Árvore Geradora de Peso Mínimo

Algoritmos e Estruturas de Dados - EC

June 30, 2014

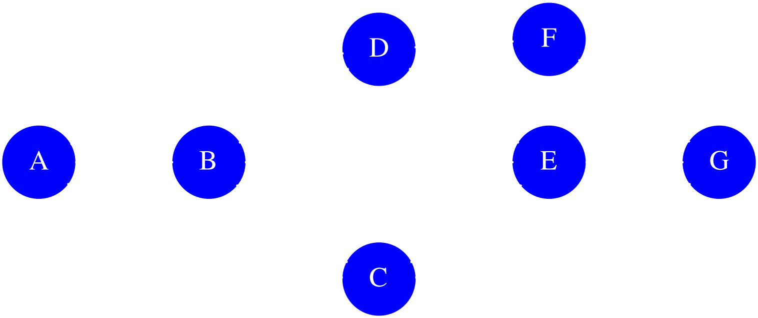
Motivação

Problema

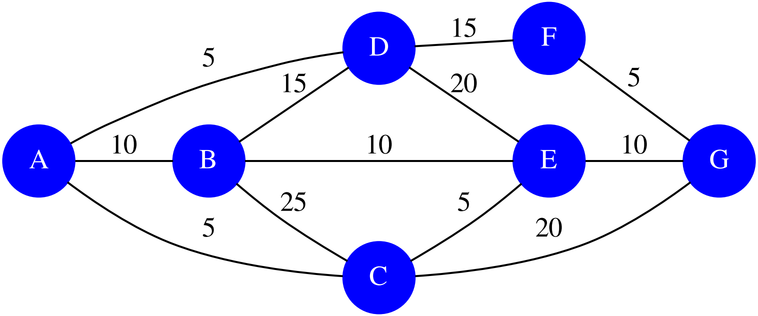
Temos um mapa com n localizações. Queremos construir estradas de mão dupla de modo que exista pelo menos um caminho entre quaisquer duas localizações do mapa. Só podemos construir estradas especificadas em uma lista com m elementos.
Sabendo que cada estrada tem um custo de construção, como podemos alcançar nosso objetivo com custo mínimo?

Motivação

Apenas as localizações do mapa:



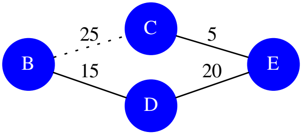
As localizações e as estradas que podemos construir, com seus custos de construção:



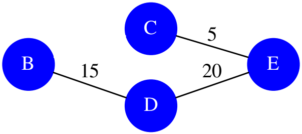
Motivação

Caracterização

Note que, na nossa resposta final, não haverá ciclos. Eles são desnecessários para garantir a **conectividade** e apenas adicionam custo à nossa resposta.
Nossa resposta será **uma árvore**.



$\longrightarrow$



Custo $= 65$

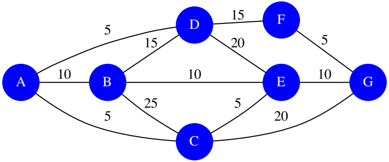
Custo $= 40$

Definições

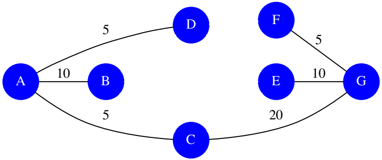
Árvore Geradora de Peso Mínimo - AGPM

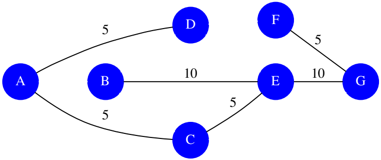
Definimos uma Árvoce Geradora de um grafo $G = (V, E)$ como sendo um subgrafo conexo de G e que contém todos os vértices V.
Definimos ainda o custo de uma árvore geradora como sendo a soma dos custos de suas arestas.
Logo, uma **Árvore Geradora de Peso Mínimo (AGPM)** de um grafo é tal que não existe uma outra árvore geradora com custo menor.

Árvore Geradora de Peso Mínimo - AGPM



Grafo original





Custo = 55

Custo = 40

Árvore Geradora de Peso Mínimo - AGPM

Observação

Se o grafo não for conexo, teremos uma **Floresta** geradora de peso mínimo.
Ela será composta de uma árvore para cada componente conexo do grafo.

Árvore Geradora de Peso Mínimo - AGPM

Estratégias

Examinaremos dois algoritmos para resolver o problema de se encontrar uma AGPM de um grafo:

- O Algoritmo de Prim,
- O Algoritmo de Kruskal.

Ambos se baseam em estratégias **gulosas**. Como vimos com o algoritmo de Dijkstra, uma estratégia gulosa é caracterizada pela tomada de decisões **ótimas locais** para atingir o estado **ótimo global**, que é a resposta.

AGPM - Algoritmo de Prim

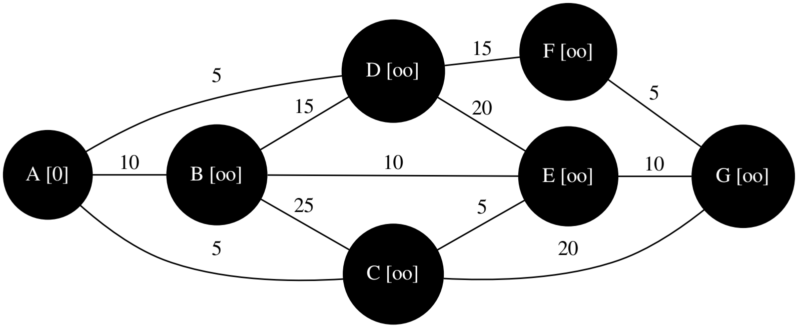
Ideia Geral

Em cada iteração do algoritmo de Prim aumentamos a resposta atual em 1 vértice.
Escolhemos uma vértice como inicial e o adicionamos na árvore. A cada passo posterior, adicionamos um vértice que ainda não está na árvore e se liga com a árvore através de uma aresta de menor peso.

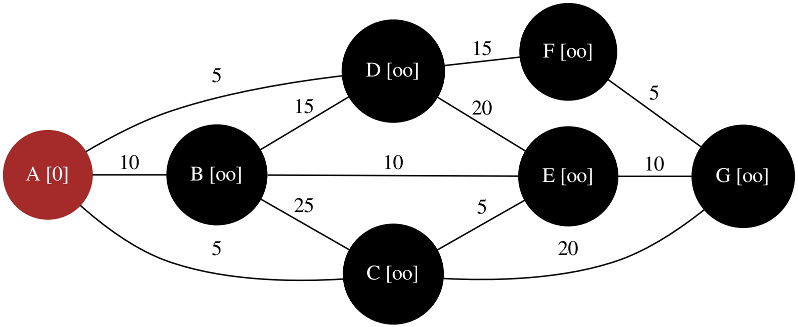
O Algoritmo de Prim: Pseudo-Código

Defina D[v] $= \infty$, marcado[v] $= 0$, predecessor[v] $=$ NULL $\forall v \in V$.
Defina D[v_1] $= 0$.
Enquanto houver vértices não marcados (e alcançáveis*) faça:
    Seja v_{at} o vértice não marcado com menor D.
    Defina marcado[v_{at}] $= 1$.
    Para toda aresta $e = (v_{at}, v_{prox})$ faça:
        Se marcado[v_{prox}] $= 0$ E D[v_{prox}] $> W(e)$:
            Defina D[v_{prox}] $= W(e)$.
            Defina predecessor[v_{prox}] $= v_{at}$.
retorne D[v_2]

