Problema I. Invertendo Setas

Arquivo-fonte:	`setas.c`, `setas.cpp` ou `setas.java`
Tempo limite:	`2 segundos`
Autor:	`Dâmi Henrique`

Bibi e Bibika estão jogando um jogo simples onde o juiz, a cada rodada, faz um desenho com vários círculos e setas ligando alguns deles.

Bibi deve contar a menor quantidade X de setas que precisam ser invertidas para existir ao menos um caminho de A até B e Bibika deve contar a menor quantidade Y de setas invertidas para existir ao menos um caminho no sentido contrário, de B até A. Ganha o jogo quem encontrar o menor valor. Caso não exista, independente da quantidade de setas invertidas, um caminho entre A–>B ou B–>A, o jogo termina empatado.

Como o juiz em algumas rodadas faz um desenho muito grande, fica bastante complicado checar a veracidade das respostas dadas pelas meninas. Sua tarefa é automatizar esse processo para ele.

Entrada

A primeira linha de cada caso de teste contêm quatro inteiros C, S, A e B, sendo C a quantidade de círculos, S a quantidade de setas, A e B os extremos do jogo. Cada uma das próximas S linhas contêm dois inteiros C1 e C2, representando uma seta ligando o círculo C1 ao círculo C2.

Saída

Para cada caso de teste, exiba o nome da vencedora e a quantidade Q de setas invertidas, no formato Bibi: Q ou Bibika: Q. Caso o jogo termine empatado, exiba Bibibibika.

Limites

$1 \leq C \leq 10^4$

$0 \leq S \leq 5 \times 10^5$

$1 \leq A, B \leq C$

Exemplos

Entrada	Saída
6 7 1 5 1 2 1 6 3 2 4 2 4 6 5 4 5 3	Bibika: 1

Entrada	Saída
3 2 1 2 1 2 2 3	Bibi: 0

Aula de hoje

Nesta aula veremos

- Menor caminho

Busca pelo menor caminho

Definição

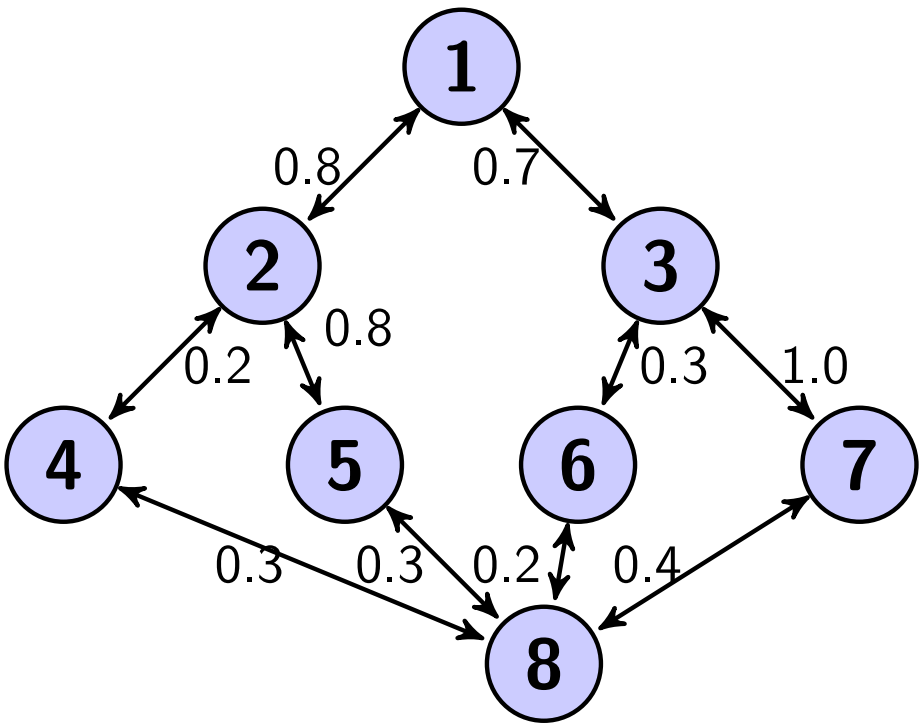
Encontrar um caminho entre dois vértices de um grafo tal que a soma dos pesos das arestas nesse caminho seja minimizada.

