Teoria dos GRAFOS

Aula 08: Caminho Mínimo em Grafos

Prof. José Alberto S. Torres

Problema do **Caminho Mínimo**

- Na teoria dos grafos e na otimização combinatória, um dos problemas mais fundamentais e ubíquos é o do **caminho mínimo de origem única**, conhecido pela sigla SSSP (do inglês, Single-Source Shortest Path).
- O problema é formalmente definido da seguinte maneira:
  - dado um grafo ponderado G = (V, E), onde V é um conjunto de vértices e E é um conjunto de arestas, e um vértice de origem específico $s \in V$, o objetivo é encontrar o caminho de menor peso total de s para todos os outros vértices $v \in V$

IESB
Centro Universitário

Problema do **Caminho Mínimo**

- O "peso" de um caminho é calculado como **a soma dos pesos de todas as arestas que o compõem**.
- Esses pesos podem representar uma variedade de métricas do mundo real, como distância, tempo de viagem, custo monetário ou latência de rede.
- A solução para o problema SSSP não é apenas um único valor, mas **um conjunto de caminhos** que formam uma "árvore de caminhos mínimos" **enraizada na origem s**.

Aplicações Iniciais

- A relevância do problema do caminho mínimo torna-se evidente ao observar sua vasta gama de aplicações práticas, a exemplo de:
  - **Navegação e Logística:** Sistemas de navegação, como GPS e serviços de mapas online. O objetivo é encontrar a rota mais rápida ou mais curta entre dois pontos geográficos.
  - **Roteamento de Redes**: Em redes de computadores e telecomunicações, o problema é determinar o caminho mais eficiente para que os pacotes de dados viajem do roteador de origem ao de destino. O "custo" pode ser a latência, a largura de banda disponível ou o número de saltos (hops) entre roteadores.

IESB
Centro Universitário

Algoritmo de **Djkistra**

- O problema é justamente como **construir essas árvores de destino**.
- Para grafos em que **todos os pesos das arestas são não negativos**, o algoritmo concebido por Edsger W. **Dijkstra** em 1956 permanece a **mais eficiente** para o problema SSSP.
- A sua **abordagem é caracterizada como "gulosa"** (greedy), o que significa que, a cada passo, o algoritmo faz a escolha que parece ser a melhor naquele momento, sem se preocupar com as consequências futuras.

IESB
Centro Universitário

Numerische Mathematik 1, 269—271 (1959)

A Note on Two Problems in Connexion with Graphs

By

E. W. DIJKSTRA

We consider n points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict ourselves to the case where at least one path exists between any two nodes. We now consider two problems.

Problem 1. Construct the tree of minimum total length between the n nodes. (A tree is a graph with one and only one path between every two nodes.)

In the course of the construction that we present here, the branches are subdivided into three sets:

I. the branches definitely assigned to the tree under construction (they will form a subtree);

II. the branches from which the next branch to be added to set I, will be selected;

III. the remaining branches (rejected or not yet considered).

The nodes are subdivided into two sets:

A. the nodes connected by the branches of set I,

B. the remaining nodes (one and only one branch of set II will lead to each of these nodes).

Estratégia **Gulosa**

- A lógica central do algoritmo de Dijkstra é sua natureza gulosa: a cada passo, ele toma a decisão que parece ser a melhor naquele instante, sem reconsiderar as decisões anteriores.
- A estratégia se desenrola da seguinte forma, o algoritmo mantém dois conjuntos de vértices:
  - Um conjunto de vértices "visitados" (ou finalizados), para os quais o caminho mais curto a partir da origem já foi determinado e não será alterado.
  - Um conjunto de vértices "não visitados", para os quais a distância conhecida é apenas uma estimativa que pode ser melhorada.

IESB
Centro Universitário

Estratégia **Gulosa**

- O processo iterativo consiste em selecionar o vértice do conjunto de não visitados com a menor distância estimada da origem. Este vértice é então movido para o conjunto de visitados.
- A partir deste novo vértice "finalizado" ou “fechado”, o algoritmo examina seus vizinhos e atualiza suas distâncias estimadas, caso um caminho mais curto seja encontrado por meio dele.
- Este ciclo de selecionar o mais próximo, finalizar e atualizar os vizinhos continua até que todos os vértices alcançáveis tenham sido visitados.

IESB
Centro Universitário

Algoritmo de **Djkistra**

- Seja G(V,A) um grafo orientado e **s** um vértice de G:
  - Atribua valor zero à estimativa do custo mínimo do vértice **s** (a raiz da busca) e infinito às demais estimativas;
  - Atribua um valor qualquer aos precedentes (o precedente de um vértice **t** é o vértice que precede **t** no caminho de custo mínimo de **s** para **t**);
  - Enquanto houver vértice aberto, relaxar o vértice:
    - seja **k** um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos;
    - feche o vértice **k**
    - Para todo vértice **j** ainda aberto que seja sucessor de **k** faça:
      - some a estimativa do vértice **k** com o custo da aresta que une **k** a **j**;
      - caso esta soma seja melhor que a estimativa anterior para o vértice **j**, substitua-a e anote **k** como precedente de **j**.

IESB
Centro Universitário

Algoritmo de **Djkistra**

O funcionamento do algoritmo pode ser dividido em três fases principais:

- **Inicialização:**
  - Cria-se uma estrutura de dados (por exemplo, um array) para armazenar a menor distância conhecida da origempara cada vértice. A distância para a própria origem é inicializada com 0, enquanto a distância para todos os outros vértices é inicializada com infinito. O infinito simboliza que ainda não descobrimos um caminho até esses vértices.
  - Cria-se uma estrutura para armazenar o predecessor de cada vértice no caminho mais curto. Isso permite reconstruir o caminho ao final do processo. Todas as entradas são inicializadas como nulas.
  - Todos os vértices são adicionados ao conjunto de "não visitados".
- **Iteração e Seleção:**
  - Enquanto o conjunto de não visitados não estiver vazio, o algoritmo seleciona o vértice u deste conjunto que possui o menor valor em dist[u]. Esta é a escolha gulosa: o algoritmo se compromete com o caminho para o vértice mais próximo que ainda não foi finalizado.
  - O vértice selecionado u é removido do conjunto de não visitados e adicionado ao de visitados.

IESB
Centro Universitário

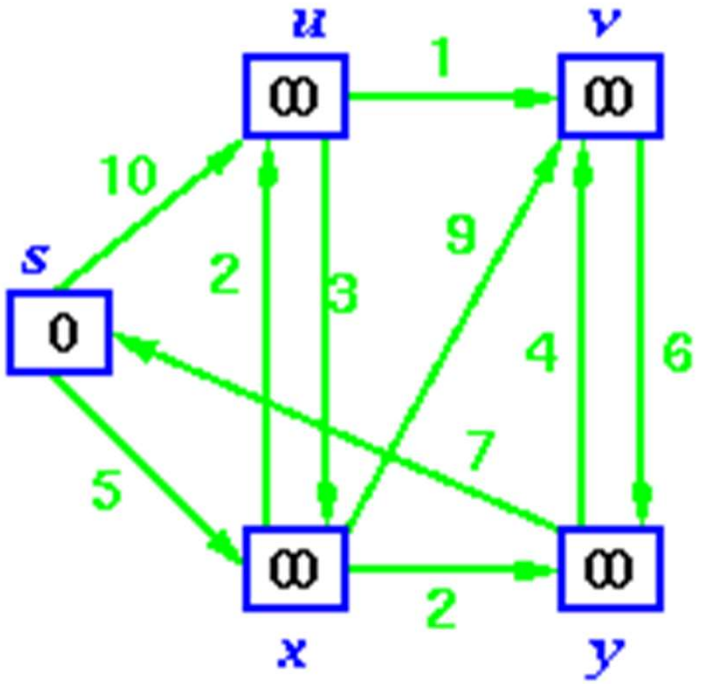
Algoritmo de **Djkistra**

- **O Processo de Relaxamento:**
  - Para o vértice u recém-selecionado, o algoritmo examina cada um de seus vizinhos v que ainda estão no conjunto de não visitados.
  - Ele calcula uma distância alternativa para v passando por u: distancia_alternativa = dist[u] + peso(u, v).
  - Se esta distancia_alternativa for menor que a distância atualmente registrada para v (dist[v]), significa que um caminho mais curto para v foi encontrado.
  - O algoritmo então "relaxa" a aresta, atualizando dist[v] para o novo valor menor e definindo o predecessor de v como u.