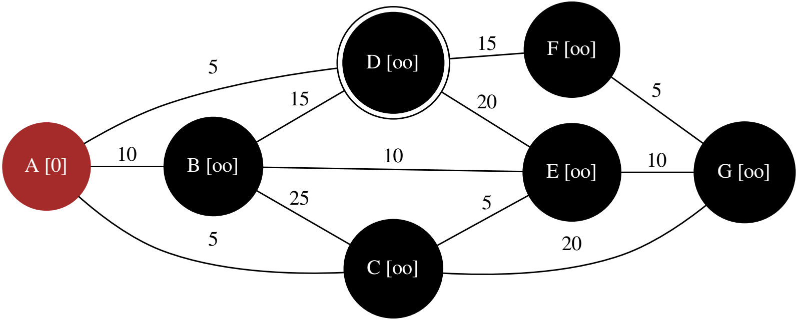
O Algoritmo de Prim - Exemplo



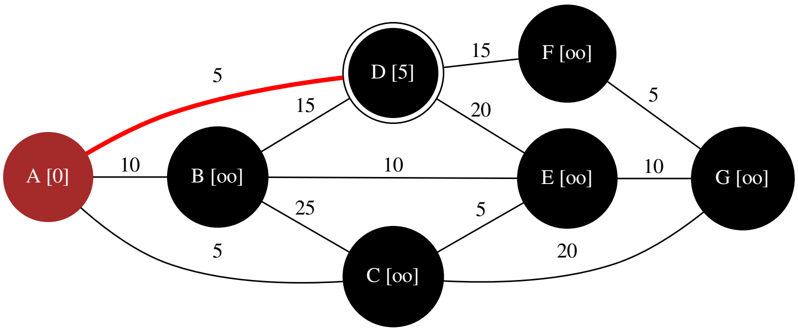
O Algoritmo de Prim - Exemplo



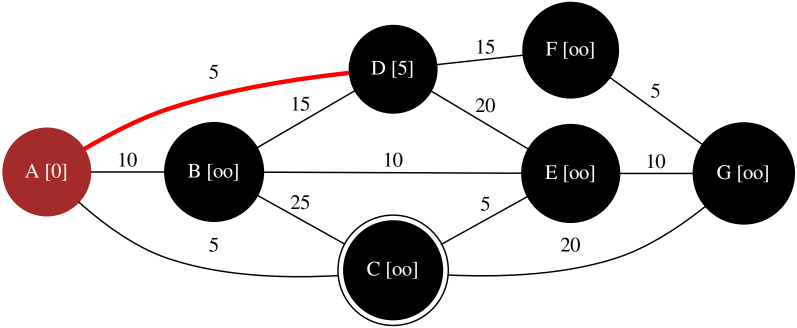
O Algoritmo de Prim - Exemplo



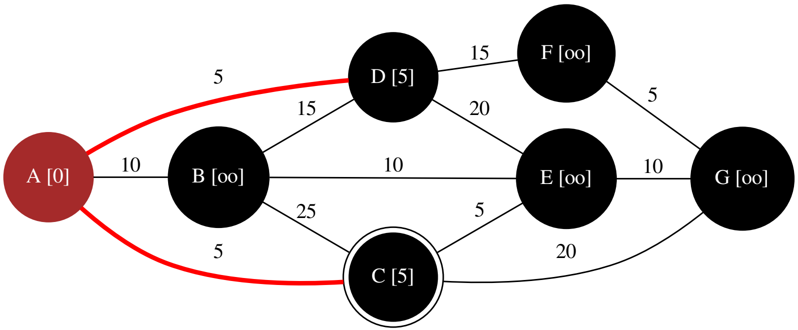
O Algoritmo de Prim - Exemplo



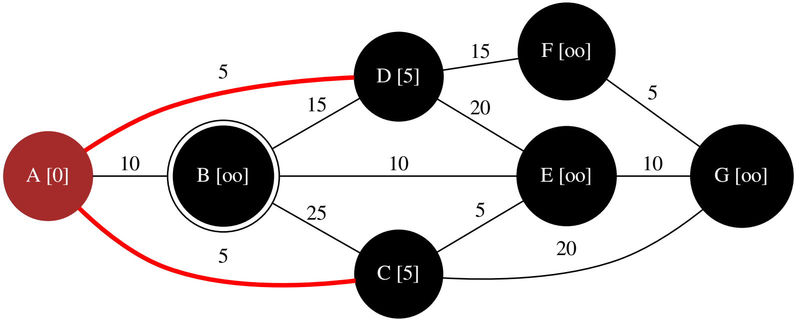
O Algoritmo de Prim - Exemplo



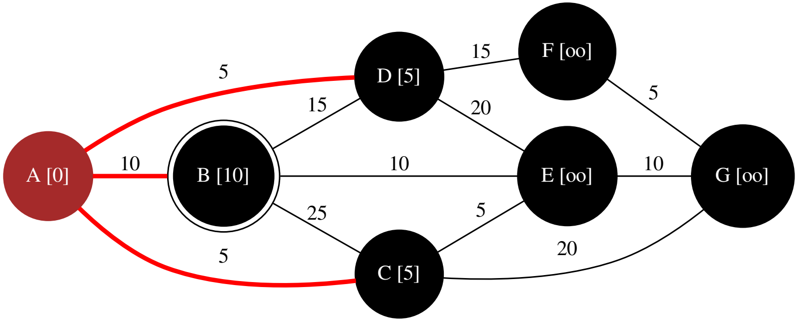
O Algoritmo de Prim - Exemplo

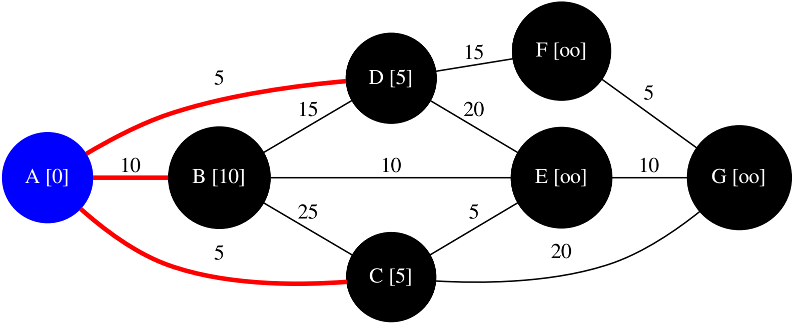


O Algoritmo de Prim - Exemplo

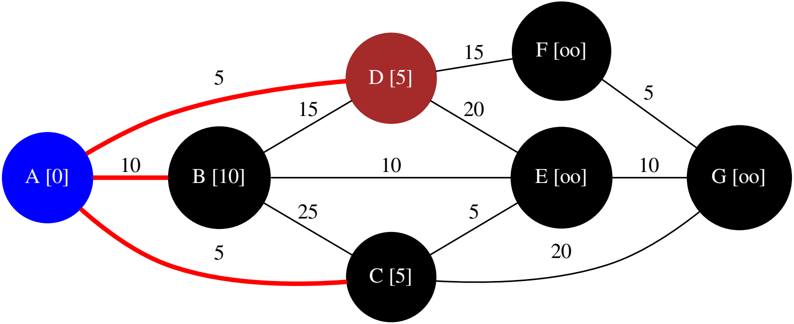


O Algoritmo de Prim - Exemplo

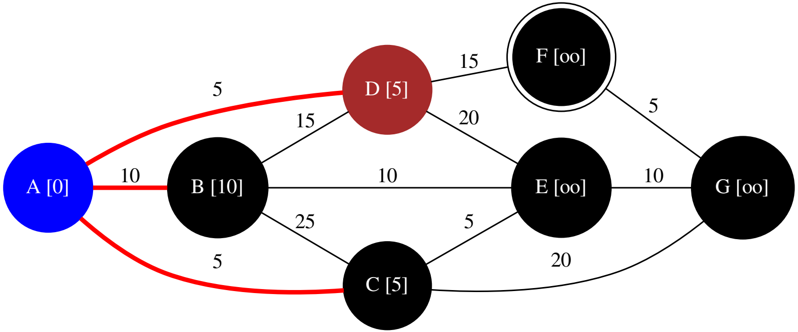



A [0]
B [10]
C [5]
D [5]
E [oo]
F [oo]
G [oo]
5
10
5
15
25
10
15
20
5
20
10
5

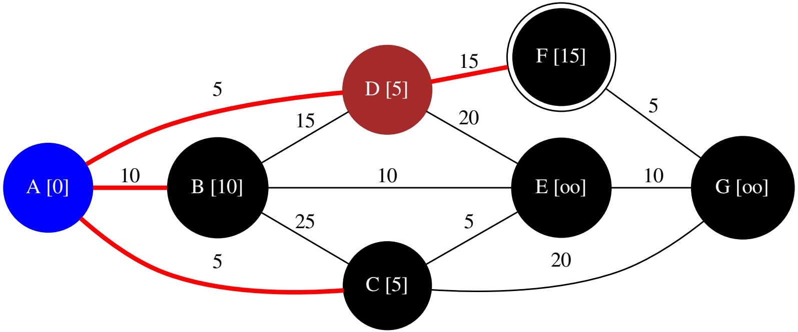
O Algoritmo de Prim - Exemplo



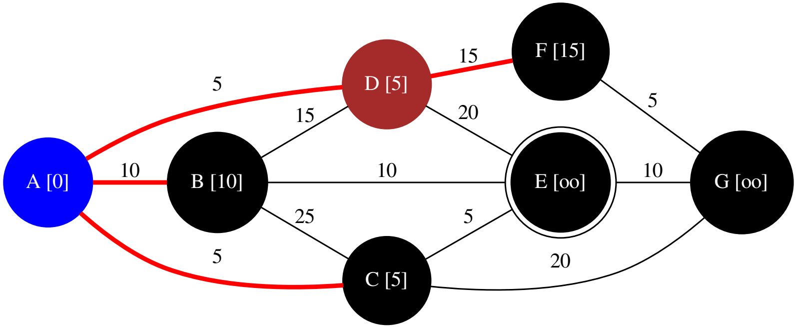
O Algoritmo de Prim - Exemplo



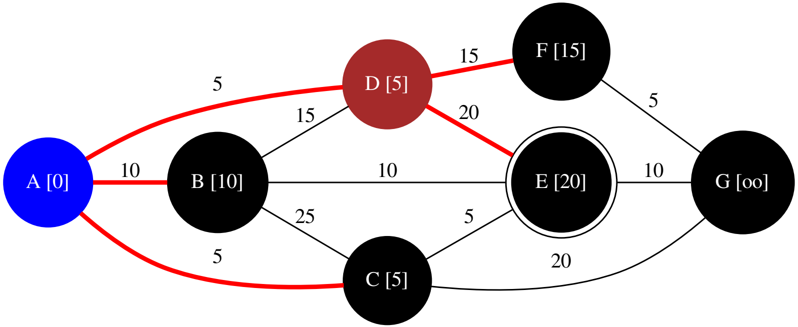
O Algoritmo de Prim - Exemplo



