Algoritmos e Estruturas de Dados II - SCC-203

Grafos: Árvores Geradoras Mínimas

Gustavo Batista

Árvores Geradoras Mínimas

- Imagine um projeto de redes de comunicações conectando *n* localidades.
- Deseja-se encontrar *n-1* conexões, cada uma ligando 2 cidades, de forma que todas as cidades sejam conectadas.
- O objetivo é encontrar entre todas as possibilidades, as *n-1* conexões que utilizam a menor quantidade de cabos.

Árvores Geradoras Mínimas

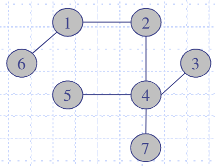
- Modelagem:
  - $G = (V, A)$: grafo conectado, não direcionado.
  - V: conjunto de cidades.
  - A: conjunto de possíveis conexões
  - $p(u,v)$: peso da aresta $(u,v) \in A$, ou seja o custo de cabo para conectar u a v.
- Objetivo:
  - Encontrar um subconjunto $T \subseteq A$, acíclico, que conecta todos os vértices de G e cujo peso total é minimizado.

Árvores Geradoras Mínimas

- Como $G = (V,T)$ é acíclico e conecta todos os vértices, T forma uma árvore chamada *árvore geradora (spanning tree)* de G.
- Sendo T um conjunto de arestas de forma a soma dos pesos é mínima, então G é uma *árvore geradora mínima (minimum spanning tree)* (AGM/MST).

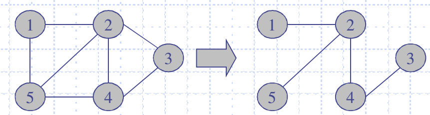
Definições: Árvore Livre

- Um grafo é uma árvore livre se for um grafo não direcionado acíclico e conectado. É comum dizer apenas que o grafo é uma árvore omitindo o "livre".



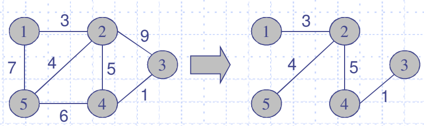
Definições: Árvore Geradora

- Uma árvore geradora (*spanning tree*) de um grafo conectado *G = (V,A)* é um subgrafo que contém todos os vértices de *G* e forma uma árvore.



Definições: Árvore Geradora Mínima

- Uma árvore geradora mínima (*minimum spanning tree*) de um grafo conectado *G = (V,A)* é uma árvore geradora cuja soma dos pesos da aresta é mínima.



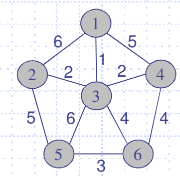
Introdução

- Existem dois algoritmos bastante conhecidos para encontrar um AGM de um grafo não direcionado *G:*
  - Algoritmo de Prim;
  - Algoritmo de Kruskal.

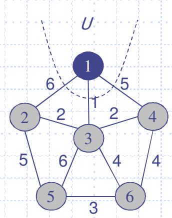
Árvore Geradora Mínima: Algoritmo de Prim

- Idéia geral para um algoritmo:
  1. Começar um vértice *v* qualquer, e adicioná-lo a um conjunto *U*;
  2. Escolher a aresta que conecta um vértice em *U* a um vértice em *V-U* tal que o peso é mínimo.
  3. Inclui o vértice da aresta escolhida em *U*.
  4. Vai para 2 enquanto $U \neq V$.

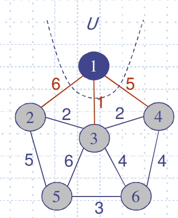
Árvore Geradora Mínima: Algoritmo de Prim



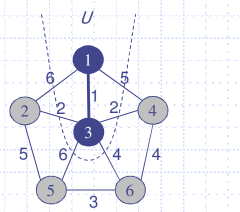
Árvore Geradora Mínima: Algoritmo de Prim



