Árvore Geradora Mínima

Laboratório de Programação Competitiva I

Pedro Henrique Paiola

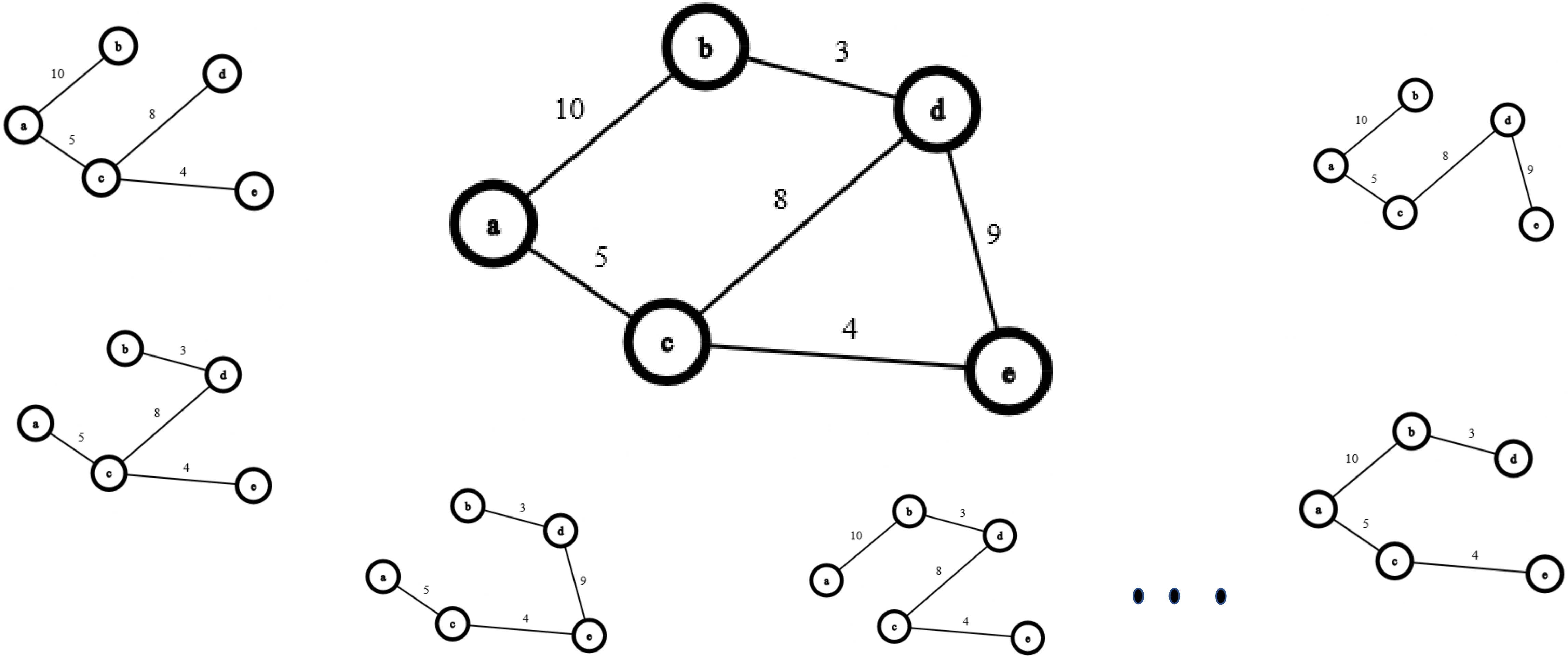
Rene Pegoraro

Wilson M Yonezawa

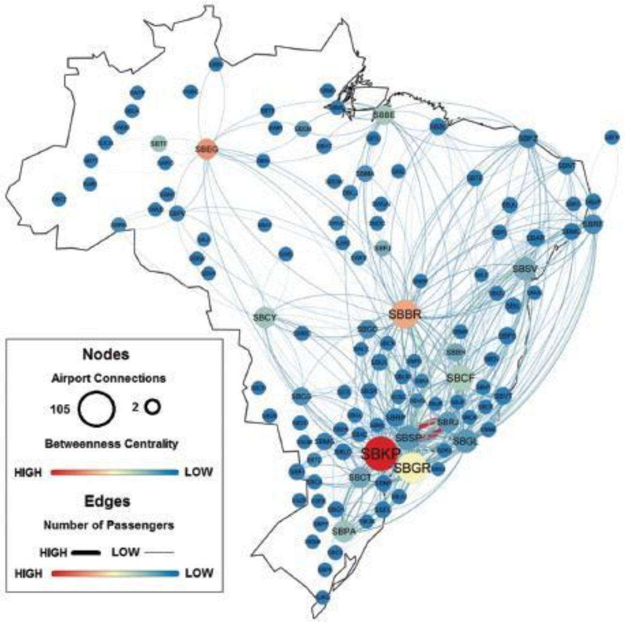
Árvore Geradora (Spanning Tree)

- Árvore Geradora de um grafo G = (V, E) é um **<u>subconjunto</u> de arestas de E** que forma uma <u>árvore</u> que **conecta <u>todos</u> os vértices de V**.
- Importante:
  - Grafo não-dirigido (não orientado)
  - Grafo conexo (existe um caminho entre qualquer par de vértices)
  - Grafo ponderado (peso na aresta)
