Árvores Espalhadas Mínimas

Disciplina: Laboratório de Programação II

Professor: Luciano Brum
Email: lucianobrum18@gmail.com
Site: https://sites.google.com/view/brumluciano

Assunto da aula de hoje:

Árvores Espalhadas Mínimas

Tópicos

- Conceito Básico.
- Algoritmo de Kruskal.
- Algoritmo de Prim.
- Resumo.

Conceitos Básicos

- Problema 1: em projeto de circuitos, é necessário tornar os pinos de vários componentes eletricamente equivalentes, juntando a fiação deles.

- Para interconectar ‘n’ pinos, podemos usar um arranjo de ‘n-1’ fios, cada um conectando 2 pinos.

- De todos arranjos possíveis, o que usar menos fio é o mais desejável.

Conceitos Básicos

- Problema 2: Uma companhia telefônica deseja criar uma rede interligando um conjunto de cidades.

- O custo para unir duas cidades por meio de cabos é conhecido mas, como toda boa companhia, esta deseja minimizar seus gastos, fazendo as ligações mais baratas possíveis sem deixar de cobrir nenhuma das cidades.

Conceitos Básicos

- É possível modelar esses problemas com grafos conectados não-orientados $G = (V,E)$, onde:
    - V é o conjunto de pinos, ou cidades.
    - E é o conjunto de interconexões possíveis entre pares de fios ou cabos.
    - Para cada aresta $(u,v) \in E$, temos um peso $w(u,v)$ especificando o custo para conectar u e v.

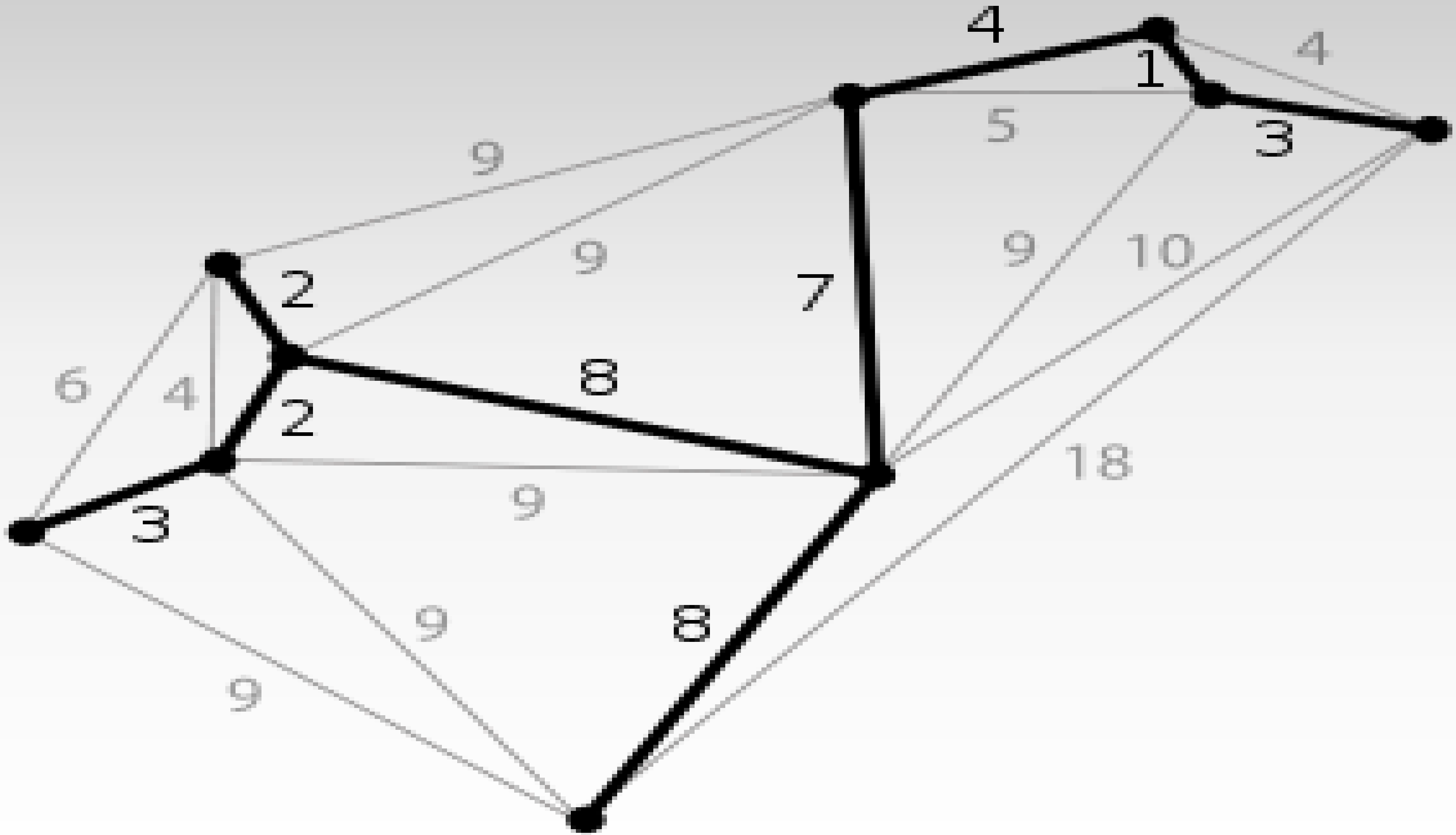
Conceitos Básicos

- Objetivo a ser alcançado: encontrar um subconjunto acíclico T ⊆ E que conecte todos os vértices e cujo peso total seja minimizado.

- w(T) = $\sum_{(u,v)\in T} w(u,v)$

- Como T é acíclico e conecta todos vértices, será formada uma árvore espalhada. O problema de determinar T é chamado de problema da árvore espalhada mínima (ou árvore de extensão mínima ou árvore geradora mínima).

Conceitos Básicos



Conceitos Básicos

- Serão examinados dois algoritmos para o cálculo da árvore geradora mínima:
  - Algoritmo de Kruskal;
  - Algoritmo de Prim;
- Ambos são algoritmos gulosos.

Conceitos Básicos

- Antes, será examinado um algoritmo genérico para o cálculo da árvore geradora mínima.

- **GENERIC-MST**(G,w)
- 1 A := {}
- 2 **while** A não formar uma árvore espalhada
- 3 **do** encontre uma aresta (u,v) que é segura para A
- 4 A := A **união** {(u,v)}
- 5 **return** A

Conceitos Básicos

```
GENERIC-MST(G,w)
1 A := {}
2 while A não formar uma árvore espalhada
3       do encontre uma aresta (u,v) que é segura para A
4           A := A união {(u,v)}
5 return A
```

- Inicialização: Na linha 1 inicializamos o conjunto de arestas como vazio.
- Manutenção: Na linha 2 temos um laço que a cada interação adiciona apenas arestas seguras e na linha 4 temos a operação que adiciona o vértice sendo analisado no conjunto de arestas A da árvore mínima.
- Término: Na linha 5 todas arestas foram adicionadas em uma árvore espalhada mínima.

