DCC206 – Algoritmos 1

Aula 05 – Árvores geradoras mínimas

Professor Renato Vimieiro

Introdução

- Considere o seguinte problema: como manter *n* cidades conectadas à rede elétrica com o menor custo possível?
  - Minimizar custo associado à instalação das linhas de transmissão (por exemplo)
- Esse problema pode ser resolvido modelando-o com um grafo:
  - Vértices: cidades
  - Arestas: linhas de transmissão
  - Peso: custo associado (qtd cabos, torres, …)
- Solução:
  - Encontrar uma árvore geradora de custo mínimo

Definição

- Uma **árvore geradora** de um grafo G=(V,E) é um subgrafo de G com todos os seus vértices e um subconjunto máximo de arestas que não formam ciclos
- Seja p(u,v) o peso associado à aresta (u,v) em G. O custo total de uma árvore geradora é:
  - $p(T) = \sum_{(u,v)\in E} p(u,v)$
- Uma árvora geradora *T* é **mínima** (ou tem custo mínimo) se, e somente se, $p(T) \leq p(T')$ para qualquer árvore geradora *T’*

Algoritmos gulosos

- As soluções que veremos para resolução do problema de encontrar a árvore geradora mínima de um grafo são baseadas numa técnica de programação conhecida como **algoritmos gulosos**
- A técnica de construção de algoritmos gulosos busca uma solução ótima global a partir de soluções ótimas locais
  - A cada iteração escolhe-se uma solução parcial que produza o melhor resultado imediato
  - A solução ótima pode não envolver a escolha de um ótimo local, pode-se escolher uma solução parcial 'ruim' que leva ao ótimo global do problema
  - Exemplo: Escolha da melhor rota para percorrer um conjunto de cidades. A escolha gulosa envolve eleger o melhor caminho imediato (entre uma cidade e suas vizinhas, sem considerar o custo total do caminho

Algoritmos gulosos

- Algoritmos guloso são heurísticos: podem ou não resultar na solução ótima do problema
  - Em geral heurísticas são usadas em situações onde encontrar o ótimo de maneira exata pode ser computacionalmente inviável
  - Espera-se que os resultados sejam bons (próximos do ótimo)
- Os algoritmos para encontrar árvore geradora mínima são heurísticas, mas foi provado que eles encontram a solução ótima

Algoritmo genérico para MST

- As estratégias gulosas que estudaremos são baseadas no seguinte algoritmo genérico:
- Inicialmente, $S = \varnothing$
- Enquanto $|S| < |V|-1$
  - (u,v) = selecionaAresta(E)
  - Se (u,v) é segura para S, inserir (u,v) em S
- Retornar S

Algoritmo genérico para MST

- Existem dois pontos a se discutir nesse algoritmo
  - A escolha da aresta
  - A definição de aresta segura
- A escolha da aresta é dependente do algoritmo como veremos a seguir
- A definição de aresta segura requer a definição de outros conceitos

