SBVESDD: Estruturas de Dados

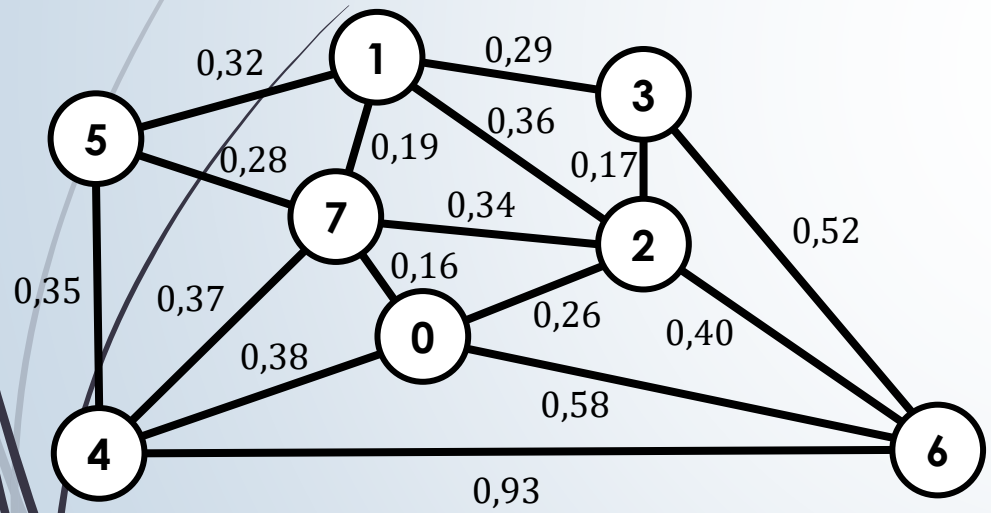
Aula 12: Estruturas de Dados Não-Lineares - Grafos Ponderados

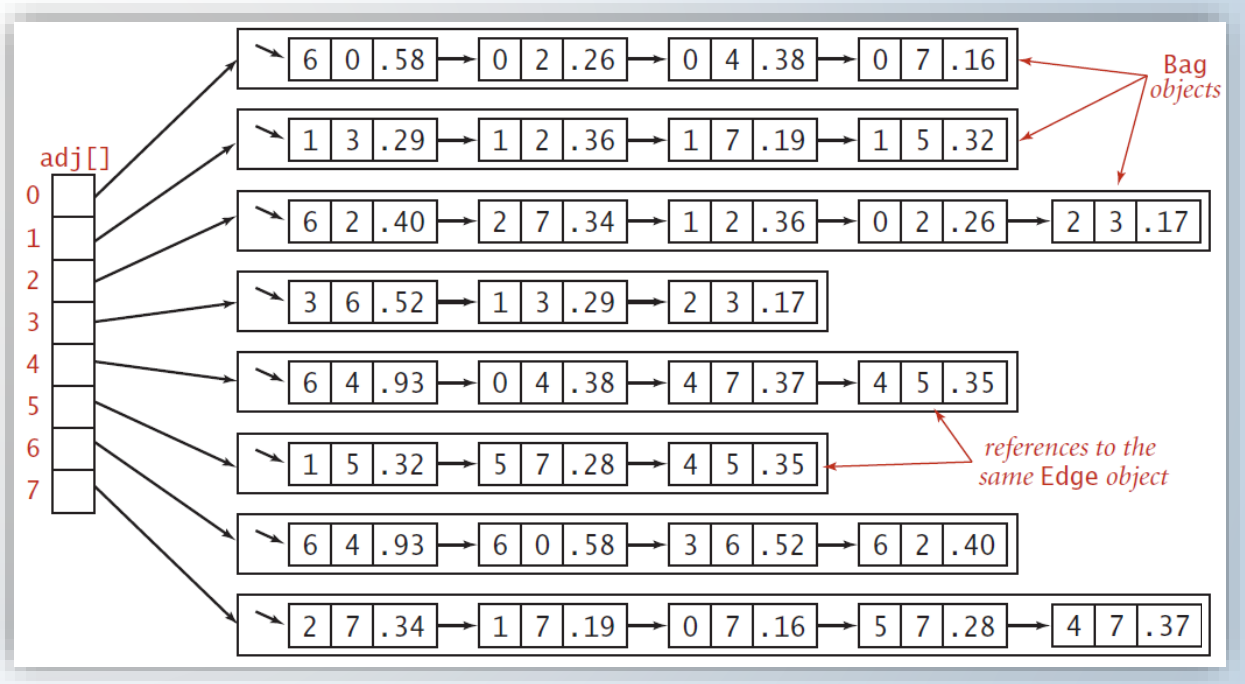
Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Grafos Ponderados

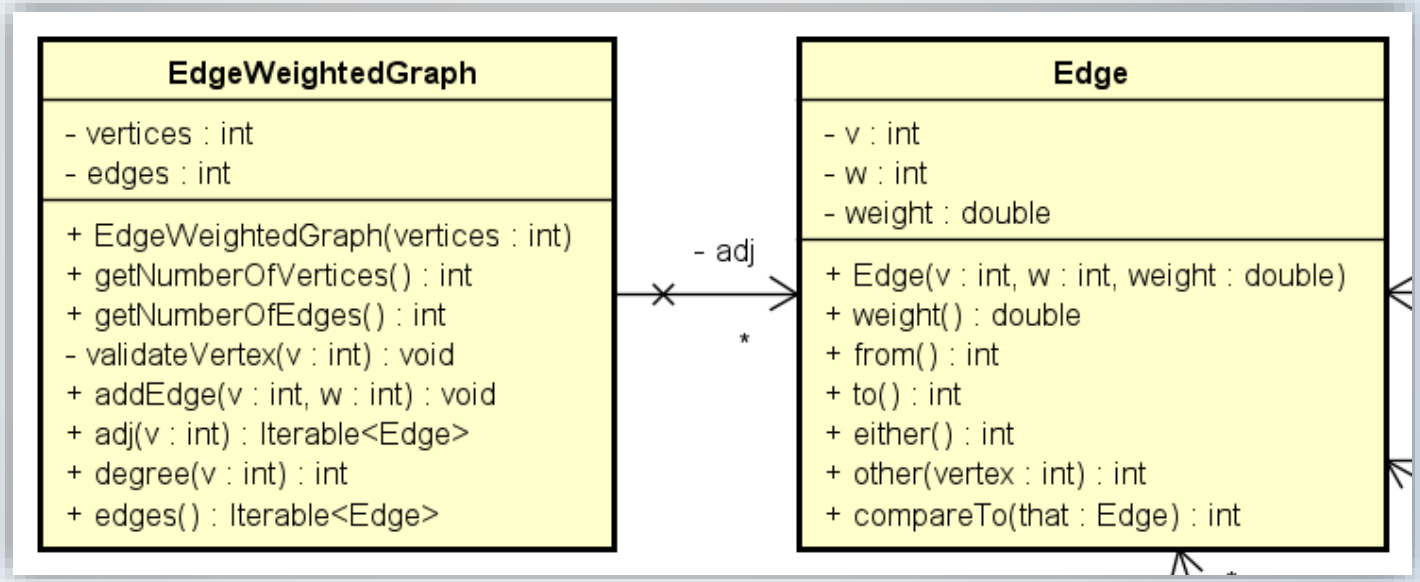
- **Grafo Ponderado:** um grafo em que pesos ou custos estão associados às arestas.





Grafos Ponderados

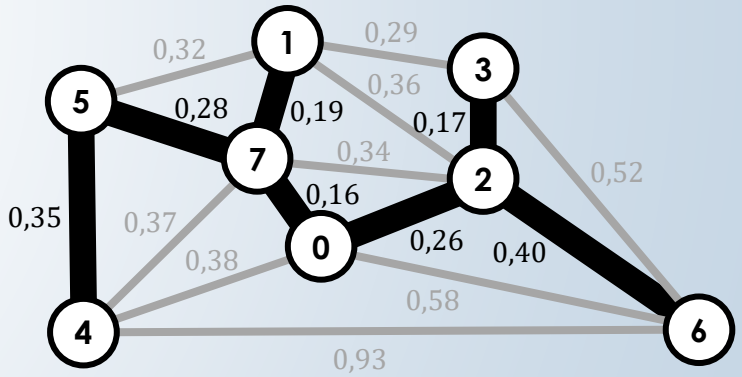
API das classes para Grafos Ponderados



Árvores Geradoras Mínimas

MST (*Minimum Spanning Trees*)

- **Definição:** para um grafo ponderado, uma árvore geradora mínima é um subgrafo conexo sem ciclos, que inclui todos os vértices e cujo peso (a soma dos pesos de todas as arestas) não é maior que o peso de qualquer outra árvore geradora.
- **Importância e Aplicações:** dado que uma MST representa o menor custo envolvido ao se construir um grafo ponderado, ou mesmo o custo de um grafo ponderado que representa algo existente, ele é aplicado na construção de vários tipos de redes como comunicação, elétrica, hidráulica, aeronáutica etc.

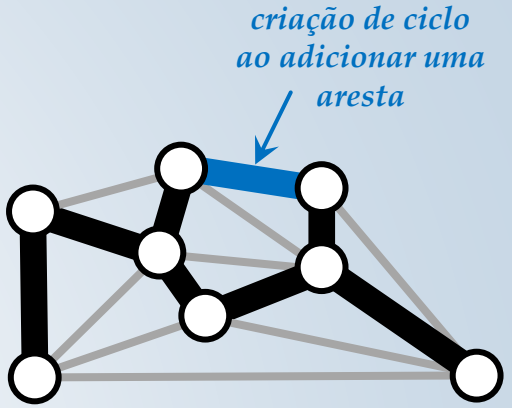


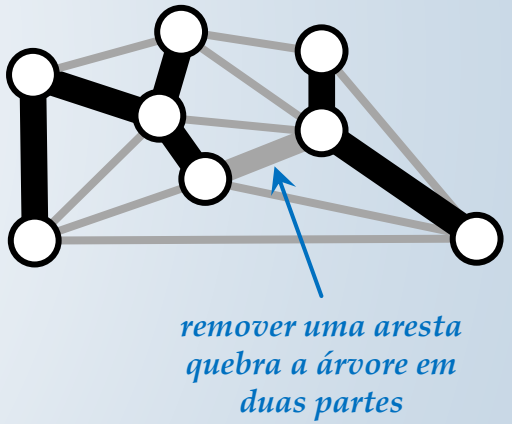
Árvores Geradoras Mínimas
MST (*Minimum Spanning Trees*)

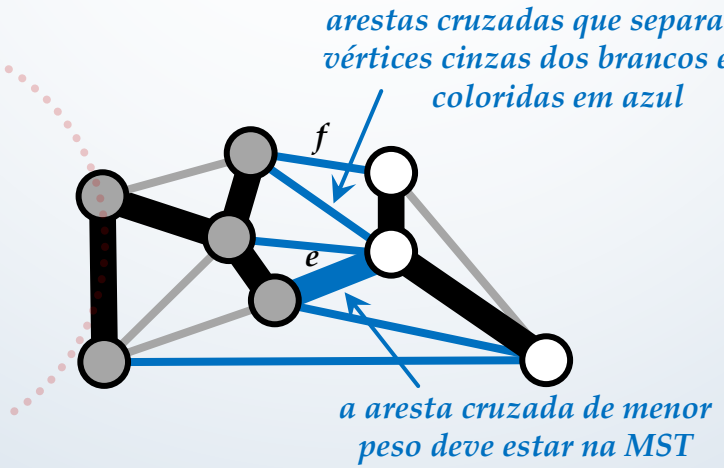
- Veremos dois algoritmos gulosos para a computação de MSTs:
  - **Algoritmo de Prim-Jarník;**
  - **Algoritmo de Kruskal.**
- Para esses algoritmos, assumiremos que:
  - O grafo é conexo. Caso não seja, é necessário adaptar o algoritmo para calcular a MST de cada componente conexo, obtendo assim uma floresta geradora mínima;
  - Os pesos das arestas não são necessariamente distâncias;
  - Os pesos podem ser zero ou negativos;
  - Nenhuma aresta tem peso igual a outra, apesar dos algoritmos conseguirem lidar com isso, mas não garantindo a obtenção de uma MST única, pois assim haverá mais de uma MST para um dado grafo.

Árvores Geradoras Mínimas
MST (*Minimum Spanning Trees*)

- **Conceitos importantes:**
  - Adicionar uma aresta que conecta dois vértices em uma árvore cria um ciclo;
  - Remover uma aresta de uma árvore a quebra em duas árvores;
  - **Propriedade do Corte:** O corte de um grafo consiste no particionamento dos vértices em dois conjuntos disjuntos não vazios. Uma aresta cruzada de um corte é a aresta que conecta um vértice de um conjunto com um vértice do outro conjunto.