Conceitos Básicos

```
GENERIC-MST(G,w)
1 A := {}
2 while A não formar uma árvore espalhada
3       do encontre uma aresta (u,v) que é segura para A
4          A := A união {(u,v)}
5 return A
```

- Parte complicada: linha 3.

- O que é uma aresta segura?

Conceitos Básicos

- Teorema 23.1: Seja G = (V, E) um grafo conectado não orientado com uma função de peso de valor real w definido em E.

- Seja um subconjunto de E que está em alguma árvore mínima correspondente a G.

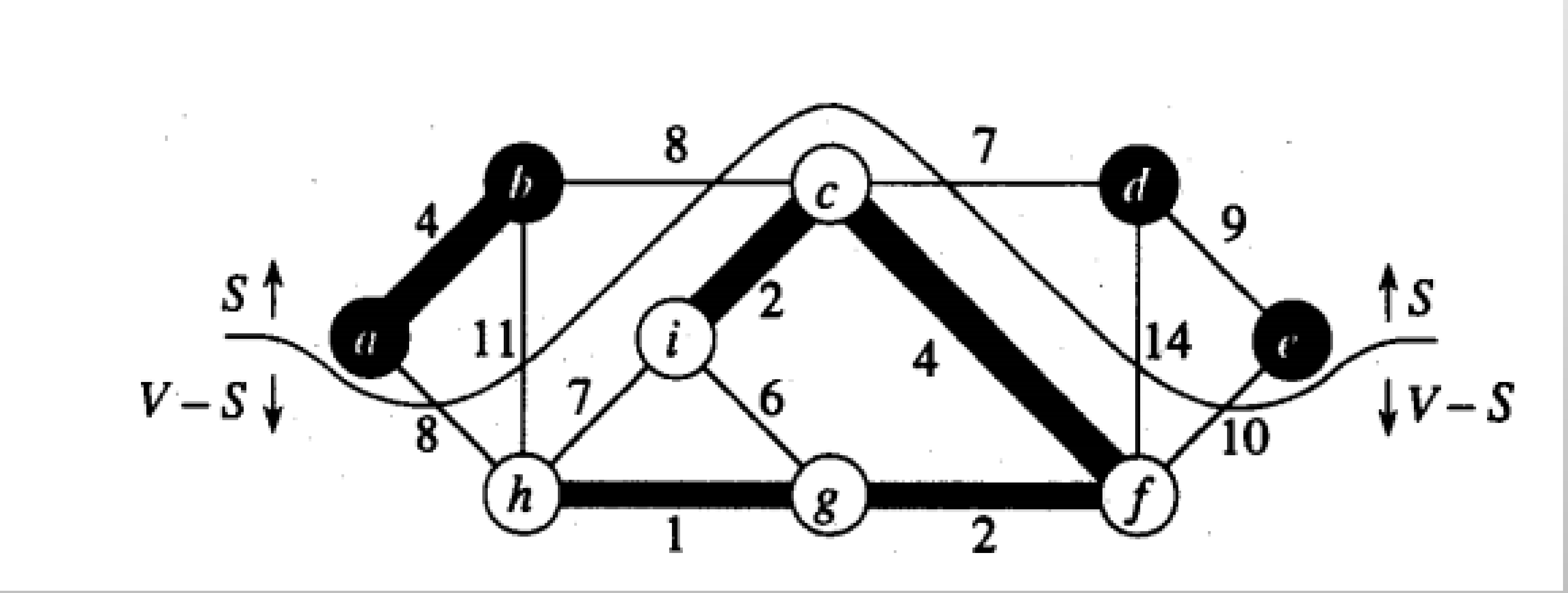
- Seja (S, V-S) qualquer corte de G que respeita A (conjunto de arestas da árvore mínima).

- Seja (u,v) uma aresta leve cruzando (S , V-S).

- Então, a aresta (u, v) é segura para A.

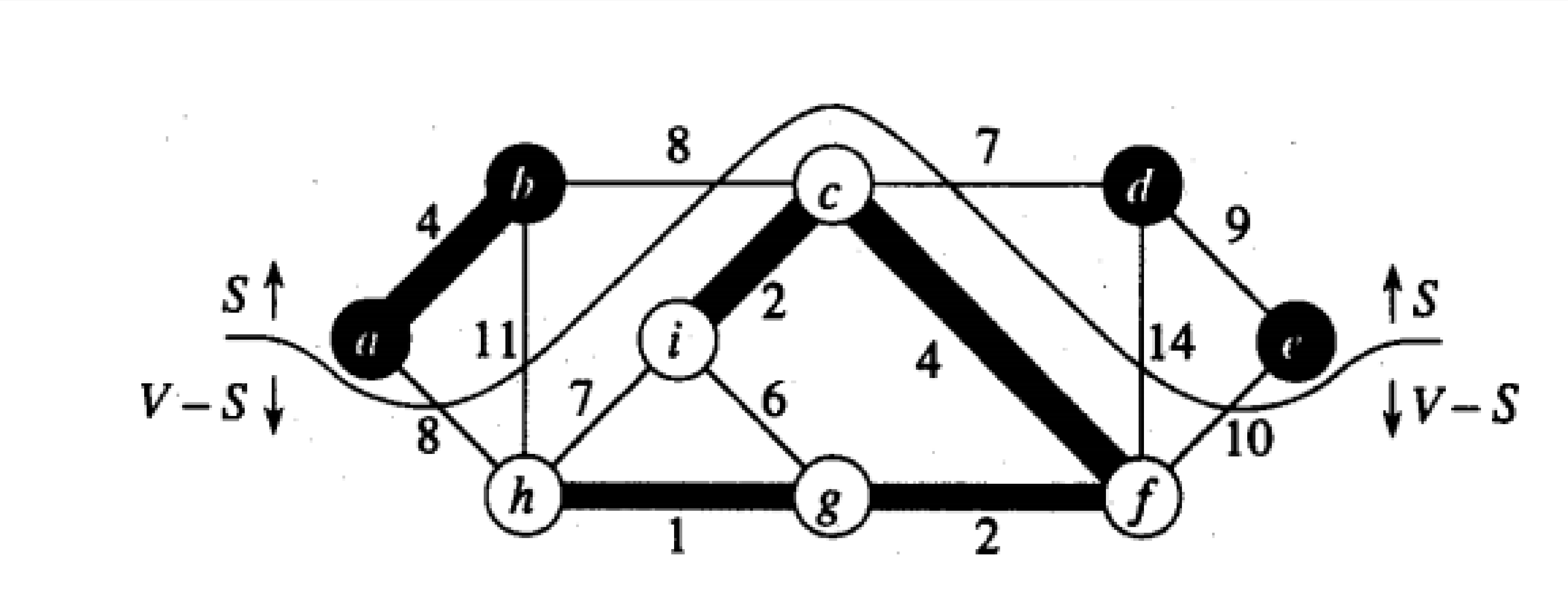
Conceitos Básicos

- Primeiro, precisamos ver algumas definições:
  - Um corte (S, V-S) de um grafo não-orientado G = (V, E) é uma partição de V.