- Em um grafo ponderado estamos interessados na árvore (árvore geradora mínima) cuja **<u>soma de todos os pesos das arestas é a menor possível</u>**

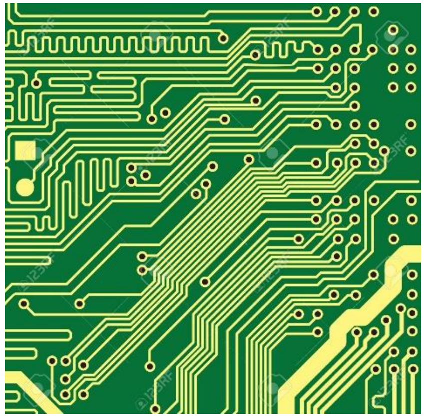
Grafo G e suas árvores geradoras



Árvore Geradora Mínima - Aplicações



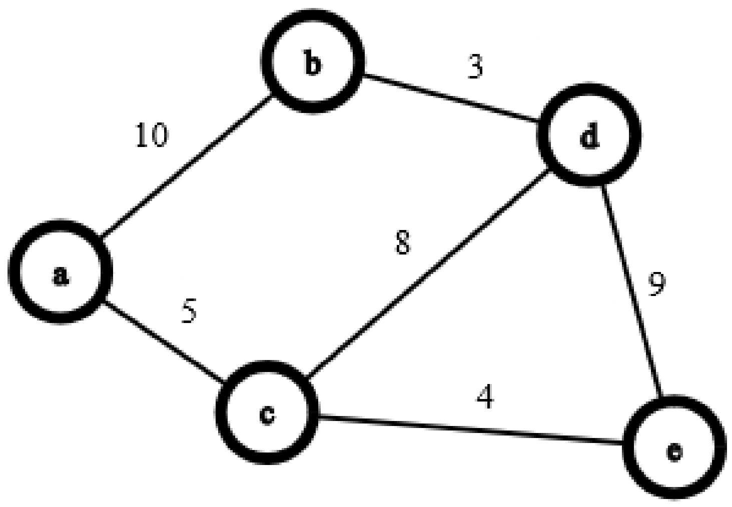
https://www.scielo.br/j/aabc/a/wCrnmTCVJSFtQhkjsdWMsqk/?lang=en#ModalFigf01



Como gerar a árvore geradora mínima?

- Estratégia gulosa
  - Algoritmo de **Kruskal** (Joseph Kruskal – 1956)
  - Algoritmo de **Prim** (Robert Prim, 1957)
  - Algoritmo de **Borůvka** (Otakar Borůvka, 1926)

