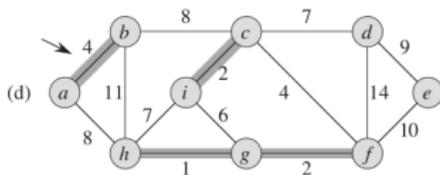
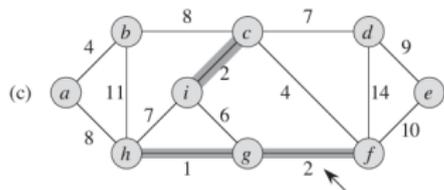
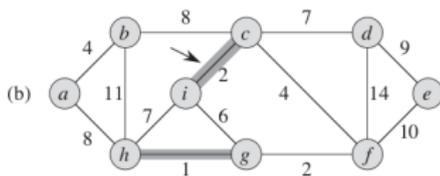
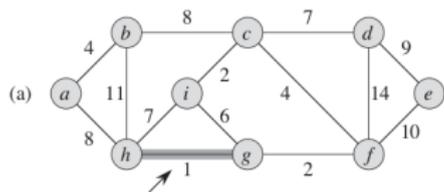
Prova

Tomamos $S = V_C$ no teorema 23.1

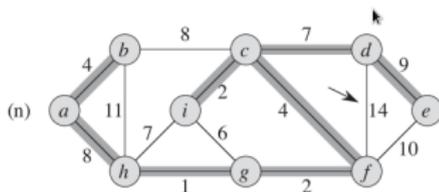
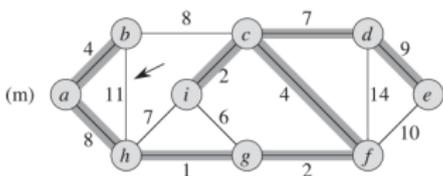
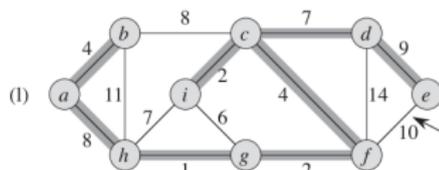
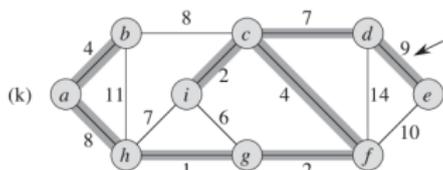
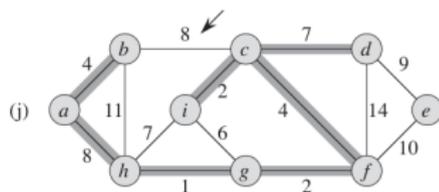
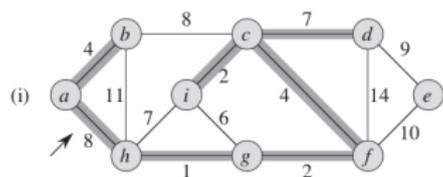
Algoritmo de Kruskal

- ▶ 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
- ▶ Cada conjunto contém os vértices de uma árvore da floresta atual

Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo de Kruskal

```
mst-kruskal(G, w)
1  A = 0
2  for cada vértice v em G.V
3    make-set(v)
4  ordenar por peso as arestas de E
5  for cada aresta (u, v) em E, em ordem de peso
6    if find-set(u) != find-set(v)
7      A = A U {(u, v)}
8      union(u, v)
9  return A
```

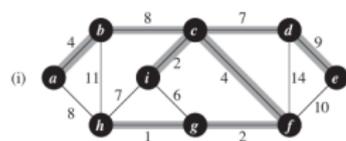
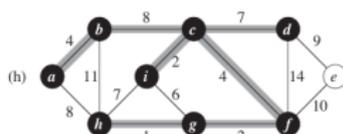
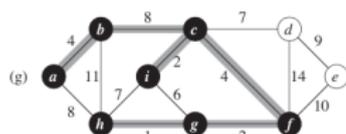
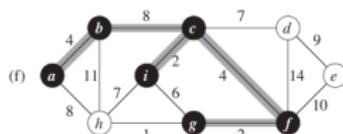
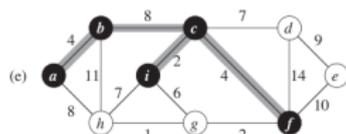
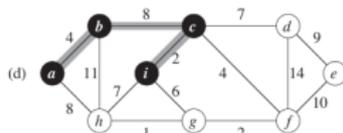
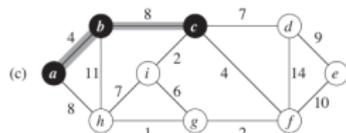
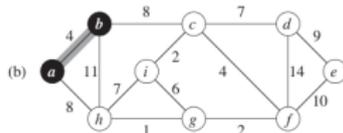
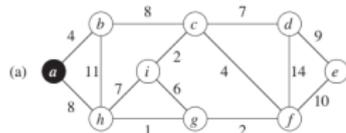
Análise do algoritmo de Kruskal