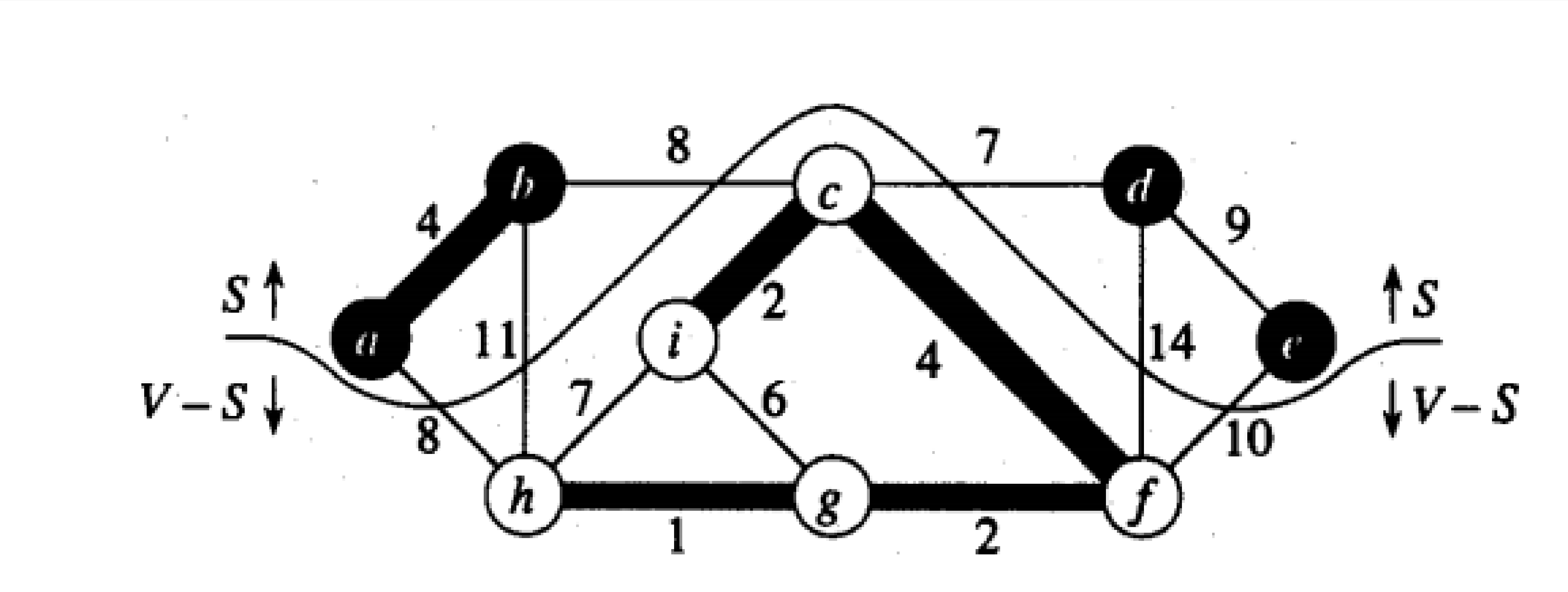
Conceitos Básicos

- Uma aresta (u,v) cruza o corte se um de seus pontos extremos está em S e outro em V-S.



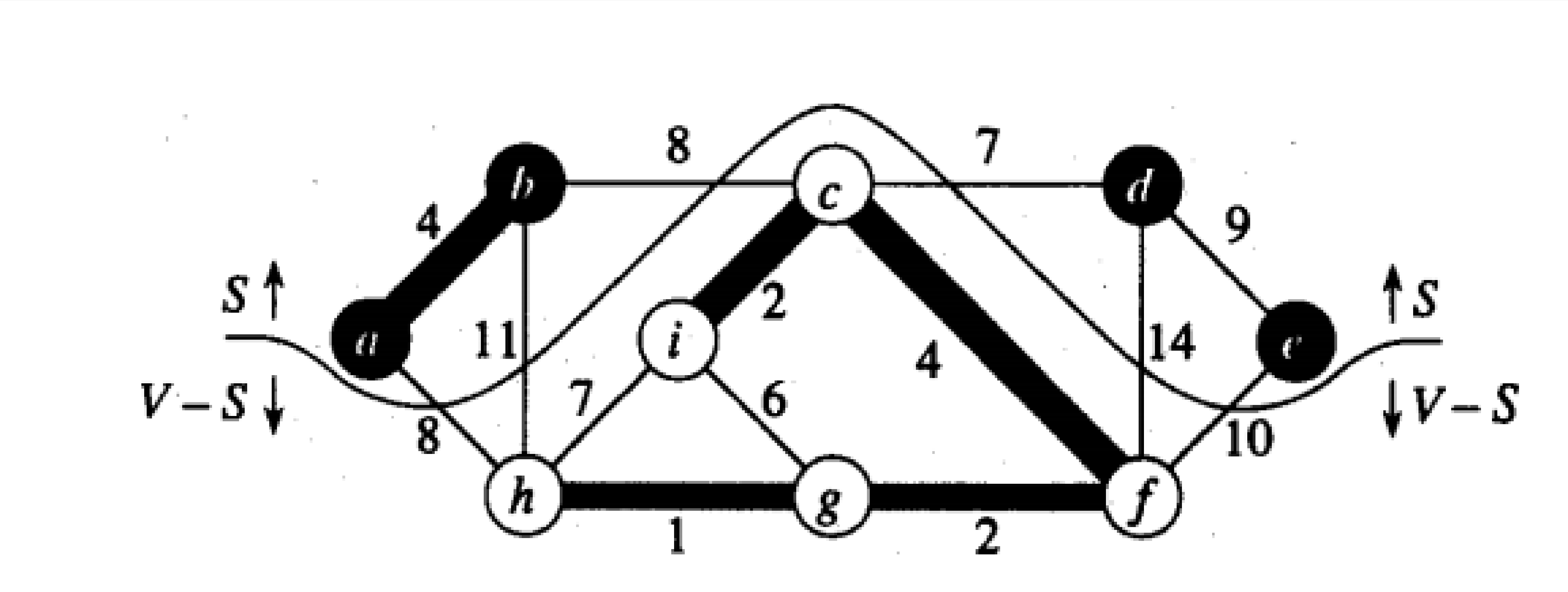
Conceitos Básicos

- Um corte respeita o conjunto A de arestas (sombreadas) se nenhuma aresta em A cruza o corte (arestas em A fazem parte da árvore espalhada).