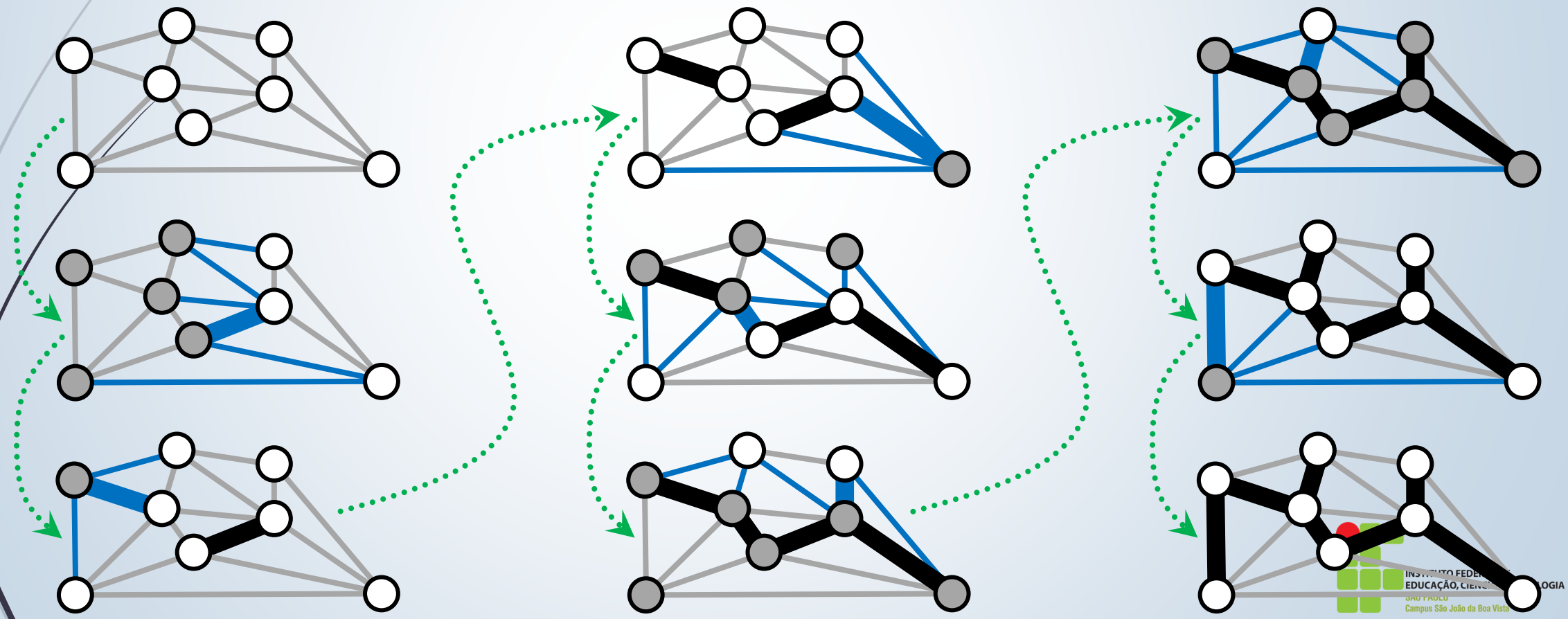




Árvores Geradoras Mínimas

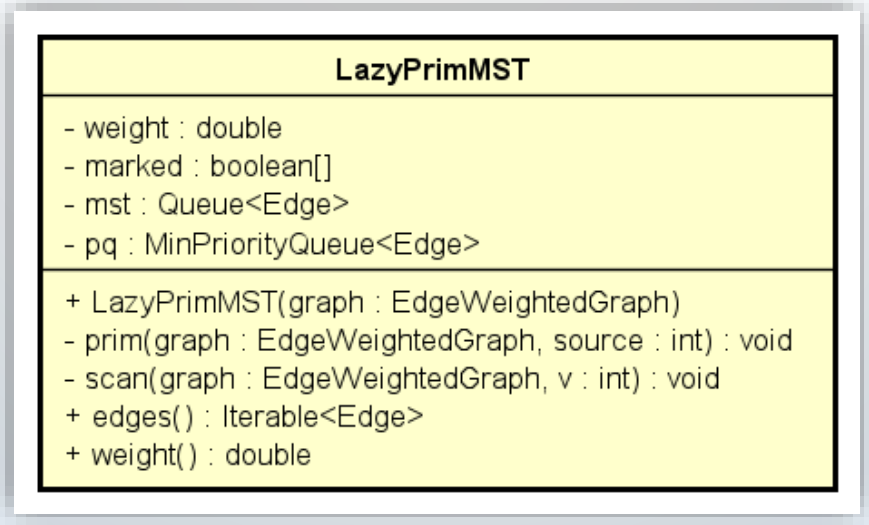
MST (*Minimum Spanning Trees*)

- O cômputo de uma MST envolve encontrar a aresta cruzada de menor peso em cada corte possível do grafo;



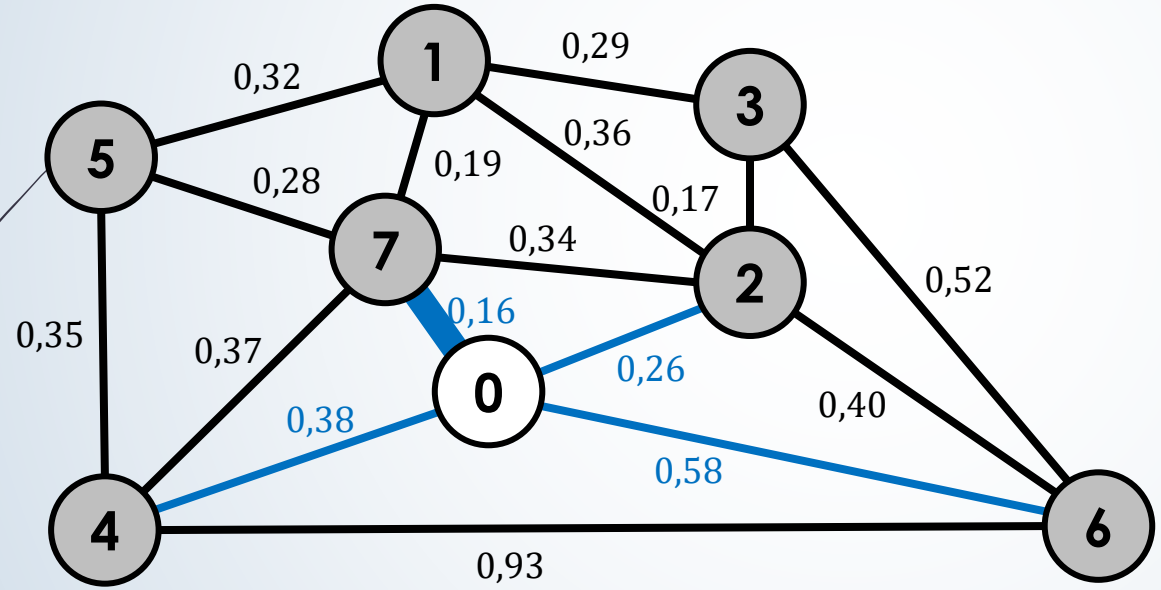
Árvores Geradoras Mínimas
Algoritmo de Prim (*Lazy*)

- **Estruturas de dados:**
  - **`boolean[] marked:`** armazena se um vértice está na árvore;
  - **`marked[v] = true:`** v está na árvore;
  - **`Queue<Edge> mst:`** arestas da MST;
  - **`MinPriorityQueue<Edge> pq:`** mantém as arestas cruzadas, comparadas pelo peso.



Simulação

Adiciona 0 à MST e todas as arestas na sua lista de adjacências à fila de prioridades.

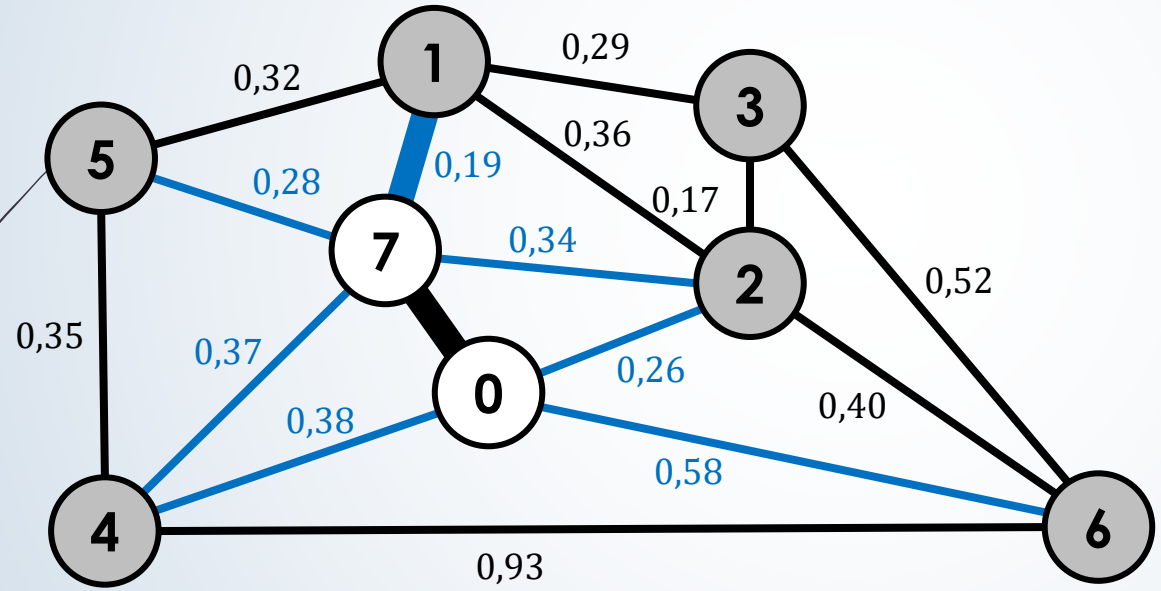


pq		
*	0-7	0,16
*	0-2	0,26
*	0-4	0,38
*	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 7 e 0 – 7 à MST e todas as arestas na sua lista de adjacências à fila de prioridades.

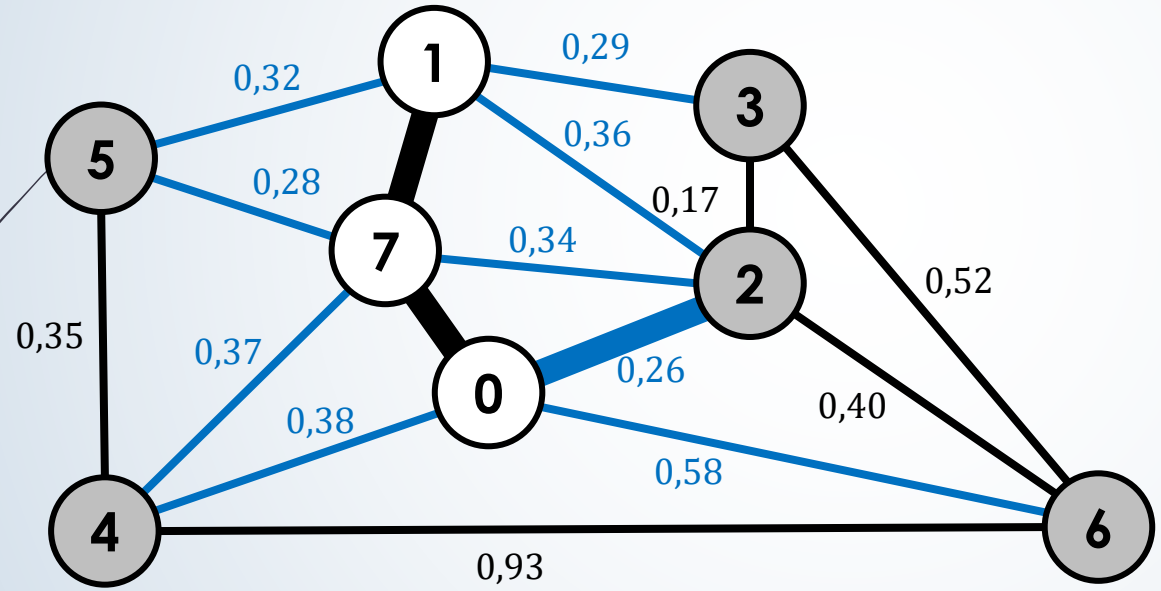


pq		
*	1-7	0,19
	0-2	0,26
*	5-7	0,28
*	2-7	0,34
*	4-7	0,37
	0-4	0,38
	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 1 e 1 – 7 à MST e todas as arestas na sua lista de adjacências à fila de prioridades.