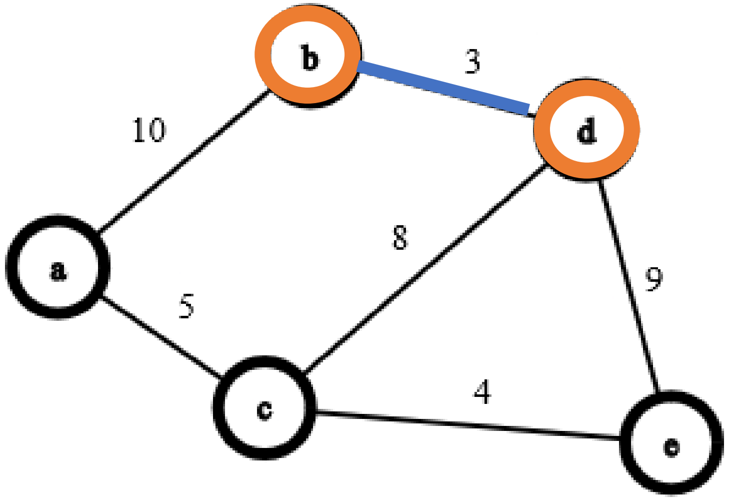
Algoritmo Kruskal



A = { }

```
Kruskal (V, E)
A = ∅  // Armazena os vértices
Para cada v ∈ V:
   Gere conjuntos disjuntos (v)
Sort E em ordem crescente de pesos
Para cada (v1, v2) ∈ E:
   se Find(v1) ≠ Find(v2) então
      A = A ∪ {(v1, v2)}
      Union(v1, v2)
   fim-se
fim-para
Retorne A
```

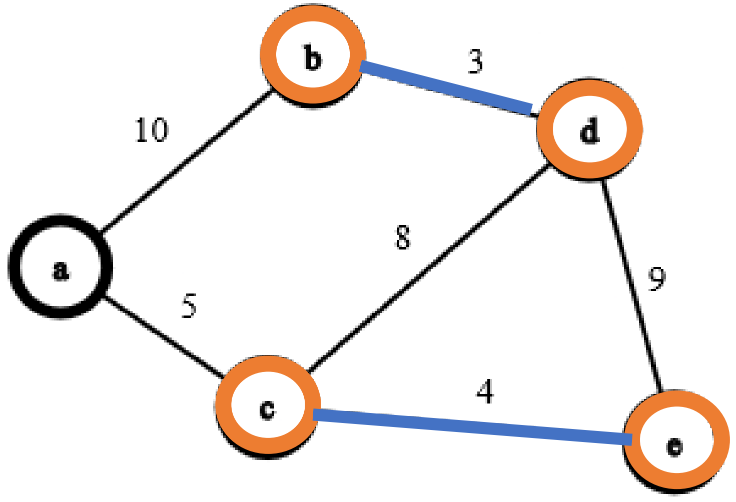
Algoritmo Kruskal



A = { (b, d)}

```
Kruskal (V, E)
A = Ø  // Armazena os vértices
Para cada v ∈ V:
   Gere conjuntos disjuntos (v)
Sort E em ordem crescente de pesos
Para cada (v1, v2) ∈ E:
   se Find(v1) ≠ Find(v2) então
      A = A ∪ {(v1, v2)}
      Union(v1, v2)
   fim-se
fim-para
Retorne A
```

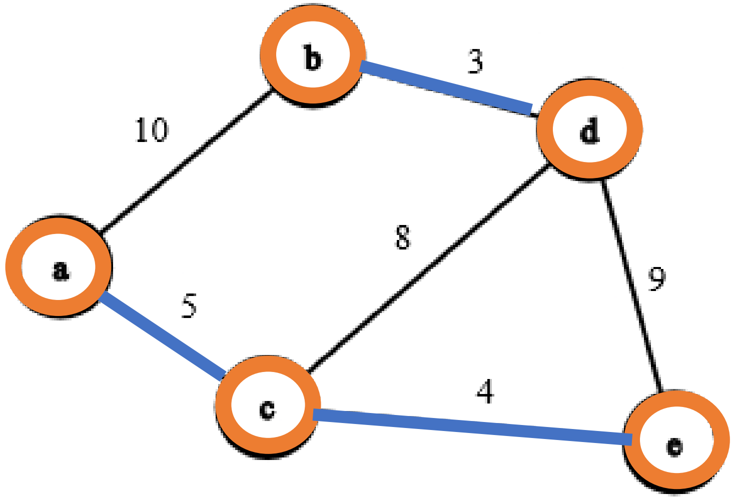
Algoritmo Kruskal



A = { (b, d),(c, e)}

```
Kruskal (V, E)
A = ∅  // Armazena os vértices
Para cada v ∈ V:
   Gere conjuntos disjuntos (v)
Sort E em ordem crescente de pesos
Para cada (v1, v2) ∈ E:
   se Find(v1) ≠ Find(v2) então
      A = A ∪ {(v1, v2)}
      Union(v1, v2)
   fim-se
fim-para
Retorne A
```

