SCC0216 - Modelagem Computacional em Grafos

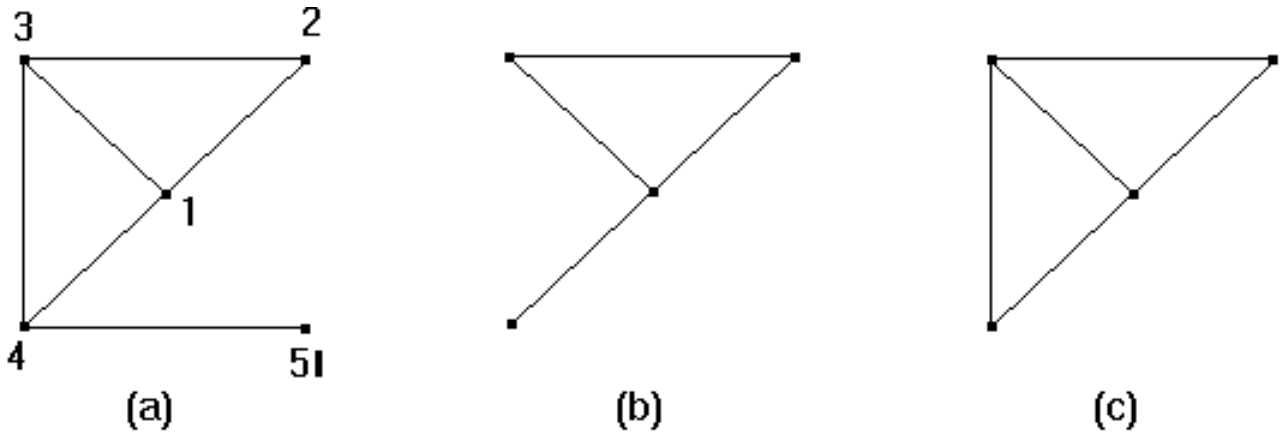
Árvores Geradoras Mínimas

Prof. Alneu (alneu@icmc.usp.br) / Profa. Rosane (rminghim@icmc.usp.br)
PAE: Alan (alan@icmc.usp.br) / Henry (henry@icmc.usp.br)

Baseado no material de aula original: Profª. Josiane M. Bueno

Sub-grafo

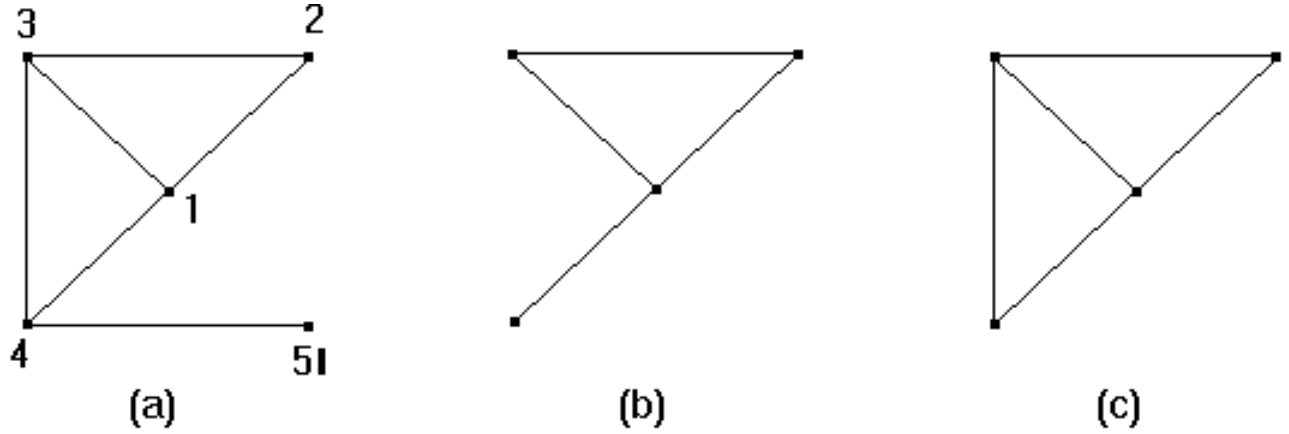
Um sub-grafo $G_2(V_2,E_2)$ de um grafo $G_1(V_1,E_1)$ é um grafo tal que V_2 está contido em V_1 e E_2 está contido em E_1