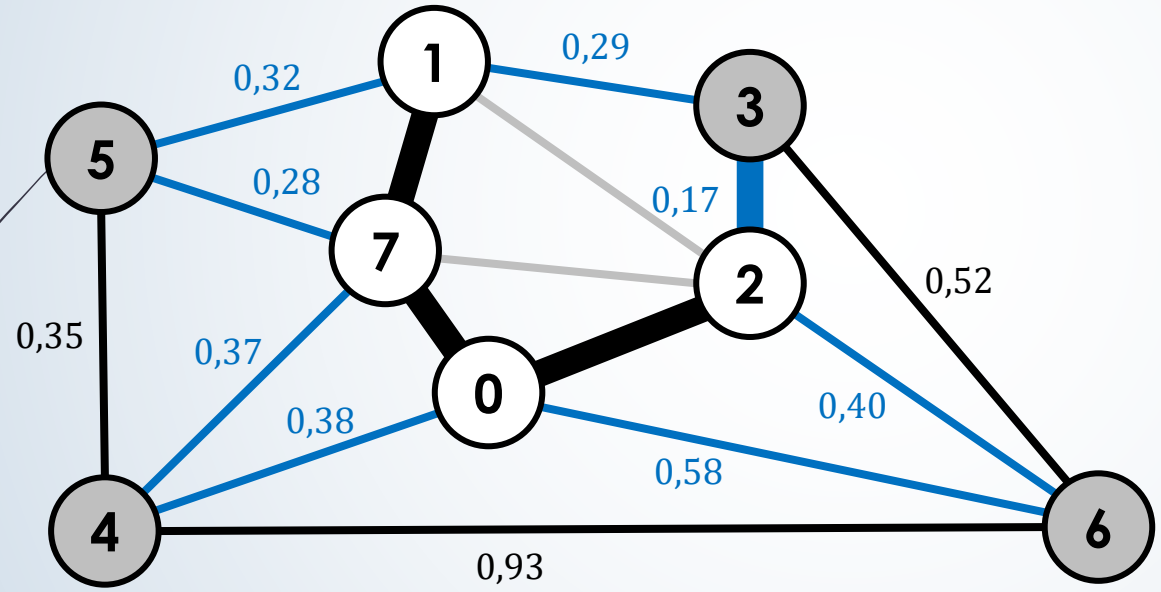


	pq	
	0-2	0,26
	5-7	0,28
*	1-3	0,29
*	1-5	0,32
	2-7	0,34
*	1-2	0,36
	4-7	0,37
	0-4	0,38
	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 2 e 0 – 2 à MST e as arestas 2 – 3 e 6 – 2 à fila de prioridades. As arestas 2 – 7 e 1 – 2 se tornam inelegíveis. **As arestas inelegíveis são as arestas não cruzadas.**

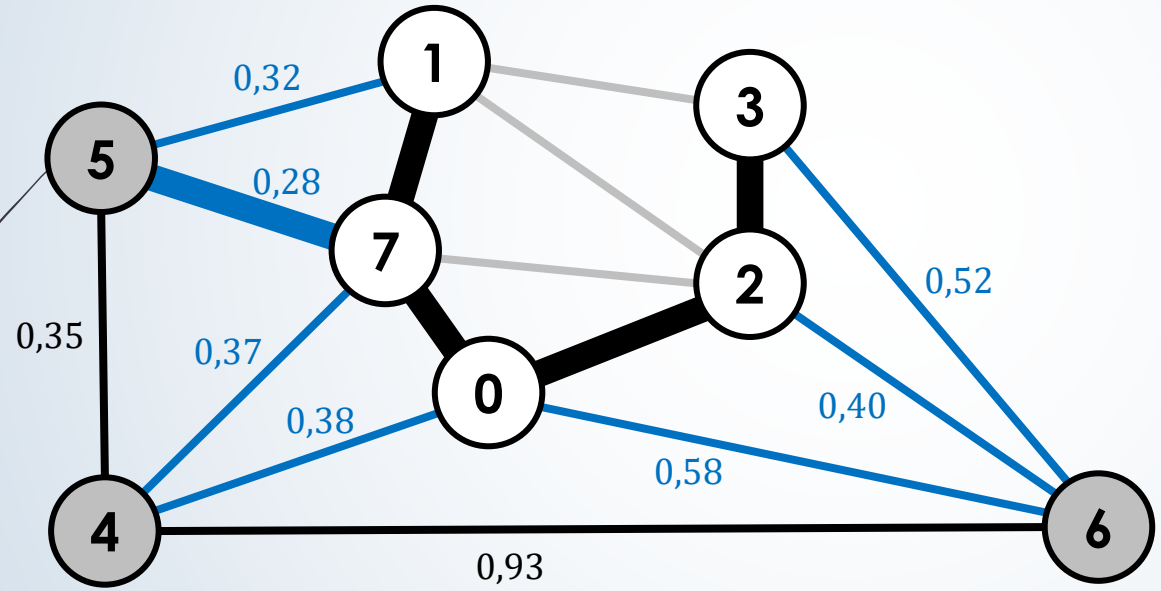


	pq	
*	2-3	0,17
	5-7	0,28
	1-3	0,29
	1-5	0,32
	2-7	0,34
	1-2	0,36
	4-7	0,37
	0-4	0,38
*	6-2	0,40
	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 3 e 2 – 3 à MST e a aresta 3 – 6 à fila de prioridades. A aresta 1 – 3 se torna inelegível.

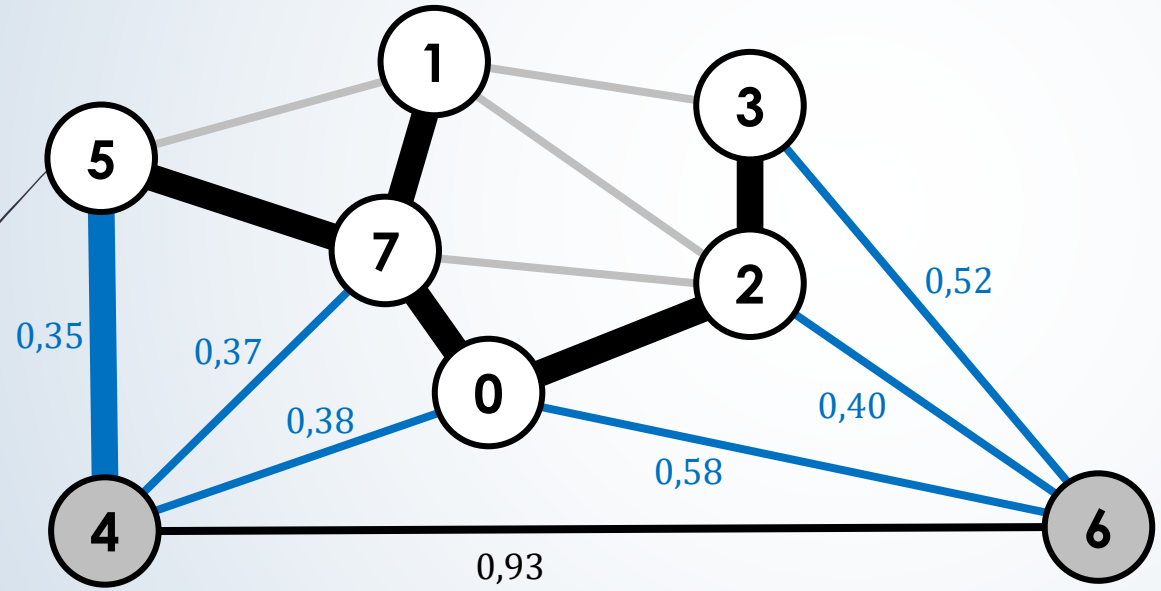


	pq	
	5-7	0,28
	1-3	0,29
	1-5	0,32
	2-7	0,34
	1-2	0,36
	4-7	0,37
	0-4	0,38
	6-2	0,40
*	3-6	0,52
	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 5 e 5 – 7 à MST e a aresta 4 – 5 à fila de prioridades. A aresta 1 – 5 se torna inelegível.

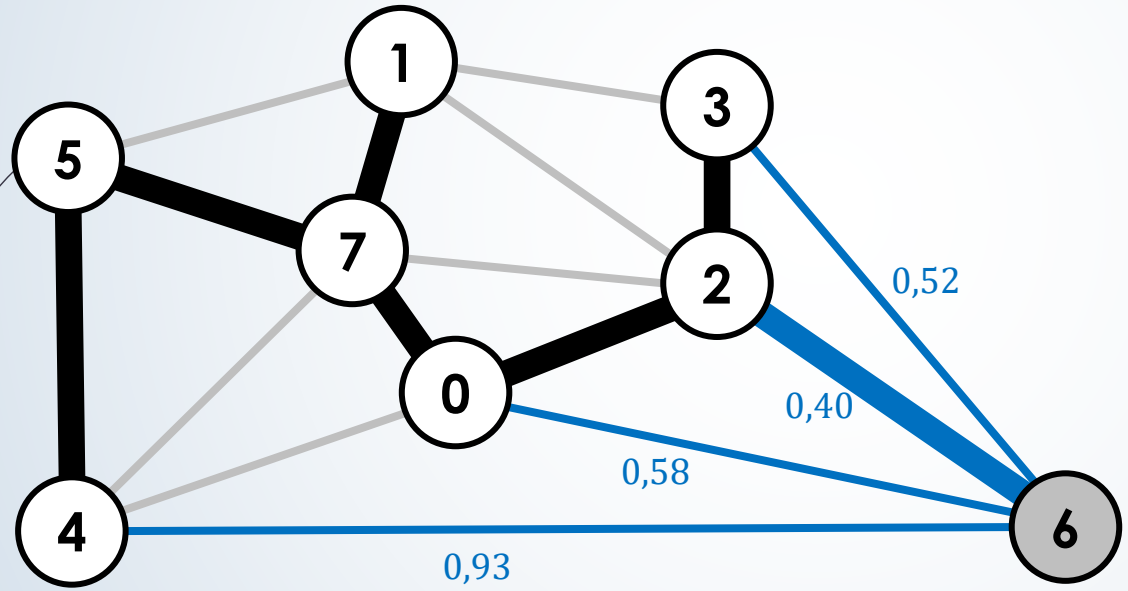


	pq	
	1-3	0,29
	1-5	0,32
	2-7	0,34
*	4-5	0,35
	1-2	0,36
	4-7	0,37
	0-4	0,38
	6-2	0,40
	3-6	0,52
	6-0	0,58

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Remove as arestas inelegíveis $1-3$, $1-5$ e $2-7$ da fila de prioridades. Adiciona 4 e $4-5$ à MST e a aresta $6-4$ à fila de prioridades. As arestas $4-7$ e $0-4$ se tornam inelegíveis.

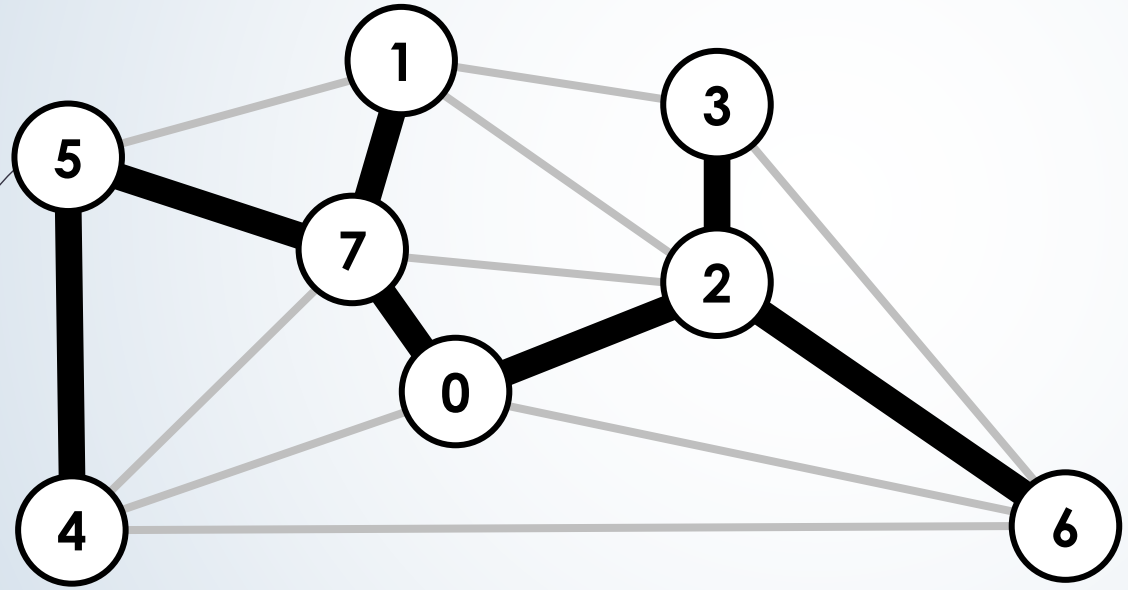


	pq	
	1-2	0,36
	4-7	0,37
	0-4	0,38
	6-2	0,40
	3-6	0,52
	6-0	0,58
*	6-4	0,93

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Remove as arestas inelegíveis $1-2$, $4-7$ e $0-4$ da fila de prioridades. Adiciona 6 e $6-2$ à MST. O restante das arestas incidentes em 6 se tornam inelegíveis.