Aplicações

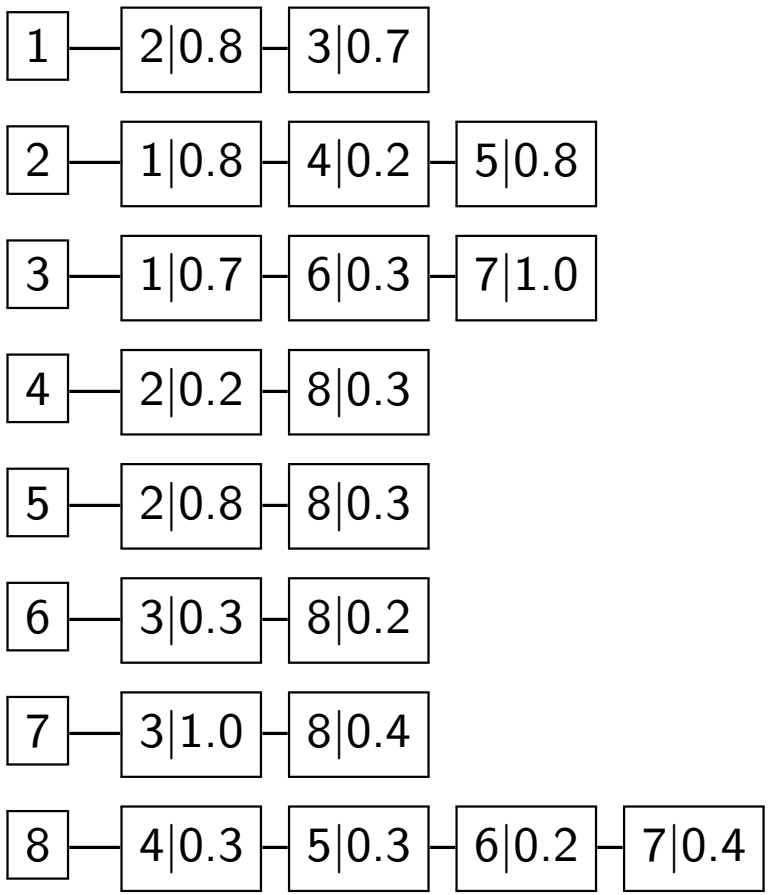
- encontrar menor rota entre duas intersecções de um mapa rodoviário
- caminho de rotas de telefonia para uma ligação com menor atraso possível
- menor número de intermediários para conseguir conversar com uma determinada pessoa

Consideraremos: grafo direcionado com pesos

Exemplos de menor caminho



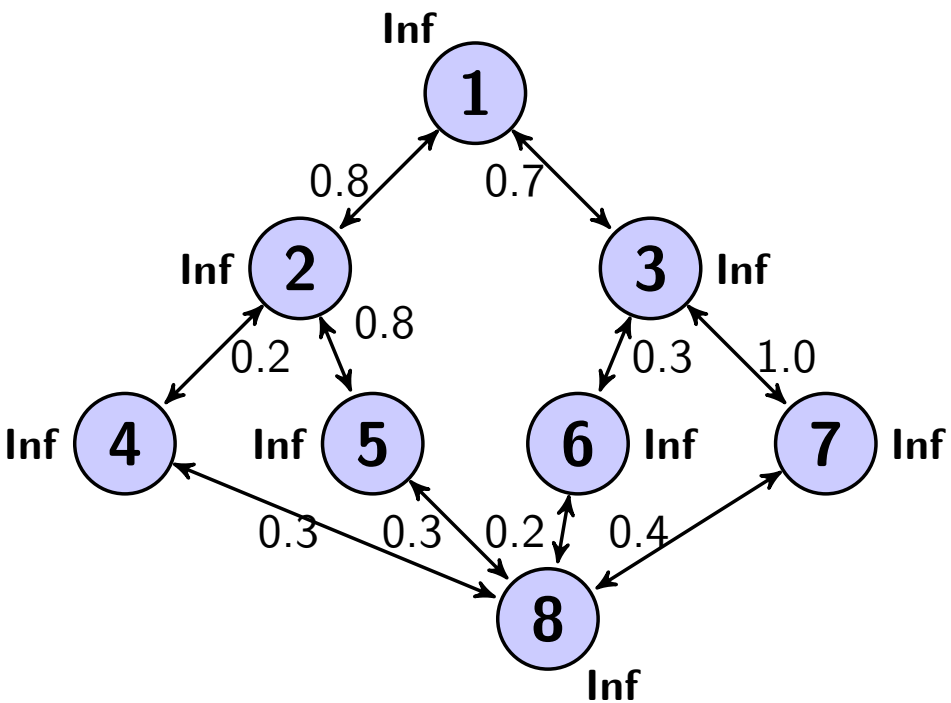
Listas de adjacência



Menor caminho por Edsger Dijkstra

Condições

- arestas com pesos não-negativos
- cada vértice guarda um valor de distância
- distâncias são registradas nos vértices, em relação ao vértice de origem
- distâncias começam infinitas