b e c são subgrafos de a

Sub-grafo induzido

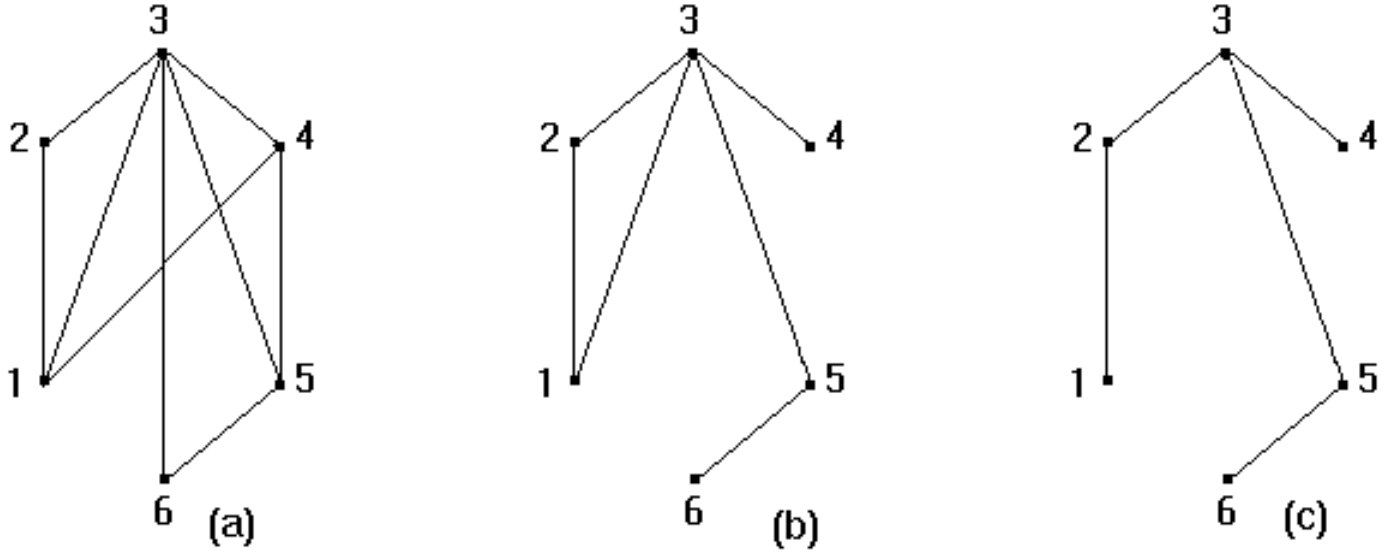
Se o sub-grafo G_2 de G_1 satisfaz: para quaisquer ***v***, ***w*** pertencentes a V_2, se **(*v*,*w*)** pertence a E_1, então **(*v*,*w*)** também pertence a E_2. Dessa forma, G_2 é dito sub-grafo induzido pelo conjunto de vértices V_2



b e c são sub-grafos de a, mas apenas c é sub-grafo induzido

Sub-grafo gerador

Sub-grafo Gerador ou sub-grafo de espalhamento de um grafo $G_1(V_1,E_1)$ é um sub-grafo $G_2(V_2,E_2)$ de G_1 tal que $V_1=V_2$. Quando o sub-grafo gerador é uma árvore, ele recebe o nome de árvore geradora (ou de espalhamento).



b e c são sub-grafos geradores de a; c é árvore geradora de a e b

Sub-grafo gerador de custo mínimo

- Formalmente...
- Dado um grafo não-orientado $G(V,E)$
  - onde w: $E \rightarrow \Re^+$ define os custos das arestas
  - queremos encontrar um sub-grafo gerador conexo T de G tal que, para todo sub-grafo gerador conexo T' de G

$$\sum_{e \in T} w(e) \leq \sum_{e \in T'} w(e)$$

Árvore geradora mínima (MST)

- Claramente, o problema só tem solução se *G* é conexo
- A partir de agora, assumimos *G* conexo
- Também não é difícil ver que a solução para esse problema será sempre uma árvore
  - Basta notar que *T* não terá ciclos pois, poderíamos obter um outro sub-grafo *T'*, ainda conexo e com custo menor que o de *T*, removendo o ciclo!