Algoritmo de Djkistra

- Inicialmente todos os vértices tem um custo infinito, exceto **s** (a raiz da busca) que tem valor 0:

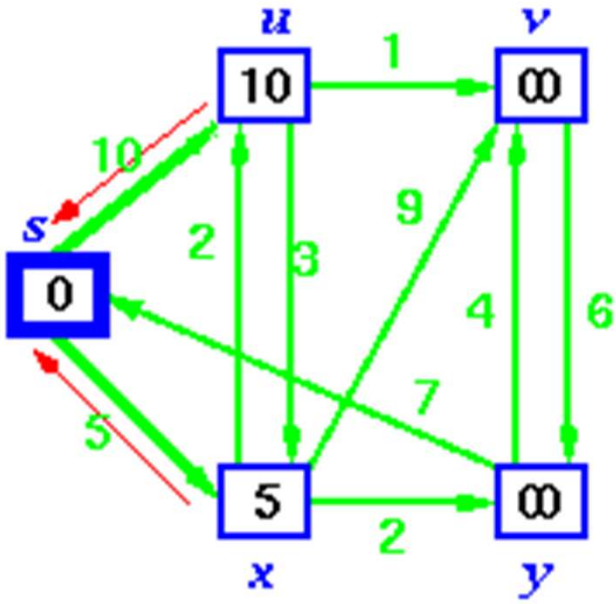
Vértices	S	U	V	X	Y
estimativa	0	∞	∞	∞	∞
precedentes	-	-	-	-	-



Algoritmo de Djkistra

- selecione **s** e recalcule as estimativas de u e x

Vértices	S	U	V	X	Y
estimativa	0	10	∞	5	∞
precedentes	-	S	-	S	-

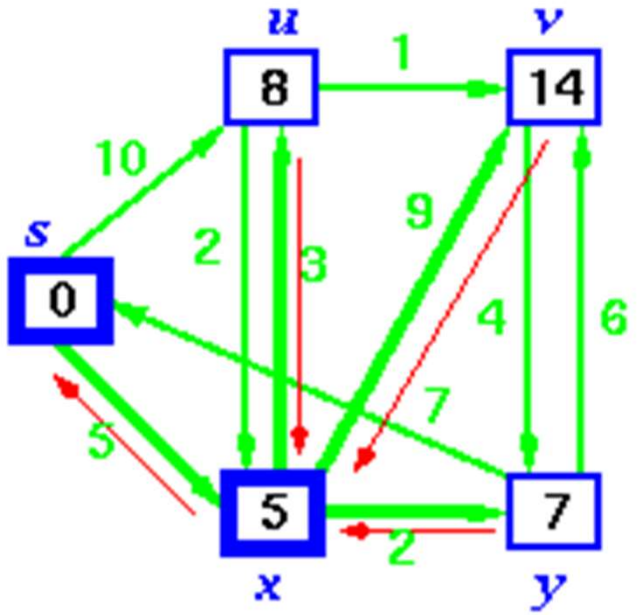


Algoritmo de Djkistra

- selecione **x** e recalcule as estimativas de u, v e y

Vértices	S	U	V	X	Y
estimativa	0	10	∞	5	∞
precedentes	-	S	-	S	-

Vértices	S	U	V	X	Y
estimativa	0	8	14	5	7
precedentes	-	X	X	S	x

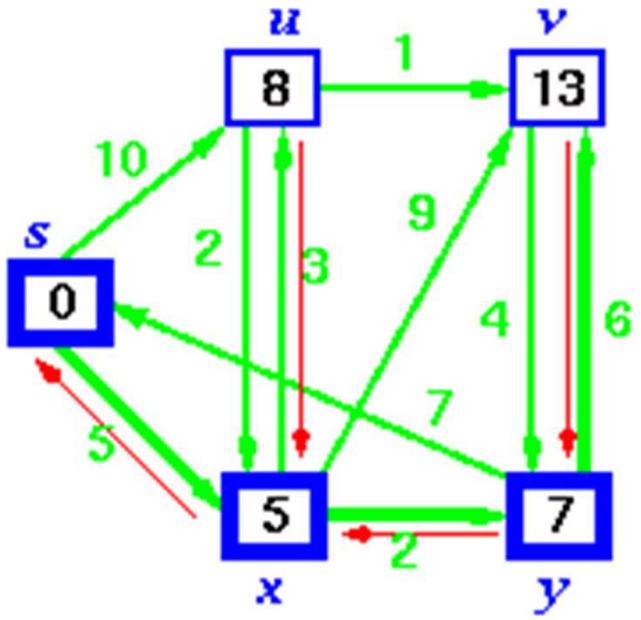


Algoritmo de Djkistra

- selecione y e recalcule a estimativa de v.

Vértices	S	U	V	X	Y
estimativa	0	8	14	5	7
precedentes	-	X	X	S	x

Vértices	S	U	V	X	Y
estimativa	0	8	13	5	7
precedentes	-	X	Y	S	x

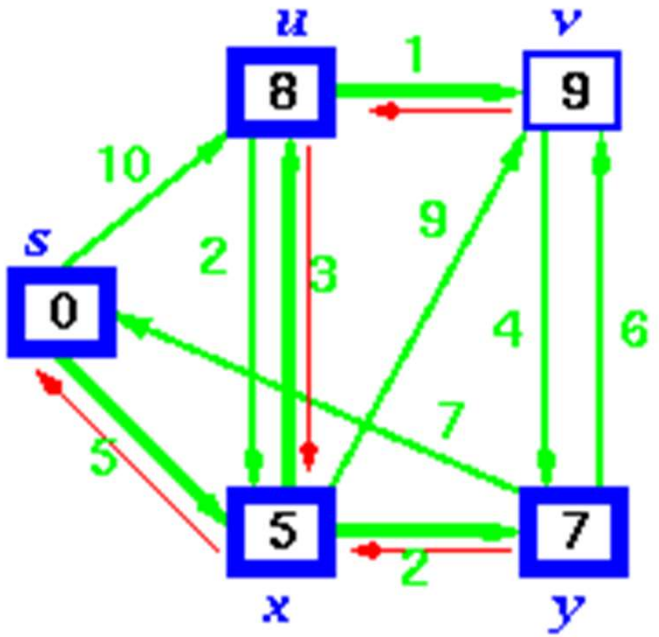


Algoritmo de Djkistra

- selecione **u** e recalcule a estimativa de v.

Vértices	S	U	V	X	Y
estimativa	0	8	13	5	7
precedentes	-	X	Y	S	x

Vértices	S	U	V	X	Y
estimativa	0	8	9	5	7
precedentes	-	X	U	S	x

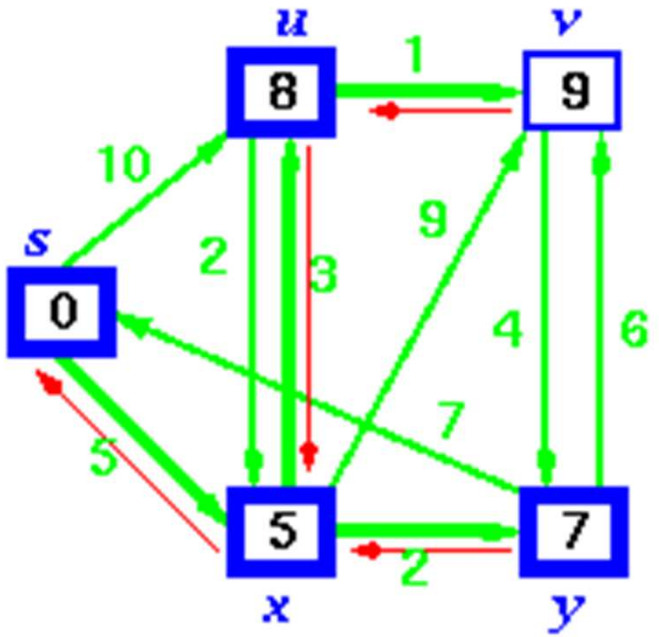


Algoritmo de Djkistra

- selecione v.