Algoritmo Djikstra: inicialização

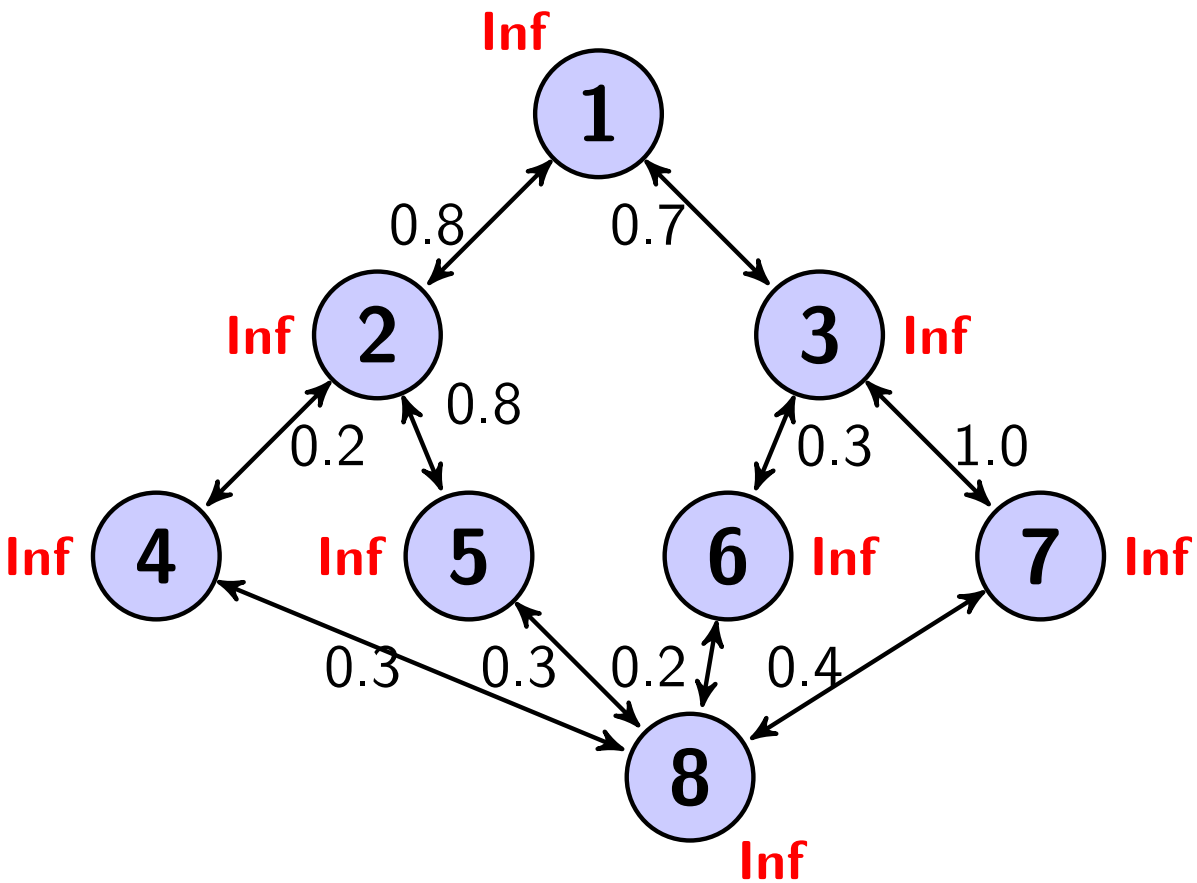
```
1  void iniciaDijkstra(Vertice origem) {
2    for (Vertice v: vertices) {  // Inicialização
3      v.distancia = Double.POSITIVE_INFINITY;
4      v.anterior = null; //vert. anterior
5      if (v.igualA(origem))
6        v.distancia = 0;// distancia da propria origem
7    }
8  }
```

