Busca em Grafos e o Algoritmo de Dijkstra

Algoritmos e Estruturas de Dados - EC

June 27, 2014

O que já vimos sobre Grafos e Buscas : Representações

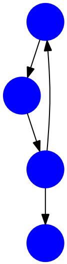
Definição Matemática

Um Grafo G é formado por um conjunto V de vértices e E de arestas. G = (V, E).
Arestas são definidas pelo par de vértices conectados por ela, $e_i = (u, v)$.
Arestas podem ser direcionadas, não direcionadas, podem ter peso, capacidade ou outros valores específicos do problema modelado pelo Grafo.

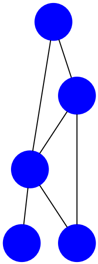
O que já vimos sobre Grafos e Buscas : Representações

Representação gráfica

Representamos o conjunto de vértices por círculos e as arestas por arcos ligando esses círculos.



Grafo direcionado

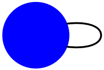


Não direcionado

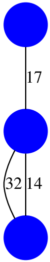
O que já vimos sobre Grafos e Buscas : Representações

Representação gráfica

Representamos o conjunto de vértices por círculos e as arestas por arcos ligando esses círculos.



Um loop



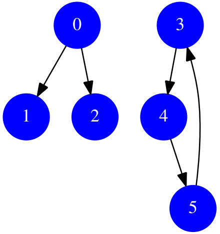
Arestas múltiplas e com peso

Representação Computacional

Para implementar um grafo, podemos escolher entre as estratégias da *Matriz de Adjacência* ou da *Lista de Incidência*.

Matriz de Adjacência

É uma matriz $|V|x|V|$ em que a célula (i,j) tem valor 1 se existe uma aresta entre os vértices (i,j) no grafo. Caso contrário, o valor da célula é 0.

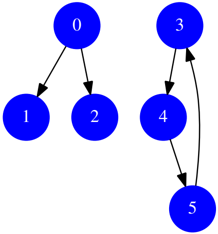


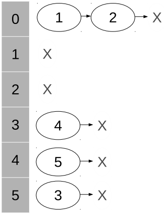
	0	**1**	**2**	**3**	**4**	**5**
0	0	1	1	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	1	0	0

O que já vimos sobre Grafos e Buscas : Representações

Lista de Incidência

É uma versão compacta da matriz de adjacência. Consiste de $|V|$ listas, uma para cada vértice do grafo. Cada elemento da *i-ésima* lista representa um dos vizinhos do vértice i no grafo.





O que já vimos sobre Grafos e Buscas : Buscas

Buscas

- Vimos que o objetivo de se fazer uma busca em um grafo é percorrer todos os seus vértices e todas as suas arestas de forma sistemática. Nesse processo devemos ter cuidado para evitar repetições, tanto de vértices quanto de arestas. No final, teremos uma **Árvore** formada pelas arestas que percorremos.
- Vimos que o algoritmo básico de busca escolhe um vértice inicial e percorre arestas incidentes a vértices visitados. Percebemos que, dependendo do critério de escolha dessas arestas, podemos ter resultados diferentes.

O que já vimos sobre Grafos e Buscas : Buscas

Busca em Profundidade

- Na busca em profundidade alteramos o algoritmo básico de busca para percorrer arestas incidentes ao vértice **mais recentemente** visitado que ainda possui arestas não marcadas.
- Vimos que podemos encontrar o vértice mais recentemente visitado utilizando uma pilha (fácil). Alternativamente podemos implementar a busca em profundidade utilizando recursão (ainda mais fácil).

Busca em Largura

- Na busca em largura tentamos percorrer primeiro as arestas incidentes ao vértice **menos recentemente** visitado que ainda possui arestas não marcadas.
- Vimos que a maneira natural de implementar a busca em largura é utilizando uma fila.

O que já vimos sobre Grafos e Buscas : Aplicações

Aplicações das Buscas

Por fim, vimos que as buscas podem ser utilizadas para resolver problemas importantes de grafos:

- Determinar se um grafo é conexo. (Busca em Largura ou Profundidade)
- Dividir o grafo em seus componentes conexos. (Buscas em Largura ou Profundidade)
- Encontrar a menor distância entre um vértice e todos os outros, **quando as arestas têm pesos iguais**. (Busca em Largura)
- Determinar se um grafo é bipartido/bicolorível. (Busca em Largura ou Profundidade)

O que ainda não vimos sobre Grafos e Buscas : O Algoritmo de Dijkstra

Algoritmo de Dijkstra

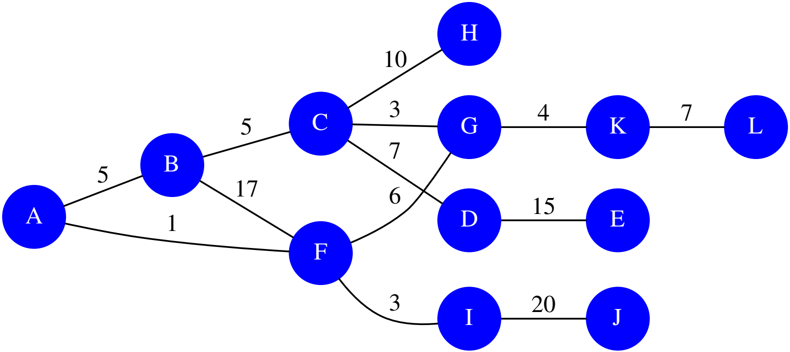
O algoritmo de Dijkstra é mais uma maneira de se realizar uma busca em grafos.
Mas dessa vez vamos começar com uma motivação diferente: o problema de encontrar um caminho de custo mínimo entre um par de vértices.

O Algoritmo de Dijkstra

Caminho de Custo Mínimo

- Um caminho, em um grafo $G = (V, E)$ é uma sequência de vértices $V = v_1, v_2, ..., v_k$ tal que, $\forall i < k - 1$ a aresta $(v_i, v_{i+1}) \in E$.
- Um caminho de A até B é um caminho cujo primeiro vértice é A e o último é B.
- Definimos o custo de um caminho V como $C(V) = \sum_{i=1}^{|V|-1} W(v_i, v_{i+1})$ onde $W(u, v)$ é o custo da aresta (u, v).
- Por fim, um caminho $V_{A,B}$ de A até B é de custo mínimo se $\nexists V'_{A,B}$ tal que $C(V'_{A,B}) < C(V_{A,B})$.

O Algoritmo de Dijkstra



O caminho A, B, C, H é um caminho **mínimo** de A até H. Seu custo é 20.
O caminho A, F, G, C, H também é um um caminho **mínimo** de A até H.
A sequência A, B, F, G, C, H é um caminho de A até H. Mas não é mínimo, seu custo é 41.

O Algoritmo de Dijkstra

A Estratégia

- Com o algoritmo de Dijkstra resolvemos o seguinte problema: dado um grafo G = (V, E) com pesos nas arestas e dois vértices $v_1, v_2 \in V$ queremos encontrar um (ou todos) caminho mínimo entre $v_1 e v_2$.
- Durante a busca, utilizamos uma estratégia **gulosa** para escolher quais vértices visitar primeiro. Dizemos que é uma estratégia gulosa porque, informalmente, fazemos **escolhas ótimas locais** para atingir um estado **ótimo global**.
- À cada iteração descobrimos um pouco mais da resposta final: à cada iteração definimos a menor distância entre o vértice origem v_1 e algum outro vértice do grafo.