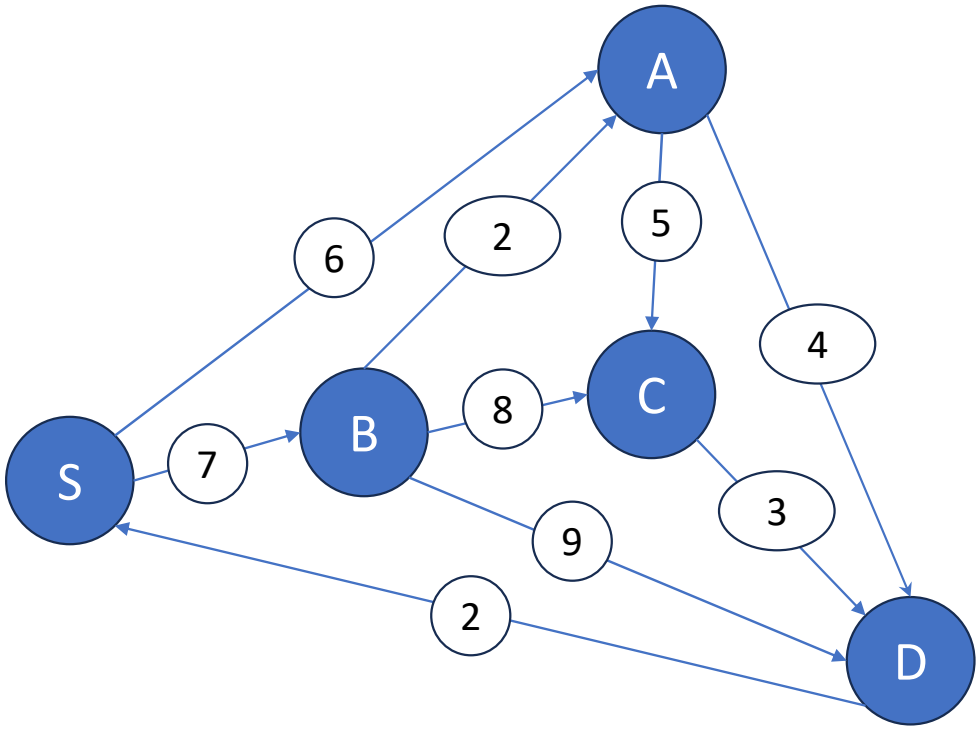
Vértices	S	U	V	X	Y
estimativa	0	8	9	5	7
precedentes	-	X	U	S	x

Vértices	S	U	V	X	Y
estimativa	0	8	9	5	7
precedentes	-	X	U	S	x



Atividade 1 – Aplique Dijkstra

Vértices	S	A	B	C	D
estimativa	0	∞	∞	∞	∞
precedentes	-	-	-	-	-



Arestas de **Peso Negativo**

- A principal e mais importante limitação do algoritmo de Dijkstra é sua incapacidade de funcionar corretamente em grafos com arestas de peso negativo.
- A razão para essa falha reside na sua suposição gulosa fundamental: uma vez que um vértice u é selecionado e marcado como "visitado", o algoritmo assume que a distância dist[u] é a menor possível e nunca mais a reavalia.
- Uma aresta de peso negativo pode invalidar essa suposição pois é possível que, após um vértice u ser finalizado, o algoritmo descubra um caminho para um vizinho v que, por sua vez, se conecta a u ou a outro vértice já visitado através de uma aresta de peso negativo.
- Isso poderia criar um caminho para u que é, na verdade, mais curto do que aquele que foi considerado final.

IESB
Centro Universitário

O Algoritmo de **Bellman-Ford**

- Quando a presença de arestas de peso negativo é uma possibilidade, o algoritmo de Bellman-Ford surge como a alternativa adequada.
- Em vez de uma abordagem gulosa, o algoritmo Bellman-Ford utiliza programação dinâmica, que reavalia continuamente as estimativas de distância, garantindo a correção mesmo na presença de pesos negativos.

O Algoritmo de **Bellman-Ford**

- O conceito de relaxamento também é a operação fundamental do Algoritmo de Bellman-Ford.
- A singularidade de cada algoritmo não reside na operação em si, mas na estratégia que dita a ordem e a frequência com que as operações de relaxamento são aplicadas.
- Enquanto Dijkstra usa uma fila de prioridades para aplicar o relaxamento de forma seletiva e gulosa, Bellman-Ford adota uma abordagem sistemática e repetitiva, **relaxando todas as arestas do grafo múltiplas vezes.**

IESB
Centro Universitário

Ciclos negativos

- Uma vantagem crucial do Bellman-Ford é sua capacidade de detectar ciclos de peso negativo — um ciclo no grafo cujos pesos das arestas somam um valor negativo.
- Em tais ciclos, o conceito de "caminho mais curto" torna-se indefinido, pois é possível percorrer o ciclo infinitamente para diminuir o custo do caminho.
- Bellman-Ford pode identificar essa condição, algo que Dijkstra não consegue fazer.

Algoritmo de **Bellman-Ford**

- O Algoritmo de Bellman-Ford pode ser compreendido como um processo estruturado em três fases distintas: inicialização, relaxamento iterativo e, finalmente, a detecção de ciclos de peso negativo.

Algoritmo de **Bellman-Ford**

- Inicialização:
  - o algoritmo começa estabelecendo um estado inicial bem definido. O vetor de distâncias é preenchido com infinito para todos os vértices, exceto para o vértice de origem s, para o qual distance[s] é definido como 0.
  - O vetor de predecessores predecessor é inicializado com valores nulos. Esta etapa prepara o terreno para o processo iterativo de descoberta de caminhos.
- Relaxamento Iterativo (|V|-1 Vezes):
  - Ela consiste em um laço principal que se repete exatamente |V|-1 vezes.
  - Dentro de cada uma dessas iterações, o algoritmo percorre *todas* as |E| arestas do grafo, aplicando a operação de relaxamento a cada uma delas.

IESB
Centro Universitário

Algoritmo de **Bellman-Ford**

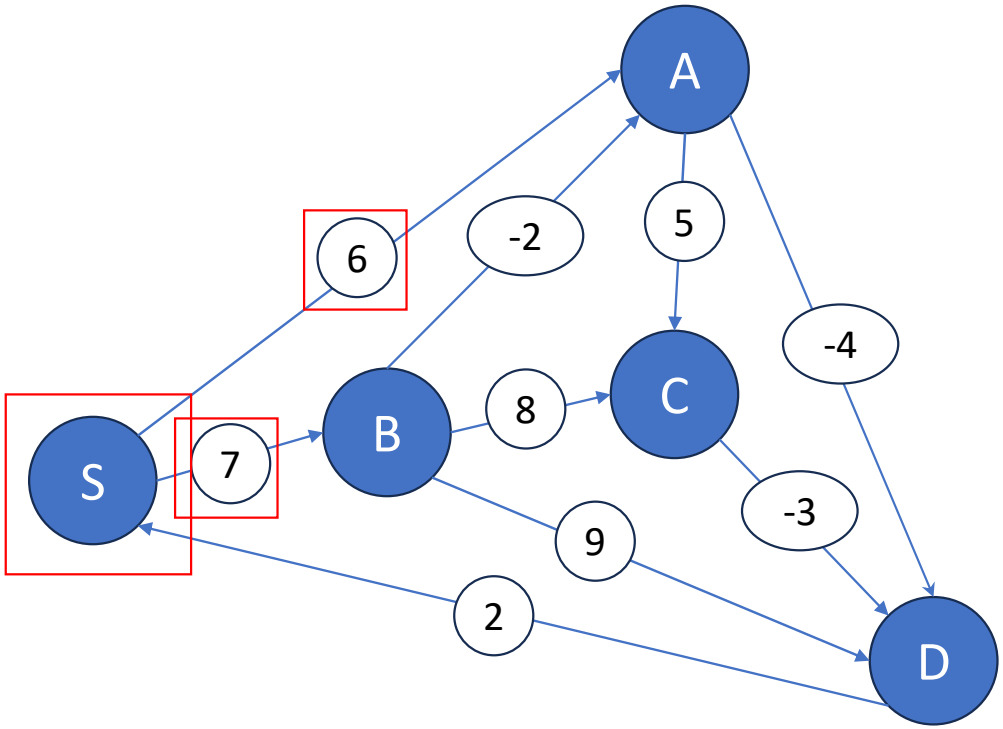
- Detecção de Ciclos de Peso Negativo:
  - Após a conclusão das |V|-1 iterações de relaxamento, o algoritmo realiza uma iteração *adicional* (a |V|-ésima) sobre todas as arestas, tentando relaxá-las novamente.
  - A condição de detecção é a seguinte: se, nesta fase final de verificação, for possível relaxar qualquer aresta — ou seja, se a condição d[u] + w(u, v) < d[v] for satisfeita para alguma aresta (u, v) —, isso serve como prova inequívoca da existência de um ciclo de peso negativo no grafo que é alcançável a partir da fonte.