Algoritmo menor caminho Dijkstra

```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);// inicialização
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6    Vertice u = verticeMenorDistancia(verticesAvisitar);
7    verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

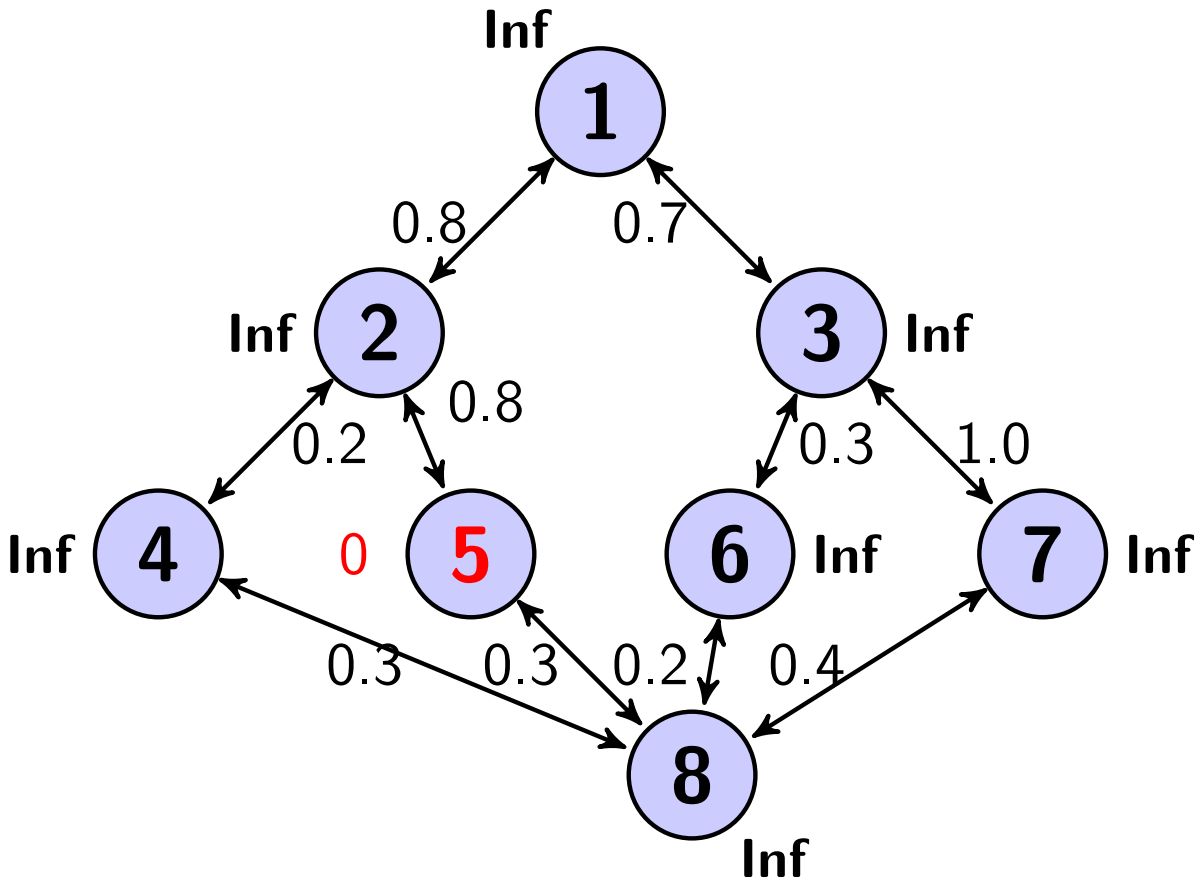
Menor caminho por Edsger Dijkstra

Inicialização: distâncias começam com Infinito

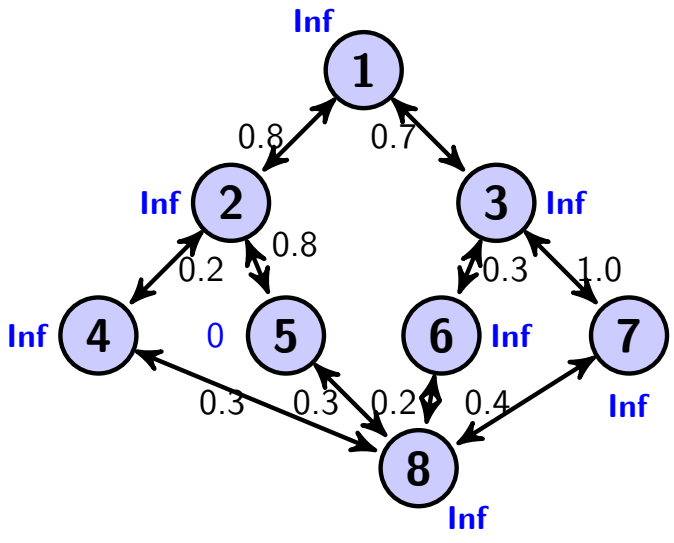


Menor caminho por Edsger Dijkstra

Inicialização: distância da origem é zero. Vértice de origem é o 5.



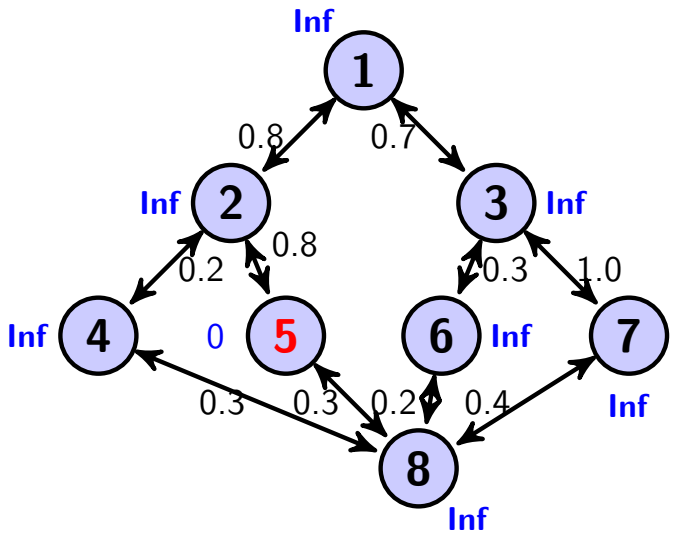
Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6    Vertice u = verticeMenorDistancia(
           verticesAvisitar);
7    verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
           visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Execução.



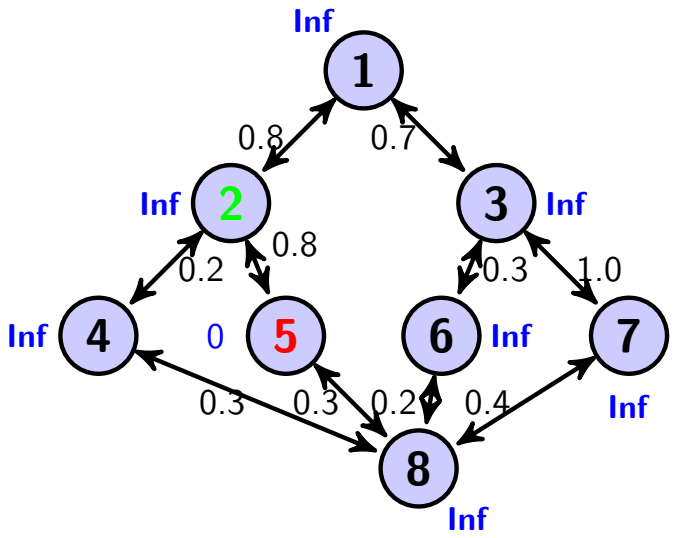