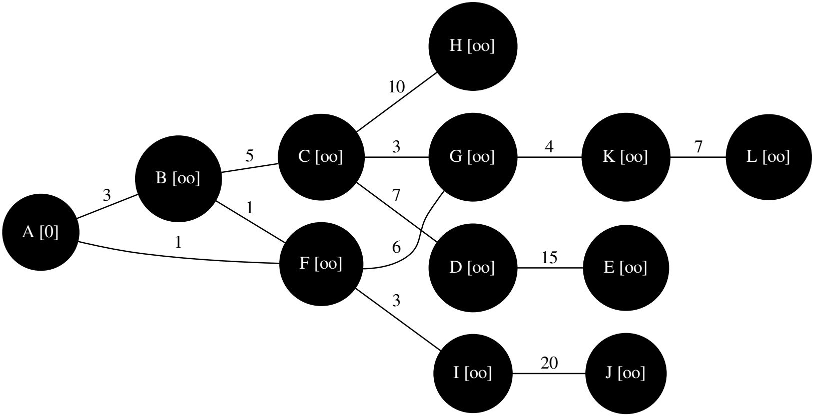
O Algoritmo de Dijkstra

O Algoritmo: Pseudo-Código

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

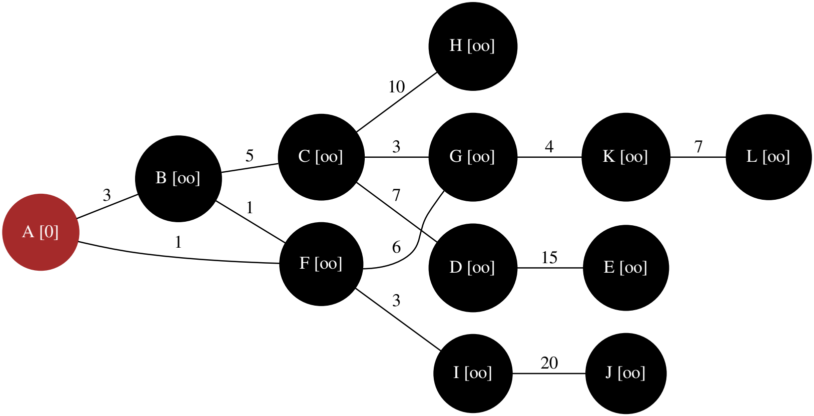
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

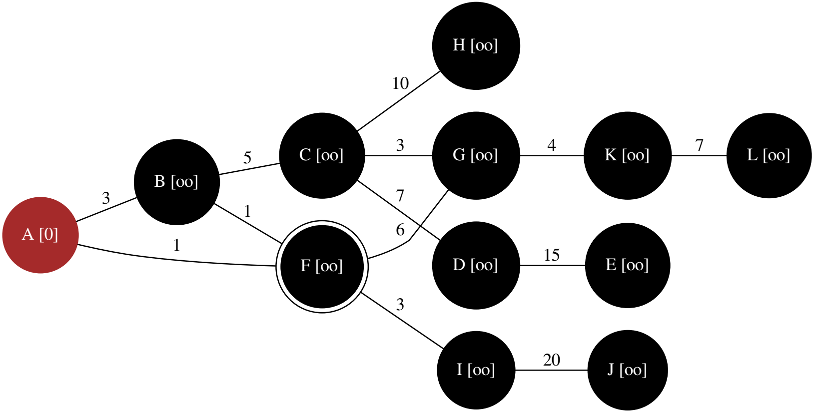


O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

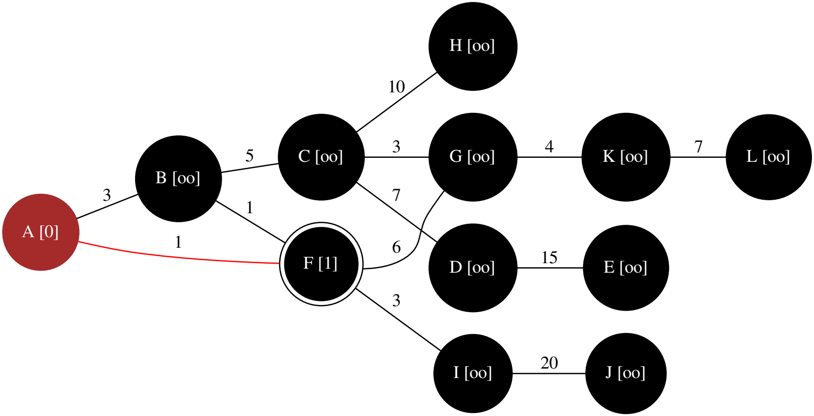


Exemplo: encontrar um caminho de custo mínimo entre A e E.



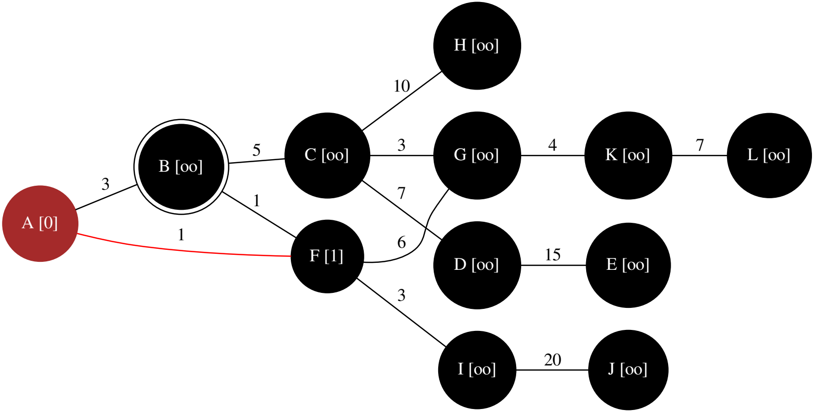
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



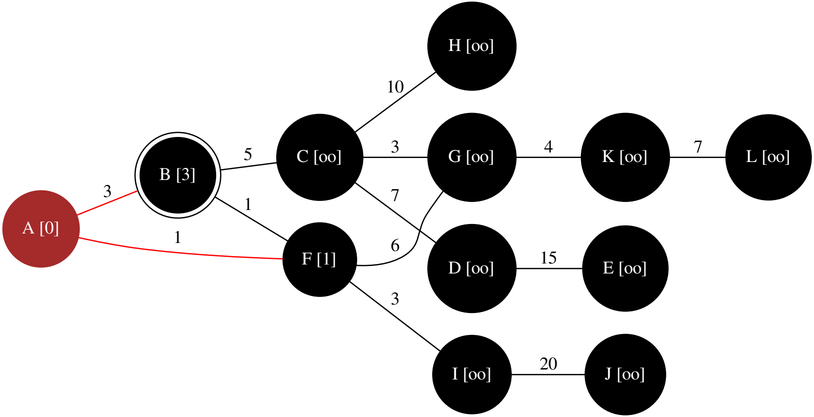
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



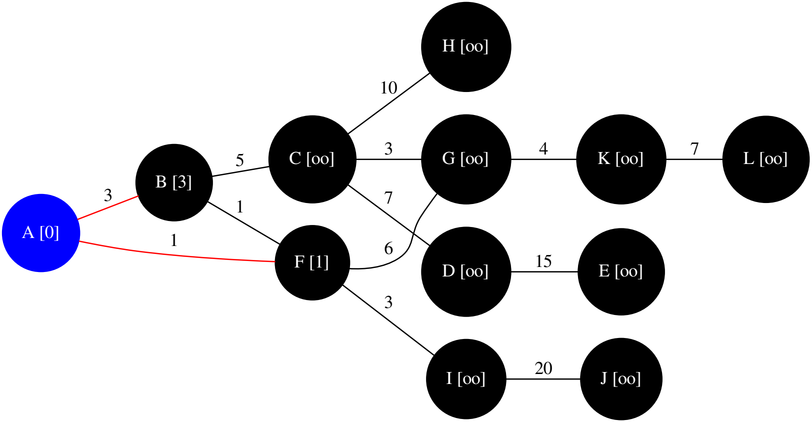
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



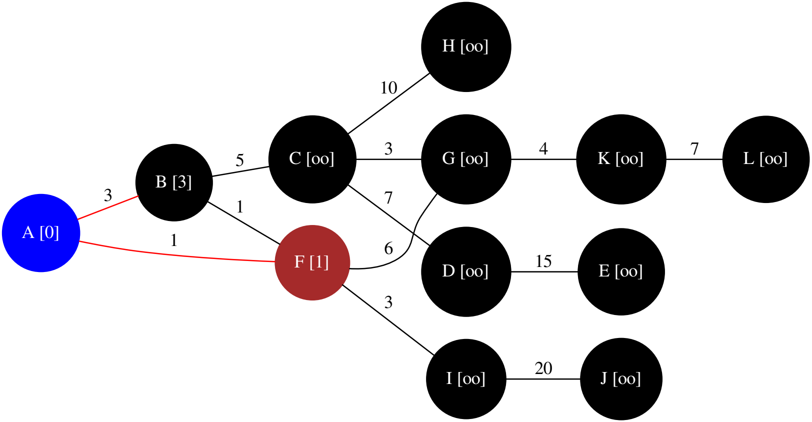
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



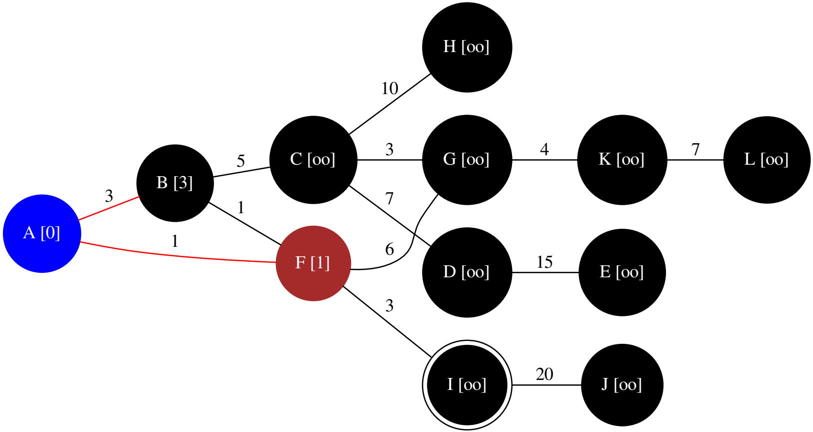
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