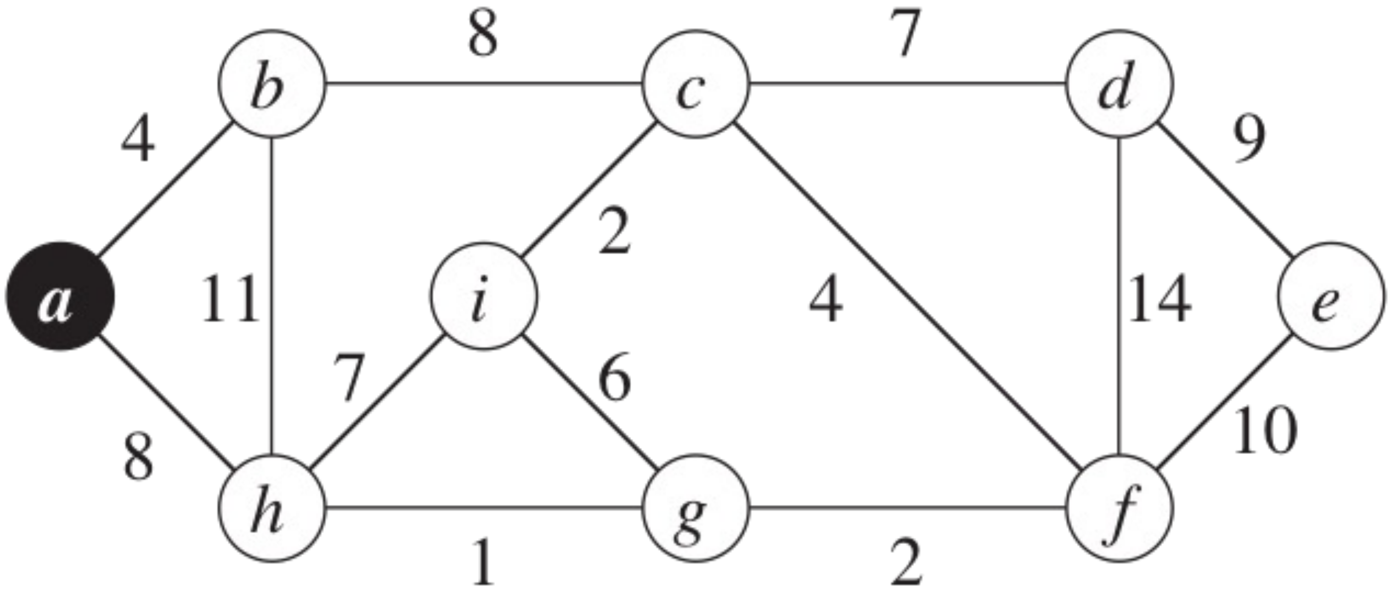
Corte e cut-set

- Seja X e Y uma partição do conjunto de vértices V ($X \cap Y = \emptyset, X \cup Y = V$)
- Tal partição é chamada de **corte (cut)** em grafos
- Um **cut-set** de um grafo é um conjunto de arestas cuja remoção desconecta o grafo
  - $C = \{(u, v) \in E \mid u \in X \land v \in Y\}$
- **Teorema**: Seja G=(V,E) uma grafo com pesos e um corte X, Y de G. Seja (u,v) a aresta do cut-set dessa partição com menor peso. A aresta (u,v) pertence a uma árvore geradora mínima. Logo (u,v) é uma aresta segura.

Corte e cut-set

- **Prova**: Seja T uma árvore geradora mínima de G. Se T não contém (u,v), então sua adição a T forma um ciclo.
- Deve existir uma aresta (w,z) nesse ciclo pertencente ao cut-set de X e Y ($w \in X$ e $z \in Y$).
- Pela escolha de (u,v), sabe-se que $p(u,v) \leq p(w,z)$
- Dessa forma, se trocarmos (w,z) por (u,v) obtemos uma árvore geradora mínima com custo não superior a T

Corte e cut-set



Algoritmo de Prim

- O algoritmo de Prim é uma modificação do algoritmo genérico que, a todo instante, escolhe a aresta de menor peso no cut-set entre os vértices que pertencem atualmente à árvore e os demais
- Em outras palavras, o algoritmo inicia com a escolha de uma origem/raiz da árvore a partir da qual novas arestas são escolhidas e vértices adicionados ao conjunto solução
- Como o algoritmo envolve muitos detalhes de implementação, o discutiremos apenas em pseudocódigo

Algoritmo Prim

- Criaremos vetores auxiliares para:
  - Identificar os antecessores dos vértices na árvore
  - Associar um rótulo com a aresta de menor peso que conecta o vértice à árvore até o momento
  - Identificar vértices que já foram incluído na árvore
- Os rótulos associados aos vértices serão usados como chaves para construir um heap dos vértices
  - Suas posições serão armazenadas em um vetor auxiliar para indicar a posição de um vértice qualquer no heap (evitar busca sequencial no heap quando for atualizar os pesos/rótulos associados ao vértice)