Árvore geradora mínima

- Árvore Geradora (*Spanning Tree*) de um grafo *G* é um sub-grafo de *G* que contém todos os seus vértices e, ainda, é uma árvore

- Árvore Geradora Mínima (*Minimum Spanning Tree* – MST) é a árvore geradora de um grafo valorado cuja soma dos pesos associados às arestas é mínimo, i.e., é uma árvore geradora de custo mínimo

Porque é um problema interessante?

- Suponha que queremos construir estradas para interligar *n* cidades
  - Cada estrada direta entre as cidades *i* e *j* tem um custo associado
  - Nem todas as cidades precisam ser ligadas diretamente, desde que todas sejam acessíveis
- Como determinar eficientemente quais estradas devem ser construídas de forma a minimizar o custo total de interligação das cidades?

Árvore geradora mínima (MST)

Como encontrar a árvore geradora mínima de um grafo *G*?

- Algoritmo Genérico
- Algoritmo de Prim
- Algoritmo de Kruskal

Árvore geradora mínima

Algoritmo Genérico

```
Generic-MST(G)
A = ∅
While A não define uma spanning tree
    encontre uma aresta (u,v) segura para A
    A = A ∪ {(u,v)}
Return A
```

- A = conjunto de arestas
- G conexo, não direcionado, ponderado
- Abordagem ‘gulosa’, MST cresce uma aresta por vez
- Aresta é ‘segura’ se mantém a condição: antes de cada iteração, A é um sub-conjunto de alguma MST

Algoritmo de Prim

```
{ Gera uma Minimum Spanning Tree do
  grafo ponderado G - Algoritmo de Prim}

Prim-MST (G)
   Escolha um vértice s para iniciar a árvore
   enquanto "Há vértices que não estão na árvore"
      Selecione a aresta com menor peso adjacente
      a um vértice pertencente à árvore e a outro
      não pertencente à árvore

      Insira a aresta selecionada e o respectivo
      vértice na árvore
   fim-enquanto
```