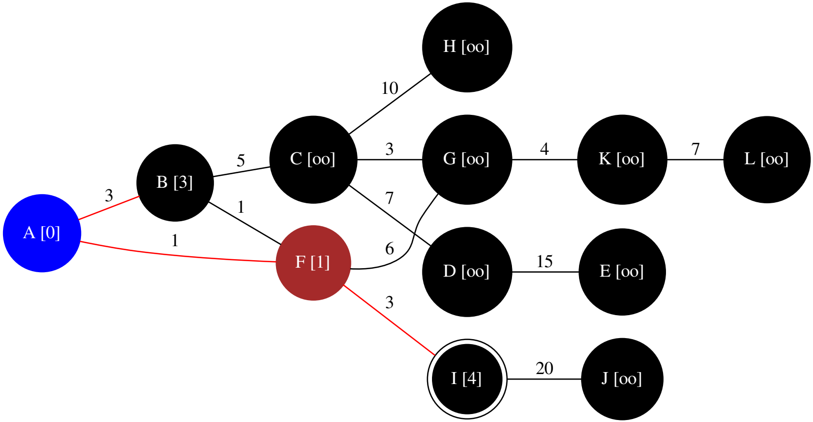
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



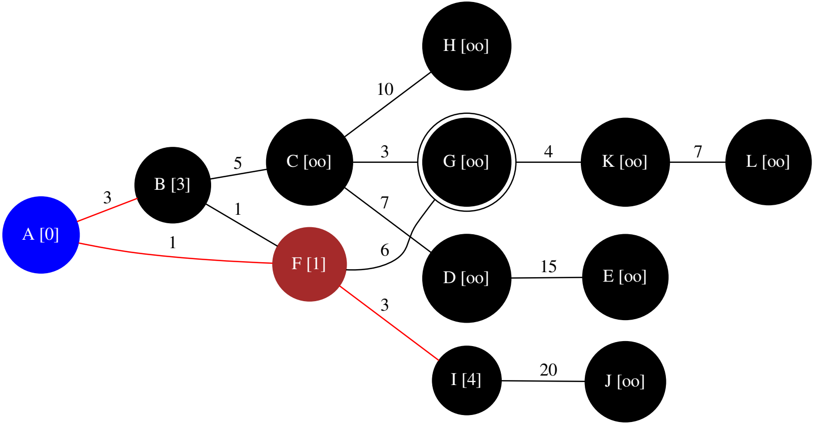
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



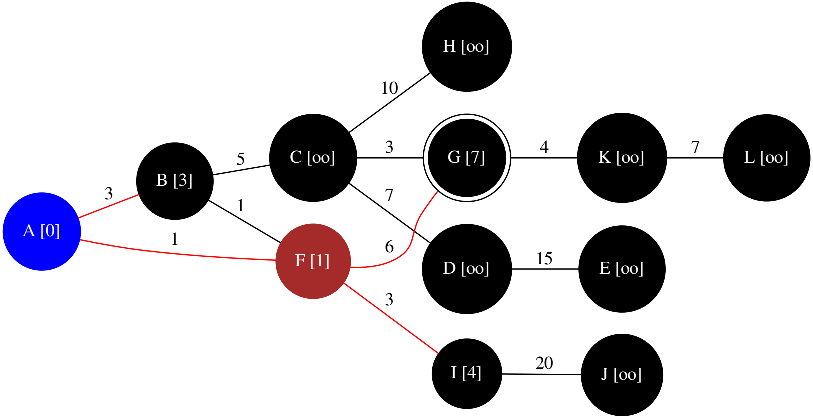
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



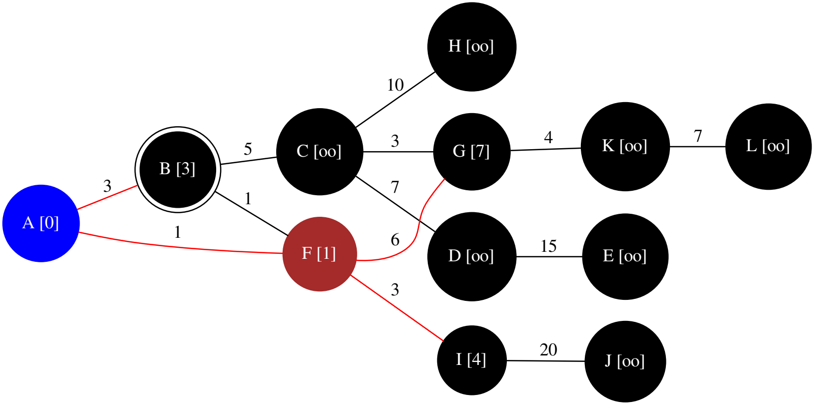
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



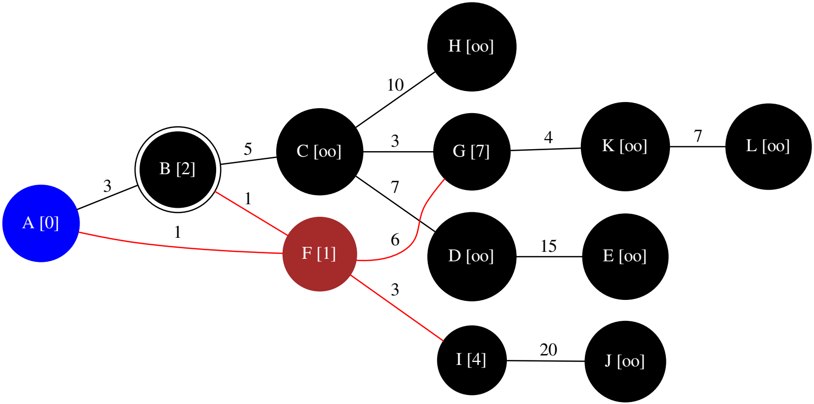
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



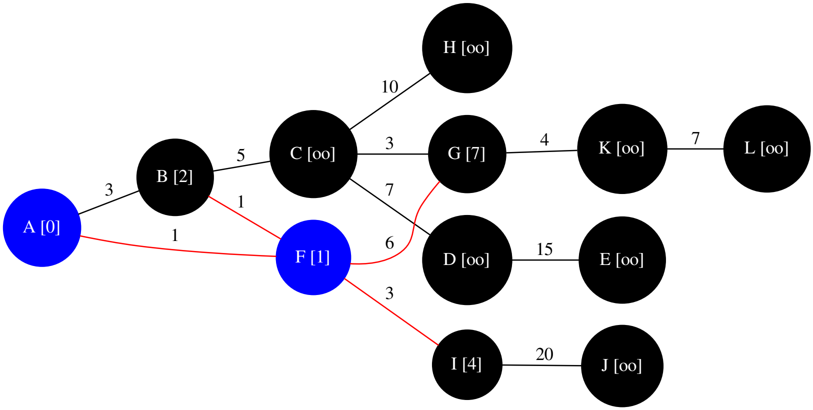
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



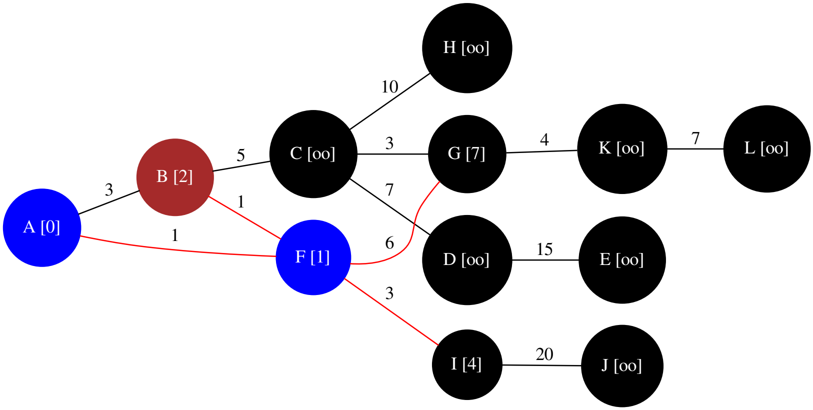
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