```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 | 8 |

u = | 5 |

u.adjacentes = | 2 | 8 |

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

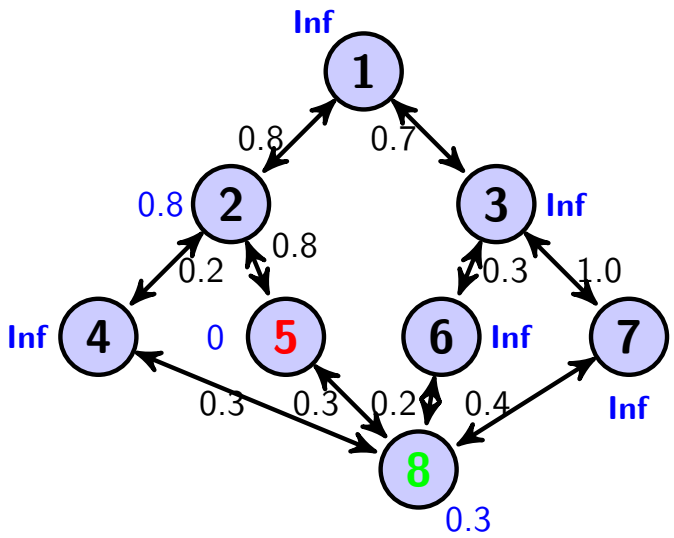
verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 | 8 |

u = | 5 |

u.adjacentes = | 2 | 8 |

adj= | 2 |

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 | 8 |

u = | 5 |

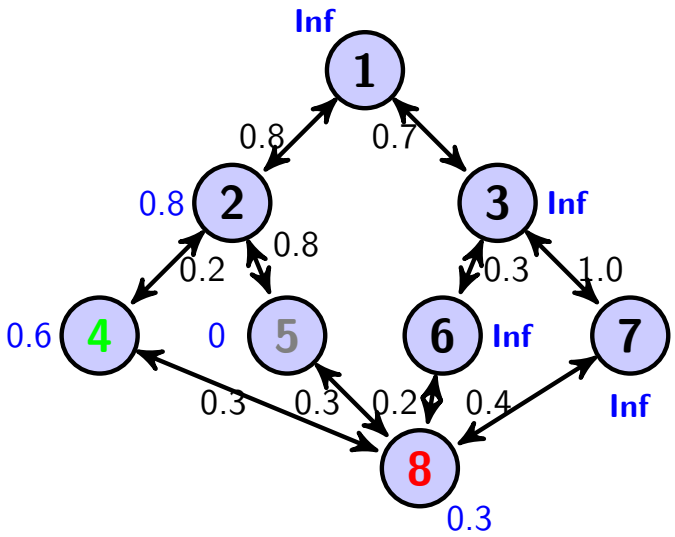
u.adjacentes = | 2 | 8 |

adj= | 8 |

Atualiza distância do vértice 8.