IESB
Centro Universitário

Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	∞	∞	∞	∞
precedentes	-	-	-	-	-

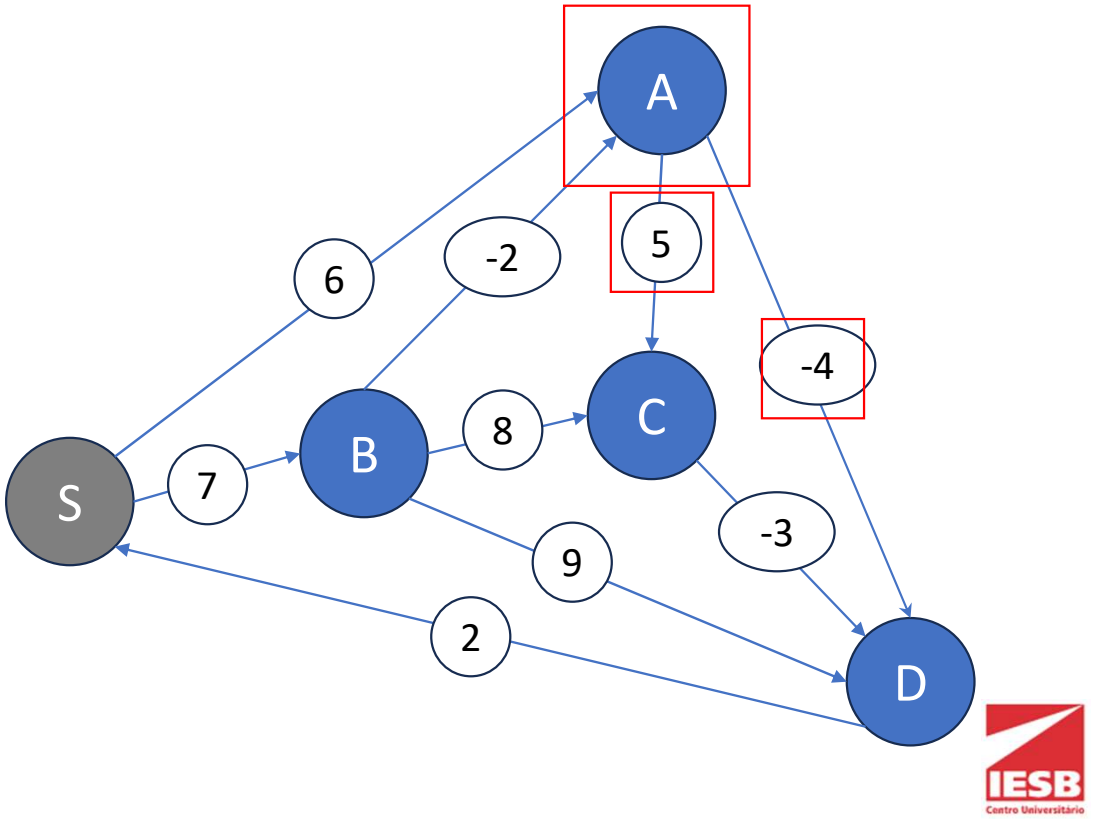
Vértices	S	A	B	C	D
estimativa	0	6	7	∞	∞
precedentes	-	S	S	-	-



Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	6	7	∞	∞
precedentes	-	S	S	-	-

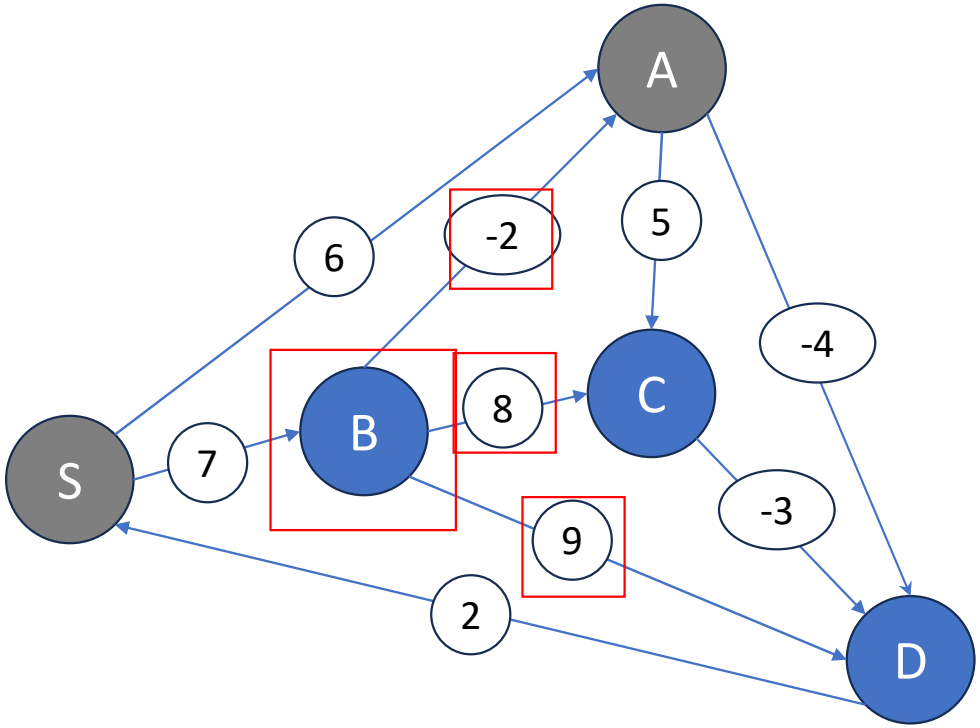
Vértices	S	A	B	C	D
estimativa	0	6	7	11	2
precedentes	-	S	S	A	A



Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	6	7	11	2
precedentes	-	S	S	A	A

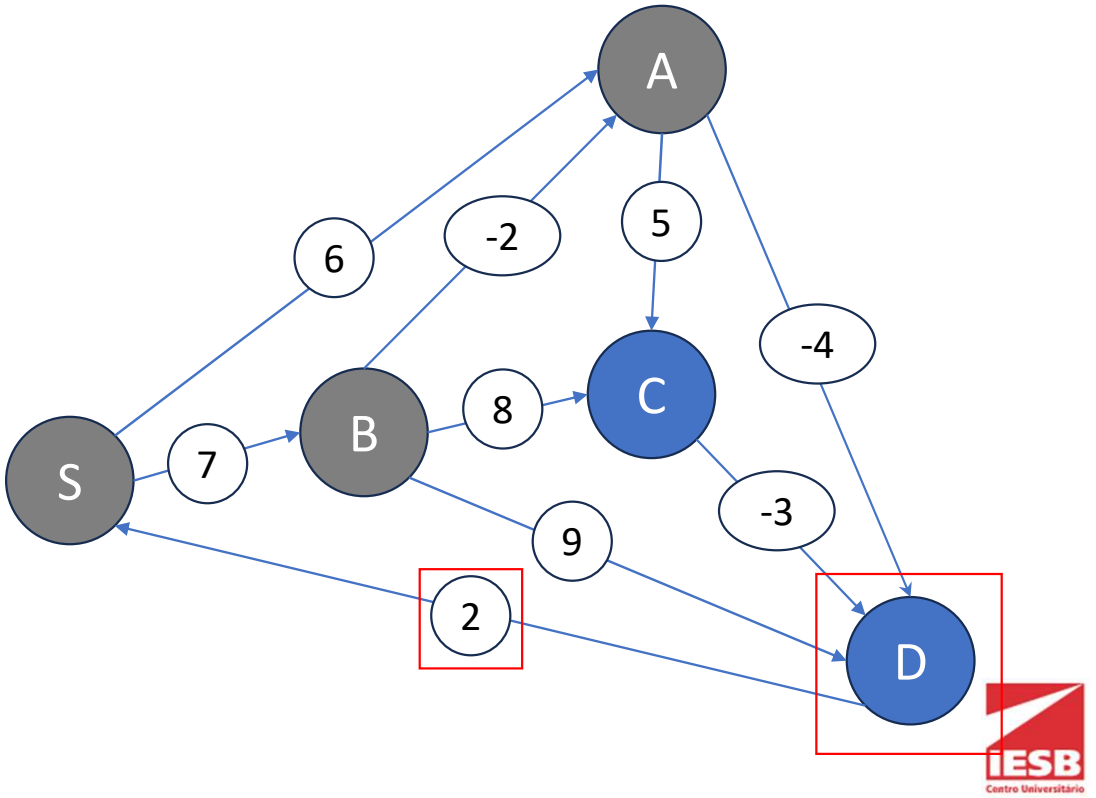
Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A



Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A

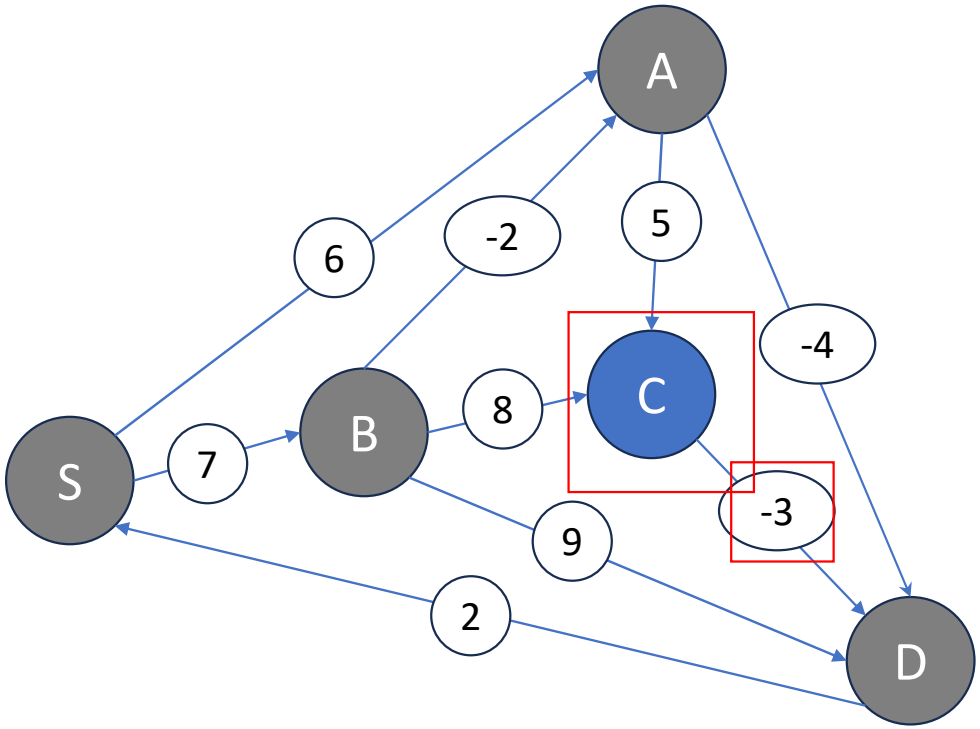
Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A



Bellman-Ford

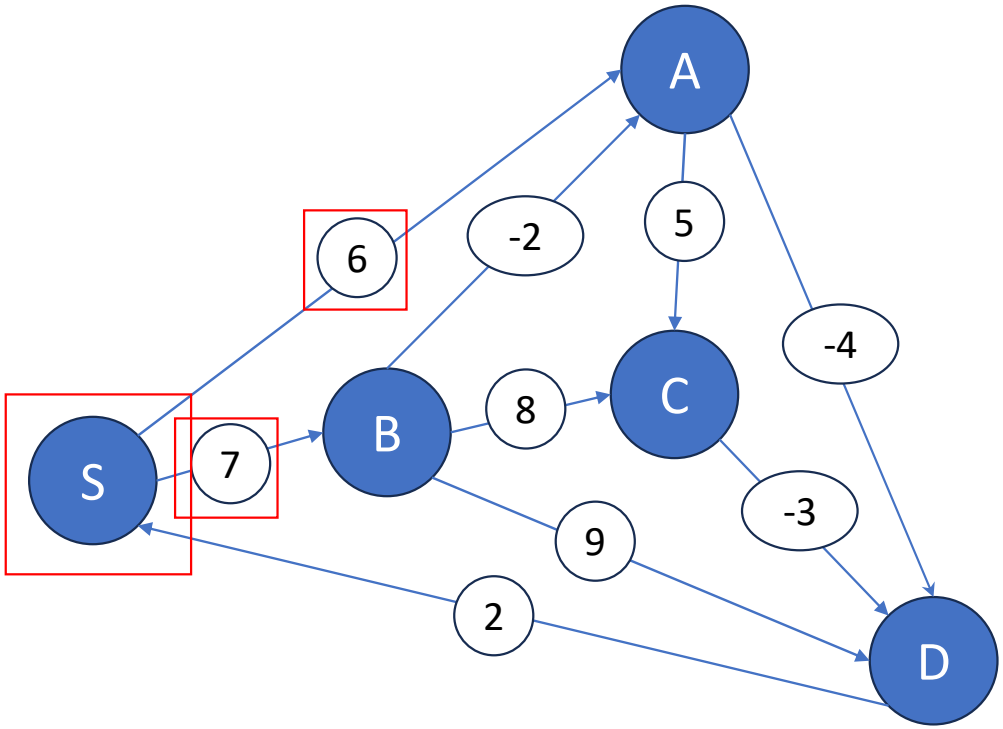
Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A

Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A



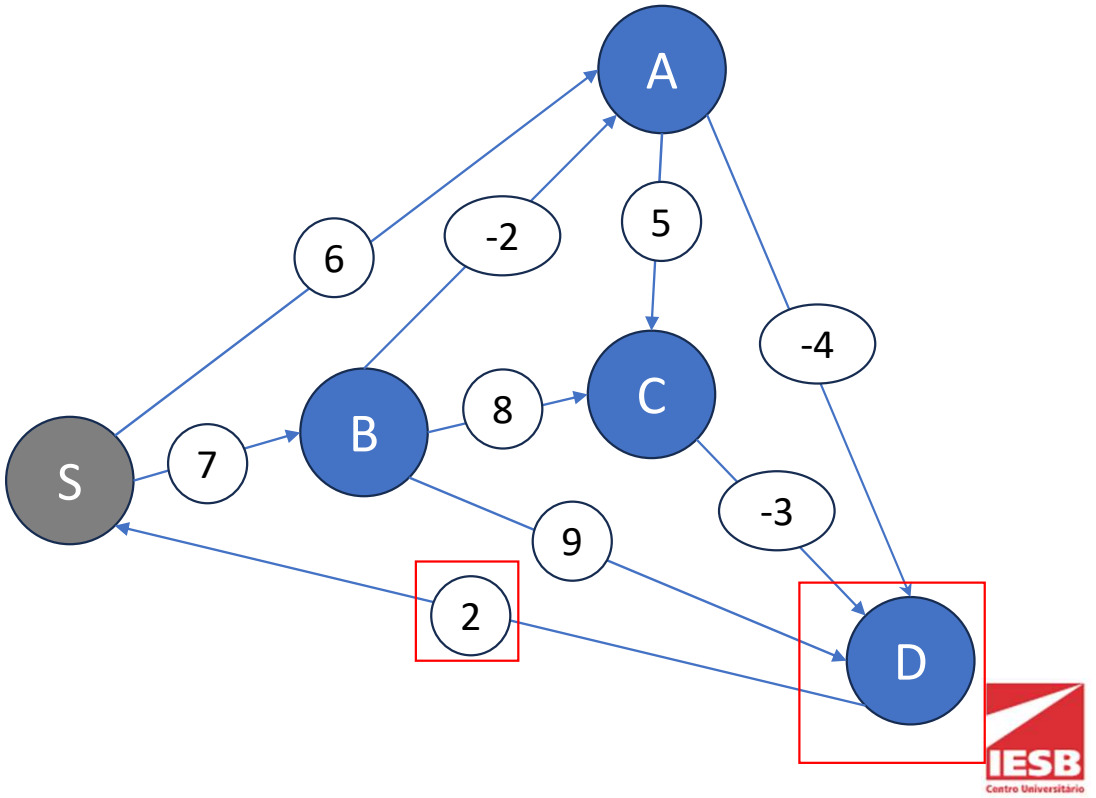
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A