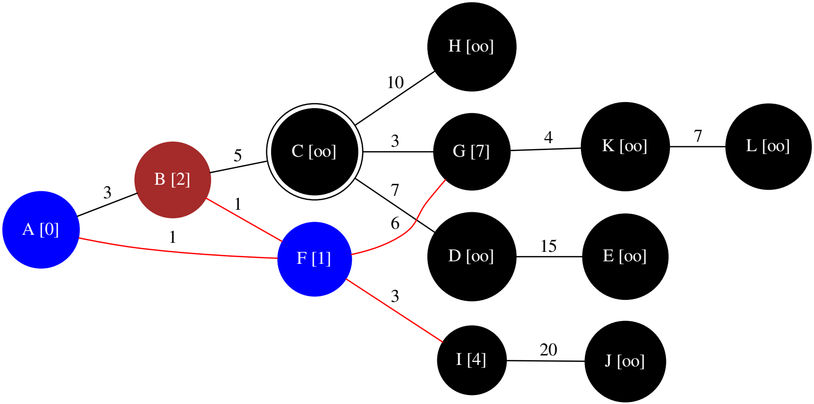


O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

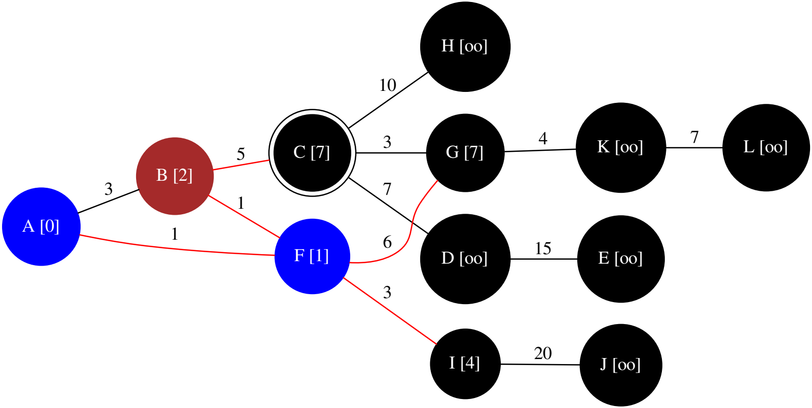


Exemplo: encontrar um caminho de custo mínimo entre A e E.



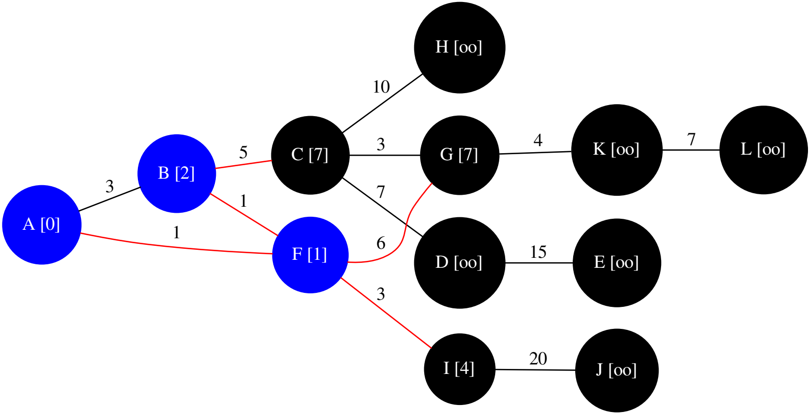
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



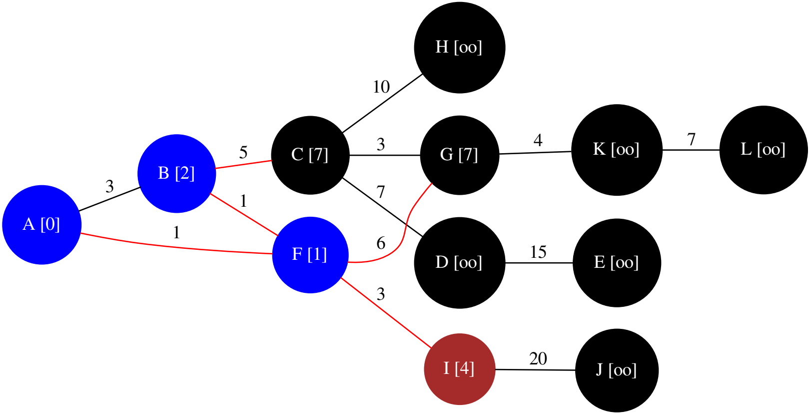
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



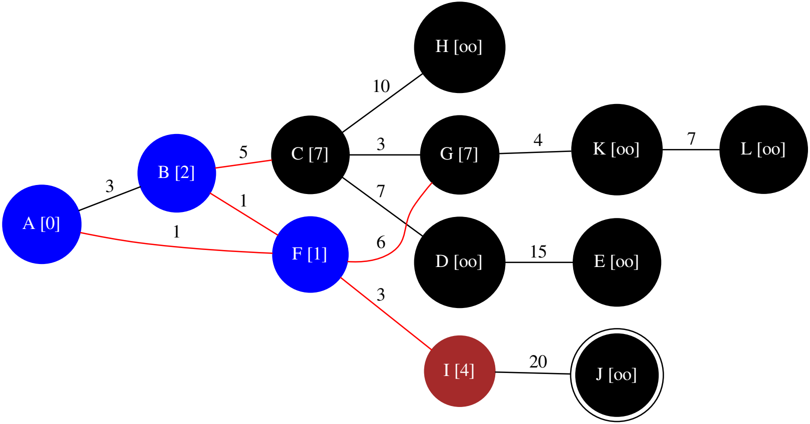
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



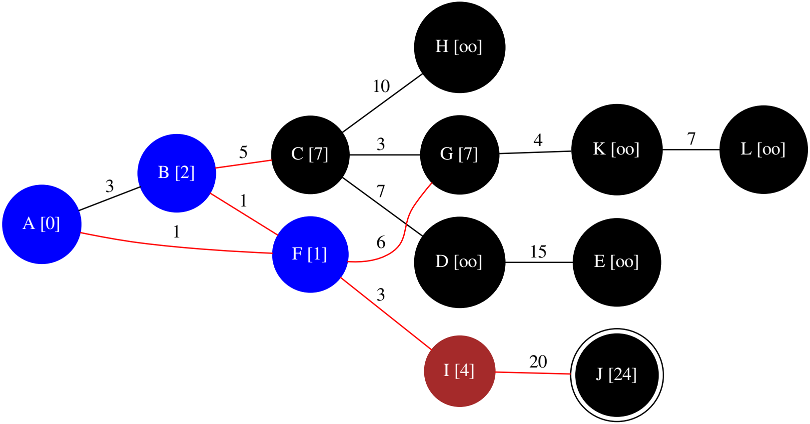
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



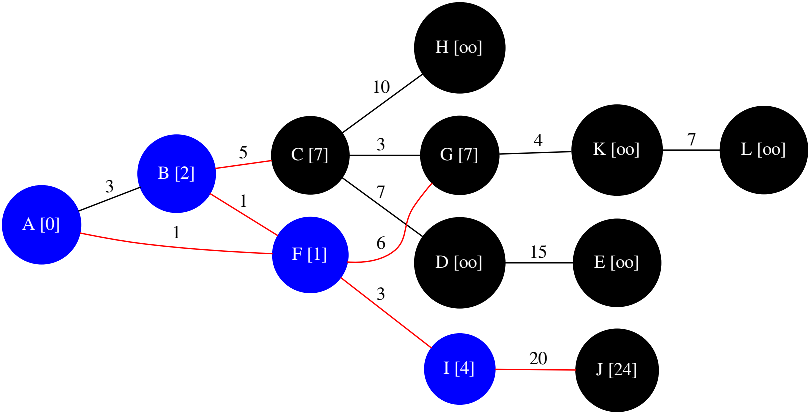
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



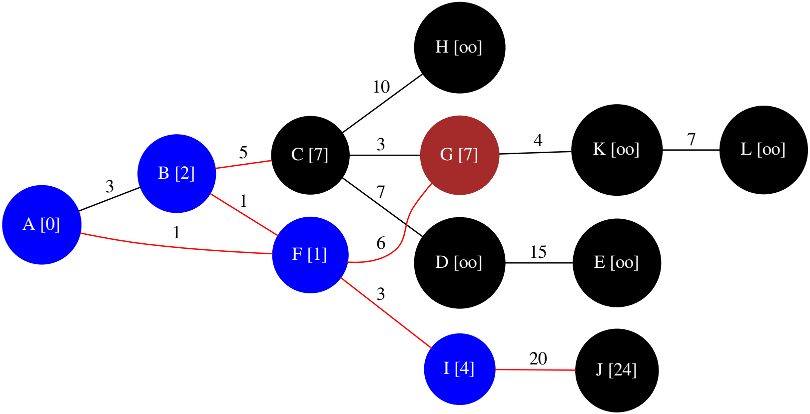
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