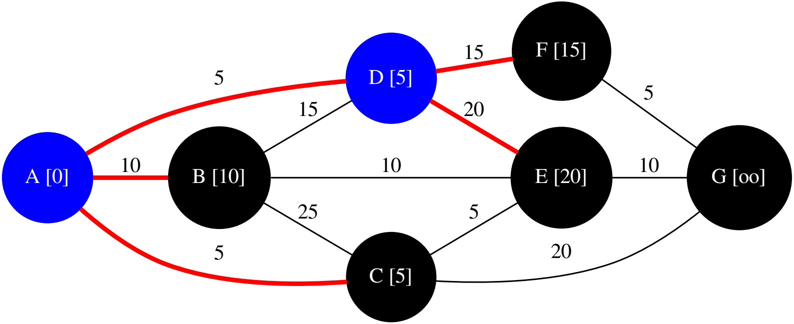
O Algoritmo de Prim - Exemplo



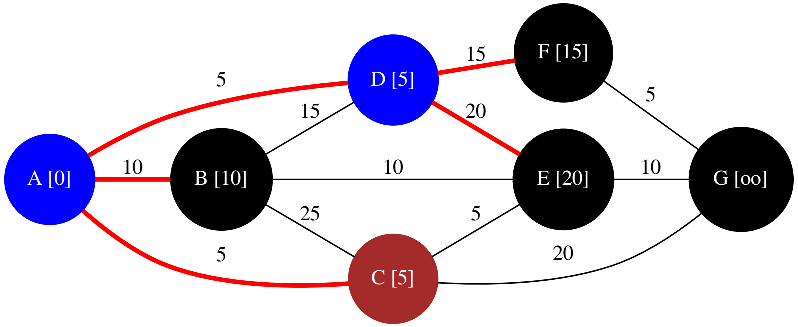
O Algoritmo de Prim - Exemplo



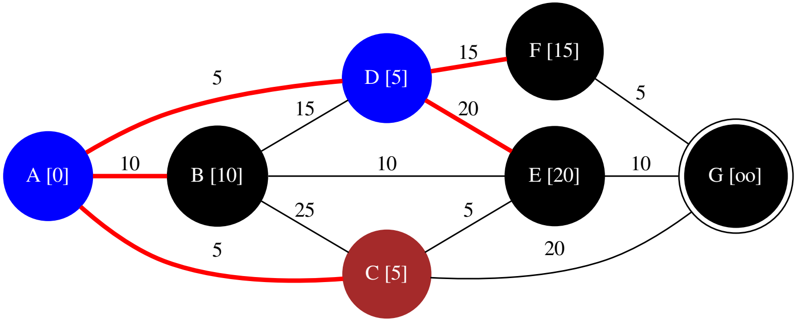
O Algoritmo de Prim - Exemplo



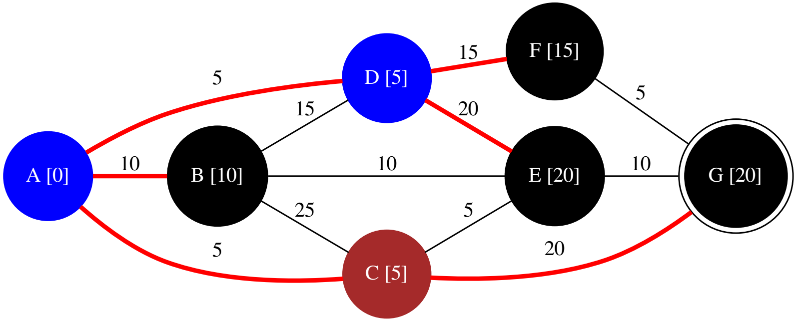
O Algoritmo de Prim - Exemplo



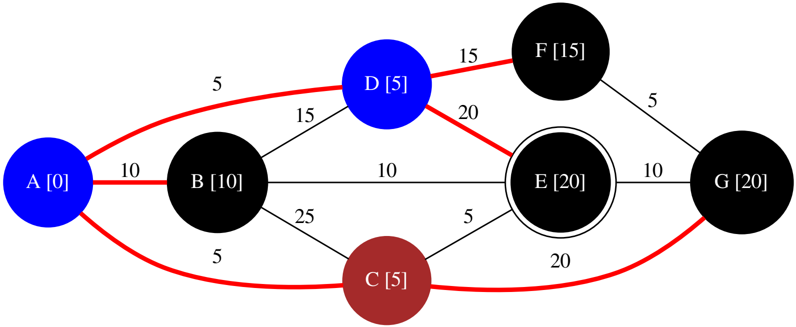
O Algoritmo de Prim - Exemplo



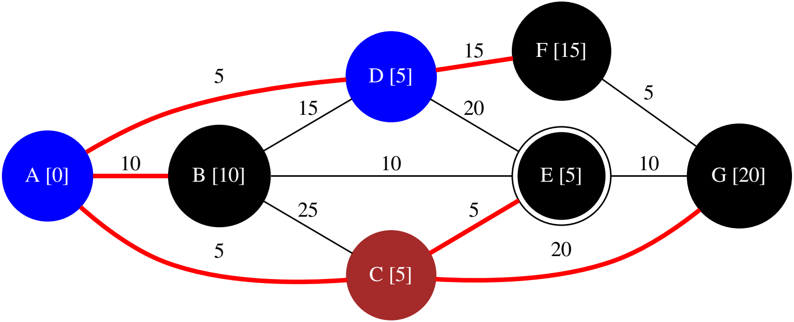
O Algoritmo de Prim - Exemplo



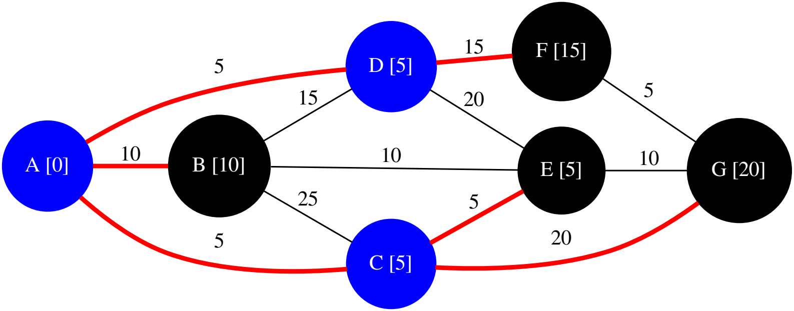
O Algoritmo de Prim - Exemplo



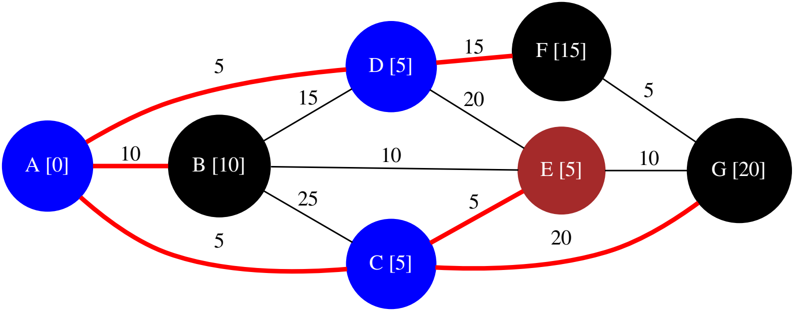
O Algoritmo de Prim - Exemplo



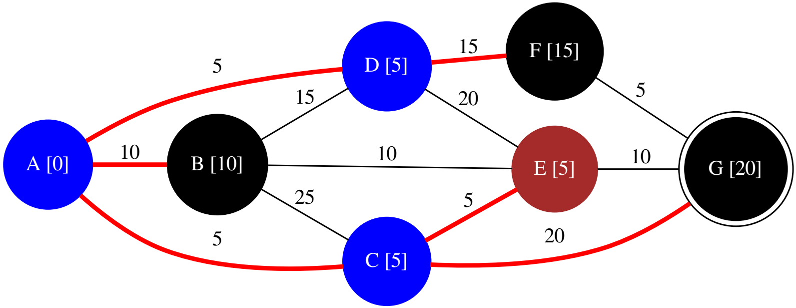
O Algoritmo de Prim - Exemplo



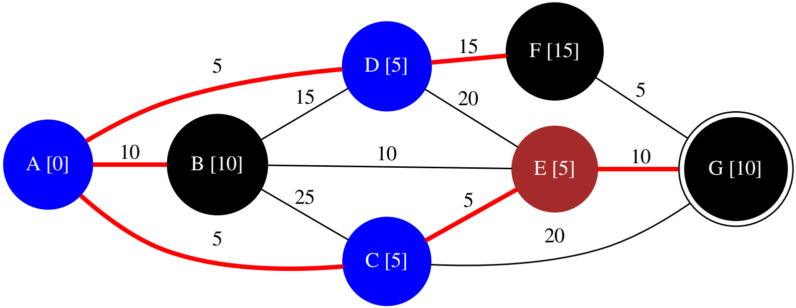
O Algoritmo de Prim - Exemplo



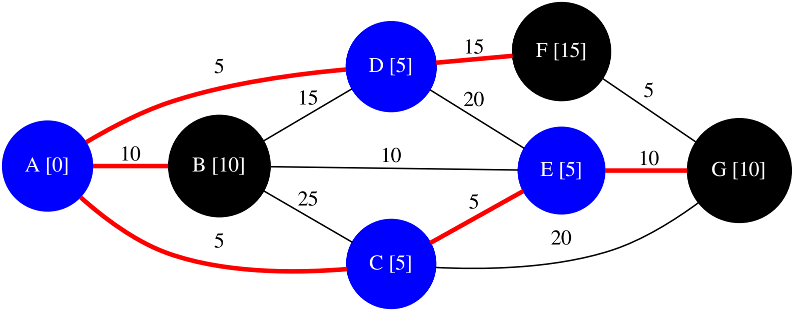
O Algoritmo de Prim - Exemplo



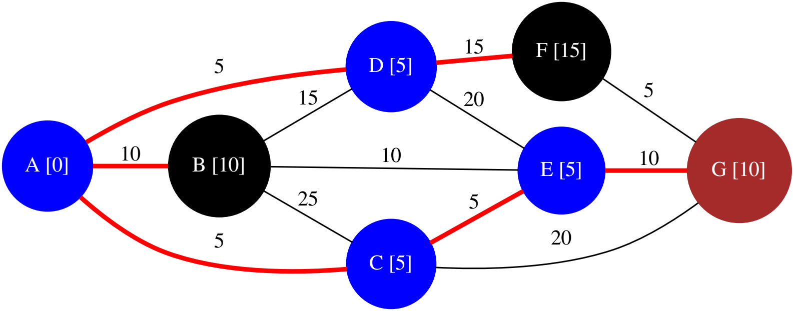
O Algoritmo de Prim - Exemplo