Os adjacentes de | 5 | acabaram, então procurar próximo a visitar.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 |

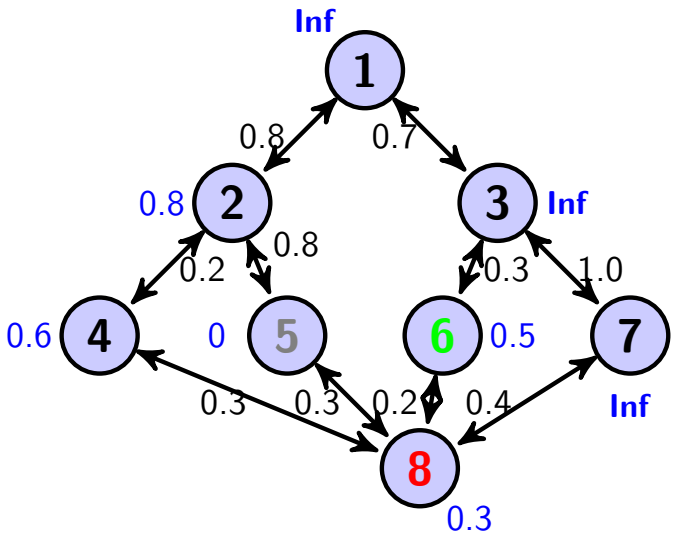
u = | 8 |, pois tem a menor distância até o momento

u.adjacentes = | 4 | 6 | 7 |

adj= | 4 |

Atualiza distância do vértice 4.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 |

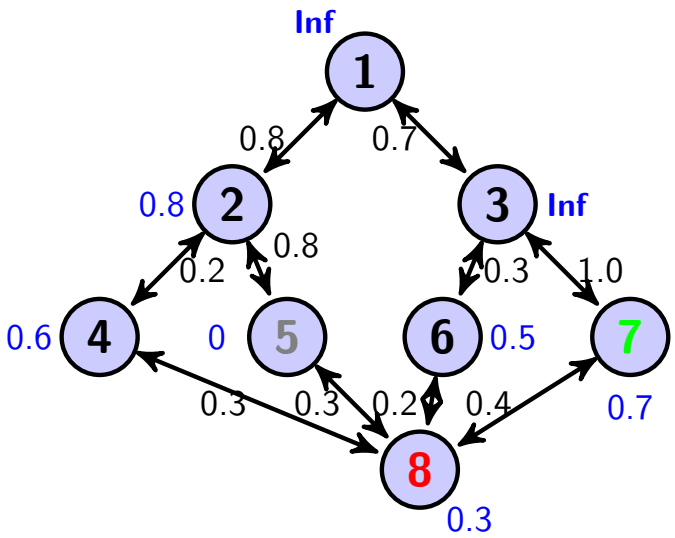
u = | 8 |, pois tem a menor distância até o momento

u.adjacentes = | 4 | 6 | 7 |

adj= | 6 |

Atualiza distância do vértice 6.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 6 | 7 |

u = | 8 |, pois tem a menor distância até o momento

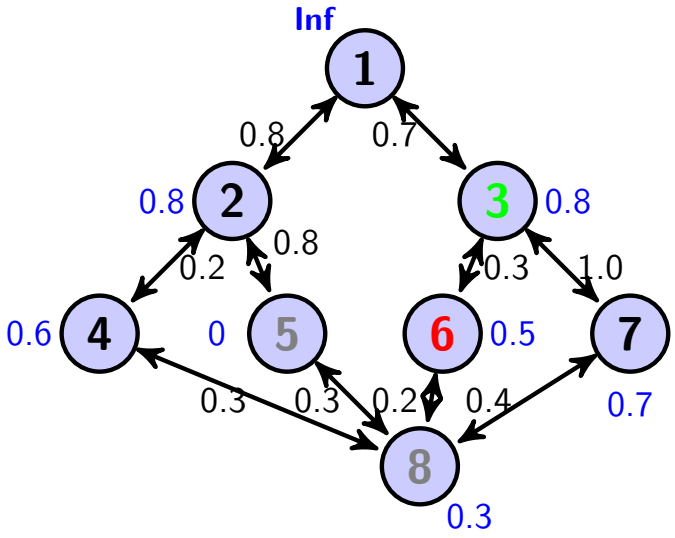
u.adjacentes = | 4 | 6 | 7 |

adj= | 7 |

Atualiza distância do vértice 7.