pq	
3-6	0,52
6-0	0,58
6-4	0,93

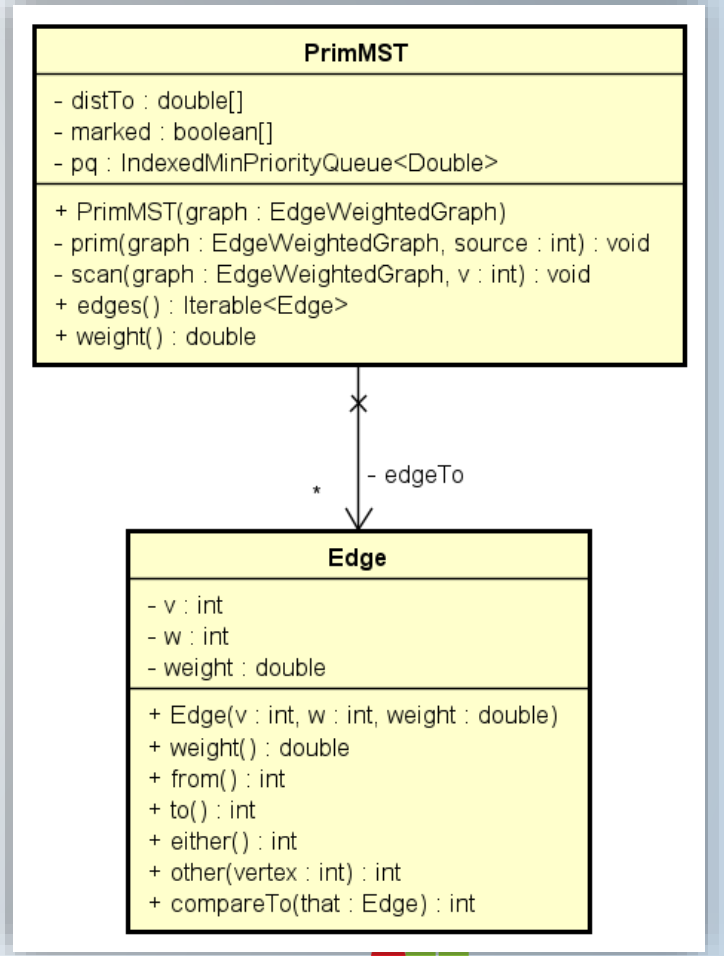
v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Árvores Geradoras Mínimas
Algoritmo de Prim (*Eager*)

- A versão *eager* (ansiosa) do algoritmo de Prim otimiza a versão *lazy* não mantendo todas as arestas de w na fila de prioridades, visto que a única aresta necessária, ou de interesse, é a que tem menor peso e que o liga à árvore.

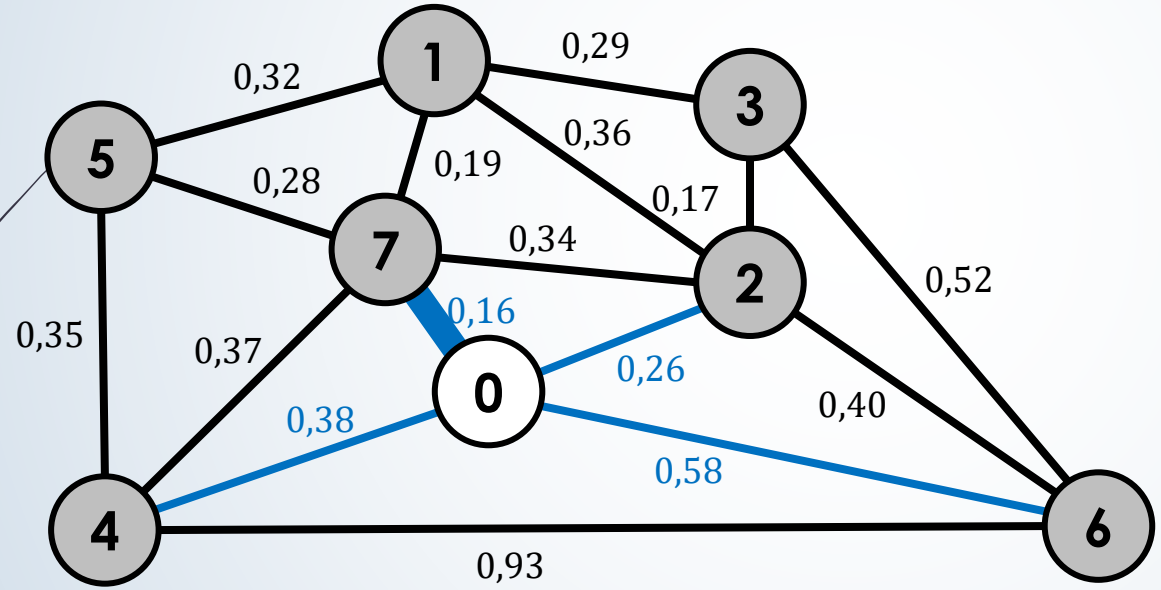
Árvores Geradoras Mínimas
Algoritmo de Prim (*Eager*)

- **Estruturas de dados:**
  - **`boolean[] marked`:** armazena se um vértice está na árvore;
  - **`marked[v] = true`:** v está na árvore;
  - **`Edge[] edgeTo`:** arestas da MST;
  - **`edgeTo[v]`:** é a aresta que conecta v na árvore;
  - **`double[] distTo`:** armazena os pesos das arestas;
  - **`distTo[v]`:** o peso da aresta que contém v na árvore;
  - **`IndexedMinPriorityQueue<Double> pq`:** mantém as arestas cruzadas, comparadas pelo peso;



Simulação

Adiciona 0 à MST e todas as arestas na sua lista de adjacências à fila de prioridades, dado que cada aresta é a melhor (e única) conexão conhecida entre um vértice da árvore e um vértice que não está na árvore.

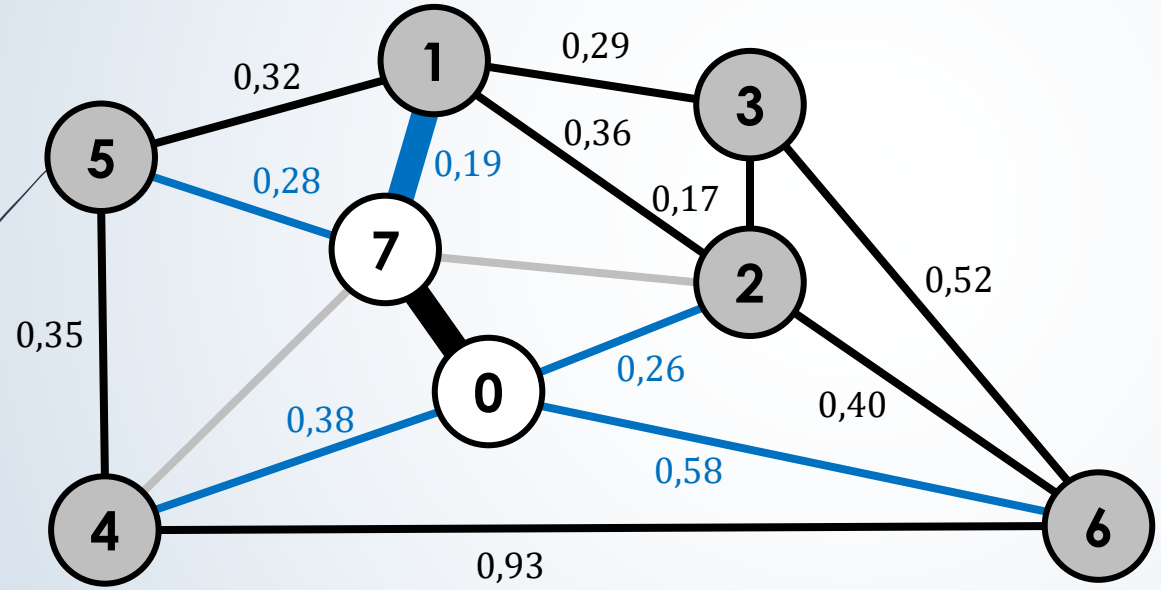


v	edgeTo[]	distTo[]
0		0,00
1		+∞
2	0-2	0,26
3		+∞
4	0-4	0,38
5		+∞
6	6-0	0,58
7	0-7	**0,16**

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 7 e 0 – 7 à MST e 1 – 7 e 5 – 7 à fila de prioridades. As arestas 4 – 7 e 2 – 7 não afetam a fila de prioridades, pois seus pesos não são menores que os pesos das conexões conhecidas da MST aos vértices 4 e 2, respectivamente.

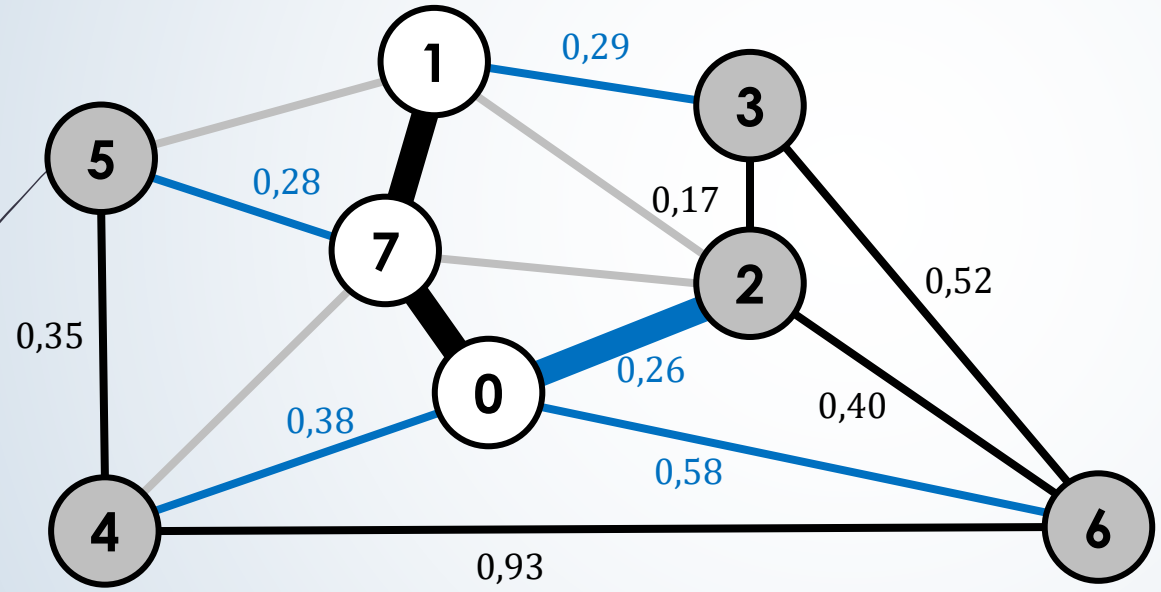


v	edgeTo[]	distTo[]
0		0,00
1	1-7	**0,19**
2	0-2	0,26
3		+∞
4	0-4	0,38
5	5-7	0,28
6	6-0	0,58
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 1 e 1 – 7 à MST e 1 – 3 à fila de prioridades.