Bellman-Ford

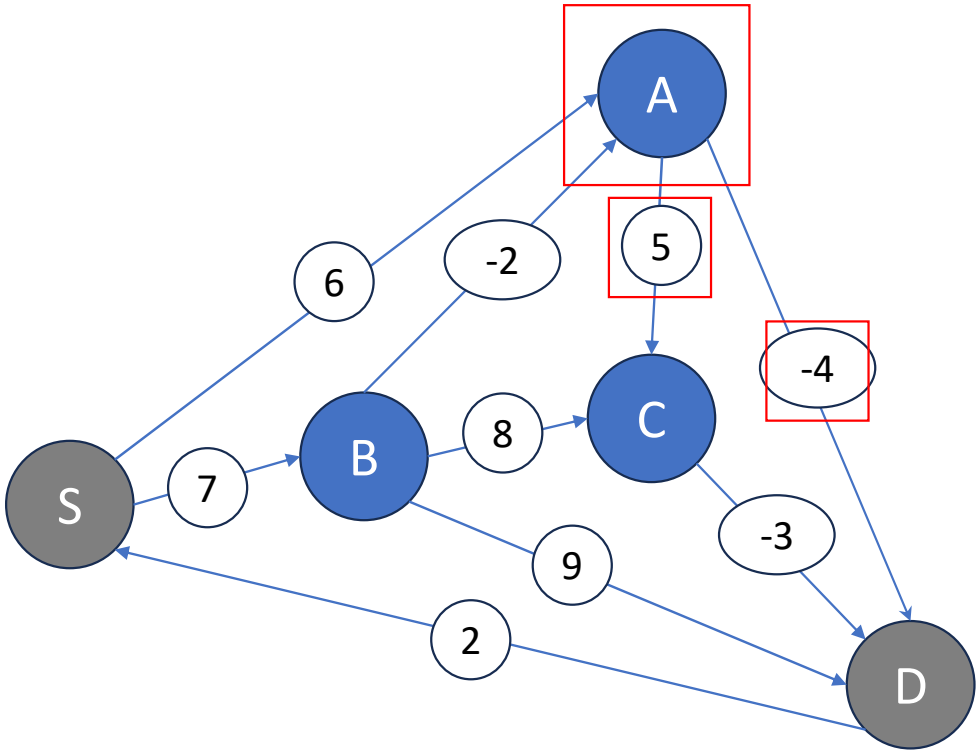
Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A



Bellman-Ford

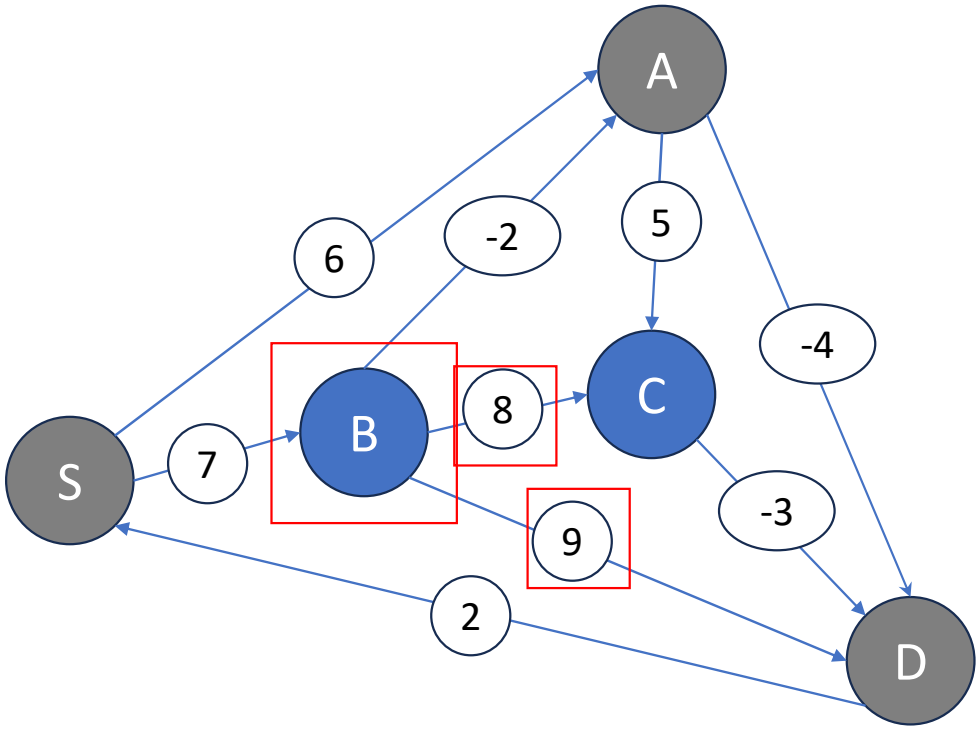
Vértices	S	A	B	C	D
estimativa	0	5	7	11	2
precedentes	-	B	S	A	A

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



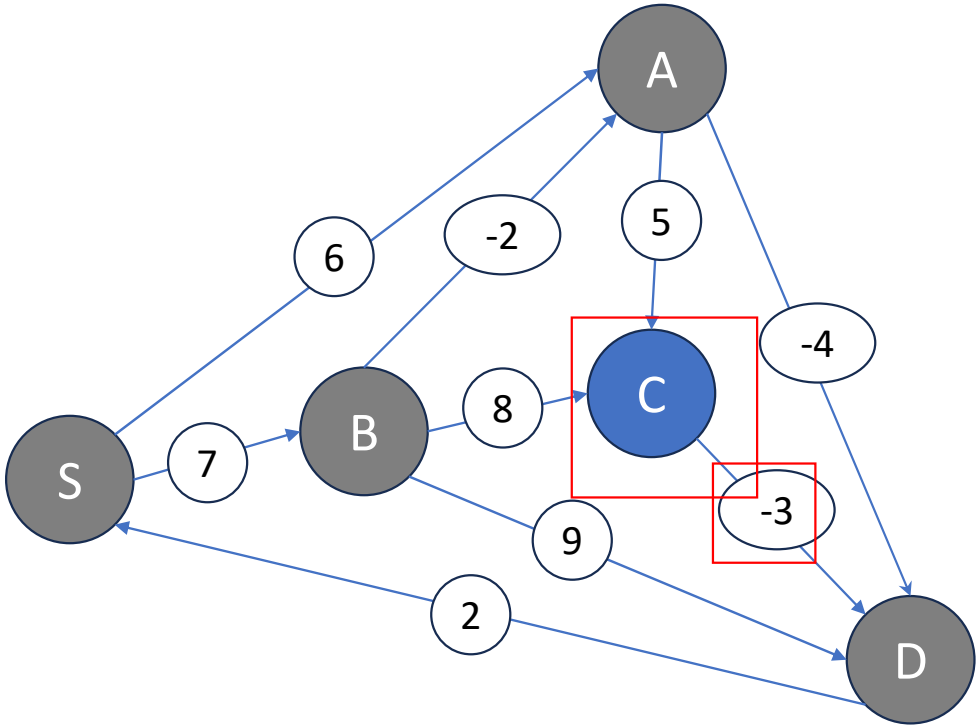
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