Árvore Geradora Mínima: Algoritmo de Prim

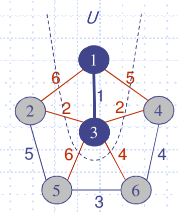


Árvore Geradora Mínima: Algoritmo de Prim



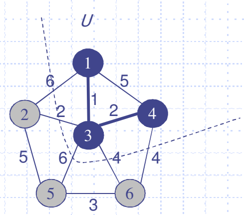
13

Árvore Geradora Mínima: Algoritmo de Prim



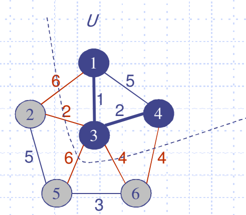
14

Árvore Geradora Mínima: Algoritmo de Prim



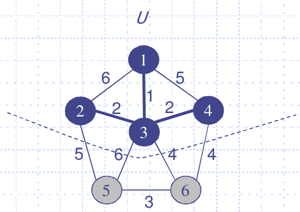
15

Árvore Geradora Mínima: Algoritmo de Prim



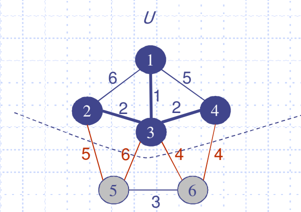
16

Árvore Geradora Mínima: Algoritmo de Prim



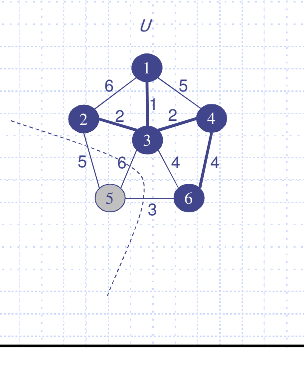
17

Árvore Geradora Mínima: Algoritmo de Prim



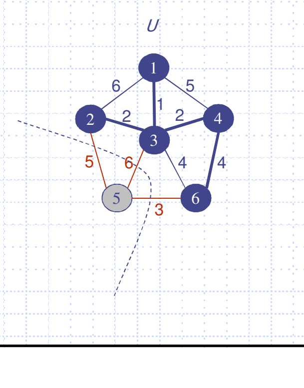
18

Árvore Geradora Mínima: Algoritmo de Prim



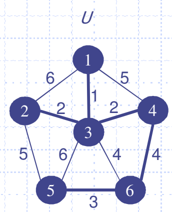
19

Árvore Geradora Mínima: Algoritmo de Prim

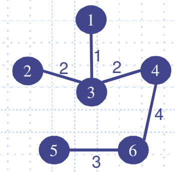


20

Árvore Geradora Mínima: Algoritmo de Prim

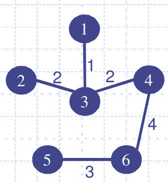


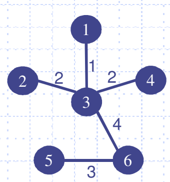
Árvore Geradora Mínima: Algoritmo de Prim



Árvore Geradora Mínima: Algoritmo de Prim

- Dado um grafo *G*, pode haver mais de uma árvore geradora mínima para *G*.





Árvore Geradora Mínima: Algoritmo de Prim

```
procedimento Prim(var Grafo: TGrafo;
                  var T: conjunto de arestas)
variáveis
       u, v: TVertice;
       U: conjunto de TVertice;
início
       T := ∅;
       U := {1};
       enquanto U ≠ V faça
       início
              seja (u, v) a aresta de menor peso tal que
                     u ∈ U e v ∈ V-U
              T := T ∪ {(u, v)};
              U := U ∪ {v};
       fim
fim
```

Algoritmo de Prim: Complexidade

- Para analisar a eficiência do algoritmo de Prim é necessário definir como será feita a seleção da aresta *(u, v)*.
- Uma implementação simples utiliza dois vetores. O primeiro *prox[i]* fornece o vértice em *U* atualmente mais próximo ao vértice *i* em *V-U*.
- O segundo vetor *mc[i]* fornece o custo da aresta *(i, prox[i])*.

25

Algoritmo de Prim: Complexidade

- A operação de encontrar *(u, v)* pode ser realizada em $O(|V|)$, percorrendo o vetor *mc*.
- Também é necessário atualizar os vetores *prox* e *mc* a cada novo vértice em *U*. Essa operação é $O(|V|)$.
- Portanto, essa implementação do algoritmo de Prim é $O(|V|^2)$.

26

Algoritmo de Prim: Complexidade

- Uma implementação mais sofisticada utiliza uma fila de prioridade para manter os vértices em *V-U*.
- A chave da fila de prioridade de um vértice $v \in V\text{-}U$ é o peso da aresta mais leve que liga *v* a um vértice de *U*.
- Se a fila de prioridade for implementada com um *heap*, então a complexidade do algoritmo de Prim é $O(|A| \log |V|)$.