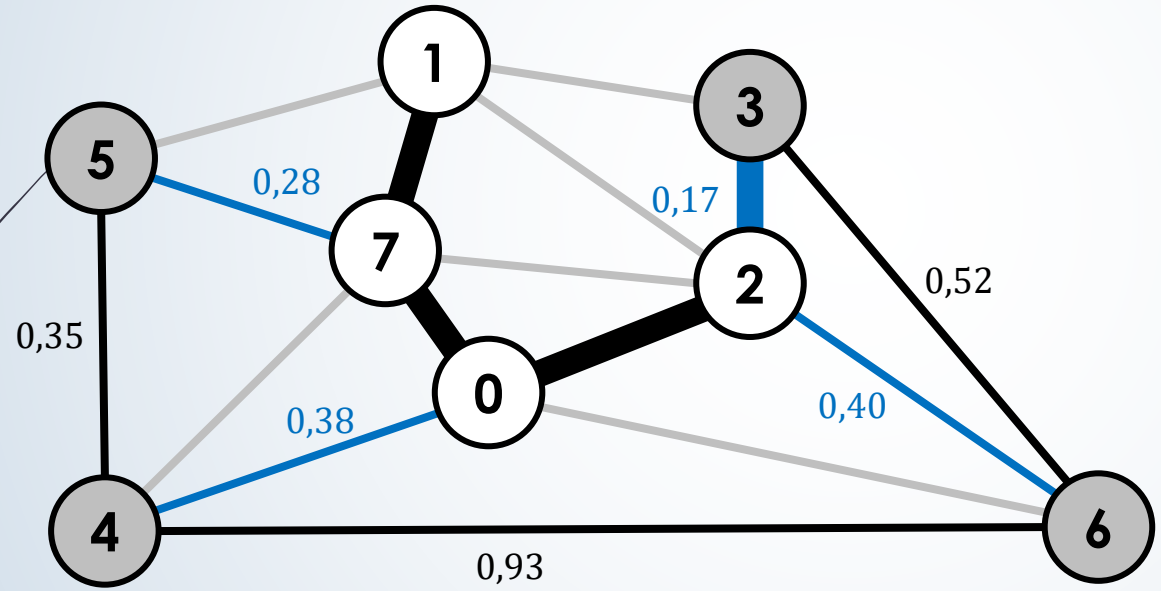


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	**0,26**
3	1-3	0,29
4	0-4	0,38
5	5-7	0,28
6	6-0	0,58
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 2 e 0 – 2 à MST, substitui 0 – 6 por 2 – 6 como a menor aresta de um vértice da árvore até 6 e substitui 1 – 2 por 2 – 3 como a menor aresta de um vértice da árvore até 3.

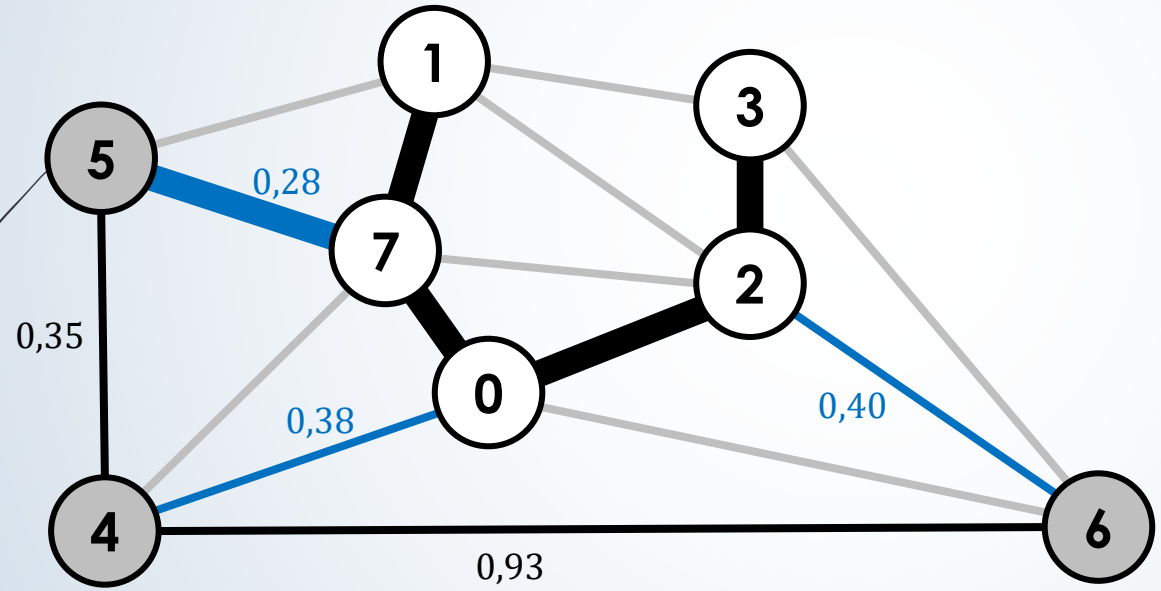


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	0,26
3	2-3	**0,17**
4	0-4	0,38
5	5-7	0,28
6	6-2	0,40
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 3 e 2 – 3 à MST.

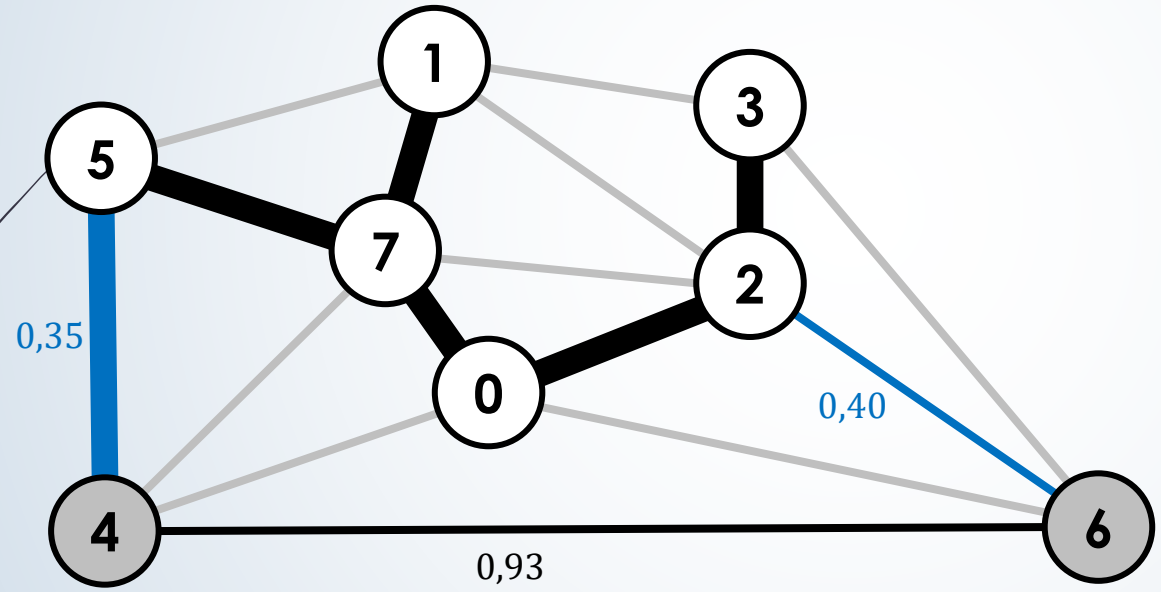


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	0,26
3	2-3	0,17
4	0-4	0,38
5	5-7	**0,28**
6	6-2	0,40
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 5 e 5 – 7 à MST e substitui 0 – 4 por 4 – 5 como a menor aresta de um vértice da árvore até 4.

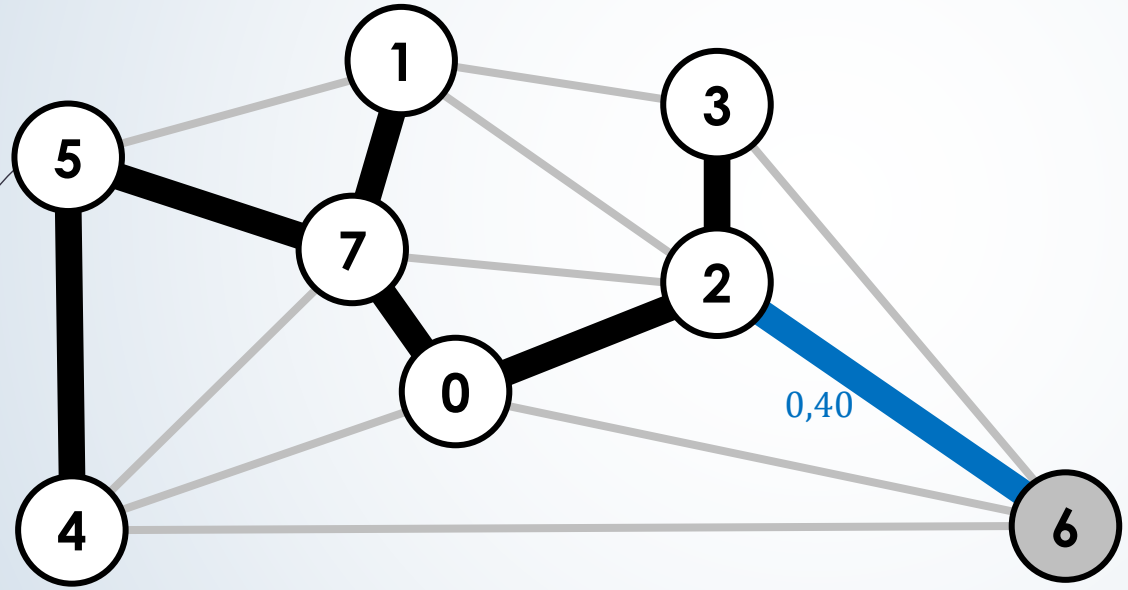


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	0,26
3	2-3	0,17
4	4-5	**0,35**
5	5-7	0,28
6	6-2	0,40
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 4 e 4 – 5 à MST.

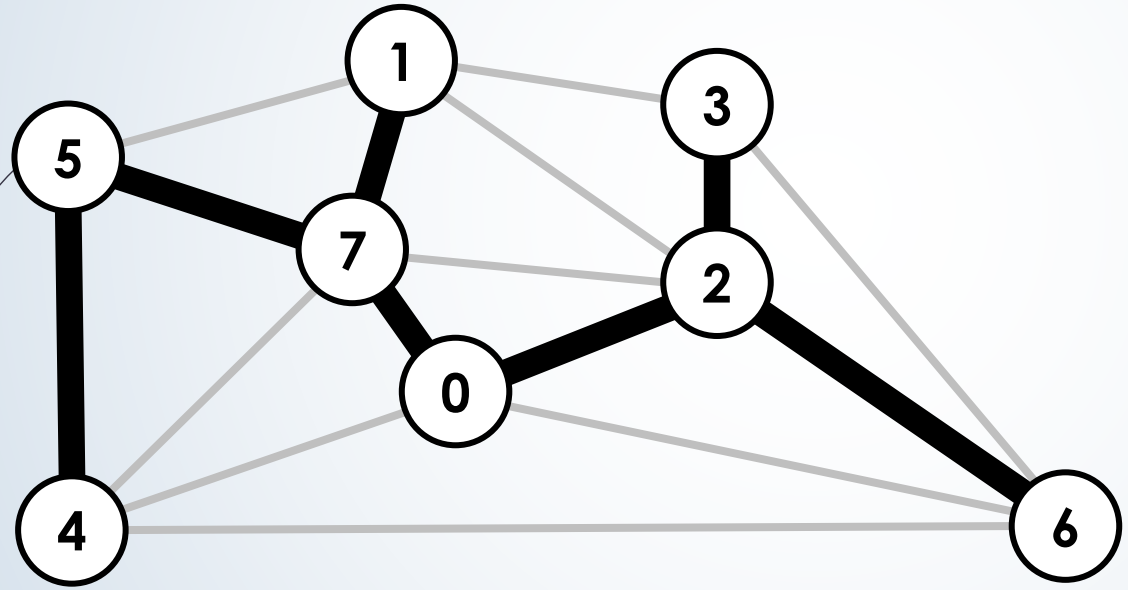


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	0,26
3	2-3	0,17
4	4-5	0,35
5	5-7	0,28
6	6-2	**0,40**
7	0-7	0,16

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Simulação

Adiciona 6 e 6 – 2 à MST.

