Grafos – Outras Implementações

Prof. Luciano Antonio Digiampietri

Representação utilizada

- Lista de adjacências
  - Por simplificação utilizaremos um arranjo de adjacências
- O grafo corresponderá a um arranjo de nós
- Cada nó possuirá:
  - Um arranjo de vizinhos
  - Um atributo com o número de vizinhos (igual a vizinhos.length)
  - Um atributo binário para indicar se já foi visitado
  - Alguns atributos adicionais que serão utilizados em outras implementações

```
class No{
  int id;
  int numVizinhos;
  No[] vizinhos;
  boolean visitado;
  int cor;
  int distancia;
  No(){}
  No(int pid){
     id = pid;
  }
}

  static No nos[];
  static No[] solucao;
```

Ciclo Hamiltoniano

Verificar se um grafo possui um Ciclo Hamiltoniano, isto é, um ciclo que passa por todos os nós uma única vez.

Implementação recursiva utilizando tentativa e erro (problema de encontrar um solução).

A partir de um nó arbitrário tentaremos montar um ciclo:

- Cada passo corresponderá a um novo nó colocado no ciclo;
- As ações possíveis correspondem a adicionar os nós vizinhos do nó atual;
  - A ação será viável se o nó vizinho ainda não pertencer ao ciclo.

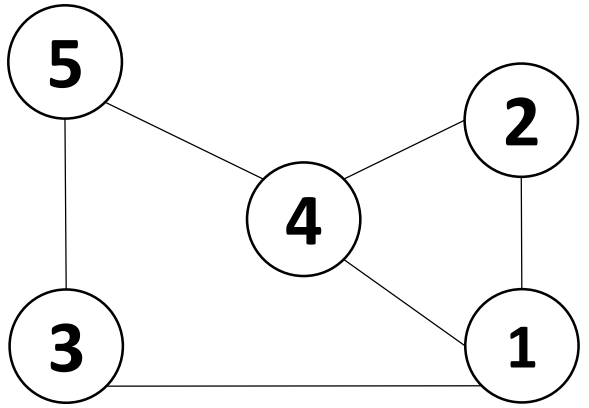
Algoritmo

```
private static boolean cicloHamiltoniano() {
   solucao = new No[nos.length];
   nos[0].visitado = true;
   solucao[0] = nos[0];
   return cicloHamiltonianoAux(1);
}
```

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```

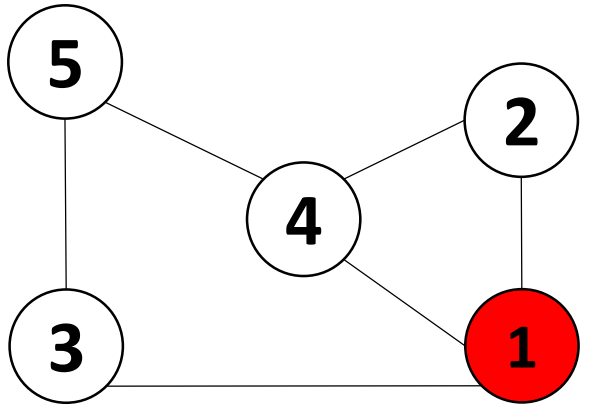
Algoritmo

```
private static boolean cicloHamiltoniano() {
    solucao = new No[nos.length];
    nos[0].visitado = true;
    solucao[0] = nos[0];
    return cicloHamiltonianoAux(1);
}
```



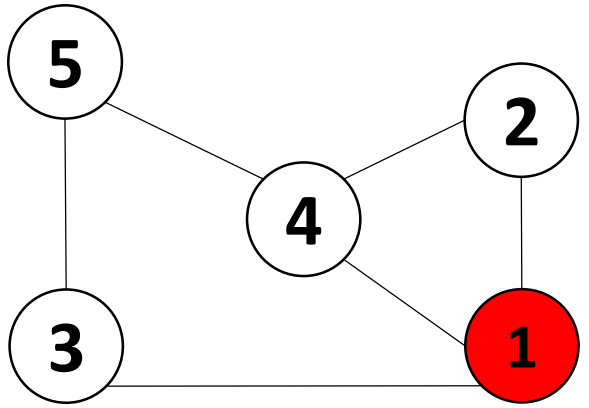
Algoritmo

```
private static boolean cicloHamiltoniano() {
    solucao = new No[nos.length];
    nos[0].visitado = true;
    solucao[0] = nos[0];
    return cicloHamiltonianoAux(1);
}
```