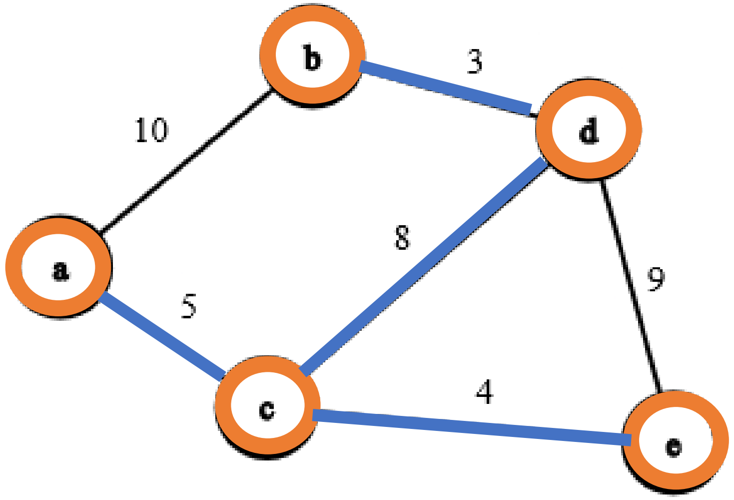
Algoritmo Kruskal



A = { (b, d),(c, e), (a,c)}

```
Kruskal (V, E)
A = ∅  // Armazena os vértices
Para cada v ∈ V:
   Gere conjuntos disjuntos (v)
Sort E em ordem crescente de pesos
Para cada (v1, v2) ∈ E:
   se Find(v1) ≠ Find(v2) então
      A = A ∪ {(v1, v2)}
      Union(v1, v2)
   fim-se
fim-para
Retorne A
```

Algoritmo Kruskal

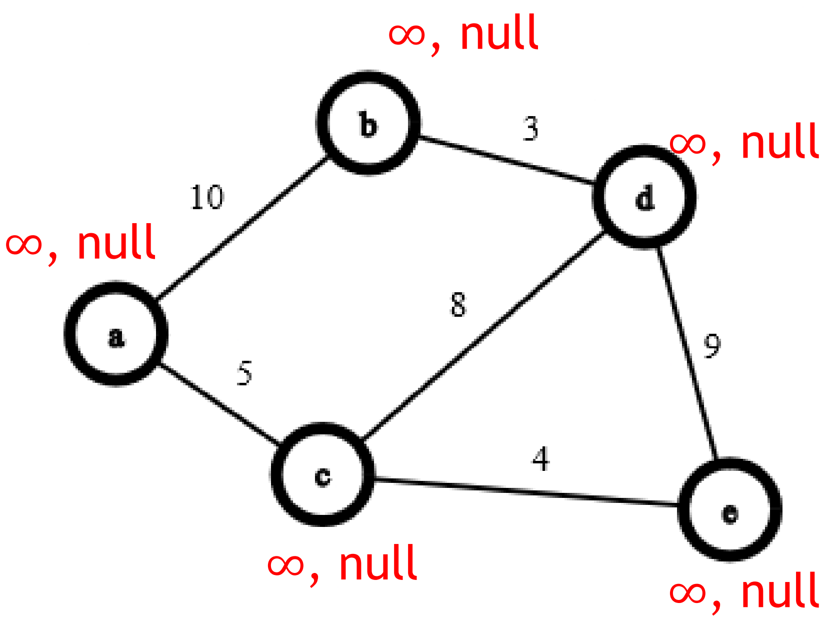


A = { (b, d),(c, e), (a,c), (c, d)}

```
Kruskal (V, E)
A = ∅  // Armazena os vértices
Para cada v ∈ V:
   Gere conjuntos disjuntos (v)
Sort E em ordem crescente de pesos
Para cada (v1, v2) ∈ E:
   se Find(v1) ≠ Find(v2) então
      A = A ∪ {(v1, v2)}
      Union(v1, v2)
   fim-se
fim-para
Retorne A
```