Os adjacentes acabaram, então procurar próximo a visitar.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 4 | 7 |

u = | 6 |, pois tem a menor distância até o momento 0.6

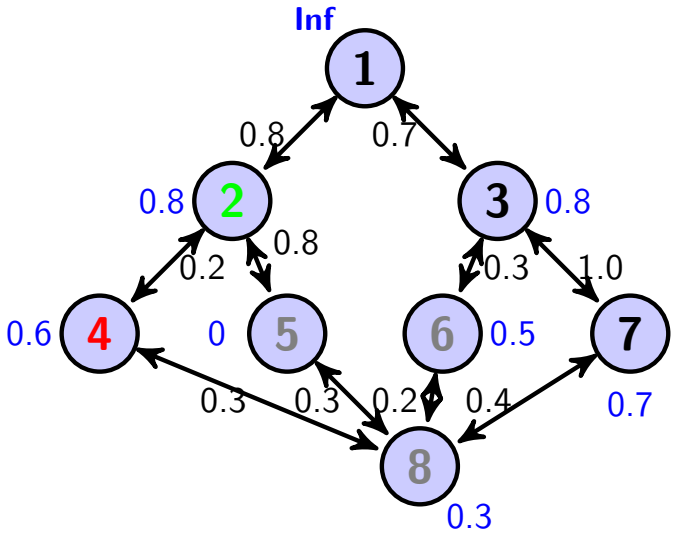
u.adjacentes = | 3 |

adj= | 3 |

Atualiza distância do vértice 3.

Os adjacentes acabaram, então procurar próximo a visitar.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar | 1 | 2 | 3 | 7 |

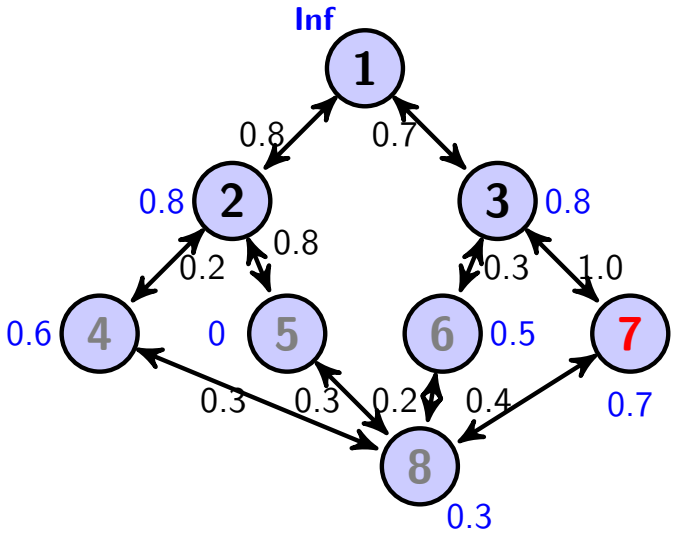
u = | 4 |, pois tem a menor distância até o momento 0.6

u.adjacentes = | 2 |

adj= | 2 |

Não atualiza distância do vértice 2, pois alternativa não é menor.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar [1][2][3]

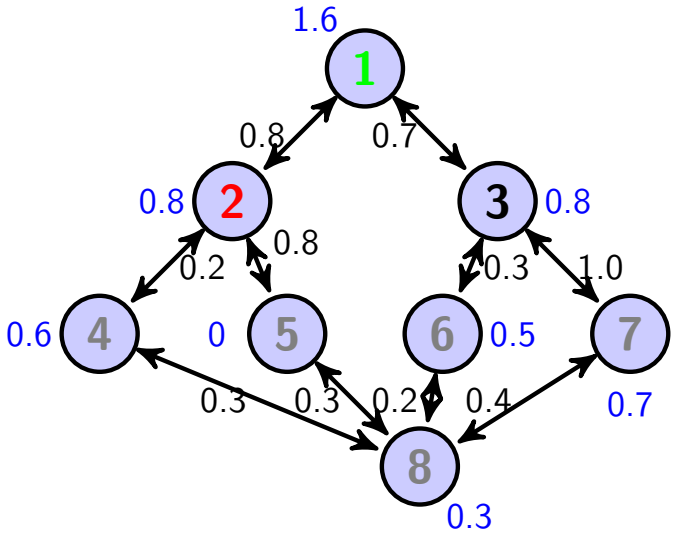
u = [7], pois tem a menor distância até o momento 0.7

u.adjacentes = [3]

adj=[3]