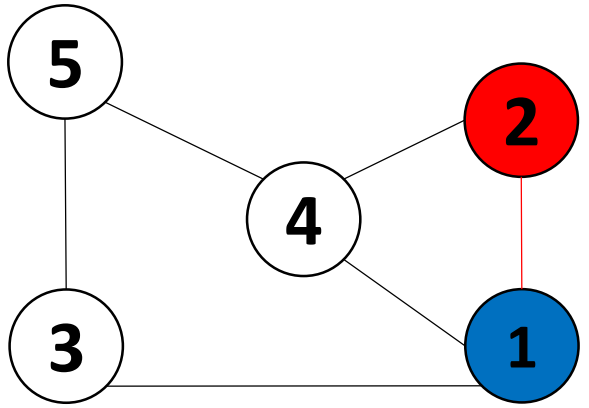
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



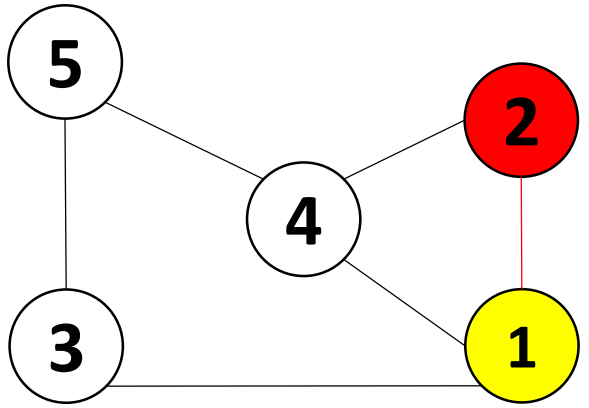
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



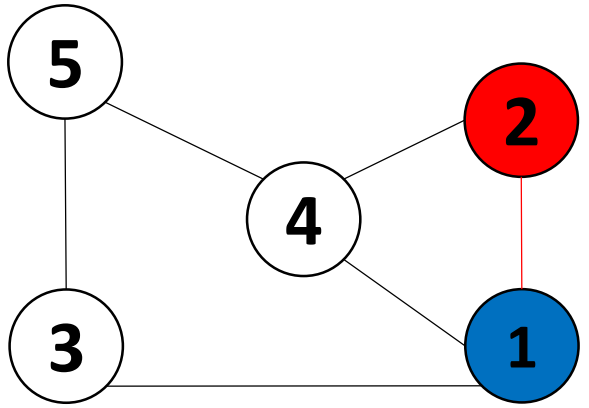
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



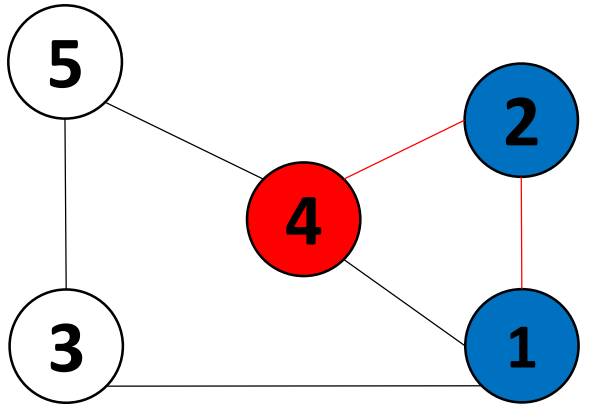
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