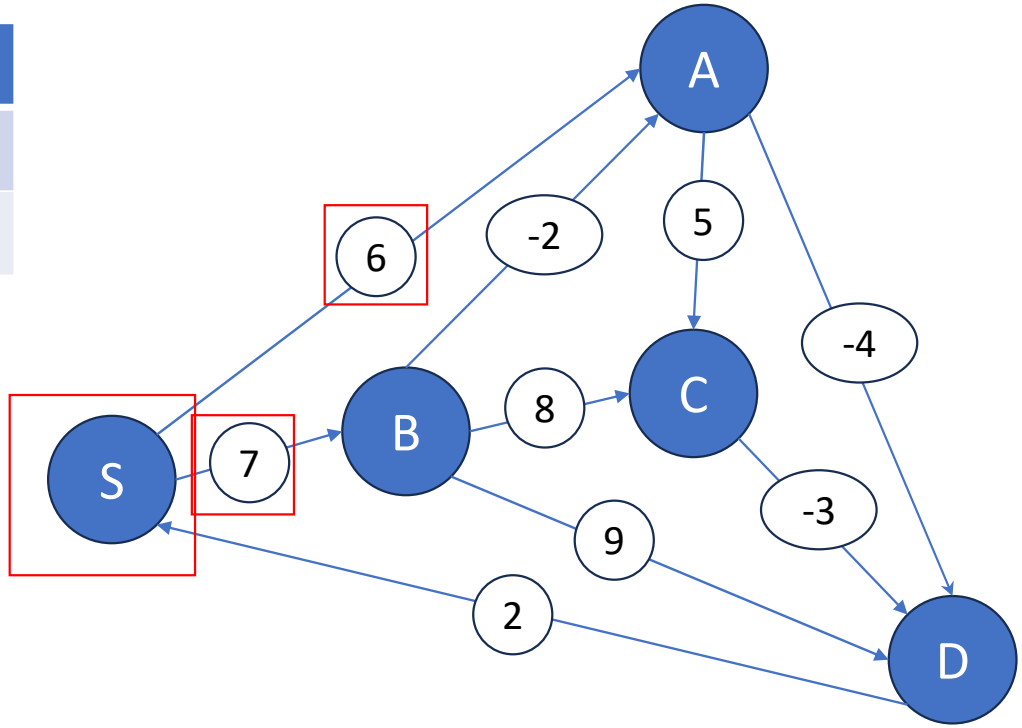
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



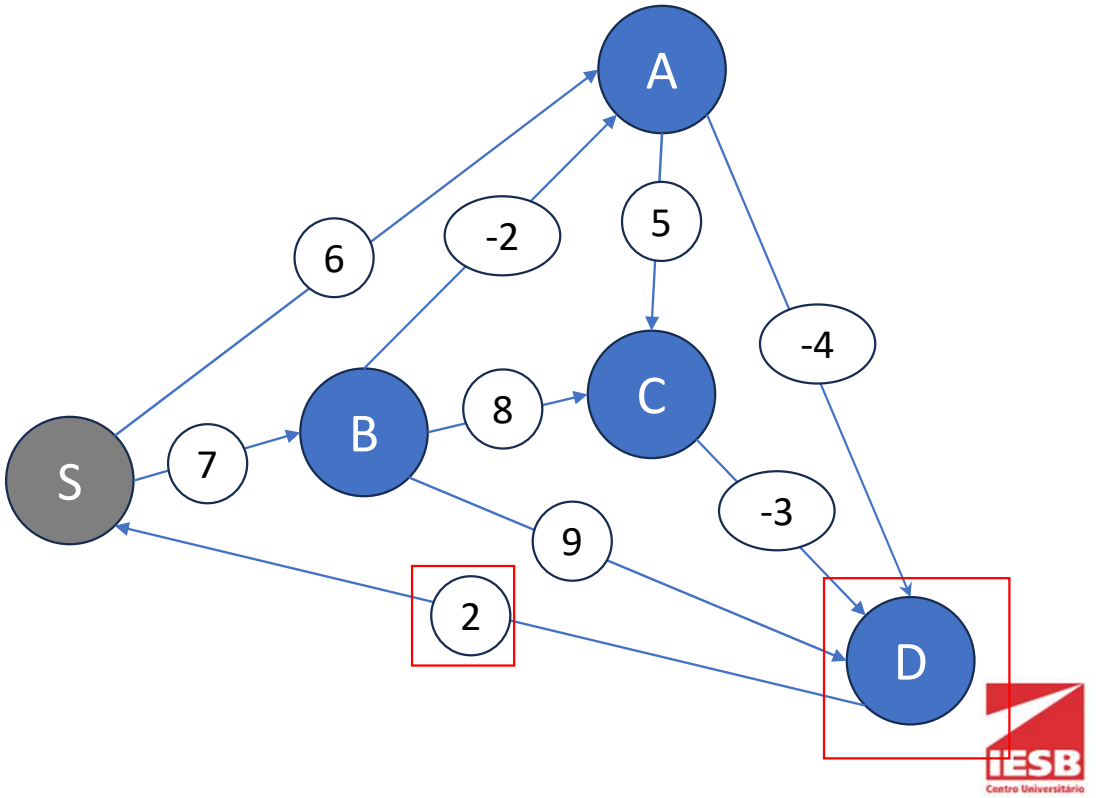
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



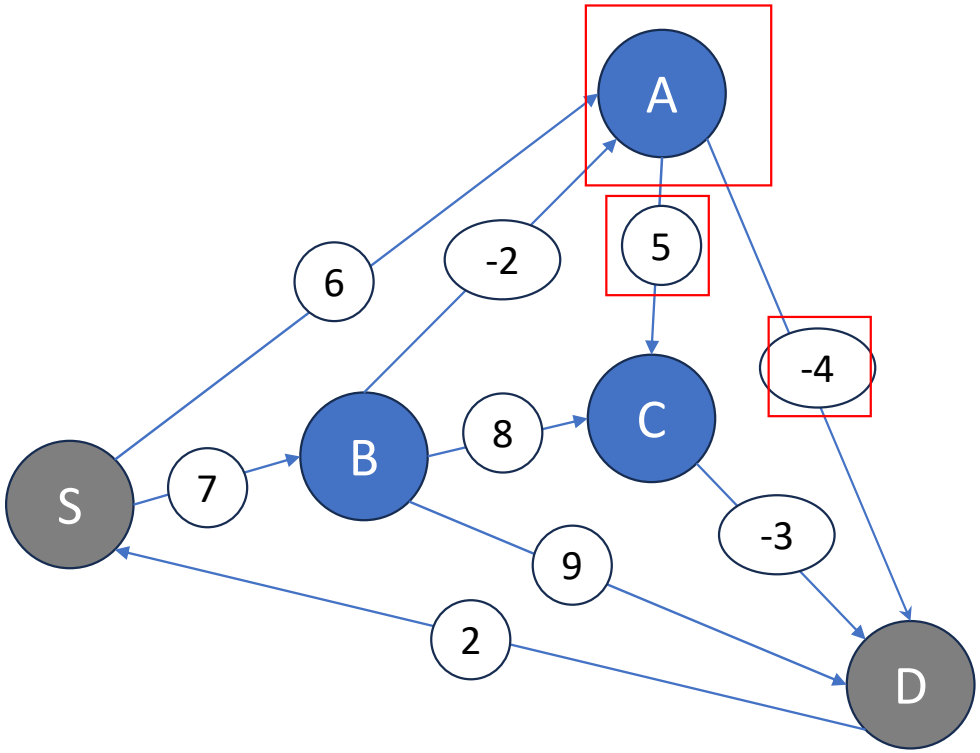
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