Algoritmo de Prim

Inicia em um determinado vértice e gera a árvore, uma aresta por vez

- Complexidade (tempo): $O(n.m)$

  n: número de vértices

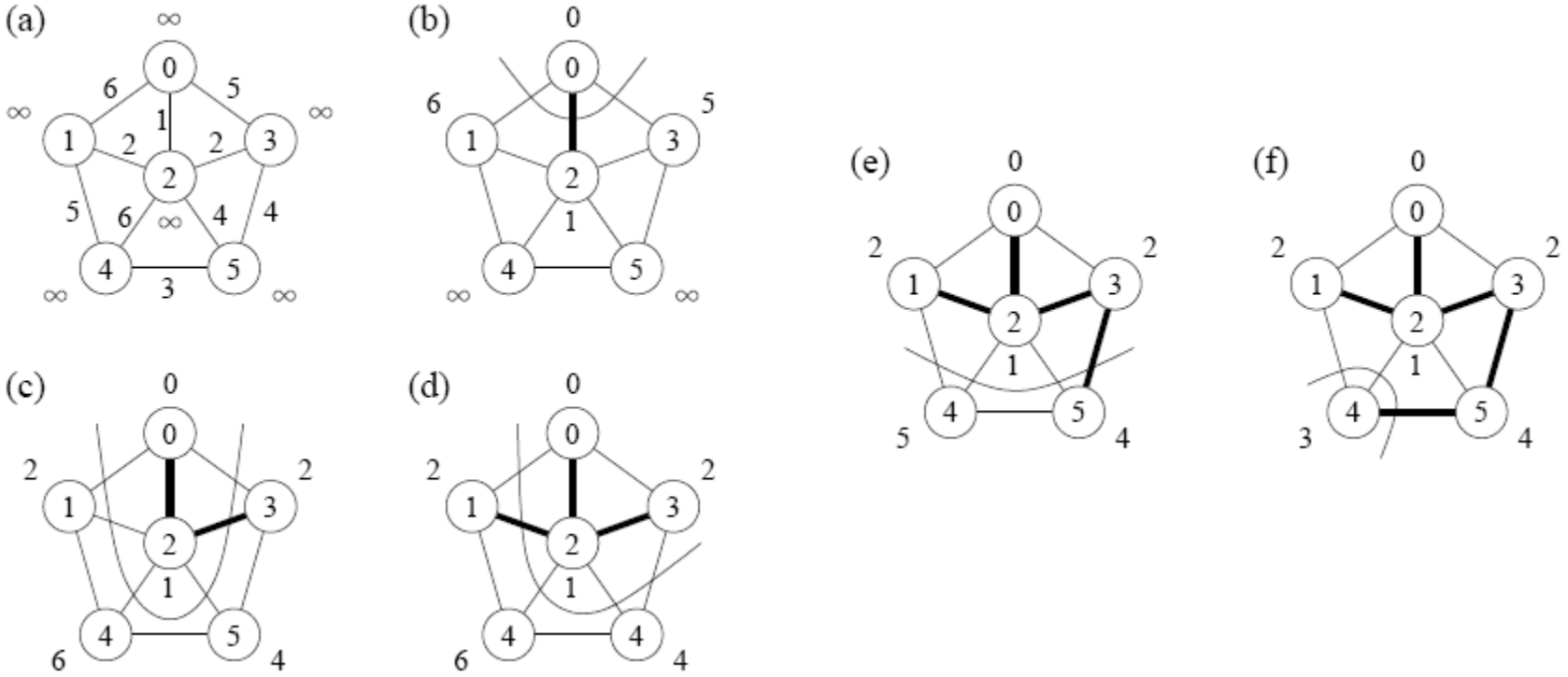
  m: número de arestas

Algoritmo de Prim

- Maneira mais eficiente de determinar a aresta de menor peso a partir de um dado vértice
  - manter todas as arestas que ainda não estão na árvore em uma fila de prioridade (*heap*)
  - prioridade é dada à aresta de menor peso adjacente a um vértice na árvore e outro fora dela

  Complexidade (tempo): $O(m.\log(n))$

Algoritmo de Prim



Algoritmo de Prim

s:origem

```
Inicialize a fila de Prioridades fp com todos o nó s
Inicializa  peso(v) como INFINITO para todo v, exceto s
Inicializa  peso(s) como 0
Inicialize antecessor(v) como -1 para todo v
Enquanto não vazia (fp)
   v <- primeiro (fp)
   elimina (v,fp)
   Enquanto u <- prox_adj(v) não nulo
              Se na_fila(fp,v) e w(u,v) <= peso(v) então
                 antecessor[v] = u
                 peso(v) = w(u,v)
              fim-se
   fim-enquanto
fim-enquanto