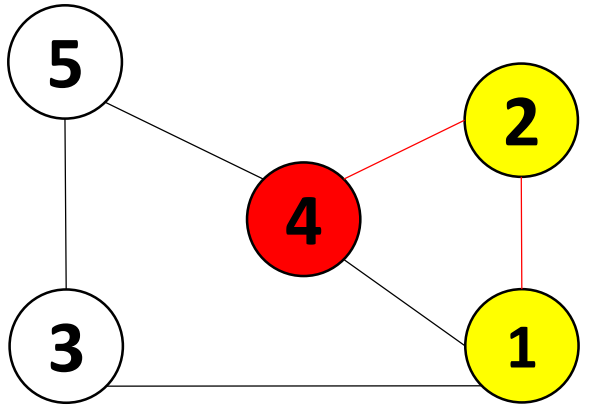
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



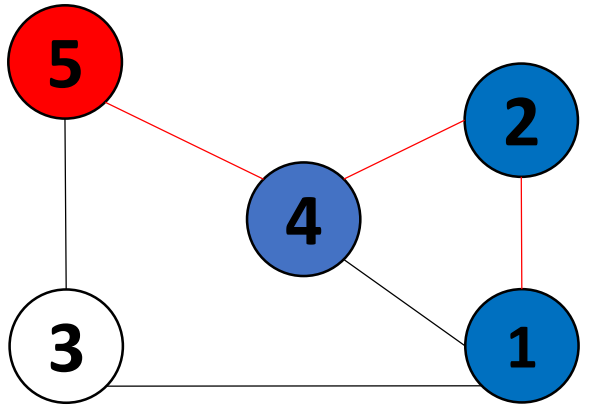
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



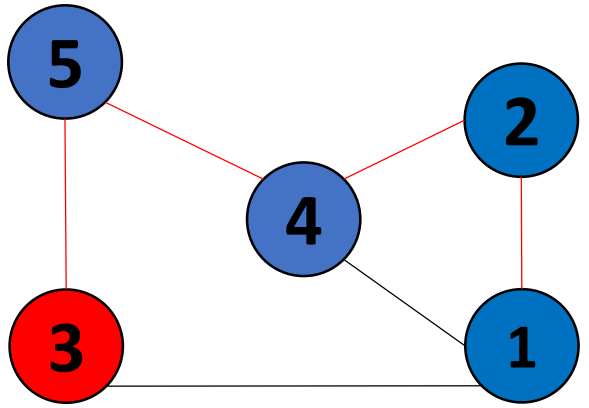
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



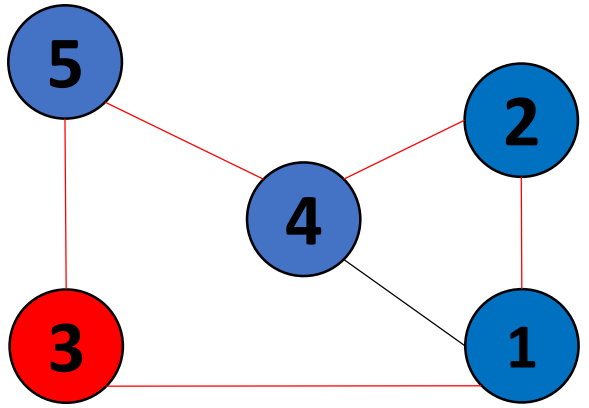
Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



Algoritmo

```
private static boolean cicloHamiltonianoAux(int i) {
   if (i==nos.length){
      for (No t: solucao[i-1].vizinhos)
          if (t == solucao[0]) return true;
      return false;
   }
   for (No t: solucao[i-1].vizinhos){
      if (!t.visitado){
      t.visitado = true;
      solucao[i] = t;
      if (cicloHamiltonianoAux(i+1)) return true;
      t.visitado = false;
      }
   }
   return false;
}
```



Caixeiro Viajante

Verificar, se houver, qual é o Ciclo Hamiltoniano de menor custo no grafo.

Implementação recursiva utilizando tentativa e erro (problema de minimização).