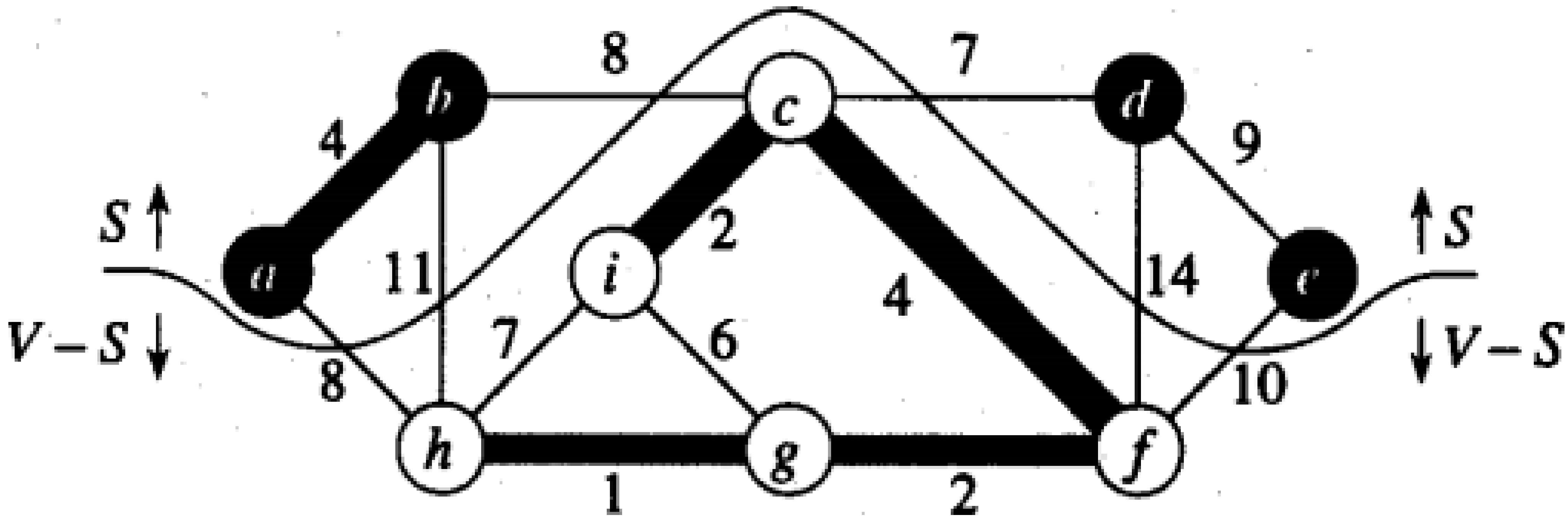
Conceitos Básicos

- Uma aresta que cruza o corte é leve quando ela possui o peso mínimo em comparação com outras arestas que cruzam o corte.



Conceitos Básicos

- Quem é a aresta segura neste caso?



Tópicos

- ~~Conceito Básico.~~
- Algoritmo de Kruskal.
- Algoritmo de Prim.
- Resumo.

Algoritmo de Kruskal

- O algoritmo de Kruskal se baseia no algoritmo genérico da árvore espalhada mínima.

- É encontrada uma aresta segura para adicionar à floresta crescente, encontrando, de todas as arestas que conectam duas árvores na floresta, uma aresta (u,v) de peso mínimo.

- Seja C1 e C2 duas árvores conectadas por (u,v). Tendo em vista que (u,v) deve ser uma aresta leve conectando C1 à alguma outra árvore, portanto (u,v) é uma aresta segura para C1.

Algoritmo de Kruskal

```
KRUSKAL(G):
A = Ø
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

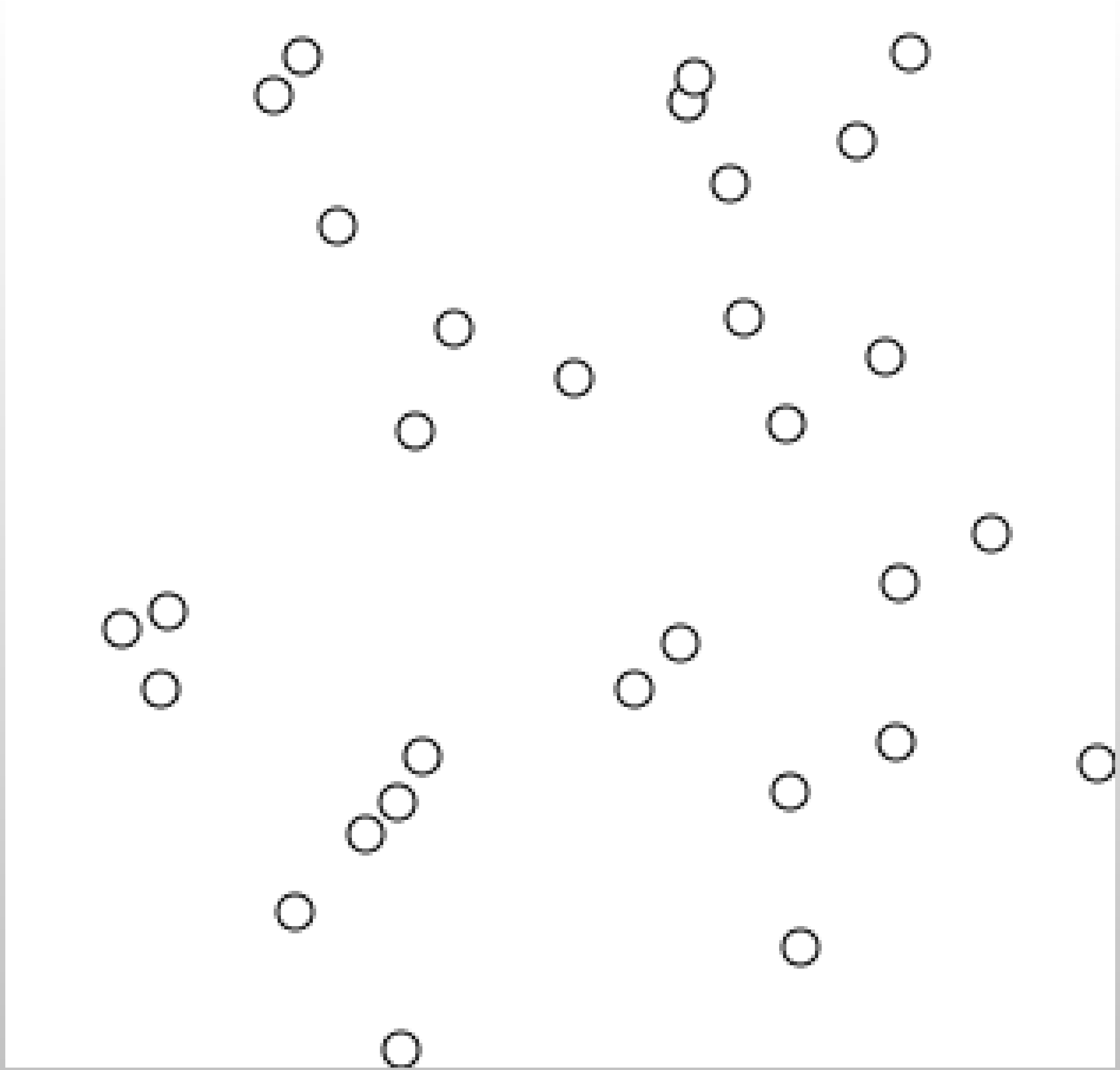
```
KRUSKAL(G):
1 A = Ø
2 foreach v ∈ G.V:
3     MAKE-SET(v)
4 foreach (u, v) in G.E ordered by weight(u, v), increasing:
5     if FIND-SET(u) ≠ FIND-SET(v):
6         A = A ∪ {(u, v)}
7         UNION(u, v)
8 return A
```

- Linha 1: Conjunto de arestas da árvore mínima é vazio (ou árvore mínima é vazia).
- Linhas 2-3: São criadas |V| árvores (ou conjuntos disjuntos), cada um contendo um vértice.
- Linha 4: arestas em E são ordenadas em ordem crescente de peso.