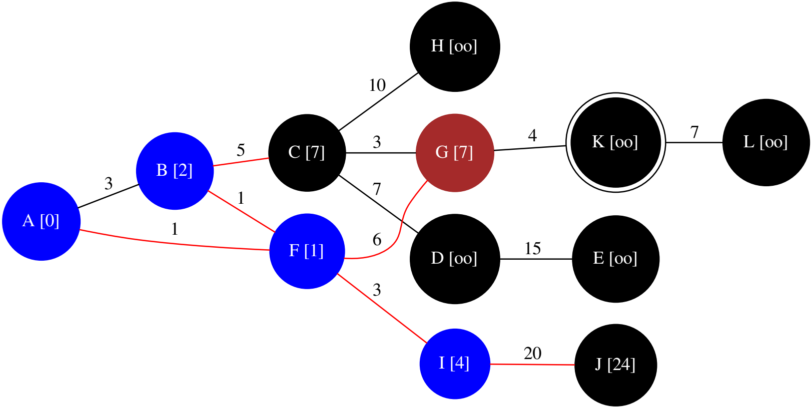
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



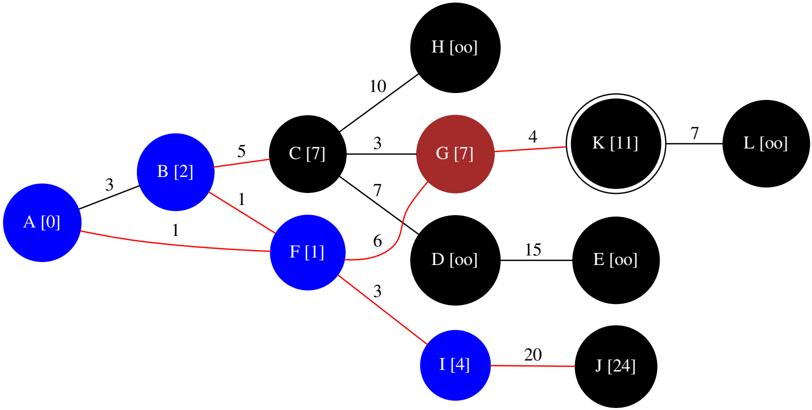
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

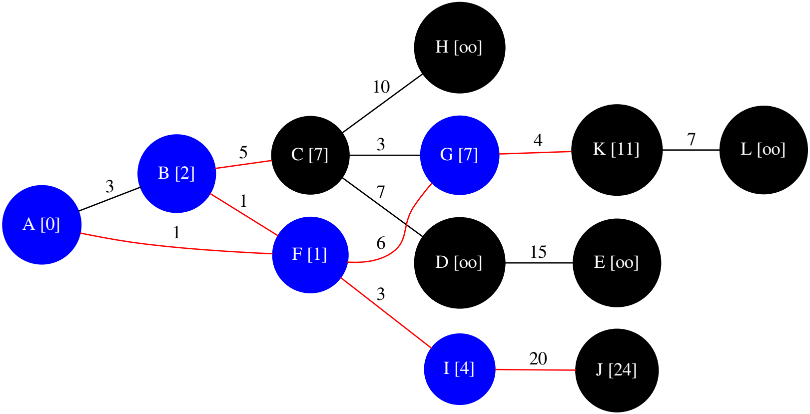


O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

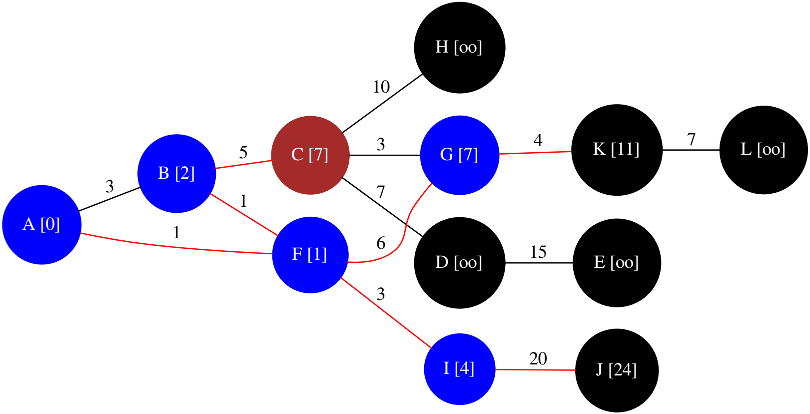


Exemplo: encontrar um caminho de custo mínimo entre A e E.

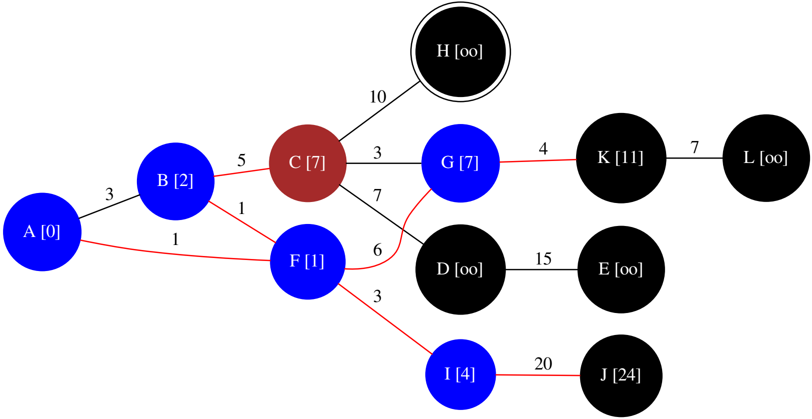


O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

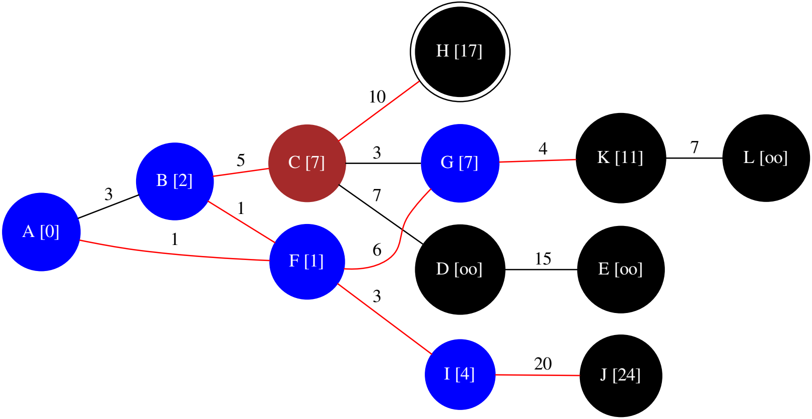


Exemplo: encontrar um caminho de custo mínimo entre A e E.



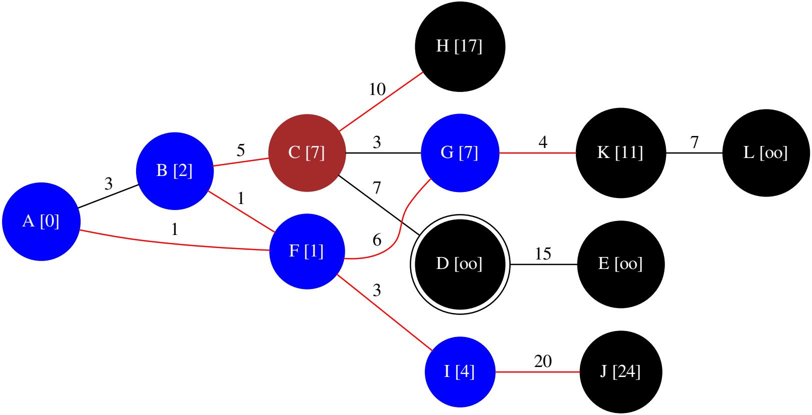
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



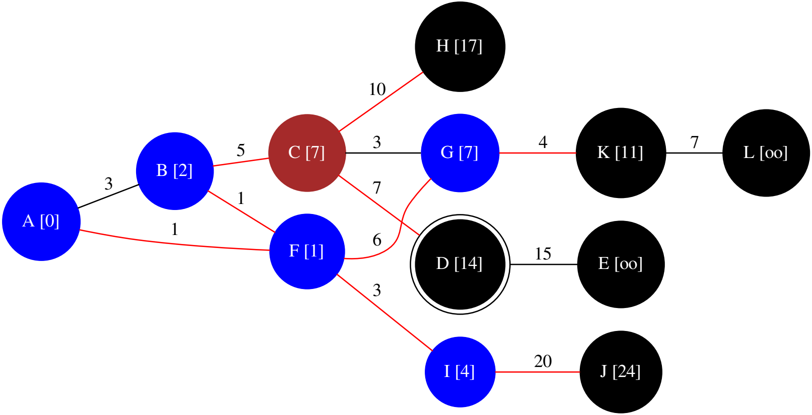
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