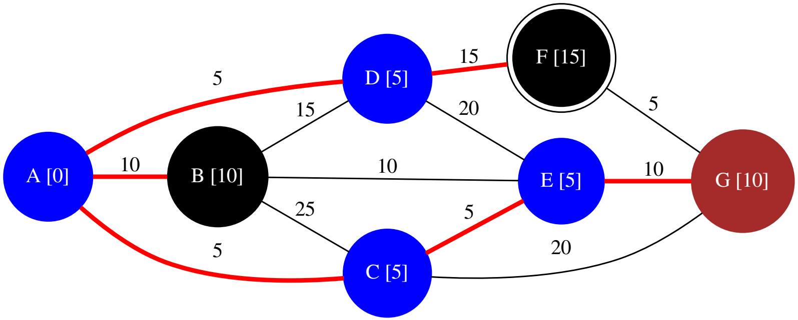
O Algoritmo de Prim - Exemplo



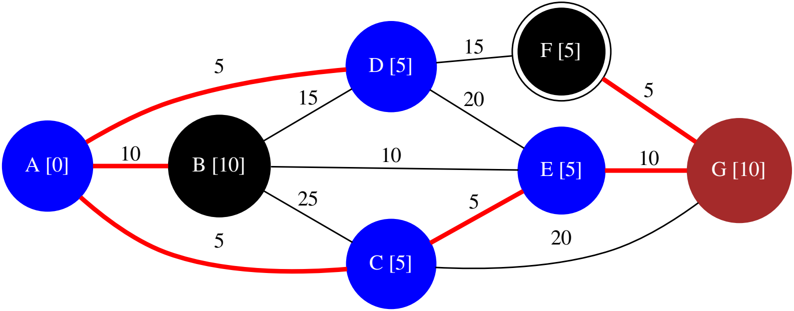
O Algoritmo de Prim - Exemplo



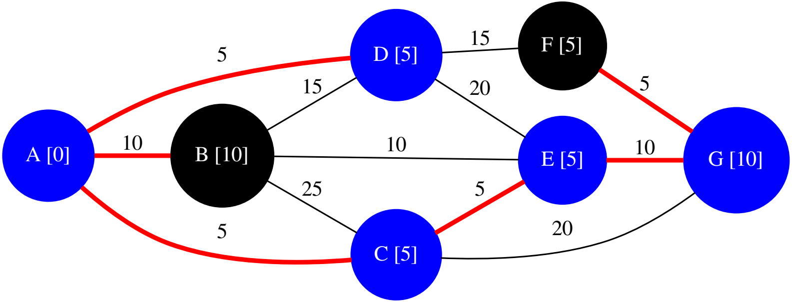
O Algoritmo de Prim - Exemplo



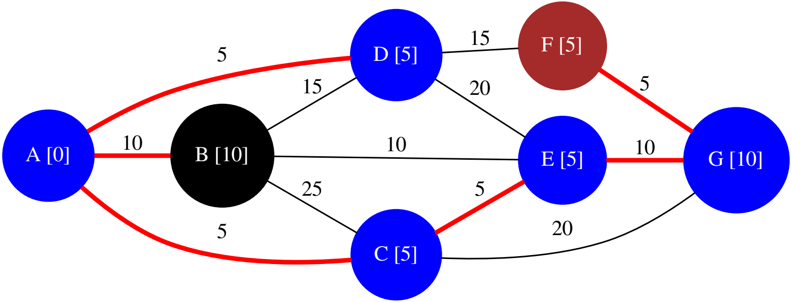
O Algoritmo de Prim - Exemplo



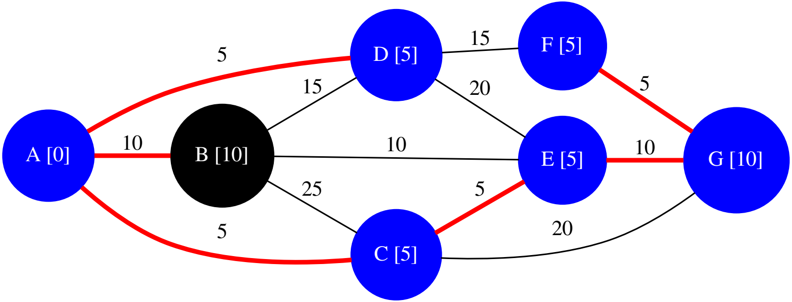
O Algoritmo de Prim - Exemplo



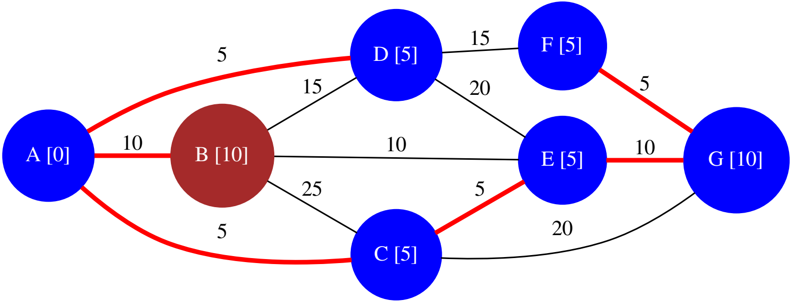
O Algoritmo de Prim - Exemplo



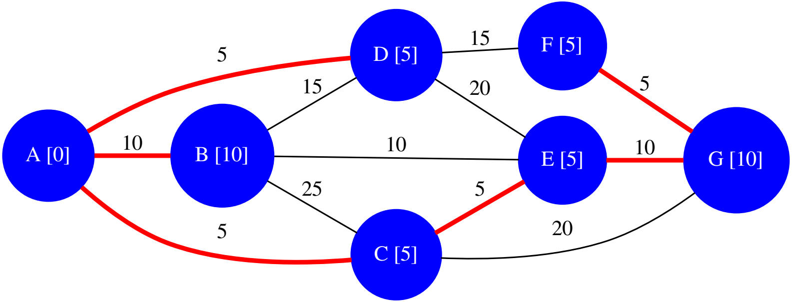
O Algoritmo de Prim - Exemplo



O Algoritmo de Prim - Exemplo



O Algoritmo de Prim - Exemplo



AGPM - O Algoritmo de Prim

Detalhes de Implementação

Exatamente como ocorreu no nosso estudo do algoritmo de Dijkstra, precisamos encontrar uma forma de recuperar o vértice com menor valor de D eficientemente. Para isso, podemos utilizar um heap mínimo.

Detalhes de Implementação - Heap

Defina D[v] $= \infty$, marcado[v] $= 0$, predecessor[v] $=$ NULL $\forall v \in V$.