- ▶ A ordenação das arestas na linha 4 demora $O(E \lg E)$
- ▶ Operações com conjuntos disjuntos (depende da implementação, supomos a implementação da seção 21.3)
 - ▶ O 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 $O(E\alpha(V))$
 - ▶ Além disso, $\alpha(|V|) = O(\lg V) = O(\lg E)$, e portanto o tempo total das operações com conjuntos disjuntos é $O(E \lg E)$
- ▶ Somando o custo de ordenação e o custo das operações com conjuntos disjuntos, temos $O(E \lg E)$. Observando que $|E| < |V|^2$, temos que $\lg |E| = O(\lg V)$, e portanto, o tempo de execução do algoritmo é $O(E \lg V)$

Algoritmo de Prim

- ▶ Baseia-se diretamente no algoritmo genérico apresentado
- ▶ As arestas do conjunto A formam uma única árvore
- ▶ A árvore começa com uma raiz arbitrária r e aumenta até alcançar todos os vértices em V
- ▶ Para cada vértice v , $v.chave$ é o peso mínimo de qualquer aresta que conecta v a um vértice da árvore; $v.chave = \infty$ se não existe nenhuma aresta deste tipo
- ▶ Em cada passo, um vértice u com a menor chave é adicionado a árvore junto com a aresta $(u.pai, u)$
- ▶ A questão principal para implementar o algoritmo de Prim de forma eficiente é tornar fácil a seleção de uma nova aresta a ser adicionada à árvore

Algoritmo de Prim



Algoritmo de Prim

```
mst-prim(G, w, r)
1  for cada u em G.V
2    u.chave = infinito
3    u.pai = NIL
4  r.chave = 0
5  Q = G.V
6  while Q != 0
7    u = extract-min(Q)
8    for cada v em u.adj
9      if v em Q e  $w(u, v) < v.chave$ 
10       v.pai = u
11       v.chave =  $w(u, v)$ 
```

Análise do algoritmo de Prim

- ▶ Depende de como a fila de prioridade é implementada
- ▶ Se a fila for implementada como um heap mínimo, o algoritmo `build-min-heap` é utilizado na inicialização nas linhas 1 a 5 no tempo $O(V)$
- ▶ O corpo do laço `while` é executado $|V|$ vezes, como cada operação `extract-min` demora $O(\lg V)$, o tempo total para todas as chamadas de `extract-min` é $O(V \lg V)$
- ▶ O laço `for` das linhas 8 a 11 é executado no total $O(E)$ vezes
- ▶ O teste de pertinência da linha 9 pode ser implementa em tempo constante
- ▶ A atribuição na linha 11 envolve uma operação implícita de `decrease-key`, que demora $O(\lg V)$, o tempo para todas as chamadas de `decrease-key` é $O(E \lg V)$
- ▶ Portanto, o tempo total do algoritmo é $(V \lg V + E \lg V) = O(E \lg V)$

Análise do algoritmo de Prim

- ▶ Se heaps de Fibonacci forem usados o tempo de execução assintótico pode ser melhorado
- ▶ `extract-min` é executado em tempo amortizado de $O(\lg V)$
- ▶ `decrease-key` é executado em tempo amortizado de $O(1)$
- ▶ Tempo total do algoritmo melhora para $O(E + V \lg V)$

Exercícios

- ▶ 23.1-2, 23.1-4
- ▶ 23.2-2, 23.2-3, 23.2-4, 23.2-5, 23.2-8

Exercícios

- 23.1-2 O professor Sabatier supõe a recíproca do teorema 23.1 dada a seguir. Seja $G = (V, E)$ um grafo conexo não orientado com uma função peso de valor real w definida em E . Seja A um subconjunto de E que está incluído em alguma árvore geradora mínima para G , seja $(S, V - S)$ qualquer conjunto corte de G que respeita A , e seja (u, v) uma aresta segura para A que cruza $(S, V - S)$. Então (u, v) é uma aresta leve para o corte. Mostre que a hipótese do professor é incorreta, fornecendo um contra-exemplo.
- 23.1-4 Forneça um exemplo simples de um grafo conexo tal que o conjunto de arestas $\{(u, v) : \text{existe um corte } (S, V - S) \text{ tal que } (u, v) \text{ é uma aresta leve cruzando } (S, V - S)\}$ não forma uma árvore geradora mínima.

Exercícios

- 23.2-2 Suponha que o grafo $G = (V, E)$ seja representado por uma matriz de adjacências. Forneça uma implementação simples do algoritmo de Prim para esse caso que seja executada no tempo $O(V^2)$.
- 23.2-3 A implementação do heap de Fibonacci do algoritmo de Prim é assintoticamente mais rápida que a implementação de heap binário para um grafo esparso $G = (V, E)$, onde $|E| = \Theta(V)$? E no caso de um grafo denso, onde $|E| = \Theta(V^2)$? De que modo $|E|$ e $|V|$ devem estar relacionados para que a implementação de heap de Fibonacci seja assintoticamente mais rápida que a implementação de heap binário?

Exercícios

- 23.2-4 Suponha que todos os pesos de arestas de um grafo sejam inteiros no intervalo de 1 a $|V|$. Com que rapidez é possível executar o algoritmo de Kruskal? E se os pesos de arestas forem inteiros no intervalo de 1 a W para alguma constante W ?
- 23.2-5 Suponha que todos os pesos de arestas de um grafo sejam inteiros no intervalo de 1 a $|V|$. Com que rapidez é possível executar o algoritmo de Prim? E se os pesos de arestas forem inteiros no intervalo de 1 a W para alguma constante W ?

Referências

- ▶ Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 23.