```

Prim

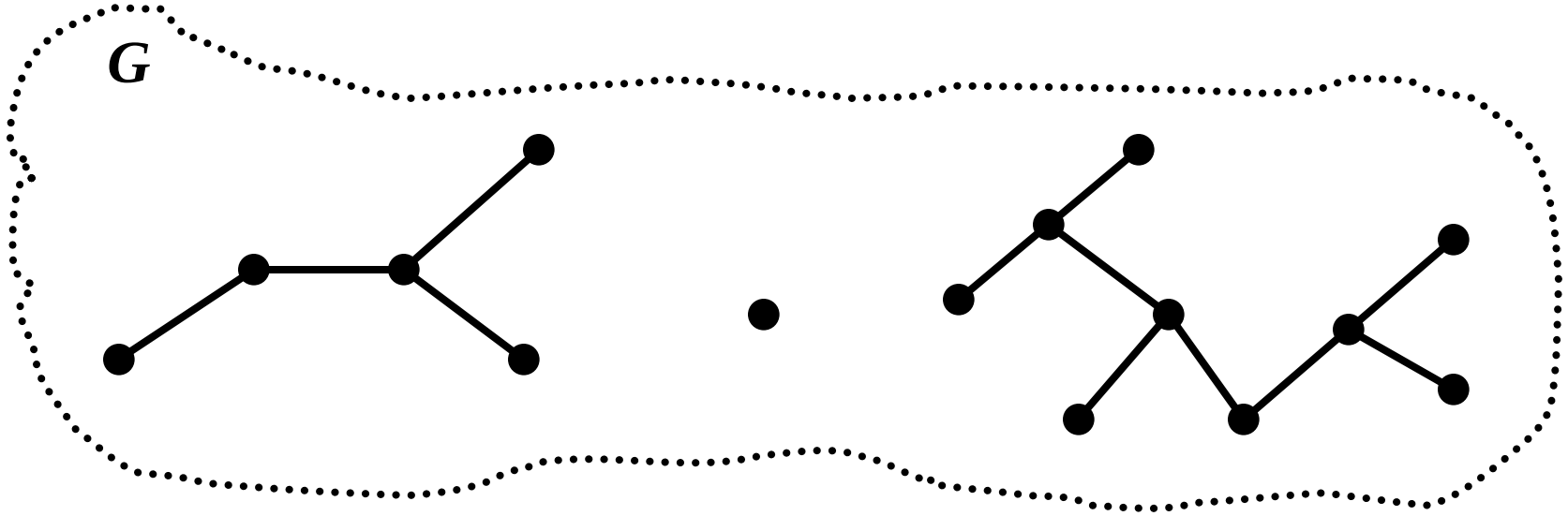
Implementação Ziviani

```
begin { AgmPrim }
  for u := 0 to Grafo.NumVertices do
    begin {Constroi o heap com todos os valores igual a Infinito}
    Antecessor[u] := -1; p[u] := Infinito;
    A[u+1].Chave := u; {Heap a ser construido}
    ItensHeap[u] := true; Pos[u] := u+1;
    end;
  n := Grafo.NumVertices;
  p[Raiz] := 0;
  Constroi(A);
  while n >= 1 do {enquanto heap nao vazio}
    begin
    u := RetiraMin(A).Chave;
    if (u <> Raiz)
    then write('Aresta de arvore: v[',u,'] v[',Antecessor[u],']');readln;
    ItensHeap[u] := false;
    if not ListaAdjVazia(u,Grafo)
       then begin
            Aux := PrimeiroListaAdj(u,Grafo); FimListaAdj := false;
            while not FimListaAdj do
              begin
              ProxAdj(u, Grafo, v, Peso, Aux, FimListaAdj);
              if ItensHeap[v] and (Peso < p[v])
              then begin
                   Antecessor[v] := u; DiminuiChave(Pos[v],Peso,A);
                   end
              end;
            end;
    end;
end; { AgmPrim }
```