Defina D[v_1] $= 0$.
Defina heap = new Heap
heap.push(v_1, 0)
Enquanto **heap contiver elementos** faça:
    Defina (v_{at}, minD) = heap.top()//pega o valor mínimo
    heap.pop()//remove o valor mínimo
    se marcado[v_{at}] = 0:
        Defina marcado[v_{at}] $= 1$.
        Para toda aresta $e = (v_{at}, v_{prox})$ faça:
            Se marcado[v_{prox}] $= 0$ E D[v_{prox}] $> W(e)$:
                Defina D[v_{prox}] $= W(e)$.
                Defina predecessor[v_{prox}] $= v_{at}$.
                heap.push(v_{prox}, D[v_{prox}])
retorne D[v_2]

AGPM - Algoritmo de Kruskal

Ideia Geral

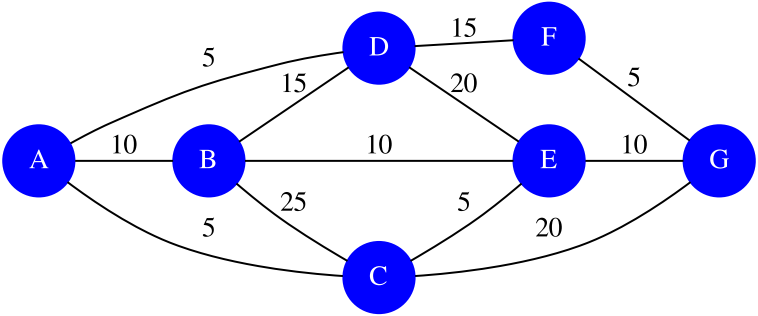
No algoritmo de Kruskal, ordenamos as arestas de forma não-decrescente. Iteramos sobre as arestas, começando pela de menor custo, e adicionamos uma aresta à resposta sempre que suas extremidades não estiverem conectadas pelas arestas adicionadas anteriormente.