```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3     MAKE-SET(v)
4 foreach (u, v) in G.E ordered by weight(u, v), increasing:
5     if FIND-SET(u) ≠ FIND-SET(v):
6         A = A ∪ {(u, v)}
7         UNION(u, v)
8 return A
```

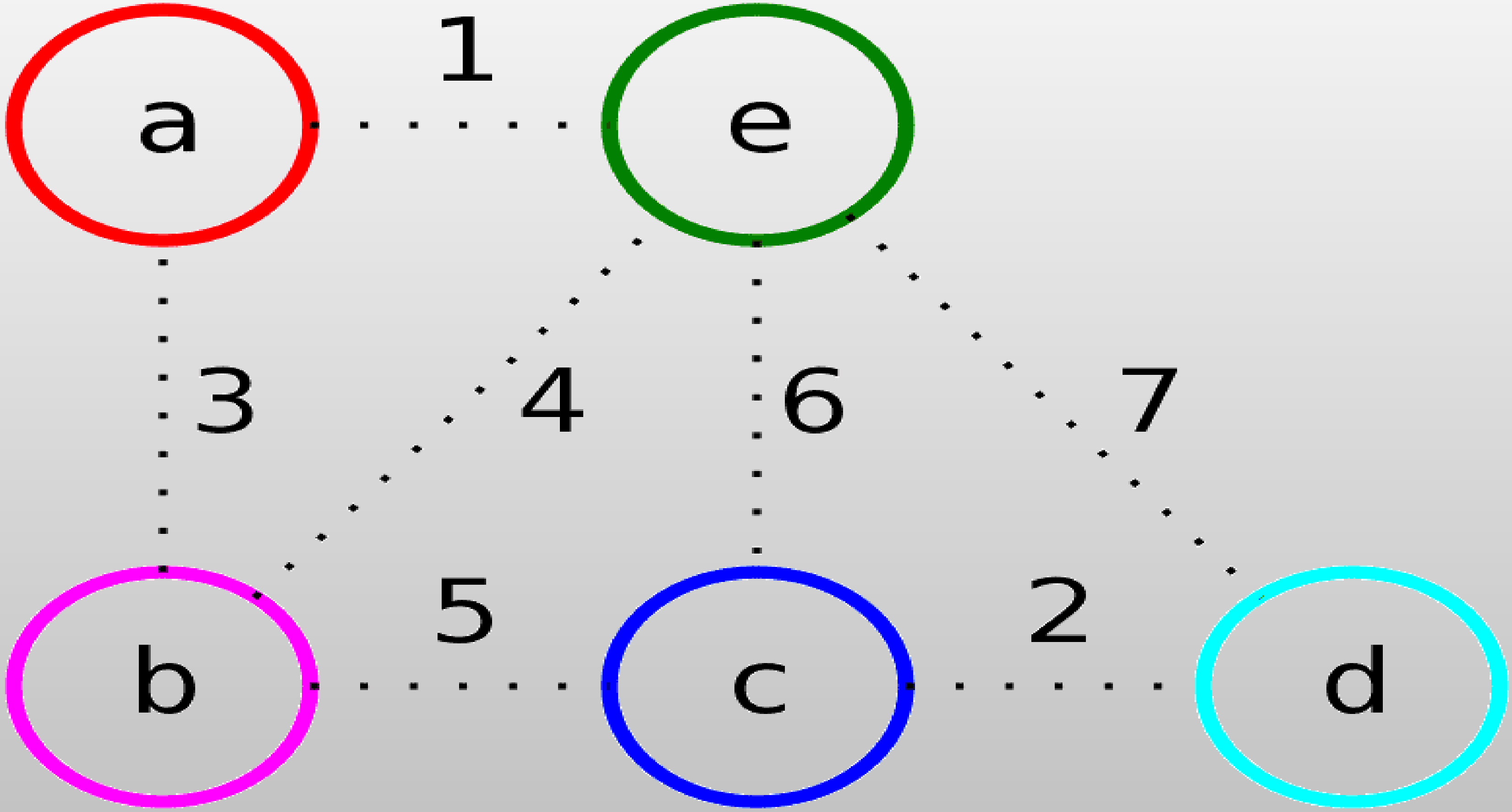
- Linhas 4-7: em ordem crescente de peso, para cada aresta é verificado se os pontos extremos (u,v) pertencem a mesma árvore.
- Se pertencem, não podem ser adicionadas, pois criariam um ciclo.
- Se não pertencem, os vértices pertencem a árvores diferentes. Nesse caso a aresta (u,v) é acrescentada a A na linha 6 e na linha 7 os vértices nas duas árvores são intercalados.

Algoritmo de Kruskal



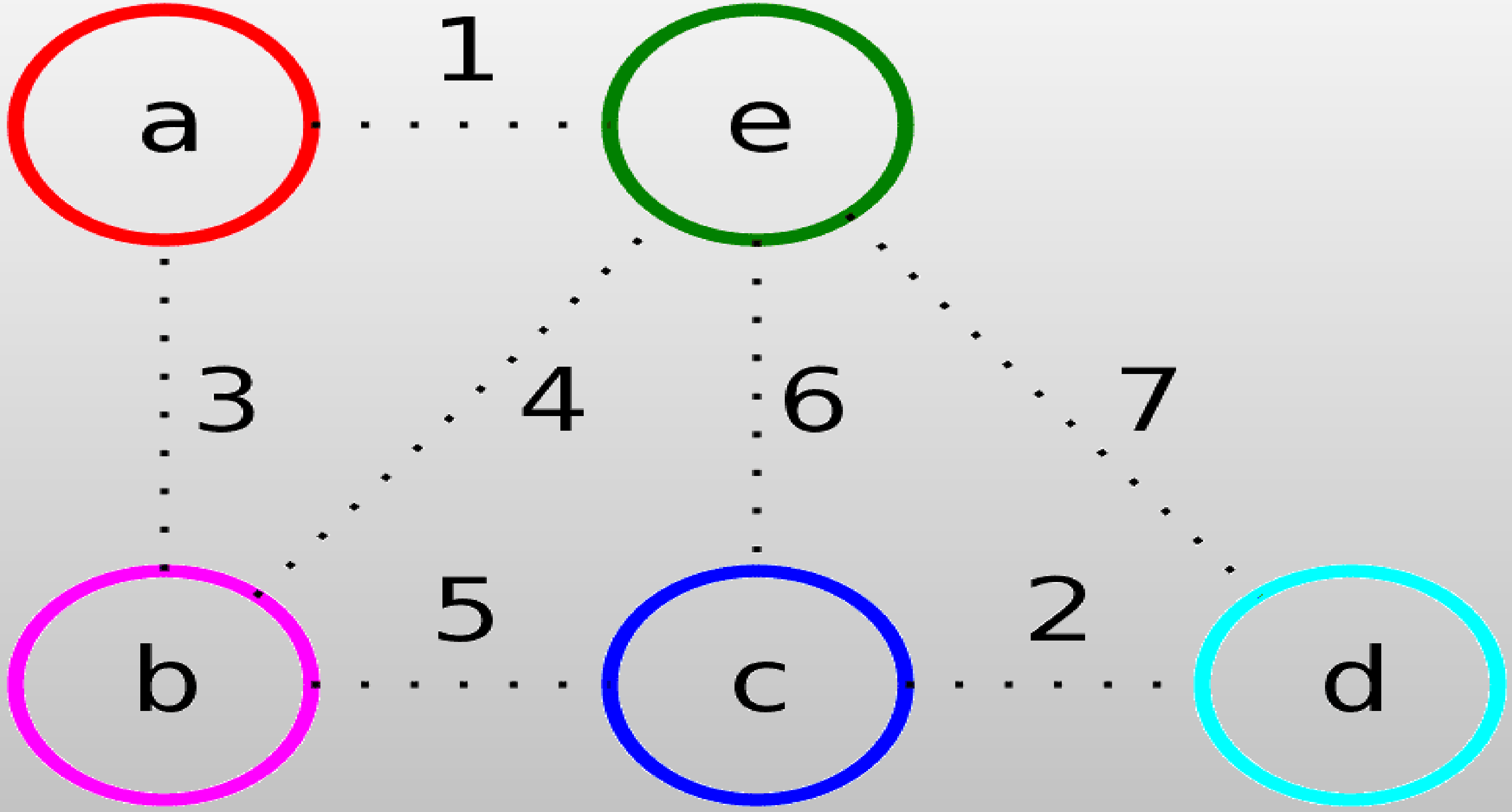
Algoritmo de Kruskal

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Algoritmo de Kruskal

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Tópico Extra

- ~~Conceito Básico.~~

- **Conjuntos Disjuntos**

Conjuntos Disjuntos

- Algumas aplicações envolvem o agrupamento de ‘n’ elementos distintos em uma coleção de conjuntos disjuntos.

- Definição: uma estrutura de dados de conjuntos disjuntos mantém uma coleção $\mathcal{G}$ = {S1,S2,S3,...,Sk} de conjuntos disjuntos dinâmicos.

- Cada elemento de um conjunto é representado por um objeto. Sendo o objeto igual a ‘x’, devemos dar suporte as seguintes operações:

Conjuntos Disjuntos

- MAKE-SET(x): Cria um novo conjunto cujo único elemento é apontado por ‘x’ (representante). Temos que ‘x’ não pode estar em outro conjunto.

- FIND-SET(x): Encontra um ponteiro para o representante do conjunto que contém ‘x’.