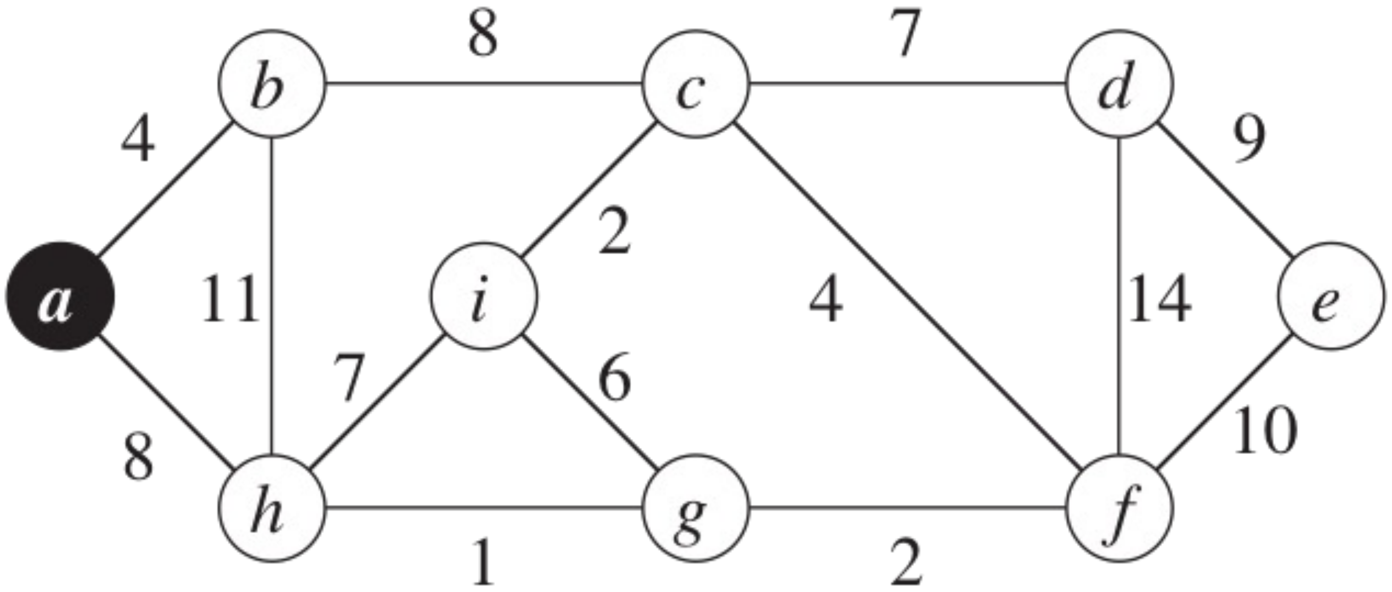
Algoritmo Prim

- Iniciaremos a construção da árvore a partir do vértice 0
- Inicializamos os rótulos dos vértices como infinito, exceto 0 que recebe rótulo 0
- Construimos um heap mínimo com todos os vértices, usando o rótulo como chave

Algoritmo Prim

- Enquanto o heap não estiver vazio
  - Remover o menor elemento (min) e marcá-lo
  - Para cada vértice v adjacente a min
    - Se v não estiver marcado e p(min,v) < rotulo[v]
      - Antecessor[v] = min
      - Heap.atualizaChave(v,p(min,v))
      - rotulo[v] = p(min,v)

Algoritmo Prim



Algoritmo de Prim

- A atribuição dos rótulos iniciais e a construção da fila de prioridade tem custo O(|V|)
- O laço principal roda O(|V|) vezes
  - Cada iteração remove o menor elemento do heap com custo O(log |V|)
  - A cada novo elemento explorado, todas as suas arestas são avaliadas para atualizar o custo mínimo (O(|E|))
    - A atualização dos custos modifica o heap (O(log |V|))
- Portanto, o custo final é O(|V| log |V| + |E| log |V|), ou seja, O(|E| log |V|)

Algoritmo de Prim

- Agora vamos verificar a corretude do algoritmo
- Para isso, vamos verificar que, em cada iteração, somente arestas seguras são escolhidas
- Sempre que um vértice é removido do heap e marcado, fixamos seu antecessor
  - Isso é equivalente a inserir uma aresta em S no algoritmo genérico
- Antes de executarmos uma nova iteração, sabemos que S respeita o corte (V-heap, heap)
  - Inicialmente, a árvore só tem a raiz e nenhuma aresta, e o heap tem o restante dos vértices
- Como o heap é mínimo, escolhemos a aresta de menor custo para inserirmos em S. Portanto, ela é segura.
- Assim, ao concluirmos a iteração, mantemos o invariante que S respeita o corte

Leitura

- Seções 23.1 e 23.2 (CLRS)