AGPM - Algoritmo de Kruskal

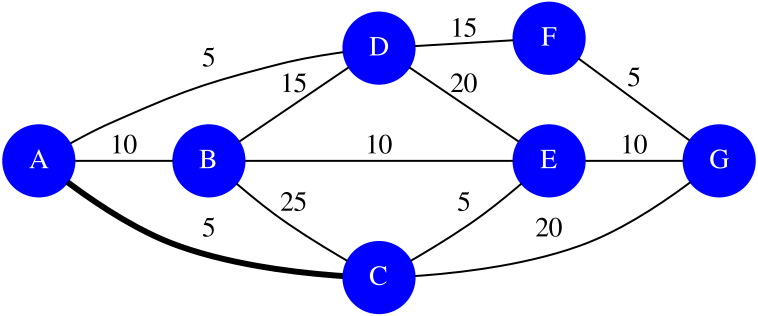
O Algoritmo de Kruskal: Pseudo-Código

Ordene o vetor de arestas $E = [(u_1, v_1, w_1), \ldots, (u_m, v_m, w_m)]$ pelo peso w_i.
Defina $Arvore = \emptyset$
Para $1 \leq i \leq m$ faça:
    Defina $e = E[i] = (u, v, w)$
    Se **não existe** um caminho de u até v utilizando arestas contidas em $Arvore$:
        Faça $Arvore = Arvore \cup \{e\}$
x