- UNION(x,y): une os conjuntos dinâmicos que contém x e y, digamos Sx e Sy, em um novo conjunto que é a união destes. Ambos devem ser disjuntos antes da operação. Escolhe-se um novo representante, sendo o de Sx ou Sy por padrão. Os conjuntos Sx e Sy são destruídos, sendo removidos da coleção $\mathscr{C}$.

Conjuntos Disjuntos

- De acordo com Cormen et al. (2002), temos as seguintes possibilidades de implementações :

  - Listas encadeadas.

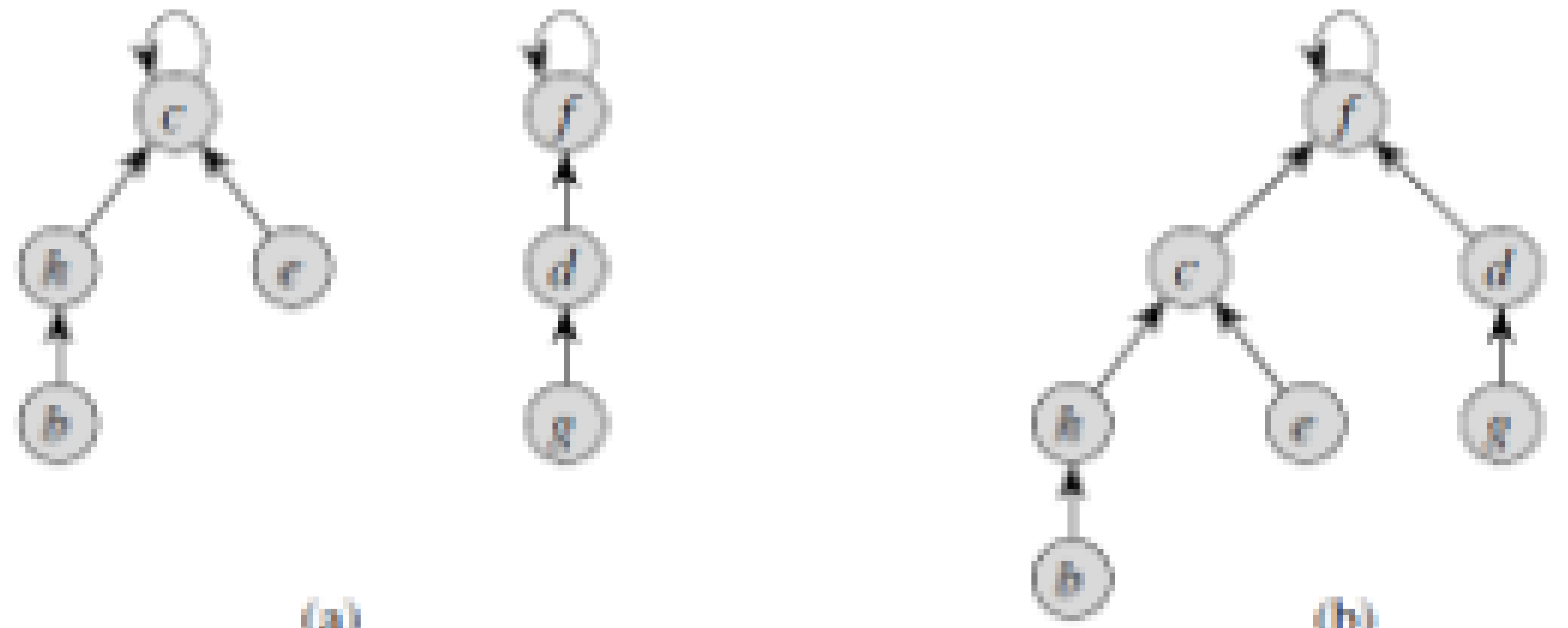
  - **Florestas de conjuntos disjuntos.**

Florestas

- Conjuntos representados por árvores enraizadas, com cada nó tendo um elemento e cada árvore representando um conjunto.

- **Em uma Floresta de Conjuntos Disjuntos, cada elemento aponta apenas para seu pai, e a raiz é seu próprio pai.**

Florestas



(a)

(b)

Florestas

- Pseudocódigo para florestas de conjuntos disjuntos:
  - **MAKE-SET(x)**
    1. **p[x]= x**
    2. **ordem[x] = 0**

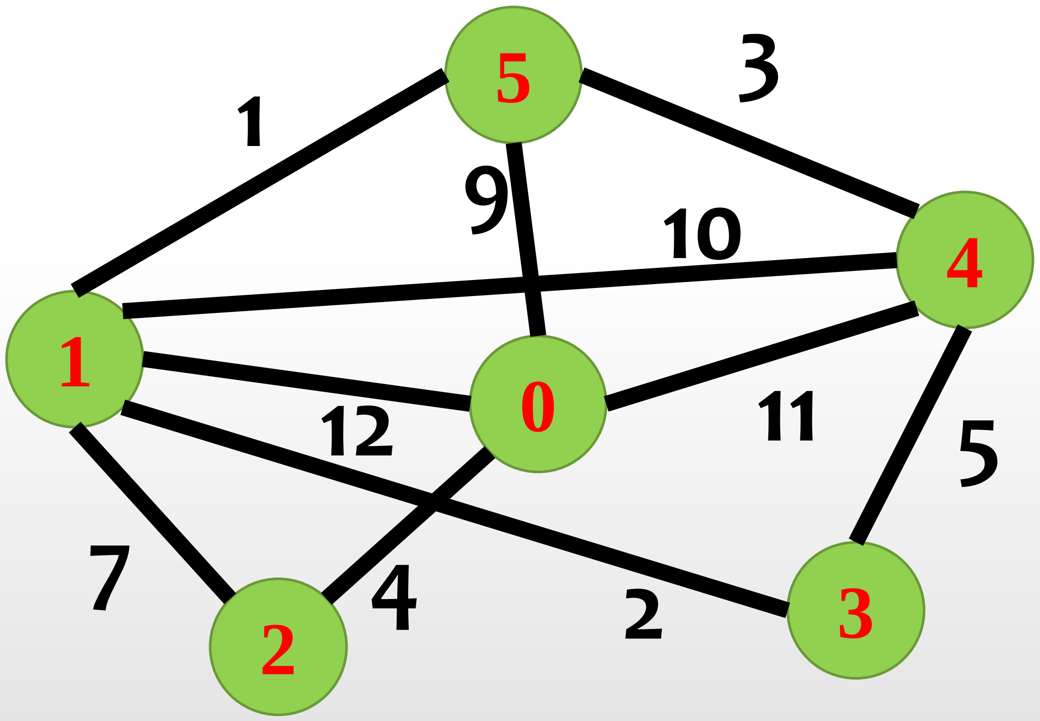
  - **FIND-SET(x)**
    1. **if x ≠ p[x]**
    2. **then p[x] = FIND-SET(p[x])**
    3. **return p[x]**

Florestas

- Pseudocódigo para florestas de conjuntos disjuntos:
  - **UNION(x,y)**
  1. **LINK(FIND-SET(x), FIND-SET(y))**

  - **LINK(x,y)**
    - **if ordem[x] > ordem[y]**
      - **then p[y] = x**
      - **else p[x] = y**
        - **if ordem[x] = ordem[y]**
        - **then ordem[y] = ordem[y]+1**

Kruskal



MAKE-SET(x)

P[x]

0	1	2	3	4	5
2	5	2	5	5	2

ordem[x]

0	1	2	3	4	5
0	0	2	0	0	1

Pesos ordenados

1	2	3	4	5	7	9	10	11

- **FIND-SET(x)**
  1. **if x ≠ p[x]**
  2. **then p[x] = FIND-SET(p[x])**
  3. **return p[x]**

- **LINK(FIND-SET(x),FIND-SET(y))**
  - **if ordem[x] > ordem[y]**
    - **then p[y] = x**
    - **else p[x] = y**
      - **if ordem[x] = ordem[y]**
      - **then ordem[y] = ordem[y]+1**

Tópicos

- ~~Conceito Básico.~~
- ~~Algoritmo de Kruskal.~~
- Algoritmo de Prim.
