Teoria dos Grafos

Árvore de Cobertura Mínima

Árvore de Cobertura - Introdução

• **Problema 1:** Os pinos de uma placa de circuito impresso devem ser conectados com a menor quantidade de fio.

• **Problema 2:** No sistema de abastecimento de água de Campina Grande, existem vários tanques de armazenamento e tratamento da água que vem do Açude de Boqueirão. Como interligar os tanques (em princípio, qulalquer par de tanques pode ser interligado), de modo a garantir o correto abastecimento e que o custo seja mínimo?

Árvore de Cobertura - Introdução

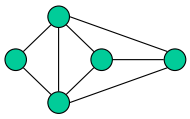
• **Problema 1:** Os pinos de uma placa de circuito impresso devem ser conectados com a menor quantidade de fio.

• **Problema 2:** No sistema de abastecimento de água de Campina Grande, existem vários tanques de armazenamento e tratamento da água que vem do Açude de Boqueirão. Como interligar os tanques (em princípio, qulalquer par de tanques pode ser interligado), de modo a garantir o correto abastecimento e que o custo seja mínimo?

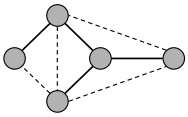
Os dois problemas acima são conhecidos como o problema da conexão mínima. Na Teoria dos Grafos é o problema de encontrar a árvore de cobertura (geradora) mínima para o grafo.

O que é uma árvore de cobertura?

- Uma *árvore de cobertura* de um grafo não dirigido G=(V,E) é um subgrafo de G que é uma árvore e contém todos os vértices de G.



Grafo



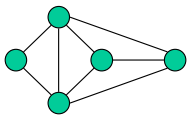
Árvore de Cobertura

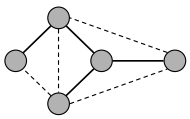
Árvore de cobertura

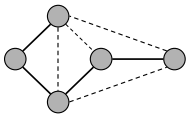
- Um grafo pode ter mais de uma árvore de cobertura?
- Um grafo não conectado pode ter uma árvore de cobertura?

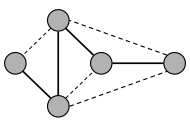
Árvore de cobertura

- Um grafo pode ter mais de uma árvore de cobertura?
- Um grafo não conectado pode ter uma árvore de cobertura?







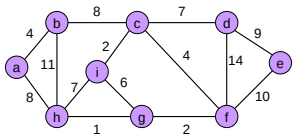


Como Encontrar uma Árvore de Cobertura?

- Para encontrar uma árvore de cobertura:
  - Se o grafo G não tem ciclos, G é uma árvore de cobertura.
  - Se G tem ciclo, é necessário remover recursivamente arestas (até achar uma árvore), mantendo o grafo conectado.

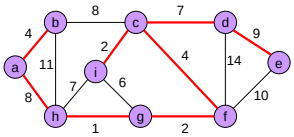
Árvore de Cobertura Mínima (MST)

- O *custo* de um subgrafo (de um grafo não dirigido ponderado) é a soma dos pesos de suas arestas.
- Uma *árvore de cobertura mínima* de um grafo não dirigido ponderado é uma árvore de cobertura com menor custo.
- Um grafo pode ter mais de uma árvore de cobertura mínima?



Árvore de Cobertura Mínima (MST)

- O *custo* de um subgrafo (de um grafo não dirigido ponderado) é a soma dos pesos de suas arestas.
- Uma *árvore de cobertura mínima* de um grafo não dirigido ponderado é uma árvore de cobertura com menor custo.
- Um grafo pode ter mais de uma árvore de cobertura mínima?



Como Encontrar uma MST?

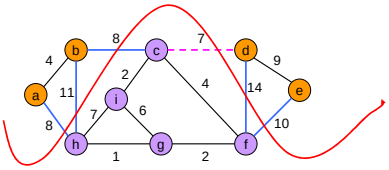
Algoritmo Genérico

1. A ← ∅
2. **while** A não é uma árvore de cobertura
3. **do** encontre uma aresta (u,v) que é segura para A
4. A ← A ∪ {(u,v)}
5. **return** A

Como Reconhecer uma Aresta Segura?

- Definir um corte no grafo, particionando o conjunto de vértices.
- Uma aresta cruza o corte se cada um dos vértices que formam a aresta está em partição diferente.
- O corte deve respeitar **A**. Isso acontece se nenhuma aresta de **A** cruza o corte.
- A aresta segura é aquela de menor peso que cruza o corte.

Aresta Segura



Algoritmos para Encontrar MST

Kruskal: Encontra a aresta segura e adiciona a árvore de cobertura que está sendo formada. A nova aresta não deve necessariamente ter um dos vértices na árvore que está sendo formada. Ou seja, uma floresta pode existir antes da MST ter sido encontrada.

Prim: Encontra a aresta segura e um dos vértices necessariamente deve pertencer a árvore que está sendo construída. Sempre existe uma única árvore parcial.

Algoritmo de Kruskal

```
1.  A ← ∅
2.  for cada vértice v ∈ V[G]
3.      do Make-Set(v)
4.  Ordene as arestas de E (ordem crescente por peso w)
5. ∀ edge (u,v) ∈ E, (considerando a ordem)
6.      if Find-Set(u) ≠ Find-Set(v)
7.              then A ← A ∪ {(u,v)}
8.                   Union(u,v)
9. return A
```


Algoritmo de Prim

```
1.  for cada vértice u ∈ V[G]
2.      do  chave[u] ← ∞ // chave[u] é o custo de u para vértice da árvore
3.          π[u] ← NIL
4.  chave[r] ← 0        // r é a raiz da árvore
5. Q ← V[G]  // Q contém todos os vértices que ainda não estão na árvore
6.  while Q ≠ ∅
7.      do  u ← Extract-Min[Q]
8.          for cada v ∈ Adj[u]
9.              do if v ∈ Q ∧ w(u,v) < chave[v]
10.             then  π[v] ← u
11.                   chave[v] ← w(u,v)
```