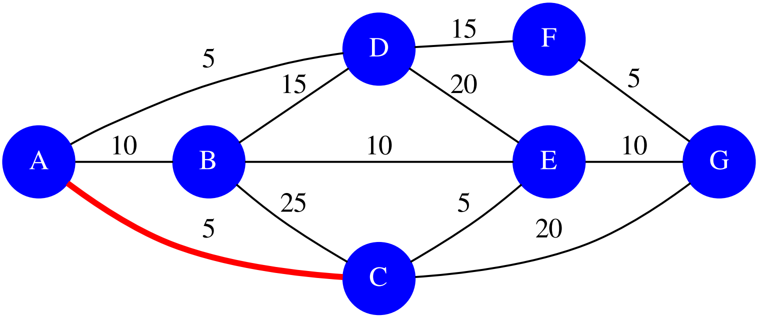
O Algoritmo de Kruskal - Exemplo



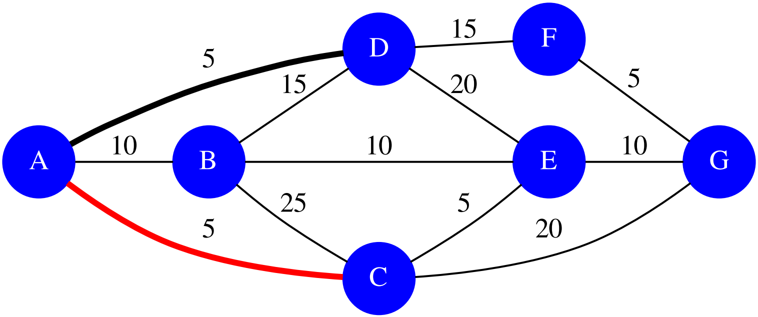
O Algoritmo de Kruskal - Exemplo



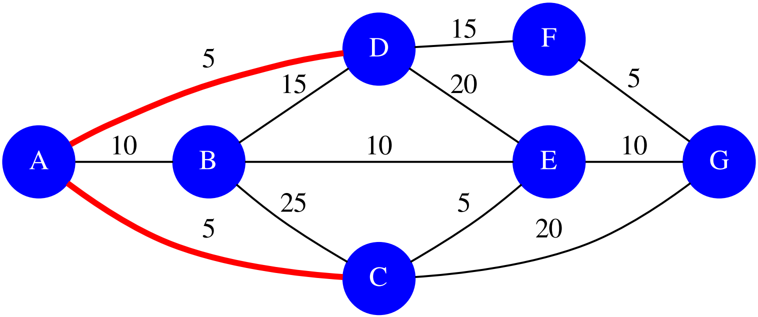
O Algoritmo de Kruskal - Exemplo



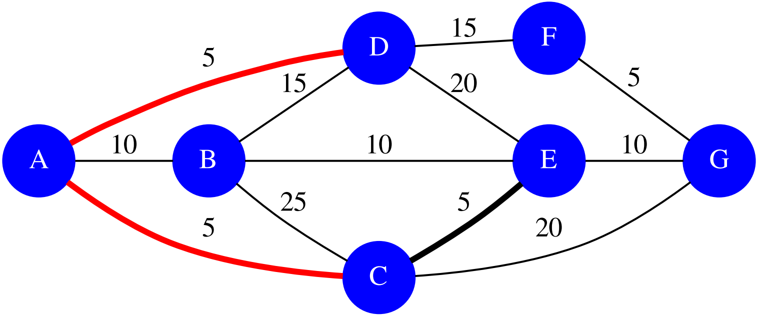
O Algoritmo de Kruskal - Exemplo



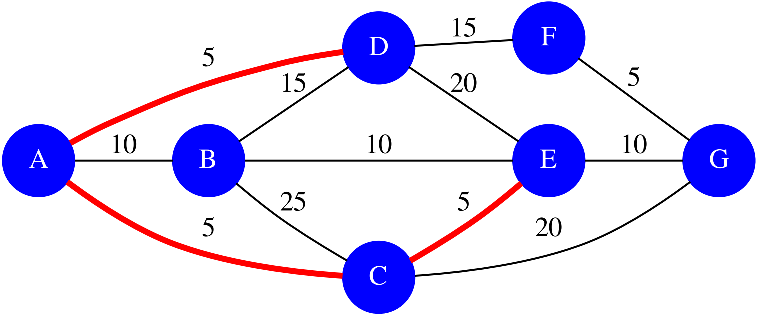
O Algoritmo de Kruskal - Exemplo



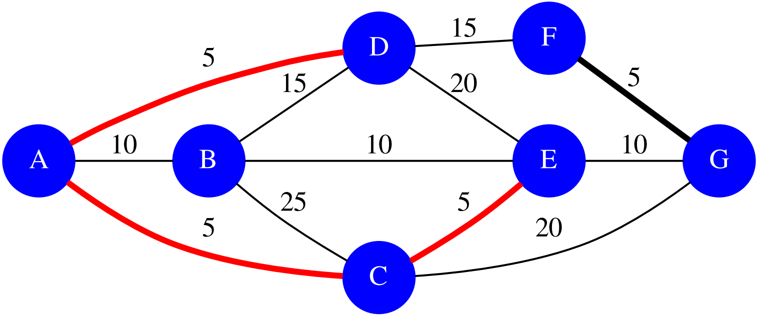
O Algoritmo de Kruskal - Exemplo



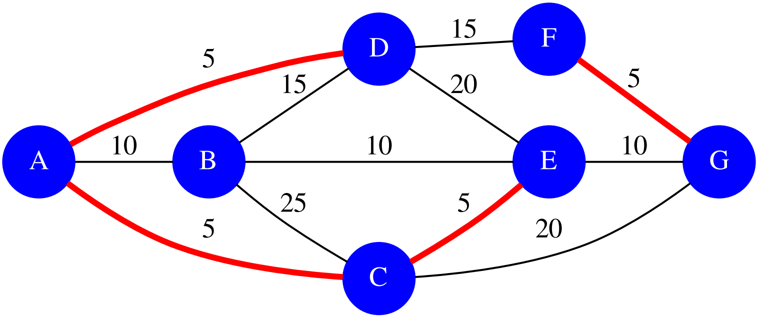
O Algoritmo de Kruskal - Exemplo



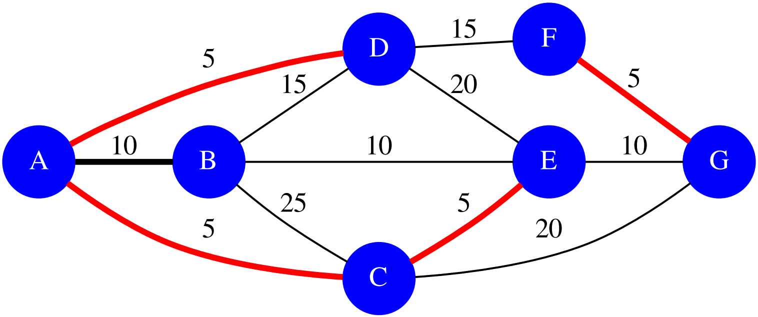
O Algoritmo de Kruskal - Exemplo



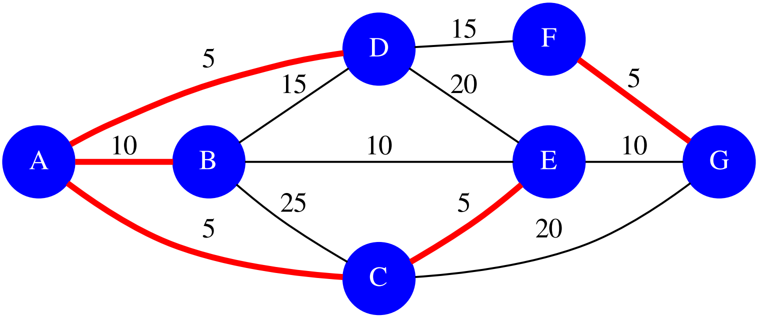
O Algoritmo de Kruskal - Exemplo



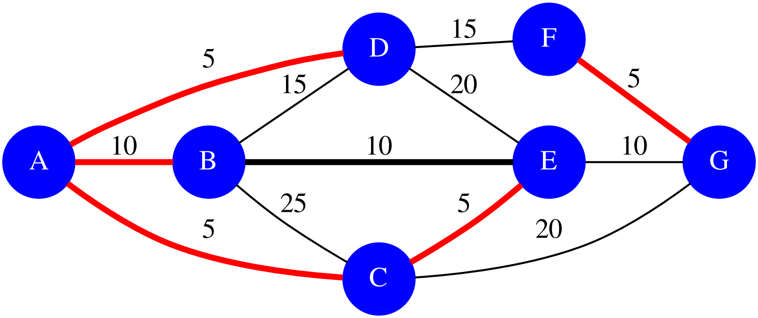
O Algoritmo de Kruskal - Exemplo



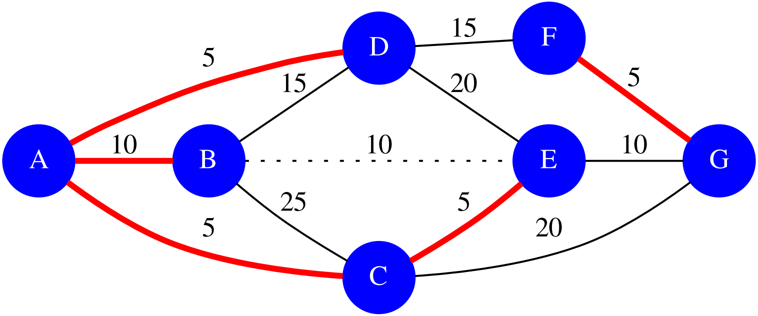
O Algoritmo de Kruskal - Exemplo



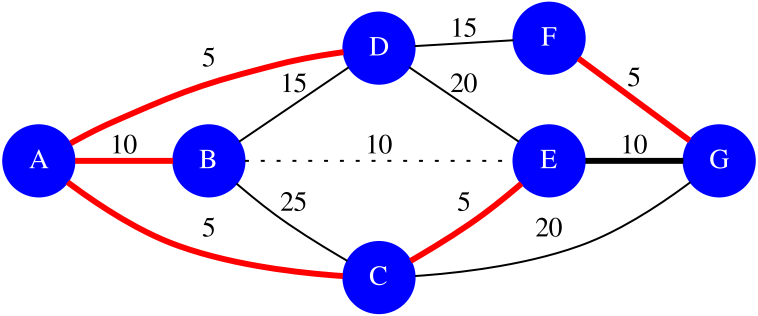
O Algoritmo de Kruskal - Exemplo



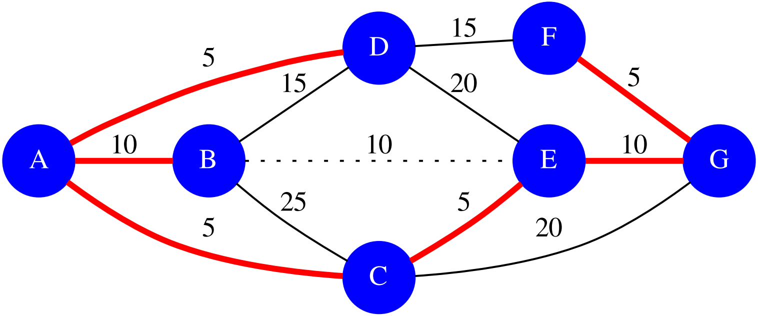
O Algoritmo de Kruskal - Exemplo



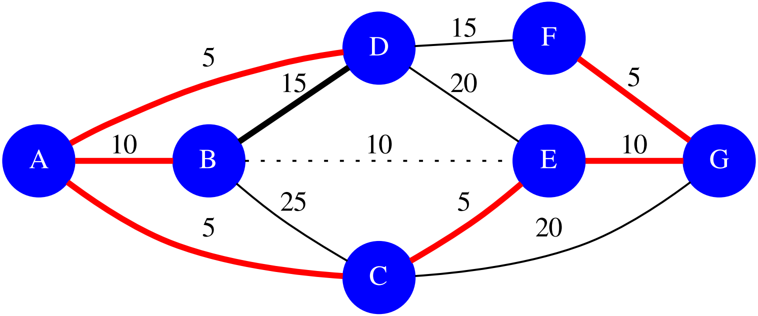
O Algoritmo de Kruskal - Exemplo



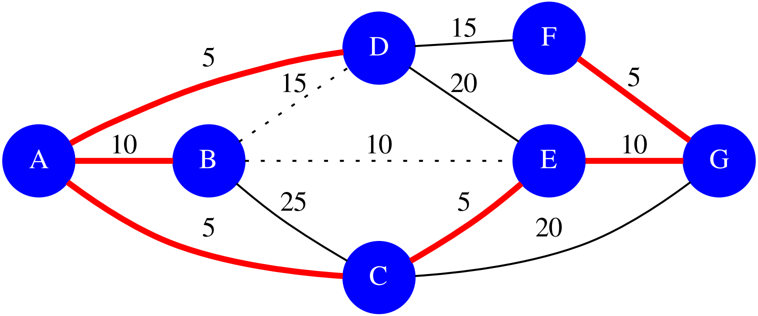
O Algoritmo de Kruskal - Exemplo



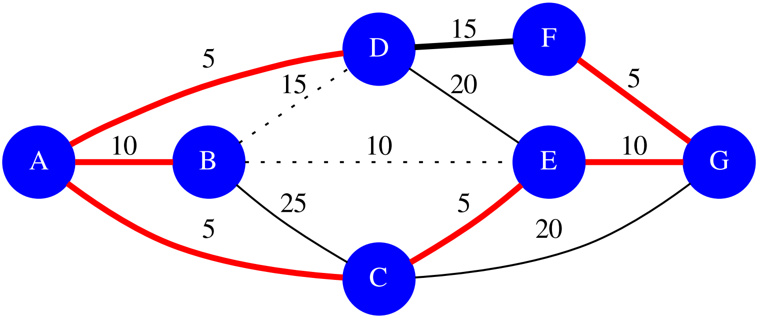
O Algoritmo de Kruskal - Exemplo



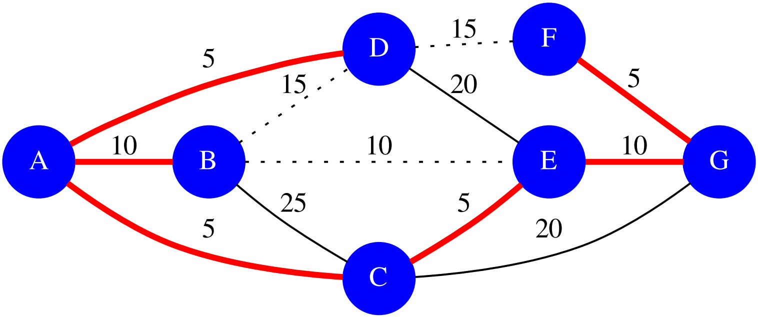
O Algoritmo de Kruskal - Exemplo



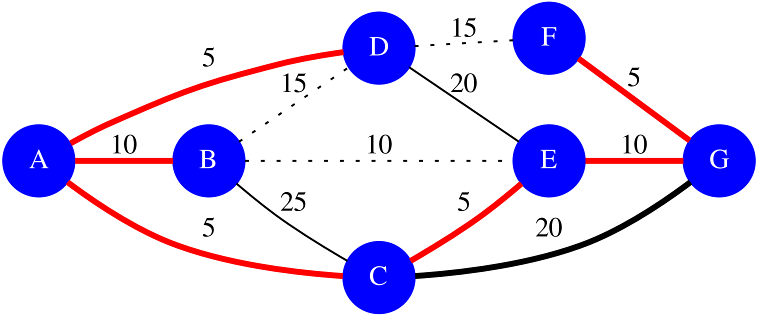
O Algoritmo de Kruskal - Exemplo



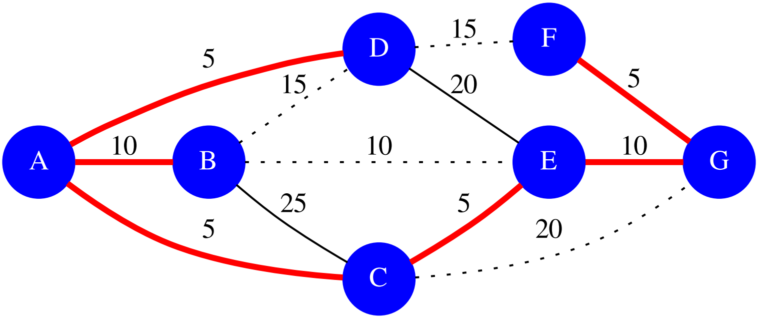