- Resumo.

Algoritmo de Prim

- O algoritmo de Prim é um algoritmo guloso (greedy algorithm) empregado para encontrar uma árvore geradora mínima (*minimal spanning tree* - *MST*) num grafo conectado, valorado e não direcionado.

- Isso significa que o algoritmo encontra um subgrafo do grafo original no qual a soma total das arestas é minimizada e todos os vértices estão interligados.

Algoritmo de Prim

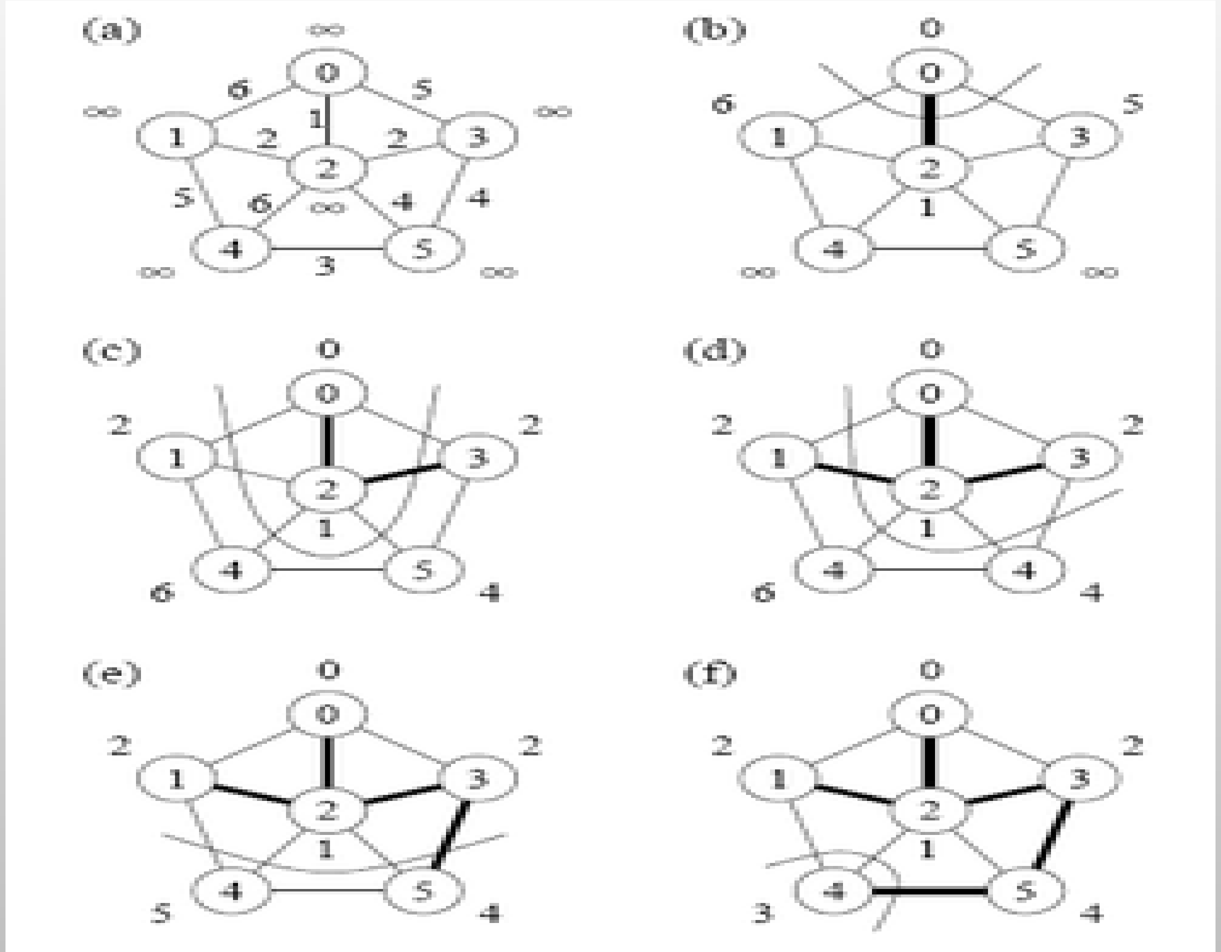
- O algoritmo de Kruskal pode ser empregado em grafos desconexos, enquanto o algoritmo de Prim precisa de um grafo **conexo**.

- O algoritmo de Prim encontra uma árvore geradora mínima para um grafo desde que ele seja valorado e não direcionado.

- O algoritmo de Prim neste caso fornecerá uma resposta ótima para este problema que não necessariamente é única.

Algoritmo de Prim

- Por exemplo, se na figura 1 os vértices deste grafo representassem cidades e as arestas fossem estradas de terra que interligassem estas cidades, como poderíamos determinar quais estradas asfaltar gastando a menor quantidade de asfalto possível para interligar todas as cidades.



Algoritmo de Prim

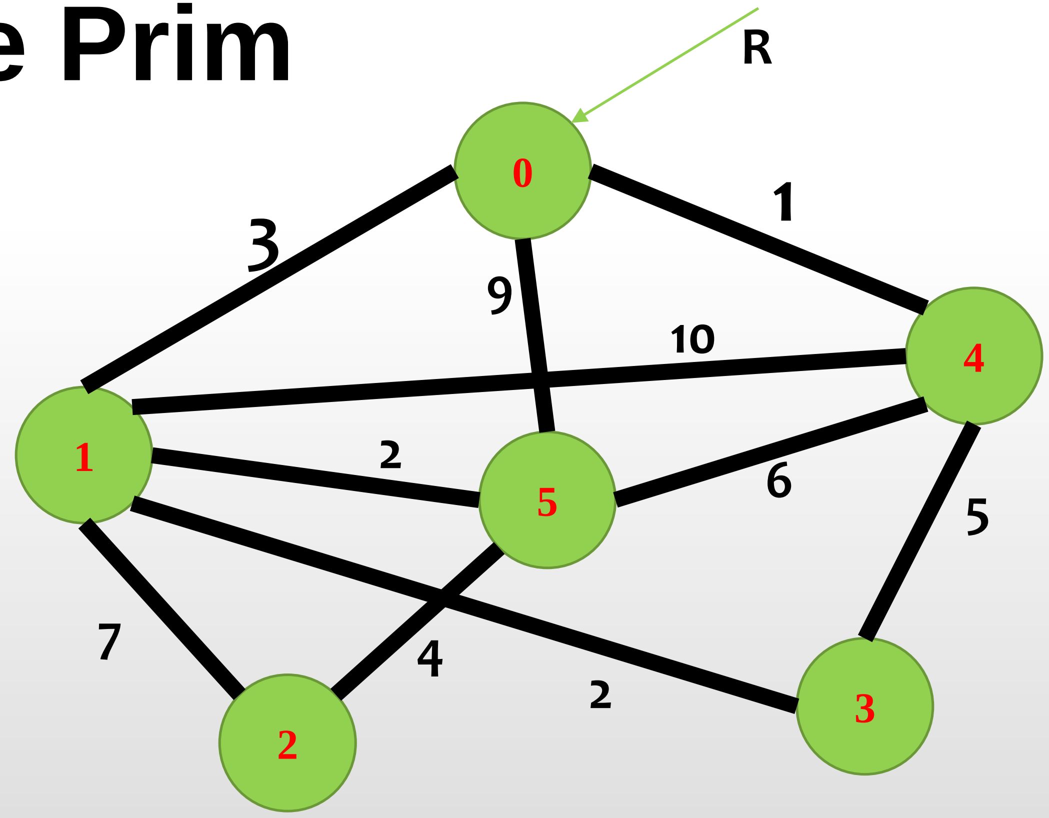
- O algoritmo de Prim pode ser derivado do algoritmo genérico.
- O subconjunto S forma uma única árvore, e a aresta segura adicionada a S é sempre uma aresta de peso mínimo conectando a árvore a um vértice que não esteja na árvore.
- A árvore começa por um vértice qualquer (no caso 0) e cresce até que "gere" todos os vértices em V .
- A cada passo, uma aresta leve é adicionada à árvore S, conectando S a um vértice de G S = (V, S).
- De acordo com o teorema anterior, quando o algoritmo termina, as arestas em S formam uma árvore geradora mínima.

Algoritmo de Prim