Complexidade: O(E*log(E))

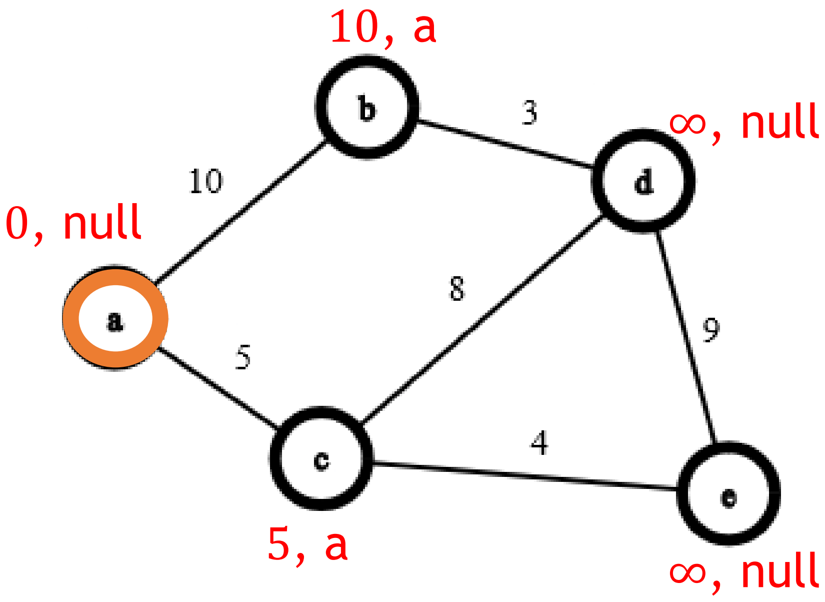
Algoritmo Prim



A = { }
Q = {a, b, c, d, e}

```
A = ∅
Para cada v ∈ V:
   CHAVE[v] = ∞
   PAI[v] = null
CHAVE[r] = 0      // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
   u = min(Q) pelo valor da Chave
   Q = Q - u
   se (PAI(u) != null) então
      A = A ∪ (u, PAI(u))
   Para cada v ∈ Adj(u):
      se (v ∈ Q e w(u, v) < CHAVE[v]) então
         PAI[v] = u
         CHAVE[v] = w
      fim-se
fim-Enquanto
Retorne A
```

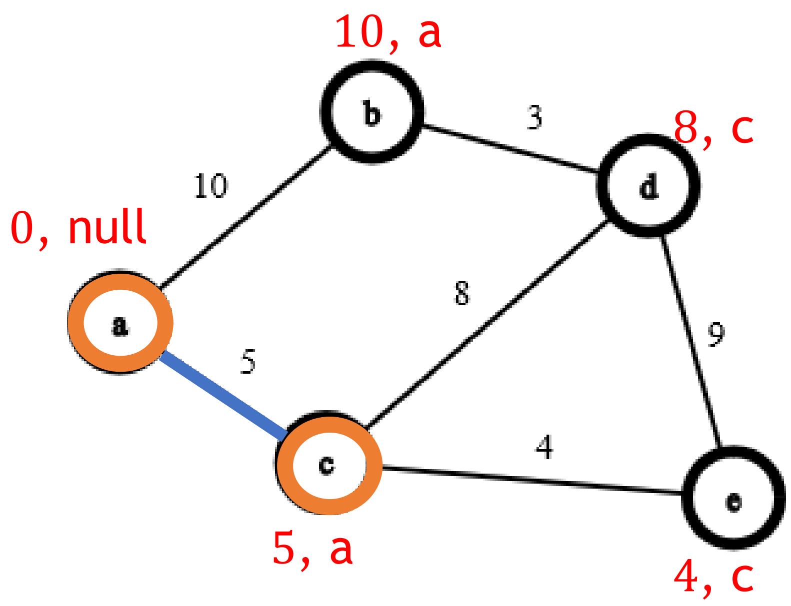
Algoritmo Prim



A = { }
Q = {b, c, d, e}

```
A = Ø
Para cada v ∈ V:
   CHAVE[v] = ∞
   PAI[v] = null
CHAVE[r] = 0     // r é qualquer vértice
Q = V
Enquanto (Q != Ø) faça
   u = min(Q) pelo valor da Chave
   Q = Q - u
   se (PAI(u) != null) então
      A = A ∪ (u, PAI(u))
   Para cada v ∈ Adj(u):
      se (v ∈ Q e w(u, v) < CHAVE[v]) então
         PAI[v] = u
         CHAVE[v] = w
      fim-se
fim-Enquanto
Retorne A
```