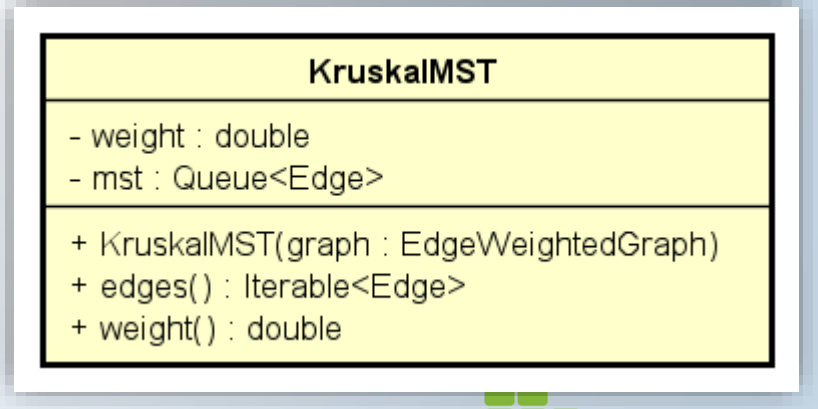


v	edgeTo[]	distTo[]
0		0,00
1	1-7	0,19
2	0-2	0,26
3	2-3	0,17
4	4-5	0,35
5	5-7	0,28
6	6-2	0,40
7	0-7	0,16

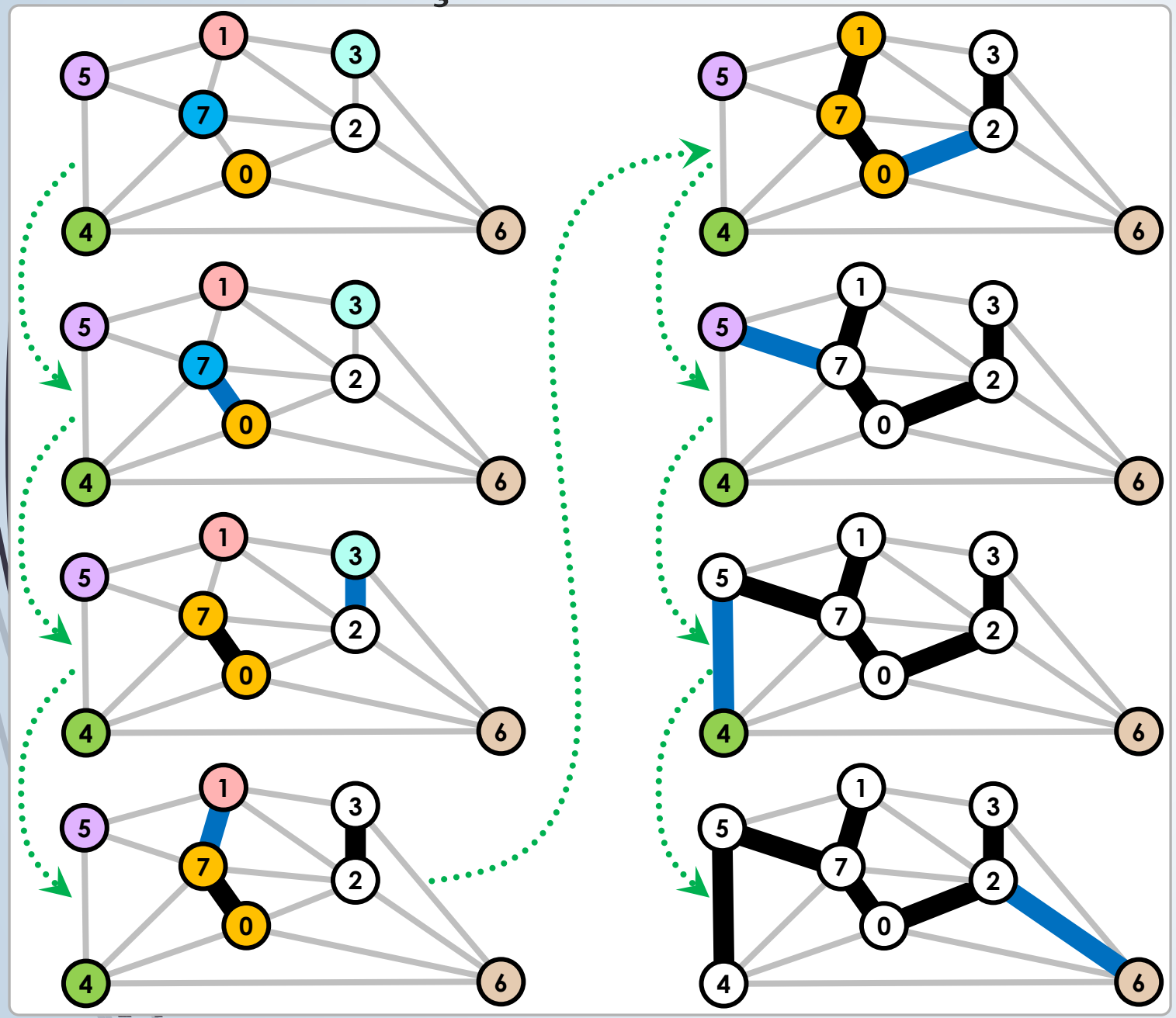
v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Árvores Geradoras Mínimas
Algoritmo de Kruskal

- O algoritmo de Kruskal consiste em processar as arestas na ordem de seus pesos (menor para o maior), inserindo na MST todas as arestas que não formam ciclos com as arestas adicionadas anteriormente, parando o processo após a inserção de $V - 1$ arestas. As arestas que vão sendo inseridas formam uma floresta que evolui gradualmente à uma árvore única, a MST;
- Internamente, uma fila de prioridades mínima será usada para obter a ordem de processamento das arestas e a estrutura de dados UnionFind será usada para verificar a conectividade/ciclo entre vértices;
- **Estrutura de dados:**
  - `Queue<Edge> mst:` arestas da MST.



Simulação

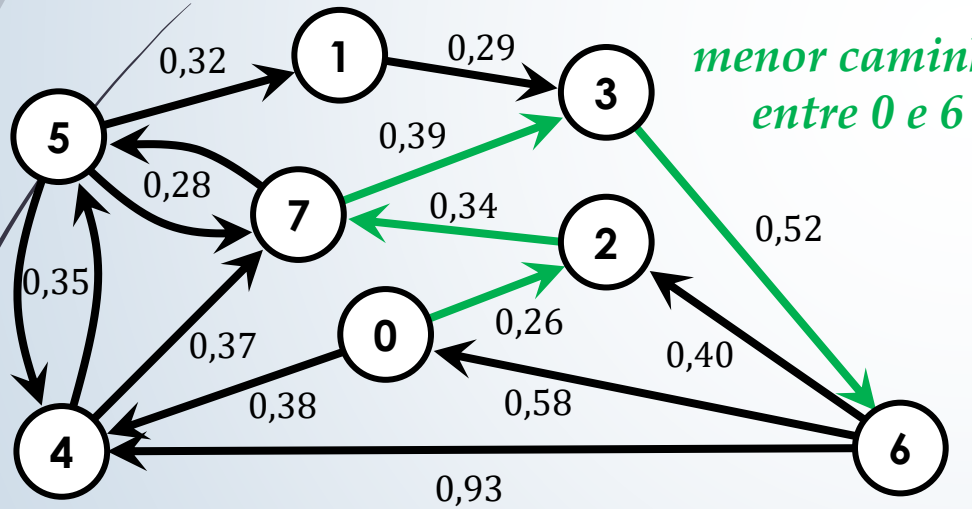


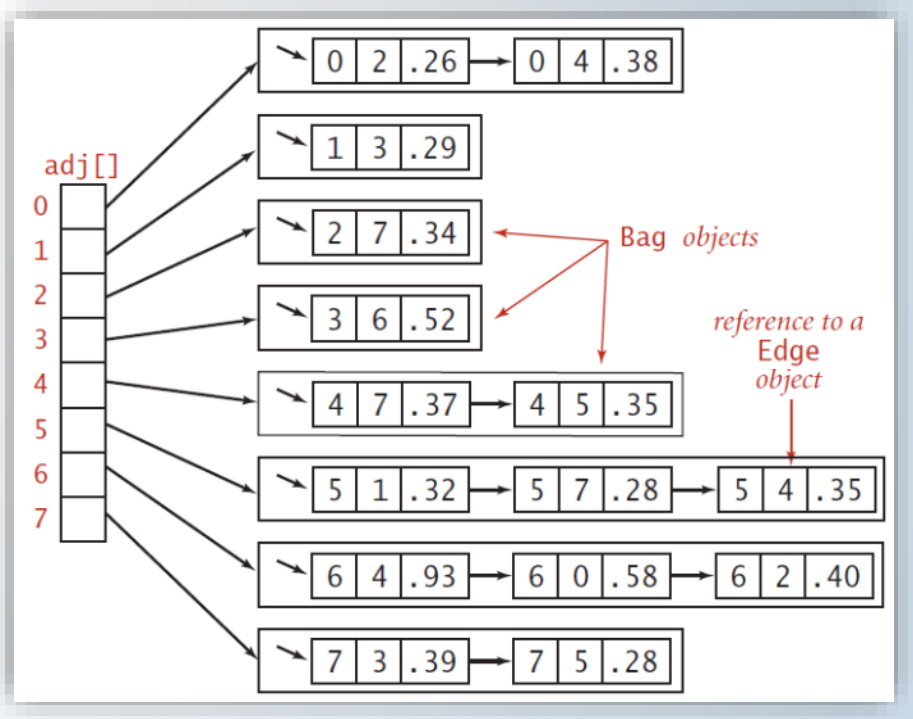
mst	pq
0-7	0,16
2-3	0,17
1-7	0,19
0-2	0,26
5-7	0,28
1-3	0,29
1-5	0,32
2-7	0,34
4-5	0,35
1-2	0,36
4-7	0,37
0-4	0,38
6-2	0,40
3-6	0,52
6-0	0,58
6-4	0,93

v	adjacentes				
0	6	2	4	7	
1	3	2	7	5	
2	6	7	1	0	3
3	6	1	2		
4	6	0	7	5	
5	1	7	4		
6	4	0	3	2	
7	2	1	0	5	4

Digrafos Ponderados

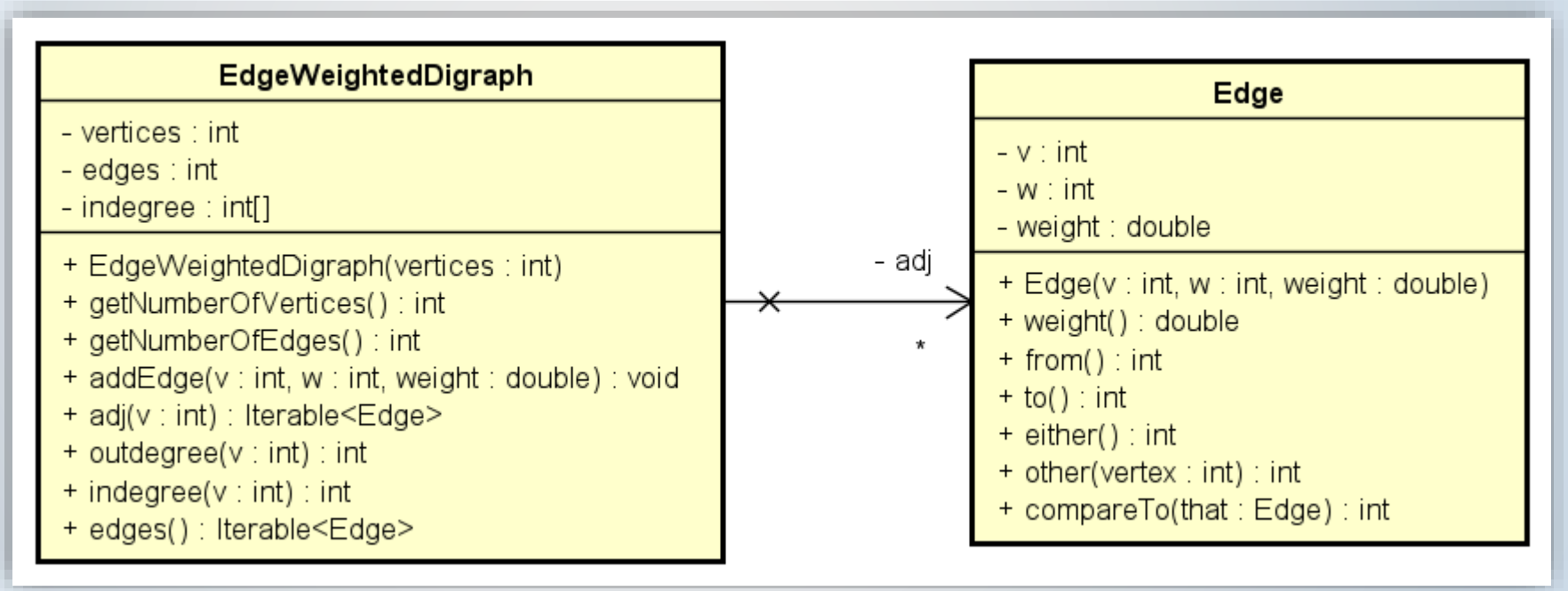
- **Digrafo Ponderado:** um digrafo em que pesos ou custos estão associados às arestas.