```
MST-PRIM(G, w, r )
1. for each u in V[G]
2.       do key[u]  ← ∞
3.                  π[u]  ← NIL
4. key[r] ← 0
5. Q ← V[G]
6. while Q is not empty
7.       do u  ← EXTRACT-MIN(Q)
8.                  for each v in Adj[u]
9.                            do if v in Q and w(u, v) < key[v]
10.                                     then π[v] ← u
11.                                              key[v]  ← w(u,v)
```

MST-PRIM(G, w, r)
1. **for each** u **in** V[G] **do**
    2. key[u] ← ∞
    3. π[u] ← NIL
4. key[r] ← 0
5. Q ← V[G]
6. **while** Q **is not empty do**
    7. u ← EXTRACT-MIN(Q)
    8. **for each** v **in** Adj[u] **do**
        9. **if** v **in** Q **and** w(u, v) < key[v] **then**
            10. π[v] ← u
            11. key[v] ← w(u,v)

Algoritmo de Prim



key[u]

0	1	2	3	4	5
0	3	4	2	1	2

π[x]

0	1	2	3	4	5
-1	0	5	1	0	1

Para mostrar na tela: analisa lista de π e key para montar a Árvore Mínima.

Q

0	1	2	3	4	5

Tópicos

- ~~Conceito Básico.~~
- ~~Algoritmo de Ordenação Topológica (usando DFS).~~
- ~~Exemplo e aplicações.~~
- Resumo.

Resumo

- Foi demonstrado o algoritmo genérico de Árvore Geradora Mínima.

- Foram demonstradas os algoritmos de Kruskal e Prim para o mesmo fim: Encontrar a Árvore Geradora Mínima.

- Foi demonstrado o funcionamento de conjuntos disjuntos.

Referências Bibliográficas

- CORMEN, Thomas H. **Algoritmos: teoria e prática**. Parte VI, Capítulo 23, páginas 454-467.

Dúvidas?

Professor Luciano Brum
email: lucianobrum18@gmail.com
https://sites.google.com/view/brumluciano