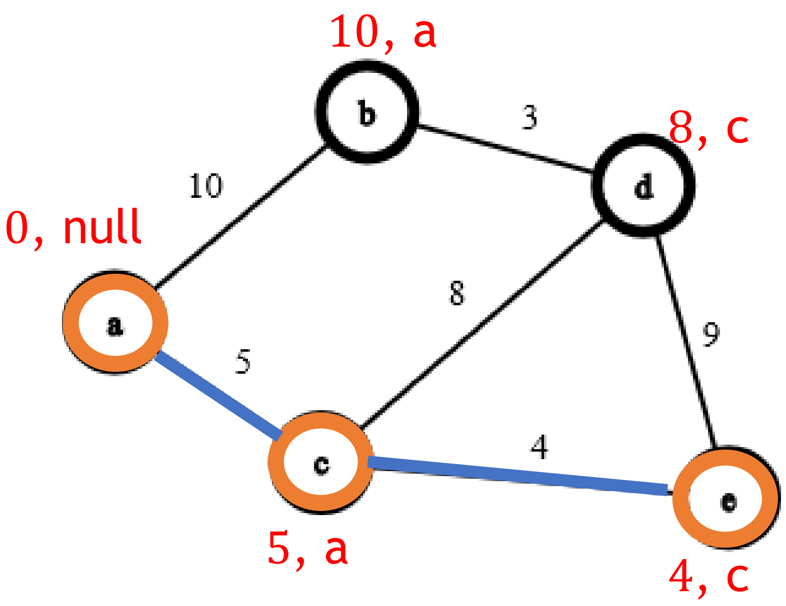
Algoritmo Prim



A = {(a,c) }
Q = {b, d, e}

```
A = ∅
Para cada v ∈ V:
    CHAVE[v] = ∞
    PAI[v] = null
CHAVE[r] = 0      // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
    u = min(Q) pelo valor da Chave
    Q = Q - u
    se (PAI(u) != null) então
        A = A ∪ (u, PAI(u))
    Para cada v ∈ Adj(u):
        se (v ∈ Q e w(u, v) < CHAVE[v]) então
            PAI[v] = u
            CHAVE[v] = w
        fim-se
fim-Enquanto
Retorne A
```