27

Árvore Geradora Mínima: Algoritmo de Prim

```
void prim(tvertice v, tgrafo *grafo, tgrafo *agm) {
    std::priority_queue<taresta> heap;
    tpeso d, peso; tvertice u, w; tapontador p;
    int i, marc[MAXNUMVERTICES];

    inicializa_grafo(agm, grafo->num_vertices);
    for (i = 0; i < grafo->num_vertices; i++) {
        marc[i] = BRANCO;
    }
    marc[v] = PRETO;
    p = primeiro_adj(v, grafo);
    while (p != NULO) {
        recupera_adj(v, p, &w, &peso, grafo);
        heap.push(cria_aresta(peso, v, w));
        p = proximo_adj(v, p, grafo);
    }
```

28

Árvore Geradora Mínima: Algoritmo de Prim

```
void prim(tvertice v, tgrafo *grafo, tgrafo *agm) {
    std::priority_queue<taresta> heap;
    tpeso d,
    int i, ma

    iniciali
    for (i =
        marc
    }
    marc[v] =
    p = primeiro_adj(v, grafo);
    while (p != NULO) {
        recupera_adj(v, p, &w, &peso, grafo);
        heap.push(cria_aresta(peso, v, w));
        p = proximo_adj(v, p, grafo);
    }
```

```
struct taresta{
    tpeso peso;
    tvertice orig, dest;

    bool operator<(const taresta &a) const {
        return peso>a.peso;
    }
};
```


Árvore Geradora Mínima: Algoritmo de Prim

```
void prim(tvertice v, tgrafo *grafo, tgrafo *agm) {
    std::priority_queue<taresta> heap;
    tpeso d, peso; tvertice u, w; tapontador p;
    int i, marc[MAXNUMVERTICES];
```

```
taresta cria_aresta(tpeso peso, tvertice orig, tvertice dest) {
    taresta aresta;

    aresta.peso = peso;
    aresta.orig = orig;
    aresta.dest = dest;
    return aresta;
}
```

```
        heap.push(cria_aresta(peso, v, w));
        p = proximo_adj(v, p, grafo);
    }
```


Árvore Geradora Mínima: Algoritmo de Prim

```
void prim(tvertice v, tgrafo *grafo, tgrafo *agm) {
    std::priority_queue<taresta> heap;
    tpeso d, peso; tvertice u, w; tapontador p;
    int i, marc[MAXNUMVERTICES];

    inicializa_grafo(agm, grafo->num_vertices);
    for (i = 0; i < grafo->num_vertices; i++) {
        marc[i] = BRANCO;
    }
    marc[v] = PRETO;
    p = primeiro_adj(v, grafo);
    while (p != NULO) {
        recupera_adj(v, p, &w, &peso, grafo);
        heap.push(cria_aresta(peso, v, w));
        p = proximo_adj(v, p, grafo);
    }
```


Árvore Geradora Mínima: Algoritmo de Prim

```
    while (!heap.empty()) {
        v = heap.top().orig; w = heap.top().dest;
        peso = heap.top().peso; heap.pop();
        if (marc[w] == PRETO) continue;
        insere_aresta(v, w, peso, agm);
        marc[w] = PRETO;
        p = primeiro_adj(w, grafo);
        while (p != NULO) {
            recupera_adj(w, p, &u, &peso, grafo);
            heap.push(cria_aresta(peso, w, u));
            p = proximo_adj(w, p, grafo);
        }
    }
}
```

Árvore Geradora Mínima: Algoritmo de Kruskal

- Uma segunda forma de encontrar uma árvore geradora mínima de um grafo *G = (V, A)* conectado conhecida como algoritmo de Kruskal.
- Inicia-se com um grafo *G′ = (V, ∅)*.
- Cada vértice é um componente conectado de si mesmo.
- A cada iteração do algoritmo, são construídos componentes conectados cada vez maiores.

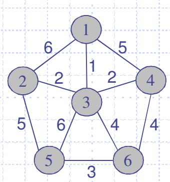
33

Árvore Geradora Mínima: Algoritmo de Kruskal

- Para crescer os componentes conectados, os vértices em *A* são analisados por ordem ascendente de peso.
- Se uma aresta conecta dois vértices em dois componentes separados, então a aresta é adicionada a *T*.
- Se uma aresta conecta dois vértices do mesmo componente, então ela é descartada, pois criaria um ciclo.