Digrafos Ponderados

API das classes para Digrafos Ponderados



Menor Caminho

SP (*Shortest Path*)

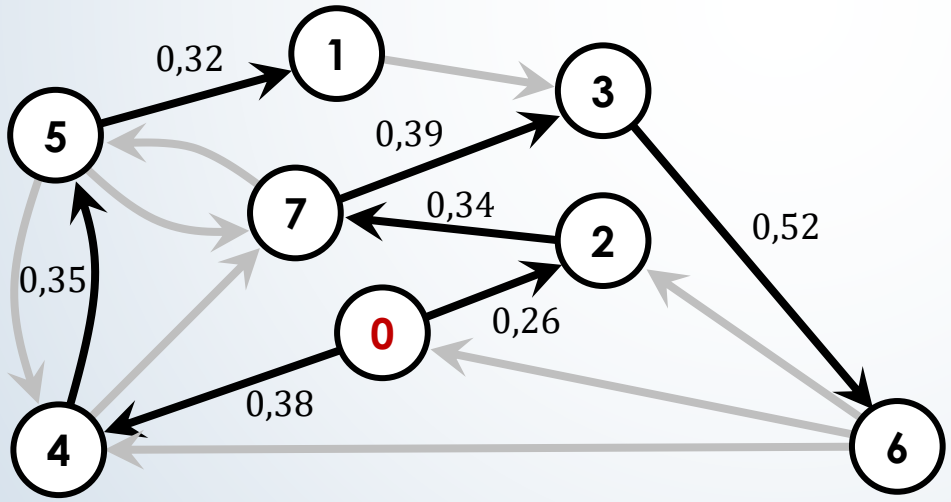
- **Definição:** O menor caminho de um vértice s a um vértice t em um grafo ponderado é um caminho direto de s a t com a propriedade de que nenhum outro caminho tem menor peso;
- **Importância e Aplicações:** um menor caminho em um grafo, envolvendo dois vértices, representa o menor custo envolvido em se caminhar do vértice fonte ao vértice de destino, sendo assim, uma das aplicações mais facilmente notada é do cálculo do menor caminho entre cidades em um mapa. Outra possibilidade de aplicação seria no roteamento de pacotes em uma rede de computadores, em que o melhor caminho a se tomar é o que visita os nós da rede através de conexões mais rápidas.

Menor Caminho
SP (*Shortest Path*)

- Veremos um algoritmo para a computação dos menores caminhos entre vértices em um digrafo ponderado:
  - **Algoritmo de Dijkstra.**
- Para esse algoritmo, assumiremos que:
  - Os caminhos são direcionados;
  - Os pesos das arestas não são necessariamente distâncias;
  - Nem todos os vértices precisam ser alcançáveis;
  - Os pesos serão positivos;
  - Ciclos são ignorados;
  - Os menores caminhos não são necessariamente únicos;
  - Arestas paralelas e loops são permitidos.

Menor Caminho
Algoritmo de Dijkstra

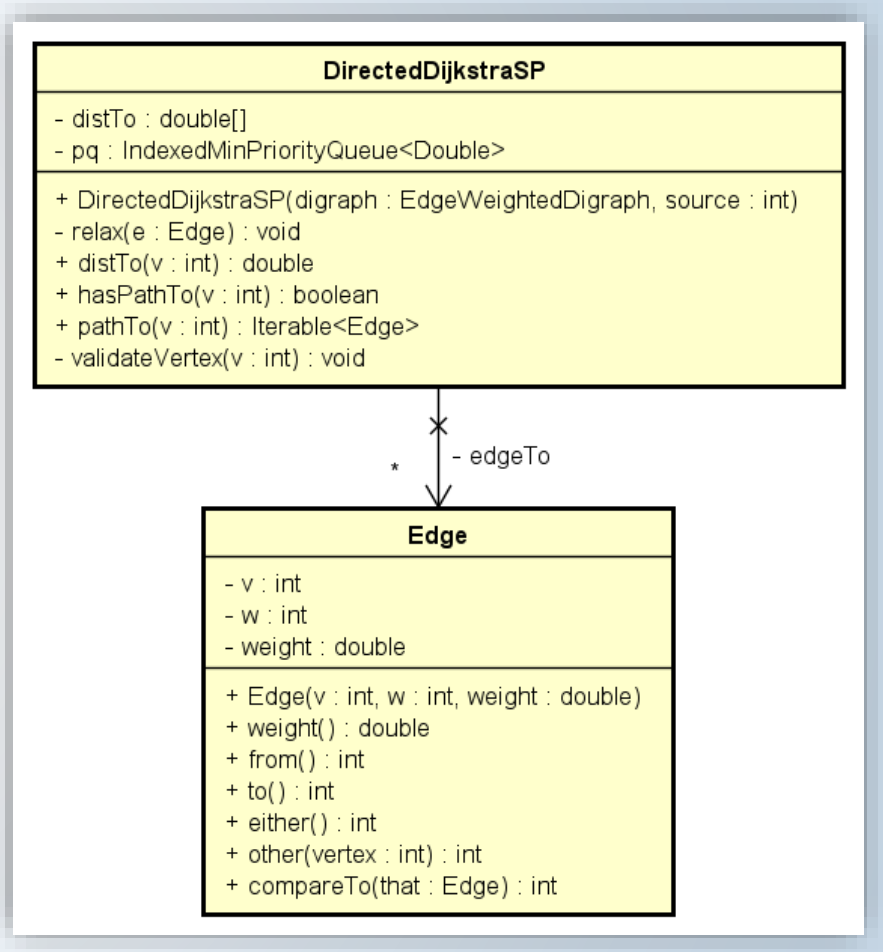
- O algoritmo de Dijkstra é análogo ao algoritmo de Prim. Enquanto no algoritmo de Prim constrói-se a MST adicionando novas arestas a cada passo, no algoritmo de Dijkstra a SPT (*Shortest Path Tree*) é construída, passo a passo, relaxando-se vértices que não fazem parte da SPT que tenham o menor valor **`distTo[v]`**.



v	edgeTo[]		distTo[]
0	`null`		0,00
1	5→1	0,32	1,05
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6	3→6	0,52	1,49
7	2→7	0,34	0,60

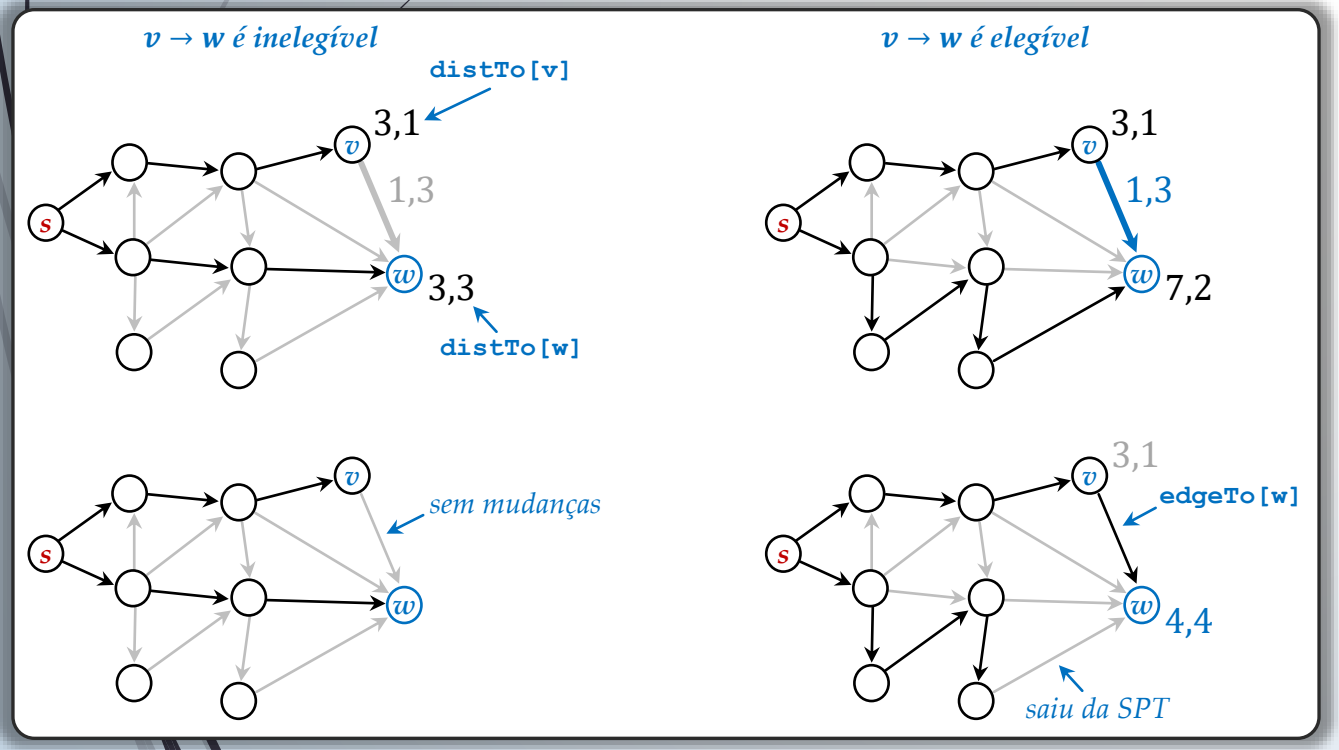
Menor Caminho
Algoritmo de Dijkstra

- **Estruturas de dados:**
  - **`Edge[] edgeTo`:** menor caminho entre vértices;
  - **`edgeTo[v]`:** última aresta no menor caminho $s \rightarrow v$;
  - **`double[] distTo`:** armazena aa distância dos menores caminhos;
  - **`distTo[v]`:** distância do menor caminho entre $s \rightarrow v$;
  - **`IndexedMinPriorityQueue<Double> pq`:** fila de prioridades dos vértices próximos a serem relaxados;



Menor Caminho
Algoritmo de Dijkstra

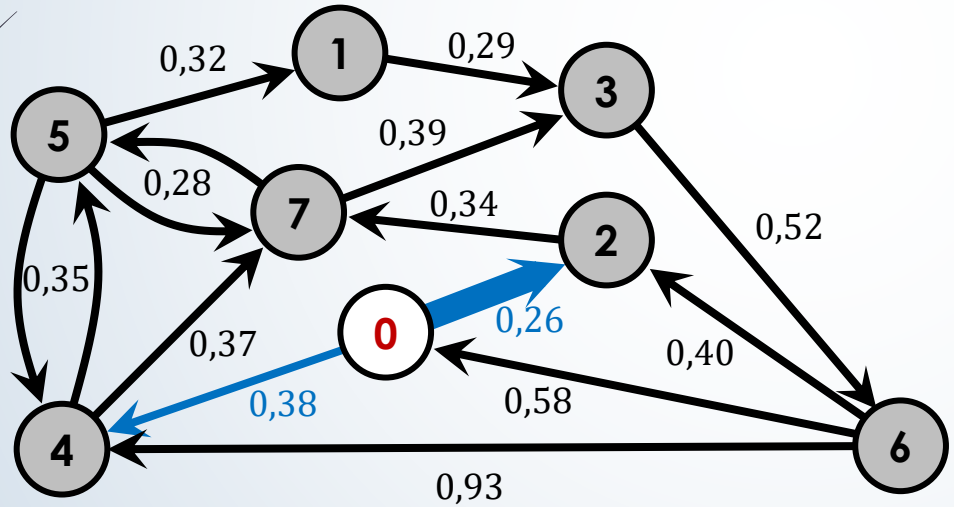
- **Relaxamento de Arestas:** relaxar uma aresta $v \rightarrow w$ significa testar se a melhor maneira conhecida até o momento de ir de s a w é ir de s a v, e então usar a aresta de v a w e, caso seja, atualizar as estruturas de dados para refletir essa possibilidade.




```
private void relax( Edge e ) {
    int v = e.from();
    int w = e.to();
    if ( distTo[w] > distTo[v] + e.weight() ) {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
    }
}
```

Simulação

Adiciona 0 à SPT e seus vértices adjacentes 2 e 4 à fila de prioridades.

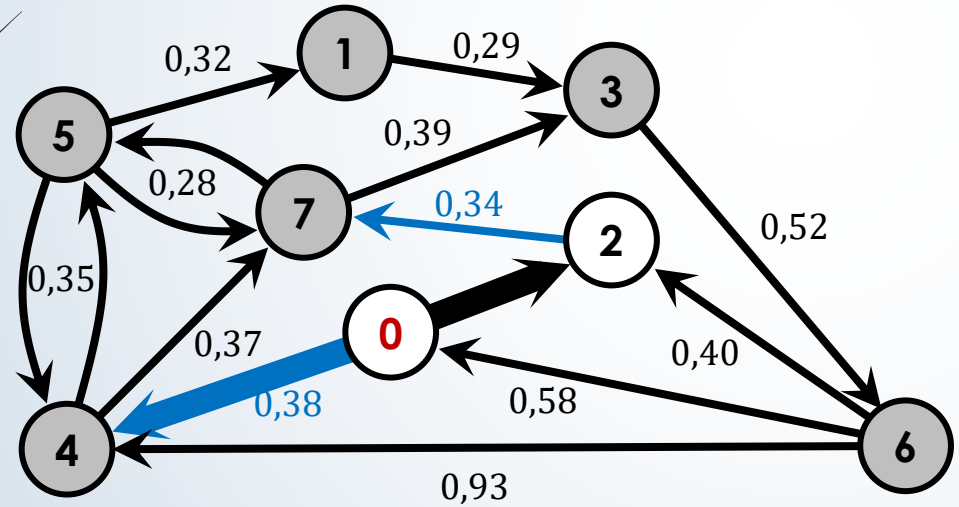


v	edgeTo[]		distTo[]
0	`null`		0,00
1			+∞
2	0→2	0,26	0,26 ←
3			+∞
4	0→4	0,38	0,38
5			+∞
6			+∞
7			+∞

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 2 da fila de prioridades, adiciona 0 → 2 à SPT e adiciona 7 à fila de prioridades.