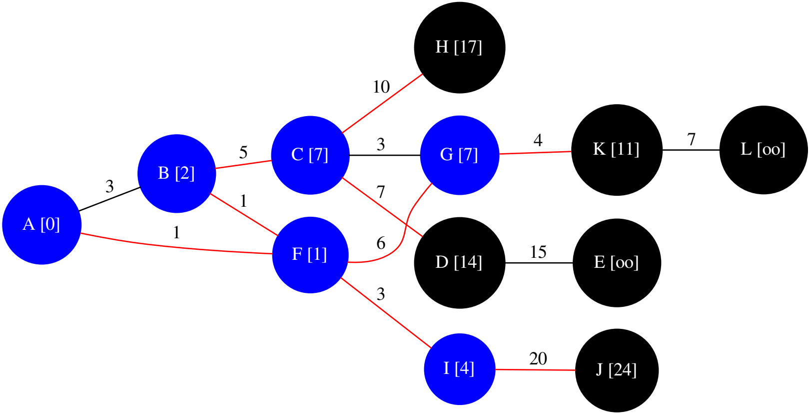
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



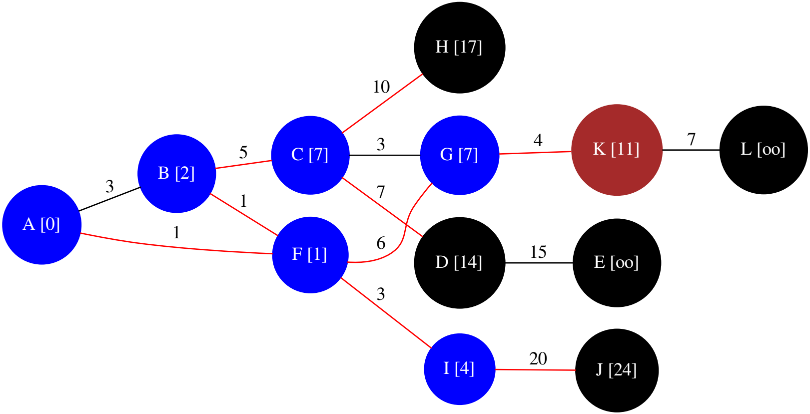
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



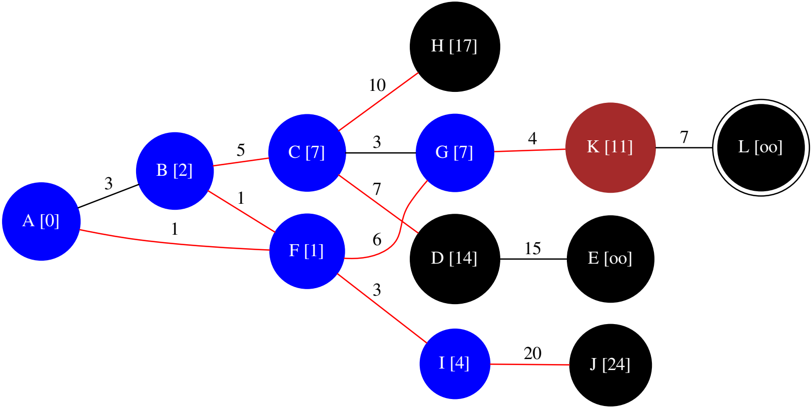
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



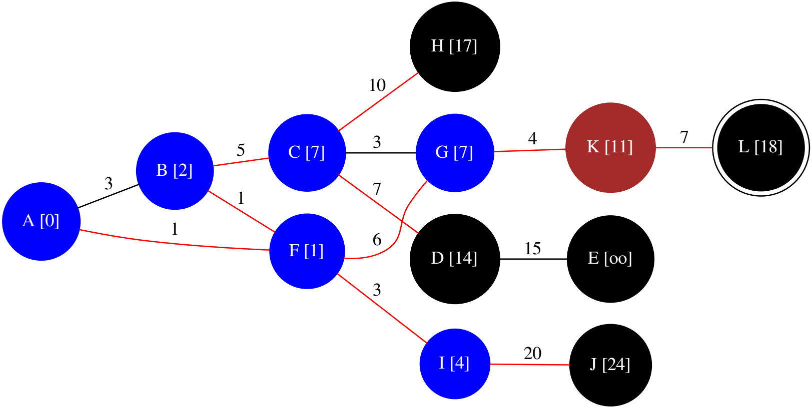
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

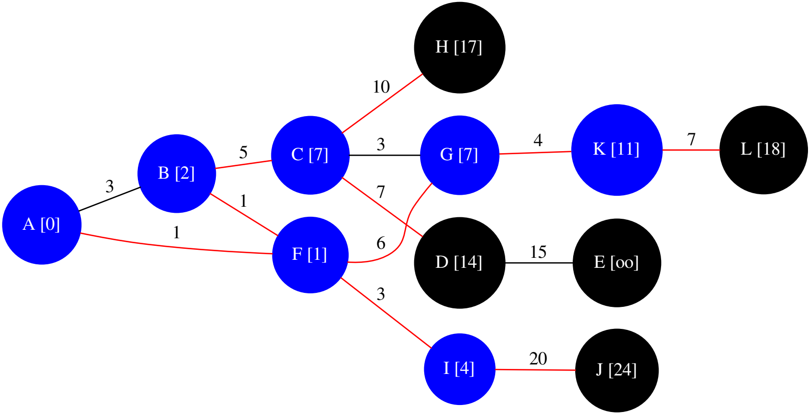


O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.

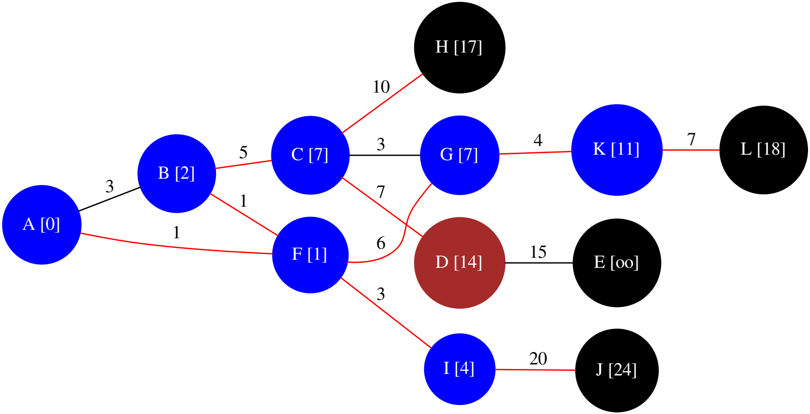


Exemplo: encontrar um caminho de custo mínimo entre A e E.



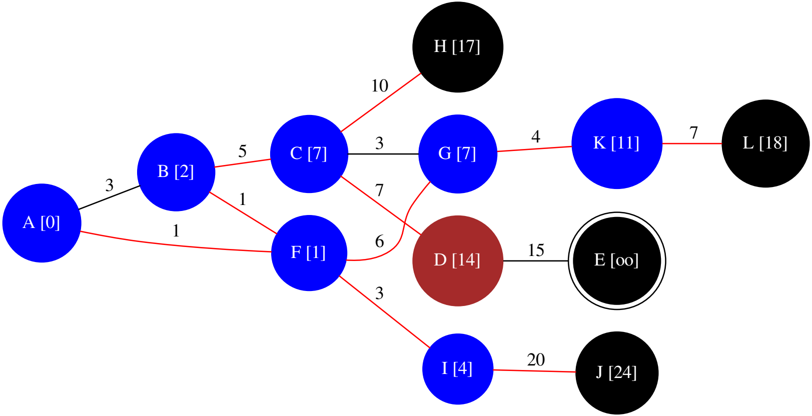
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