A partir de um dado nó procuraremos pelo ciclo de menor custo:

- Cada passo corresponderá a um novo nó colocado ciclo;
- As ações possíveis correspondem a adicionar os nós vizinhos do nó atual;
  - A ação será viável se o nó vizinho ainda não pertencer ao ciclo.

Representação

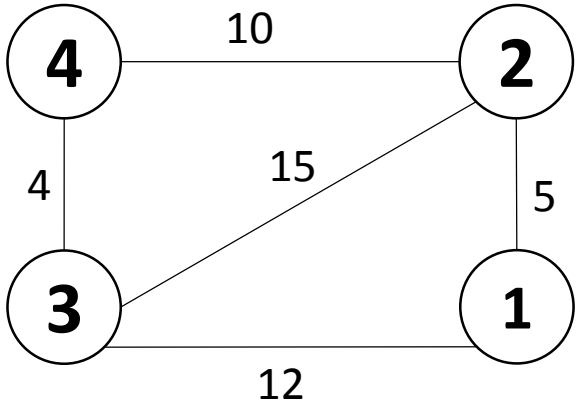
- Utilizaremos para este algoritmo uma matriz de distância entre os nós
  - Uma distância igual a “INFINITO” indicará um par de nós não adjacentes.

Algoritmo

```
private static int caixeiroViajante(int inicial ) {
    tempSolucao = new int[numNos];
    melhorSolucao = new int[numNos];
    valorMelhorSolucao = INFINITO;
    valorSolucaoAtual = 0;
    visitados[inicial] = true;
    tempSolucao[0] = inicial;
    caixeiroViajanteAux(1);
    if (valorMelhorSolucao < INFINITO) return valorMelhorSolucao;
    return -1;
  }
```

```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

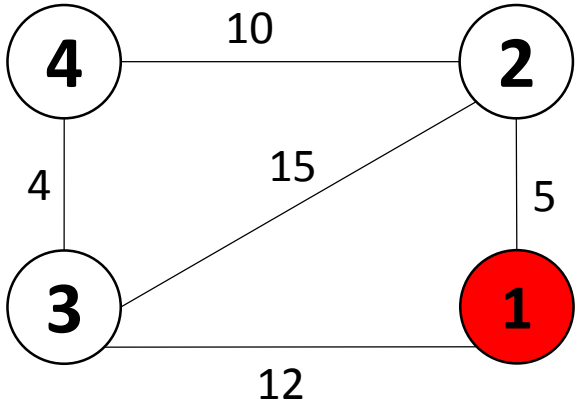