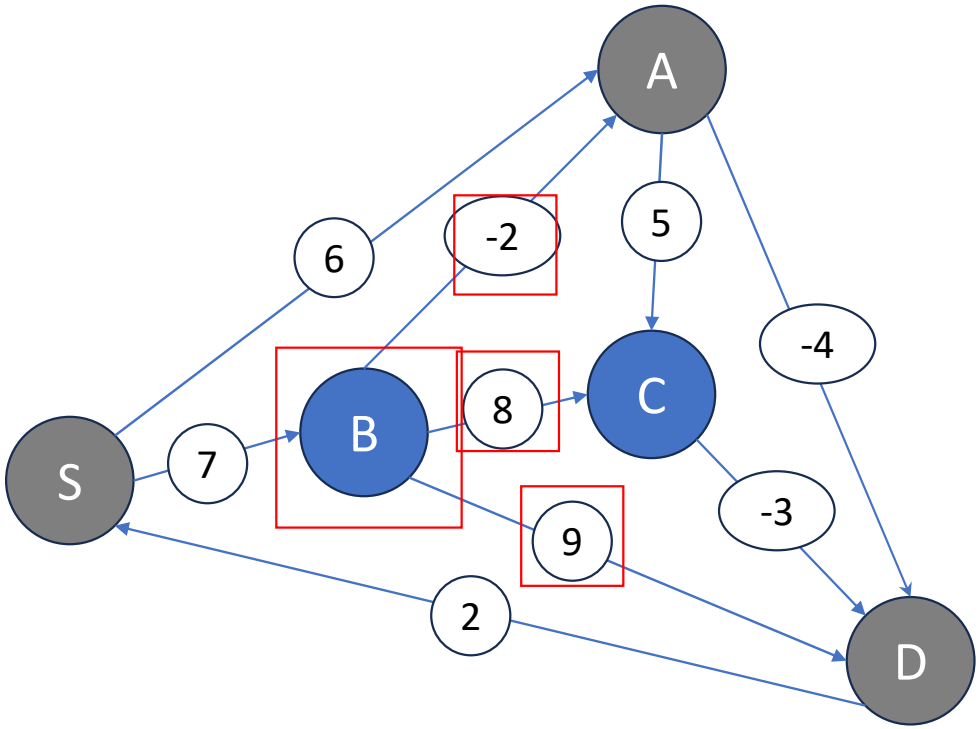
Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



Bellman-Ford

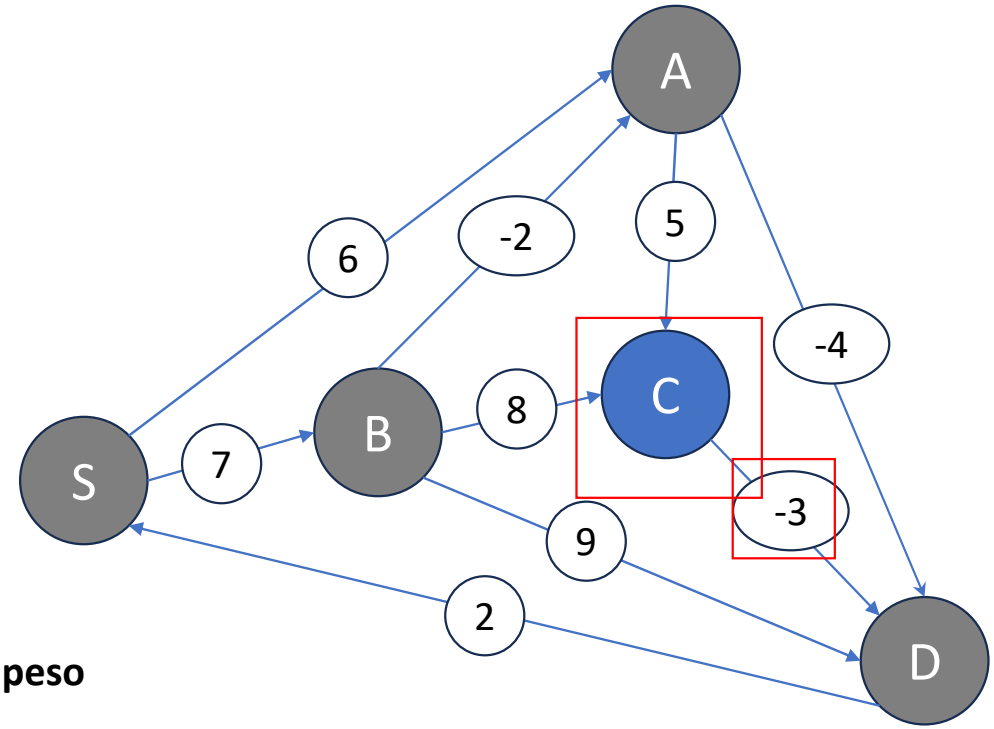
Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A



Bellman-Ford

Vértices	S	A	B	C	D
estimativa	0	5	7	10	1
precedentes	-	B	S	A	A

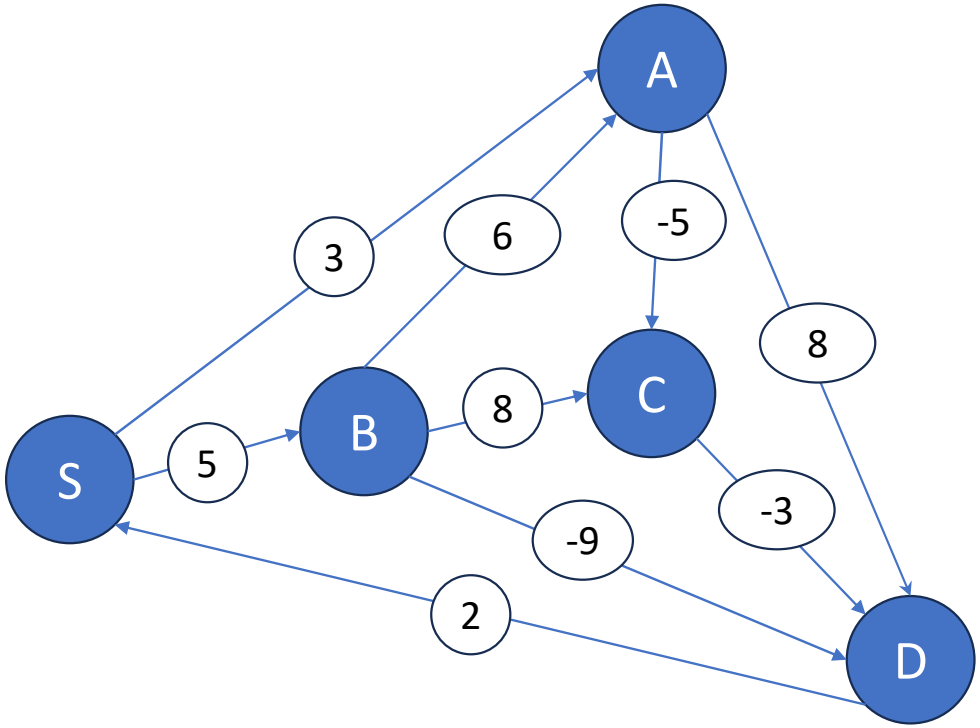
Aproximei o FIM ☺



Nenhuma distância foi reduzida, portanto **não há ciclo de peso negativo** acessível a partir de S.

Atividade 2 - Aplique Bellman-Ford

Vértices	S	A	B	C	D
estimativa					
precedentes					



Desempenho

- A diferença de desempenho deve-se à complexidade computacional de cada algoritmo, que é medida em relação ao número de Vértices (V) e Arestas (E):
  - Algoritmo de Bellman-Ford: Sua complexidade é sempre O(V×E).
  - Algoritmo de Dijkstra: Sua complexidade, com uma implementação comum (heap), é O(ElogV) ou $O(V^2)$ com uma implementação de array simples.

Desempenho

No Grafo Denso (b): O número de arestas (E) é proporcional a V^2 (pois tudo está conectado a tudo).

- **Bellman-Ford:** O(V×E) torna-se $O(V \times V^2) = O(V^3)$. (Crescimento cúbico - muito lento.)
- **Dijkstra:** O(ElogV) torna-se $O(V^2 \log V)$ (ou apenas $O(V^2)$ na implementação simples). $O(V^2)$ é *imensamente* mais rápido que $O(V^3)$, como o gráfico (b) demonstra claramente.

Desempenho

No Grafo Esparso (a): O número de arestas (E) é proporcional a V (cada vértice tem ~4 arestas).

- **Bellman-Ford:** $O(V \times E)$ torna-se $O(V \times V) = O(V^2)$.
- **Dijkstra:** $O(E \log V)$ torna-se $O(V \log V)$ (ou $O(V^2)$ na implementação simples).

Atividade - Algoritmos

- Crie os algoritmos de DIJKSTRA e Bellman-Ford.
- Não é preciso utilizar a estrutura de grafos construída anteriormente.


DÚVIDAS