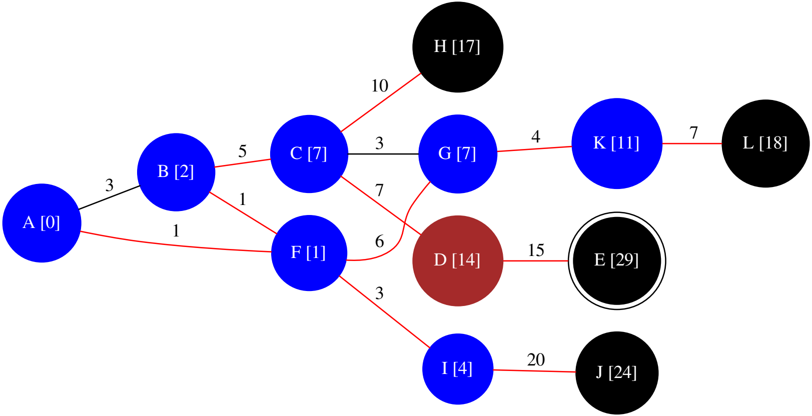
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



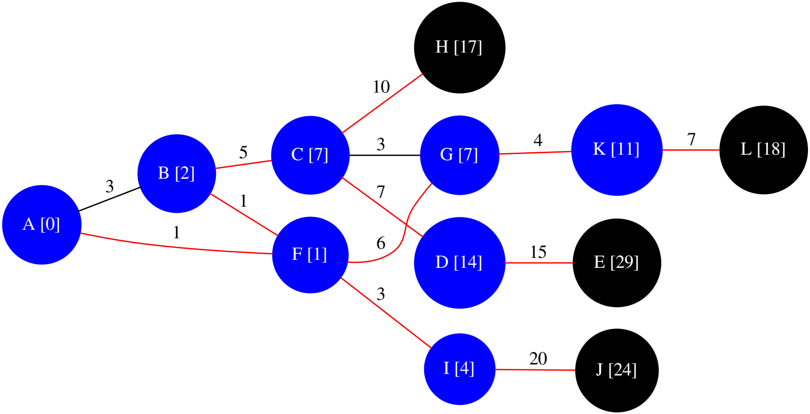
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



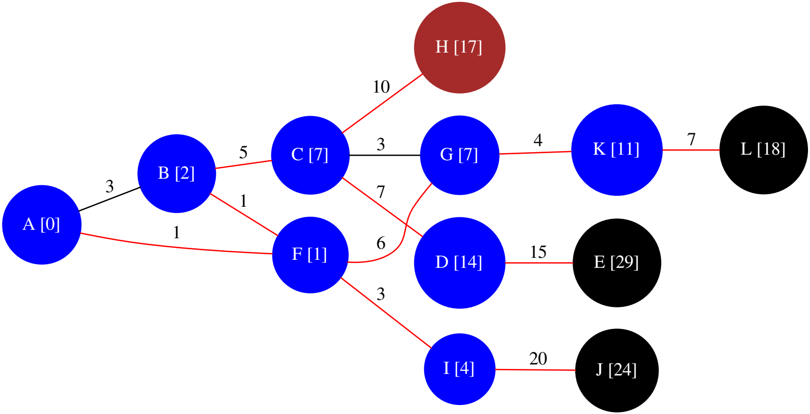
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



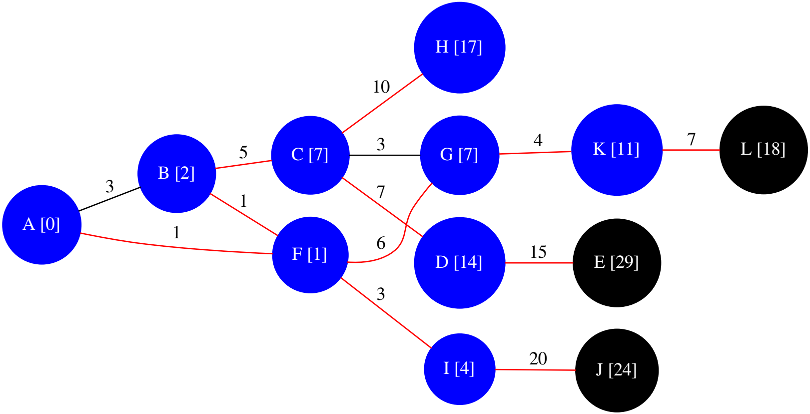
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



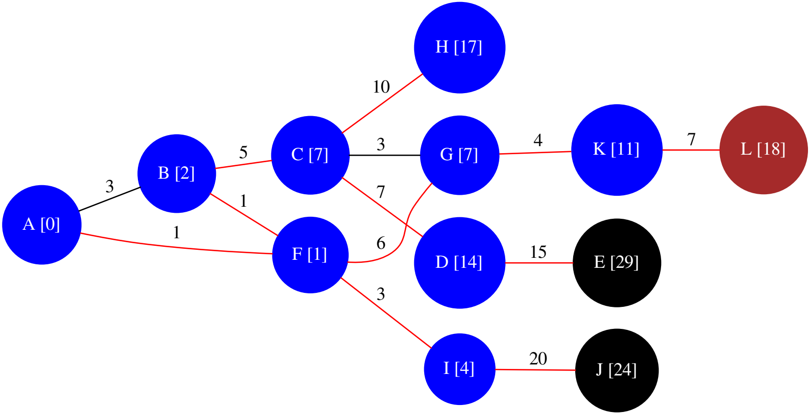
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



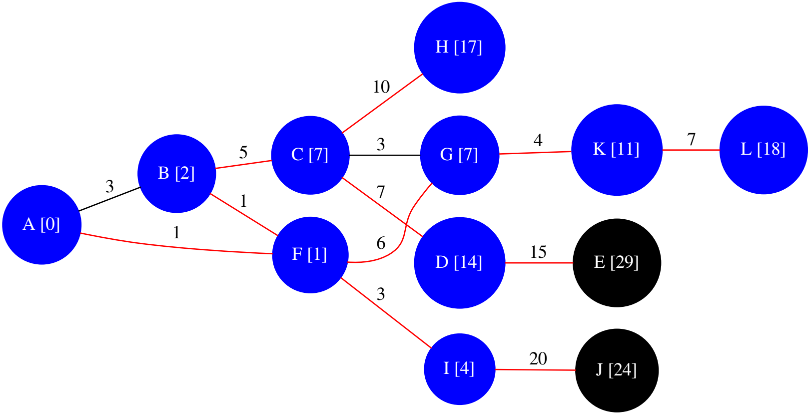
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



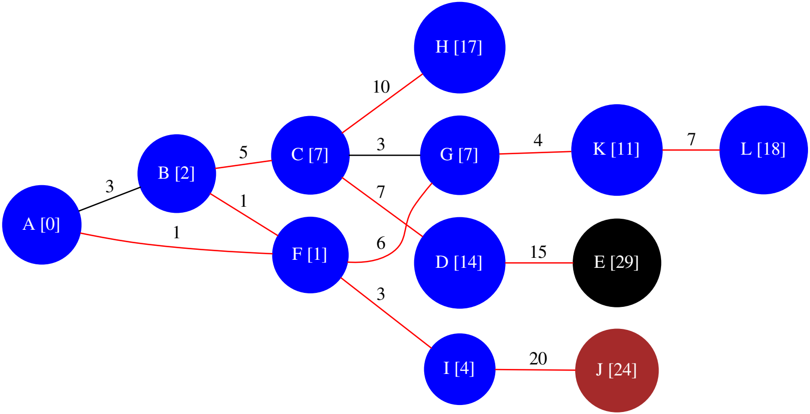
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



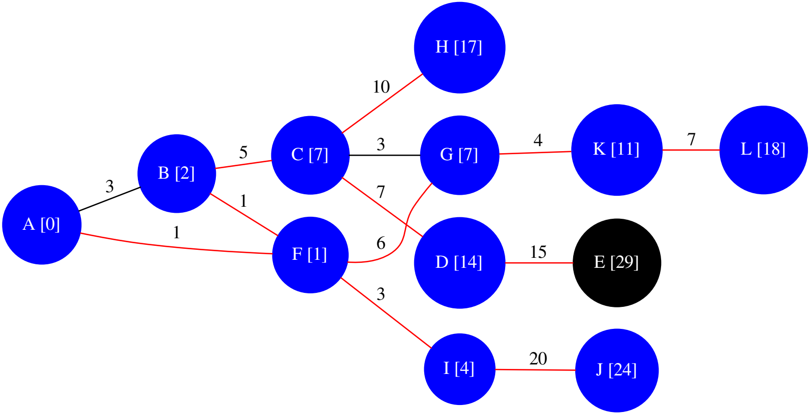
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