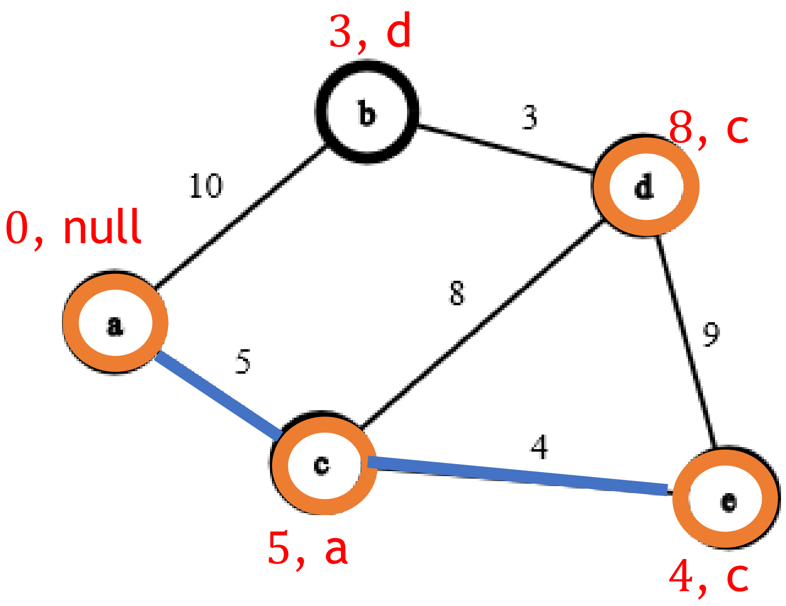
Algoritmo Prim



A = {(a,c), (c, e) }
Q = {b, d}

```
A = ∅
Para cada v ∈ V:
    CHAVE[v] = ∞
    PAI[v] = null
CHAVE[r] = 0     // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
    u = min(Q) pelo valor da Chave
    Q = Q - u
    se (PAI(u) != null) então
        A = A ∪ (u, PAI(u))
    Para cada v ∈ Adj(u):
        se (v ∈ Q e w(u, v) < CHAVE[v]) então
            PAI[v] = u
            CHAVE[v] = w
        fim-se
fim-Enquanto
Retorne A
```

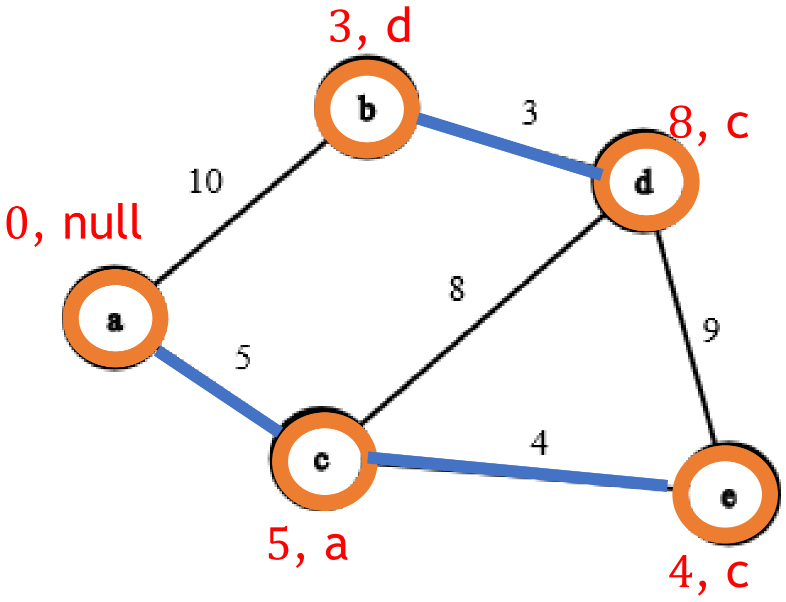
Algoritmo Prim



A = {(a,c), (c, e), (d,c) }
Q = {b}

```
A = ∅
Para cada v ∈ V:
   CHAVE[v] = ∞
   PAI[v] = null
CHAVE[r] = 0     // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
   u = min(Q) pelo valor da Chave
   Q = Q - u
   se (PAI(u) != null) então
      A = A ∪ (u, PAI(u))
   Para cada v ∈ Adj(u):
      se (v ∈ Q e w(u, v) < CHAVE[v]) então
         PAI[v] = u
         CHAVE[v] = w
      fim-se
fim-Enquanto
Retorne A
```