O Algoritmo de Kruskal - Exemplo



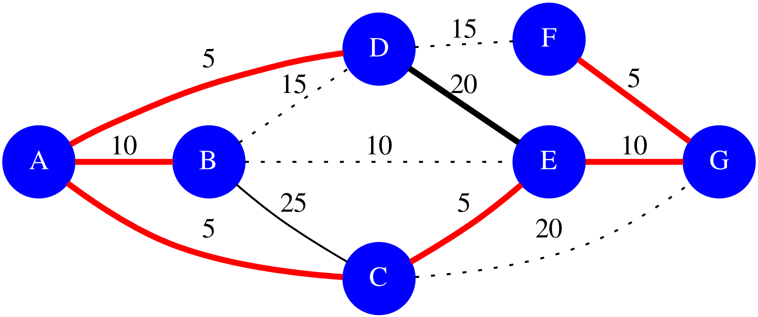
O Algoritmo de Kruskal - Exemplo

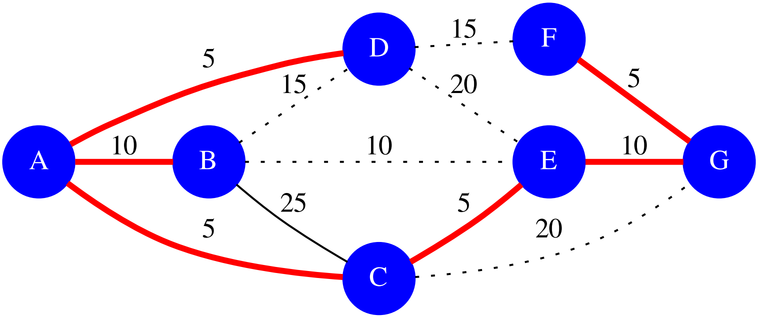


O Algoritmo de Kruskal - Exemplo

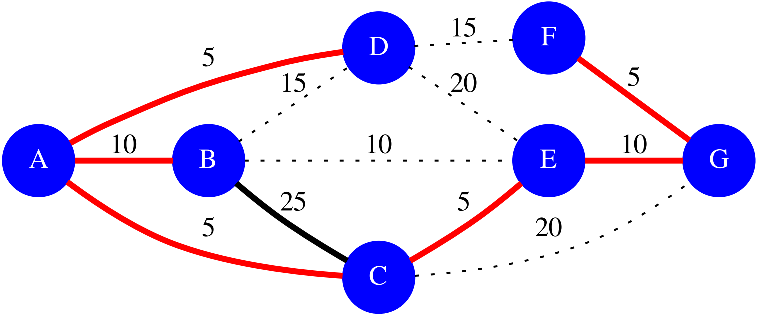


O Algoritmo de Kruskal - Exemplo

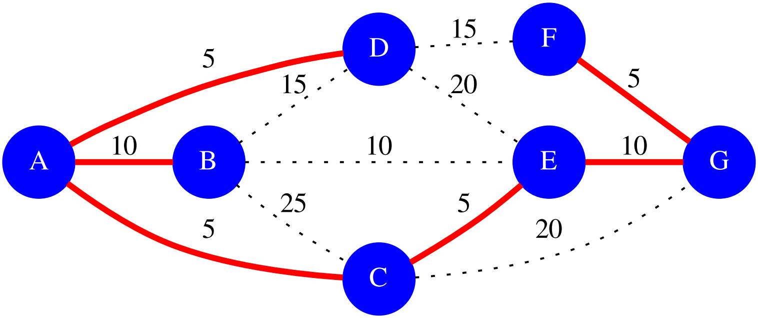



A
B
C
D
E
F
G
5
10
5
15
25
15
20
10
5
20
5
10

O Algoritmo de Kruskal - Exemplo



O Algoritmo de Kruskal - Exemplo



AGPM - Algoritmo de Kruskal

Detalhes de Implementação

Como saber se dois vértices estão ligados por algum caminho utilizando arestas já adicionadas à resposta?

Detalhes de Implementação

Como saber se dois vértices estão ligados por algum caminho utilizando arestas já adicionadas à resposta?

R: Podemos utilizar conjuntos disjuntos.

O Algoritmo de Kruskal: Pseudo-Código

Ordene o vetor de arestas $E = [(u_1, v_1, w_1), \ldots, (u_m, v_m, w_m)]$ pelo peso w_i.
Defina $Arvore = \emptyset$
Execute MakeSet(n)
Para $1 \leq i \leq m$ faça:
    Defina $e = E[i] = (u, v, w)$
    Se $find(u) \neq find(v)$
        Faça $Arvore = Arvore \cup \{e\}$
        union(u, v)

Detalhes de Implementação

- As estratégias de *compressão de caminho* e *união por rank* ajudam a melhorar a eficiência da implementação.
- Na implementação do algoritmo de Prim, a representação do grafo por lista de incidência é, em geral, mais coveniente.
- Já para o Kruskal, é mais prático guardar apenas uma lista de arestas.