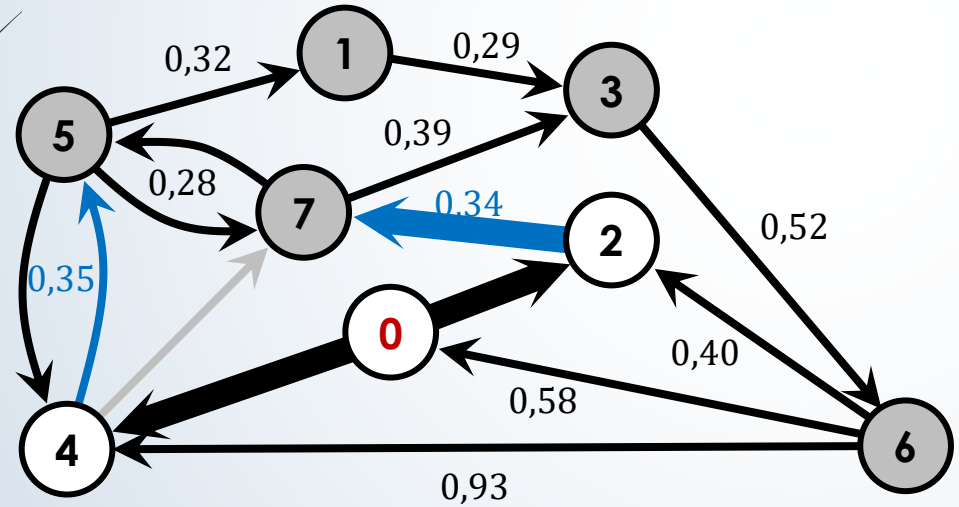


v	edgeTo[]		distTo[]
0	null		0,00
1			+∞
2	0→2	0,26	0,26
3			+∞
4	0→4	0,38	0,38 ←
5			+∞
6			+∞
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 4 da fila de prioridades, adiciona 0 → 4 à SPT e adiciona 5 à fila de prioridades.
A aresta 4 → 7 é inelegível.

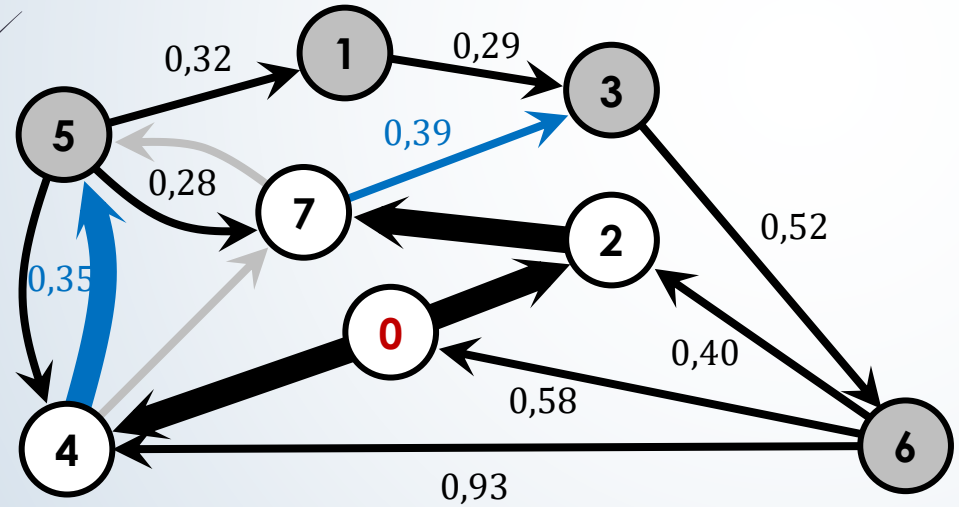


v	edgeTo[]		distTo[]
0	null		0,00
1			+∞
2	0→2	0,26	0,26
3			+∞
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6			+∞
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 7 da fila de prioridades, adiciona 2 → 7 à SPT e adiciona 3 à fila de prioridades.
A aresta 7 → 5 é inelegível.

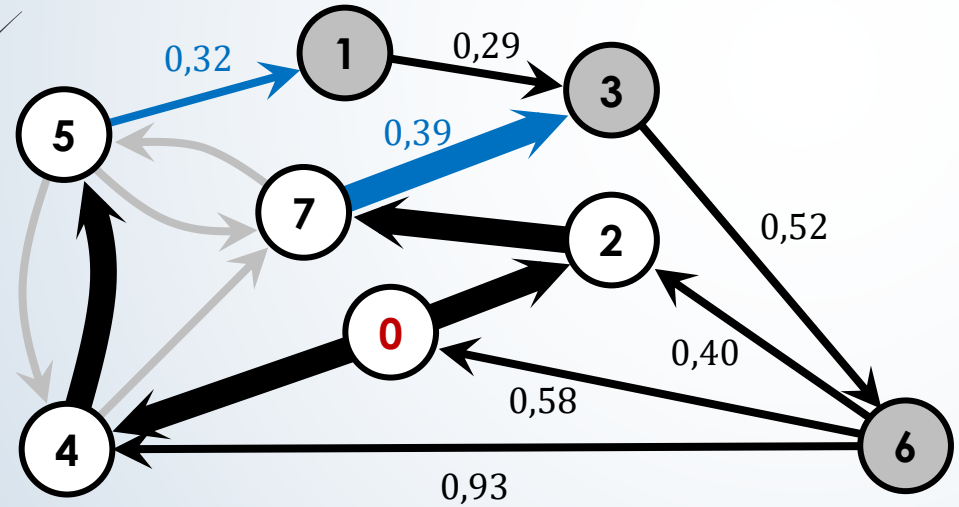


v	edgeTo[]		distTo[]
0	`null`		0,00
1			+∞
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6			+∞
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 5 da fila de prioridades, adiciona 4 → 5 à SPT e adiciona 1 à fila de prioridades.
A aresta 5 → 7 é inelegível.

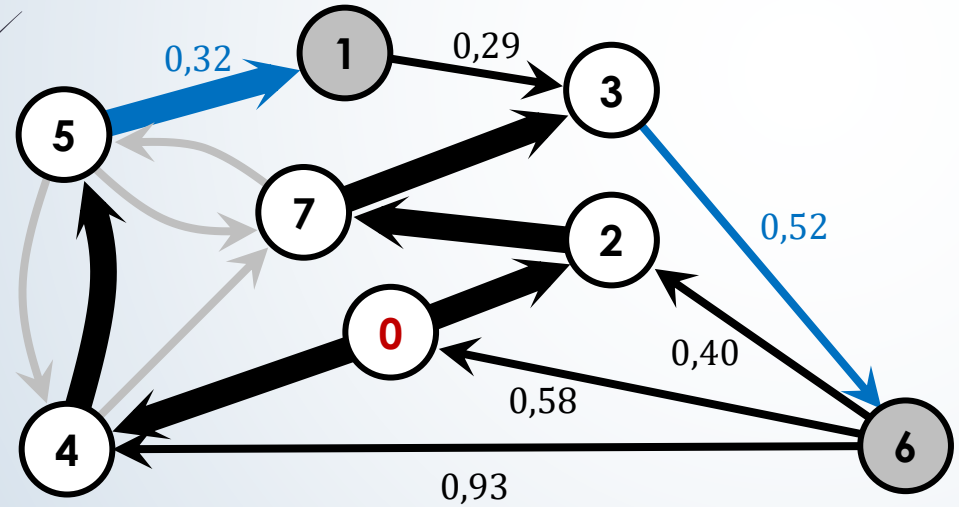


v	edgeTo[]		distTo[]
0	`null`		0,00
1	5→1	0,32	1,05
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6			+∞
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 3 da fila de prioridades, adiciona 7 → 3 à SPT e adiciona 6 à fila de prioridades.

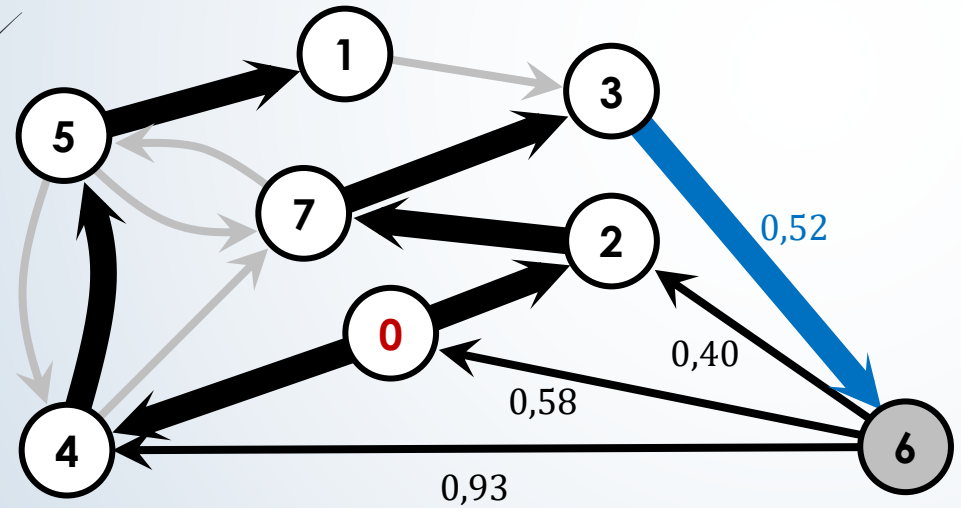


v	edgeTo[]		distTo[]
0	`null`		0,00
1	5→1	0,32	1,05
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6	3→6	0,52	1,49
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 1 da fila de prioridades e adiciona 5 → 1 à SPT. A aresta 1 → 3 é inelegível.

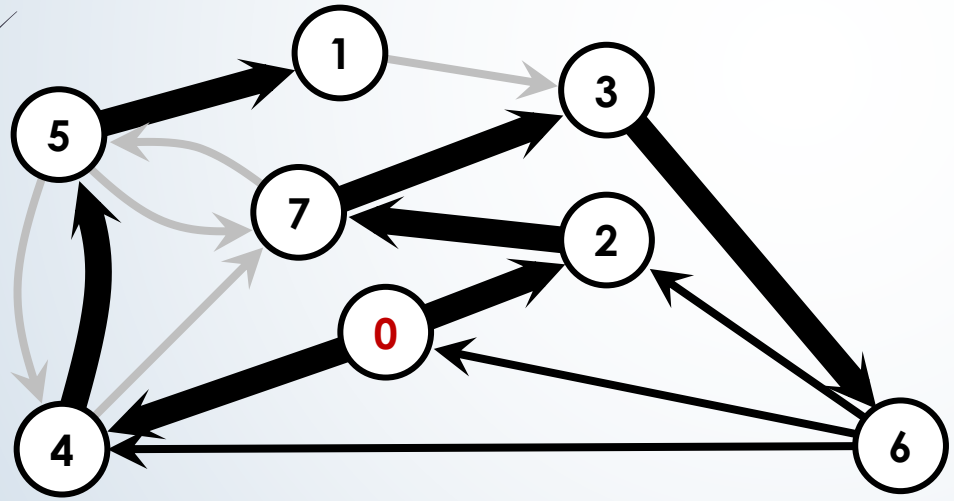


v	edgeTo[]		distTo[]
0	`null`		0,00
1	5→1	0,32	1,05
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6	3→6	0,52	1,49
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Simulação

Remove 6 da fila de prioridades e adiciona 3 → 6 à SPT.



v	edgeTo[]		distTo[]
0	`null`		0,00
1	5→1	0,32	1,05
2	0→2	0,26	0,26
3	7→3	0,37	0,97
4	0→4	0,38	0,38
5	4→5	0,35	0,73
6	3→6	0,52	1,49
7	2→7	0,34	0,60

v	adjacentes		
0	2	4	
1	3		
2	7		
3	6		
4	7	5	
5	1	7	4
6	4	0	2
7	3	5	

Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

WEISS, M. A. Data Structures and **Algorithm Analysis in Java**. 3. ed. Pearson Education: New Jersey, 2012. 614 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.