Kruskal

Floresta

- Uma Floresta é um conjunto de árvores.



Algoritmo de Kruskal

- Mais eficiente que Prim em grafos esparsos
- Não inicia em nenhum vértice em particular
  - Considera se cada aresta individualmente pode ou não pertencer à árvore geradora mínima, analisando-as em ordem crescente de custo
  - As árvores que compõem a floresta são identificadas pelos conjuntos S_i, que contém os vértices que a compõem
  - Ao final do processo, o conjunto E_T contém a solução do problema, i.e., a MST
  - Complexidade: O(m.log(m))
    - Se o teste $S_p \cap S_q = \varnothing$ for bem implementado; esse teste garante que a inclusão de e em E_T não introduz um ciclo

Algoritmo de Kruskal

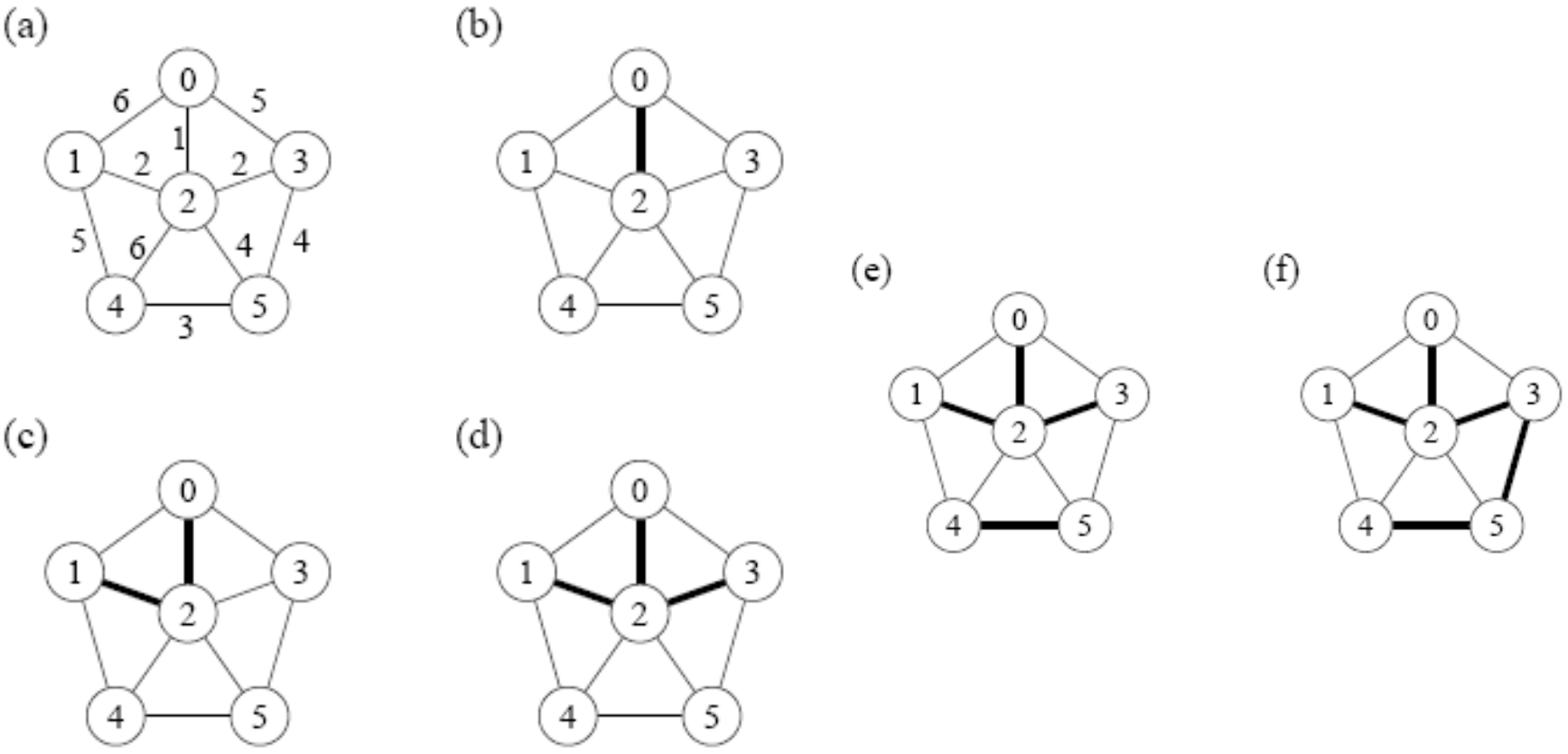
Basicamente, o algoritmo consiste em:

"Incluir em E_T todas as arestas de E em ordem crescente de peso, rejeitando, contudo, cada uma que forma ciclos com as arestas já em E_T."

- Pode ser interpretado como sendo a construção de uma árvore geradora a partir de uma floresta.

- Estado inicial: corresponde à floresta formada por n árvores triviais (um só vértice cada), i.e., $E_T = \varnothing$

Algoritmo de Kruskal – exemplo Ziviani



Algoritmo de Kruskal - Ziviani

- Sejam C1 e C2 duas árvores conectadas por (u, v):

  - Como (u, v) tem de ser uma aresta leve conectando C1 com alguma outra árvore, (u, v) é uma aresta segura para C1.
  - É guloso porque, a cada passo, ele adiciona à floresta uma aresta de menor peso.
  - Obtém uma AGM adicionando uma aresta de cada vez à floresta e, a cada passo, usa a aresta de menor peso que não forma ciclo.
  - Inicia com uma floresta de |V| árvores de um vértice: em |V| passos, une duas árvores até que exista apenas uma árvore na floresta.

Algoritmo de Kruskal - Ziviani

- Usa fila de prioridades para obter arestas em ordem crescente de pesos.

- Testa se uma dada aresta adicionada ao conjunto solução *S* forma um ciclo.

- Tratar **conjuntos disjuntos**: maneira eficiente de verificar se uma dada aresta forma um ciclo. Utiliza estruturas dinâmicas. Sempre unindo árvores disjuntas, árvores são obtidas.

Algoritmo de Kruskal

```
{ Gera uma Minimum Spanning Tree do grafo ponderado G(V,E), conexo - Algoritmo de Kruskal}
Kruskal-MST (G)

  Definir conjuntos S_j:{v_j}, 1 ≤ j ≤ n, e E_T = ∅
  Insira as arestas de E em uma fila de prioridade Q, segundo o peso (ordem crescente)
  Enquanto houver arestas na fila faça
     e = unqueue(Q)
     Seja (v,w) o par de vértices extremos de e
     Se v ∈ S_p e w ∈ S_q, S_p ∩ S_q = ∅ então
          S_p = S_p ∪ {S_q}
          eliminar S_q
          E_T = E_T ∪ {e}
  Fim Enquanto
```