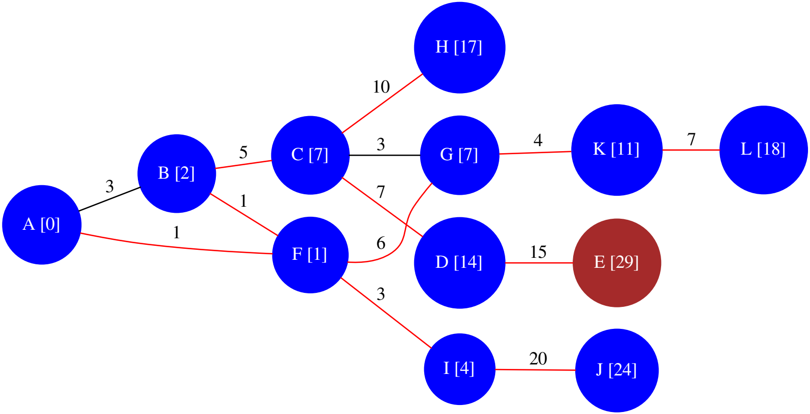
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



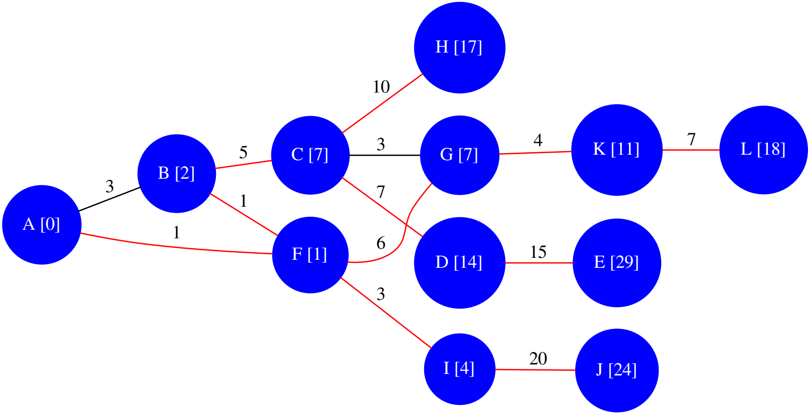
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



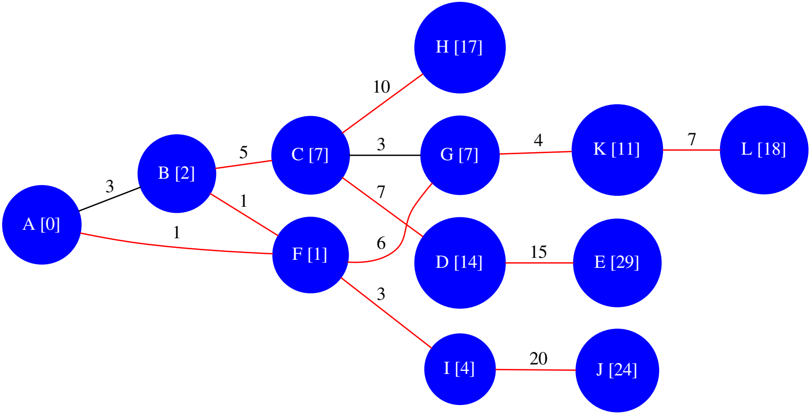
O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.



O Algoritmo de Dijkstra

Exemplo: encontrar um caminho de custo mínimo entre A e E.
O custo do caminho é 29. Para reconstruir o caminho, basta começar de E e seguir as arestas vermelhas.



O Algoritmo de Dijkstra

Implementação

O detalhe de implementação mais relevante, inclusive para a análise da complexidade do algoritmo, é a forma pela qual descobrimos qual é o vértice não marcado com menor distância. Podemos, para isso, manter uma lista com os vértices não marcados e percorrê-la para pegar o de menor valor de distância:

O Algoritmo de Dijkstra

Implementação - Lista

Defina distancia[v] $= \infty$, marcado[v] $= 0$, predecessor[v] $=$ NULL $\forall v \in V$.
Defina distancia[v_1] $= 0$.
Defina listaNaoMarcados = V
Enquanto **listaNaoMarcados contiver elementos** faça:
  Defina v_{at} **= listaNaoMarcados[0]**
  para todo $v \in listaNaoMarcados$ **faça:**
    se distancia[v] $<$ **distancia[**v_{at}**]:**
      v_{at} **= v**
  Remova v_{at} **de listaNaoMarcados**
  Defina marcado[v_{at}] $= 1$.
  Para toda aresta $e = (v_{at}, v_{prox})$ faça:
    Se marcado[v_{prox}] $= 0$ E distancia[v_{prox}] $>$ distancia[v_{at}] $+ W(e)$:
      Defina distancia[v_{prox}] $=$ distancia[v_{at}] $+ W(e)$.
      Defina predecessor[v_{prox}] $= v_{at}$.
retorne distancia[v_2]

O Algoritmo de Dijkstra

Implementação

A implementação anterior nos dá um algoritmo quadrático em $|V|$. Será que é possível fazer melhor?

O Algoritmo de Dijkstra

Implementação

A implementação anterior nos dá um algoritmo quadrático em $|V|$. Será que é possível fazer melhor?
Sim. Podemos utilizar um heap mínimo ao invés de uma lista. A ideia é semelhante à da fila utilizada na Busca em Largura:

O Algoritmo de Dijkstra

Implementação - Heap

Defina distancia[v] $= \infty$, marcado[v] $= 0$, predecessor[v] $=$ NULL $\forall v \in V$.
Defina distancia[v_1] $= 0$.
Defina heap = new Heap
heap.push(v_1, 0)
Enquanto **heap contiver elementos** faça:
    Defina (v_{at}, minDist) = 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 distancia[v_{prox}] $>$ distancia[v_{at}] $+ W(e)$:
                Defina distancia[v_{prox}] $=$ distancia[v_{at}] $+ W(e)$.
                Defina predecessor[v_{prox}] $= v_{at}$.
                heap.push(v_{prox}, distancia[v_{prox}])
retorne distancia[v_2]

O Algoritmo de Dijkstra

Implementação

Com a modificação anterior, conseguimos recuperar o vértice de menor distância em tempo logaritmo.

O Algoritmo de Dijkstra

Variações do Problema do Caminho Mínimo

Vimos como o algoritmo do Dijkstra se comporta em grafos não direcionados. E nos direcionados, o que muda?

O Algoritmo de Dijkstra

Variações do Problema do Caminho Mínimo

Vimos como o algoritmo do Dijkstra se comporta em grafos não direcionados. E nos direcionados, o que muda?
Nada! Funciona exatamente do mesmo jeito.

O Algoritmo de Dijkstra

Variações do Problema do Caminho Mínimo

Vimos o problema de encontrar o caminho mínimo com um único vértice de origem. Como fazer para considerar um **conjunto** de vértices de origem? (Imagine que cada vértice de origem é a casa de um familiar seu em uma cidade para a qual você está viajando. O vértice de destino é o aeroporto onde você desembarca. Qual dos seus familiares chega ao aeroporto mais rapidamente?)

O Algoritmo de Dijkstra

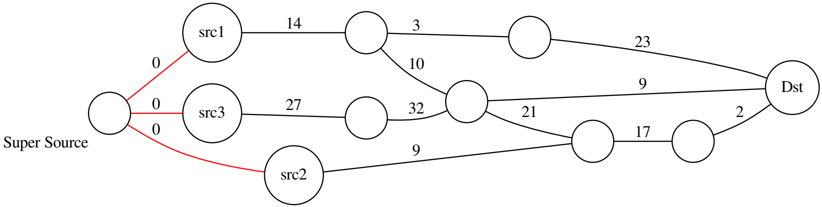
Variações do Problema do Caminho Mínimo

Vimos o problema de encontrar o caminho mínimo com um único vértice de origem. Como fazer para considerar um **conjunto** de vértices de origem? (Imagine que cada vértice de origem é a casa de um familiar seu em uma cidade para a qual você está viajando. O vértice de destino é o aeroporto onde você desembarca. Qual dos seus familiares chega ao aeroporto mais rapidamente?)

Super Source

Uma possível solução é adicionar ao grafo um nó especial chamado de Super Source. Adicione também uma aresta com peso 0 ligando o Super Source a cada um dos vértices de origem. Execute o Dijkstra com o Super Source sendo a origem.

O Algoritmo de Dijkstra



O Algoritmo de Dijkstra

Variações do Problema do Caminho Mínimo

Vimos que é fácil guardar e reconstruir **um** caminho mínimo entre um par de vértices.

Como fazer se quisermos guardar e reconstruir **todos** os caminhos mínimos?.

Variações do Problema do Caminho Mínimo

Vimos que é fácil guardar **um** caminho mínimo entre um par de vértices. Como fazer se quisermos guardar **todos** os caminhos mínimos?.

Todos os caminhos mínimos

Ao invés de guardar **um** predecessor de cada vértice, guardarmos **uma lista** com **todos** eles.

O Algoritmo de Dijkstra

Dijkstra : Todos os Caminhos Mínimos

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