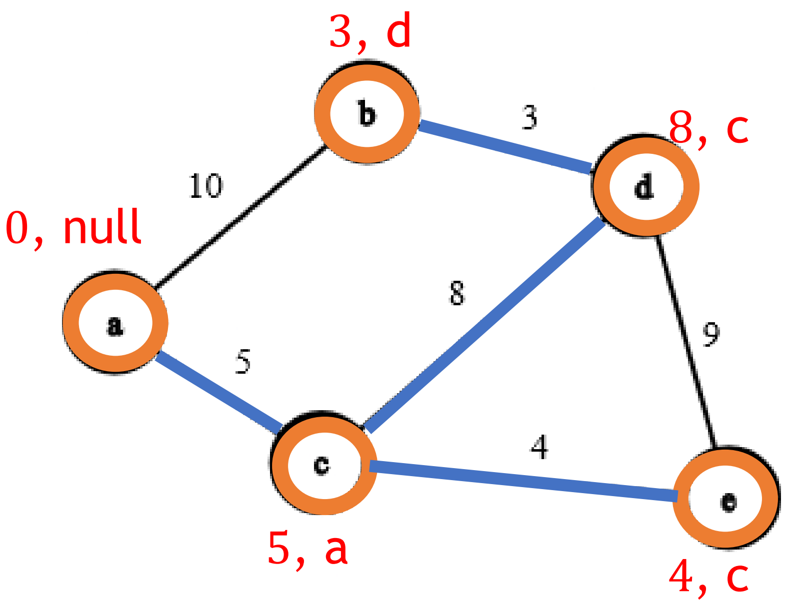
Algoritmo Prim



A = {(a,c), (c, e), (d,c), (b, d)}
Q = { }

```
A = ∅
Para cada v ∈ V:
    CHAVE[v] = ∞
    PAI[v] = null
CHAVE[r] = 0      // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
    u = min(Q) pelo valor da Chave
    Q = Q - u
    se (PAI(u) != null) então
        A = A ∪ (u, PAI(u))
    Para cada v ∈ Adj(u):
        se (v ∈ Q e w(u, v) < CHAVE[v]) então
            PAI[v] = u
            CHAVE[v] = w
        fim-se
fim-Enquanto
Retorne A
```

Algoritmo Prim



A = {(a,c), (c, e), (d,c), (b, d)}
Q = { }

```
A = ∅
Para cada v ∈ V:
   CHAVE[v] = ∞
   PAI[v] = null
CHAVE[r] = 0     // r é qualquer vértice
Q = V
Enquanto (Q != ∅) faça
   u = min(Q) pelo valor da Chave
   Q = Q - u
   se (PAI(u) != null) então
      A = A ∪ (u, PAI(u))
   Para cada v ∈ Adj(u):
      se (v ∈ Q e w(u, v) < CHAVE[v]) então
         PAI[v] = u
         CHAVE[v] = w
      fim-se
fim-Enquanto
Retorne A
```

Complexidade: O(E*log(V))

Complexidade: O(E + Vlog(V))

URI 1152 – Estradas Escuras

Programação Competitiva unesp

URI Online Judge | 1152

Estradas Escuras

Univeristy of Ulm Local Contest Alemanha

Timelimit: 3

Nestes dias se pensa muito em economia, mesmo em Byteland. Para reduzir custos operacionais, o governo de Byteland decidiu otimizar a iluminação das estradas. Até agora, todas as rotas eram iluminadas durante toda noite, o que custava 1 Dólar Byteland por metro a cada dia. Para economizar, eles decidiram não iluminar mais todas as estradas e desligar a iluminação de algumas delas. Para ter certeza que os habitantes de Byteland continuem a se sentirem seguros, eles querem otimizar o sistema de tal forma que após desligar a iluminação de algumas estradas à noite, sempre existirá algum caminho iluminado de qualquer junção de Byteland para qualquer outra junção.

Qual é a quantidade máxima de dinheiro que o governo de Byteland pode economizar, sem fazer os seus habitantes sentirem-se inseguros?

Entrada

A entrada contém vários casos de teste. Cada caso de teste inicia com dois números **m** ($1 \le m \le 200000$) e **n** ($m-1 \le n \le 200000$), que são o número de junções de Byteland e o número de estradas em Byteland, respectivamente. Seguem n conjuntos de três valores inteiros, **x**, **y** e **z**, especificando qual será a estrada bidirecional entre **x** e **y** com **z** metros ($0 \le x, y < m$ e $x \neq y$).

A entrada termina com **m**=**n**=0. O grafo especificado em cada caso de teste é conectado. O tamanho total de todas as estradas em cada caso de teste é menor do que 2^{31}.

URI 1152 – Estradas Escuras

- Solução:
  - Calcule o custo total de iluminar todas as rotas (**custo_total**)
  - Encontre uma arvore geradora mínima (MST) e calcule o custo mínimo (**custo_mínimo**)
  - Calcule a quantidade máxima de dinheiro economizado:
    - **tot_economizado = custo_total - custo_minimo**

Programação Competitiva unesp

Referências

The Algorithm Design Manual. Steven S. Skiena, Springer, 2008.

Teoria Computacional de Grafos – Os algoritmos. Jayme Luiz Szwarcfiter, Elsivier, 2018.

Algoritmos – Teoria e Prática. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Elsevier, 2012.

https://boqian.weebly.com/c-programming.html

https://www.programiz.com/dsa/kruskal-algorithm

https://www.programiz.com/dsa/prim-algorithm