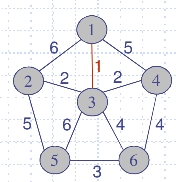
34

Árvore Geradora Mínima: Algoritmo de Kruskal



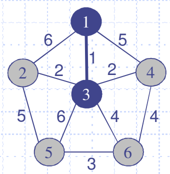
35

Árvore Geradora Mínima: Algoritmo de Kruskal

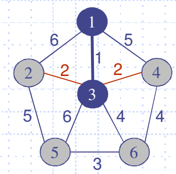


36

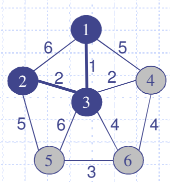
Árvore Geradora Mínima: Algoritmo de Kruskal



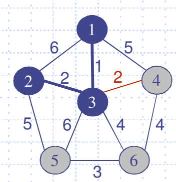
Árvore Geradora Mínima: Algoritmo de Kruskal



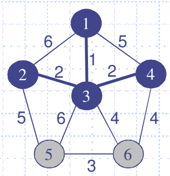
Árvore Geradora Mínima: Algoritmo de Kruskal



Árvore Geradora Mínima: Algoritmo de Kruskal

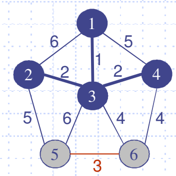


Árvore Geradora Mínima: Algoritmo de Kruskal



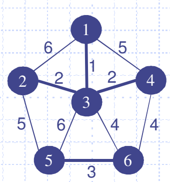
41

Árvore Geradora Mínima: Algoritmo de Kruskal



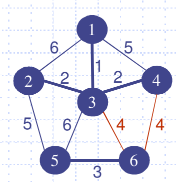
42

Árvore Geradora Mínima: Algoritmo de Kruskal



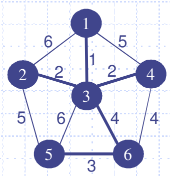
43

Árvore Geradora Mínima: Algoritmo de Kruskal



44

Árvore Geradora Mínima: Algoritmo de Kruskal



45

Árvore Geradora Mínima: Algoritmo de Kruskal

```
procedimento Kruskal(var Grafo: TGrafo;
                     var T: conjunto de arestas)
variáveis
        u, v: TVertice;
        U1,...,Un: conjunto de TVertice;
        Q: fila de prioridade;
início
        T := ∅;
        Q := as arestas de G ordenadas pelo seu peso;
        para i:=1 até Grafo.NumVertices faça
                Ui := {i};
        enquanto houver arestas em Q faça
        início
                seja (u, v) a aresta de menor peso de Q tal que
                        u ∈ Up e v ∈ Uq e Up ∩ Uq = ∅
                T := T ∪ {(u, v)};
                Up := Up ∪ Uq;
                eliminar Uq;
        fim
fim
```

Algoritmo de Kruskal: Complexidade

- O desempenho algoritmo de Kruskal depende de dois fatores principais:
  - Encontrar a aresta de menor peso;
  - Verificar a aresta conecta dois componentes distintos ($U_p \cap U_q$).
- Se *Q* for implementada como uma fila de prioridade com um *heap* e a operação de conjuntos for eficiente, então o algoritmo é *O(|A| log |A|)*.

47

Exercício

- Implementar o algoritmo de Kruskal utilizando as operações do TAD Grafo e STL.

48

Problemas com Grafos: Rede Ótica

Problema: Rede Ótica

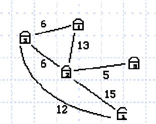
Os caciques da região de Tutuaçu pretendem integrar suas tribos à chamada "aldeia global". A primeira providência foi a distribuição de telefones celulares a todos os pajés. Agora, planejam montar uma rede de fibra ótica interligando todas as tabas. Esta empreitada requer que sejam abertas novas picadas na mata, passando por reservas de flora e fauna. Conscientes da necessidade de preservar o máximo possível o meio ambiente, os caciques encomendaram um estudo do impacto ambiental do projeto. Será que você consegue ajudá-los a projetar a rede de fibra ótica?

Problemas com Grafos: Rede Ótica

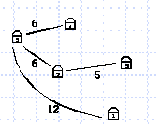
Tarefa

Vamos denominar uma ligação de fibra ótica entre duas tabas de um ramo de rede. Para possibilitar a comunicação entre todas as tabas é necessário que todas elas estejam interligadas, direta (utilizando um ramo de rede) ou indiretamente (utilizando mais de um ramo). Os caciques conseguiram a informação do impacto ambiental que causará a construção dos ramos. Alguns ramos, no entanto, nem foram considerados no estudo ambiental, pois sua construção é impossível.

Problemas com Grafos: Rede Ótica



Ramos de rede possíveis com custo ambiental associado



Interligação das tabas com menor custo ambiental

Sua tarefa é escrever um programa para determinar quais ramos devem ser construídos, de forma a possibilitar a comunicação entre todas as tabas, causando o menor impacto ambiental possível.

Problemas com Grafos: Rede Ótica

Entrada

```
5 6      <= Nr. tabas e conexões
1 2 15
1 3 12
2 4 13
2 5 5
3 2 6
3 4 6
```

Saída

```
1 3
2 3
2 5
3 4
```