Não atualiza distância do vértice 3, pois a distância alternativa não é menor.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar $\boxed{1}\boxed{3}$

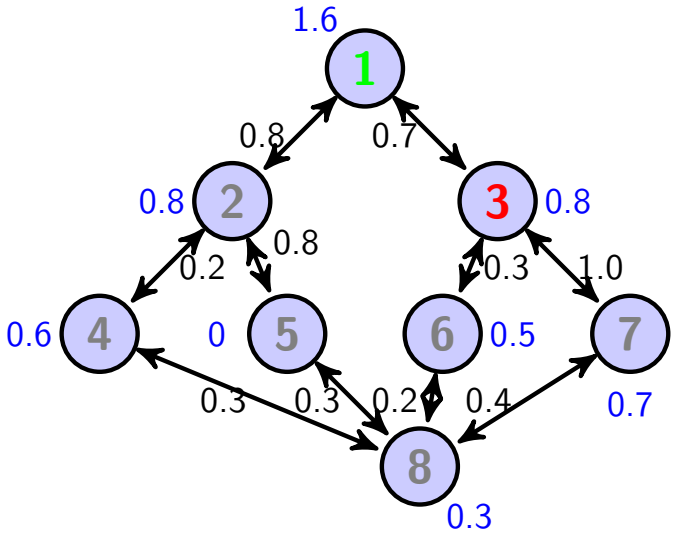
u = $\boxed{2}$, pois tem a menor distância até o momento 0.8

u.adjacentes = $\boxed{1}$

adj=$\boxed{1}$

Atualiza distância do vértice 1 de **Inf** para 1.6.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar $\boxed{1}$

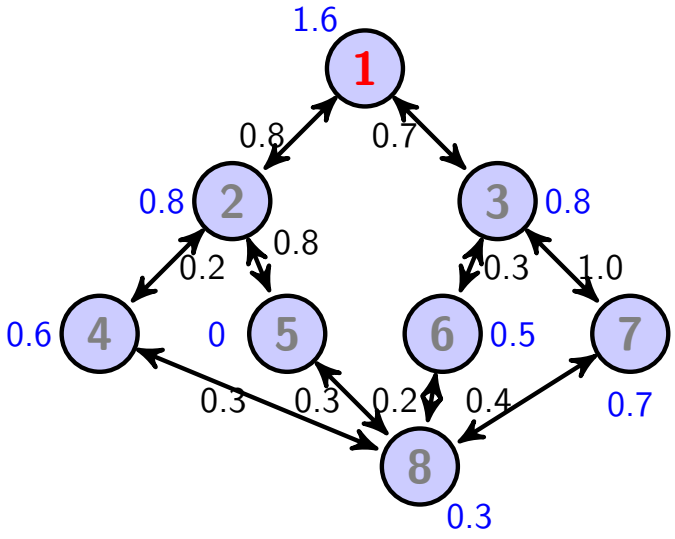
u = $\boxed{3}$, pois tem a menor distância até o momento 0.8

u.adjacentes = $\boxed{1}$

adj=$\boxed{1}$

Atualiza distância do vértice 1 de 1.6 para 1.5.

Execução.




```
1  void dijkstra(Vertice origem){
2   iniciaDijkstra(origem);
3   verticesAvisitar = g.vertices;
4
5   while (verticesAvisitar.naoVazio()) {
6     Vertice u = verticeMenorDistancia(
            verticesAvisitar);
7     verticesAvisitar.remove(u); //e visita
8    if (Double.isInfinite(u.distancia))
9      break; // esta inacessivel
10
11   for (Vertice adj: u.adjacentes) {// nao
          visitados
12     double alternativa = u.distancia + adj.peso;
13     if (alternativa < adj.distancia) {
14       adj.distancia = alternativa;
15       adj.anterior = u;
16     }
17    }
18   }
19 }
```

verticesAvisitar $\boxed{1}$

u = $\boxed{3}$, pois tem a menor distância até o momento 0.8

u.adjacentes = $\square$

Terminou o cálculo das distâncias.

Menor caminho

- usar variáveis `anterior`

```
1  Vector menorCaminho(Vertice origem, Vertice destino) {
2    dijkstra(origem);
3    Vector<Vertice> caminho = new Vector<Vertice>();
4
5    while (destino.igualA(origem)==false) {
6      caminho.add(destino);
7      destino = destino.anterior;
8    }
9    caminho.add(destino);
10
11   return(caminho);
12 }
```