```
private static int caixeiroViajante(int inicial ) {
    tempSolucao = new int[numNos];
    melhorSolucao = new int[numNos];
    valorMelhorSolucao = INFINITO;
    valorSolucaoAtual = 0;
    visitados[inicial] = true;
    tempSolucao[0] = inicial;
    caixeiroViajanteAux(1);
    if (valorMelhorSolucao < INFINITO) return valorMelhorSolucao;
    return -1;
  }
```

Solução atual = 0

Melhor solução = inf

numNos = 0

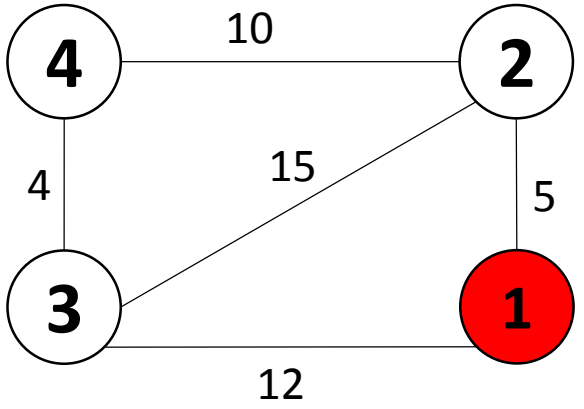


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 0

Melhor solução = inf

numNos = 1

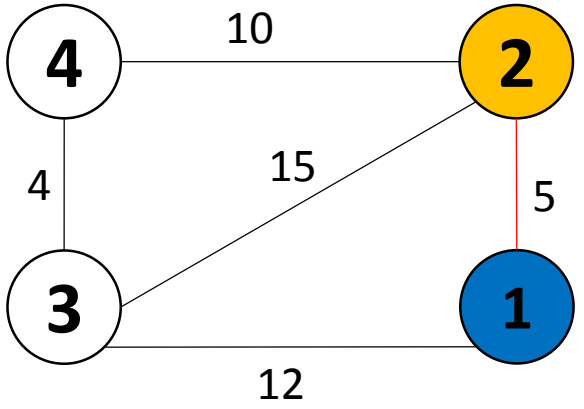


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 0

Melhor solução = inf

numNos = 1

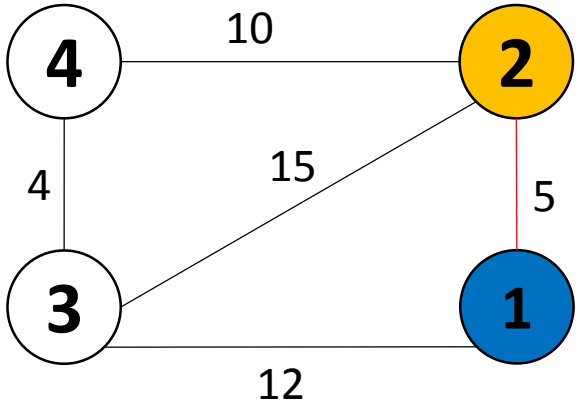


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 0

Melhor solução = inf

numNos = 1

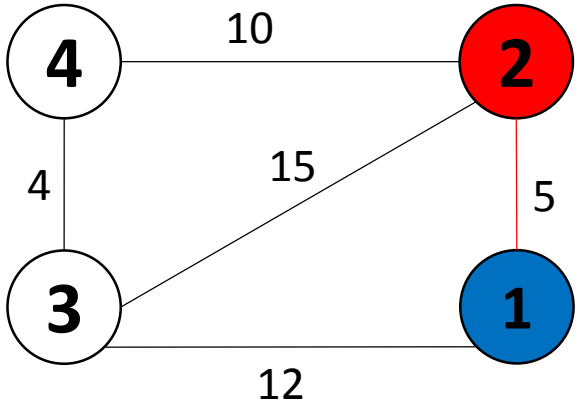


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = inf

numNos = 2

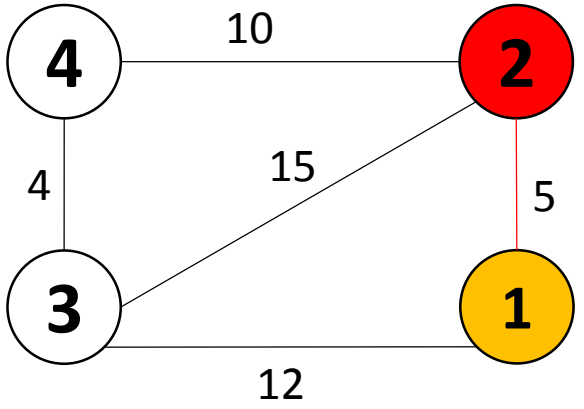


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = inf

numNos = 2

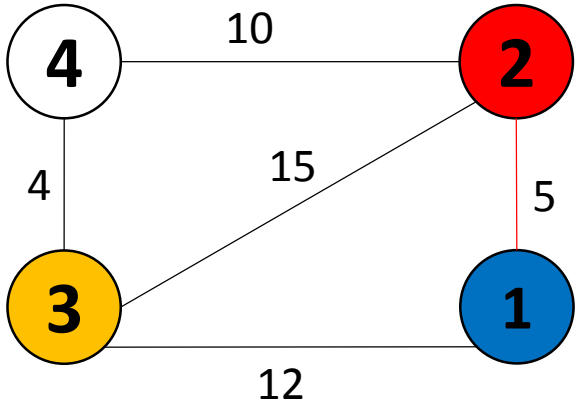


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = inf

numNos = 2

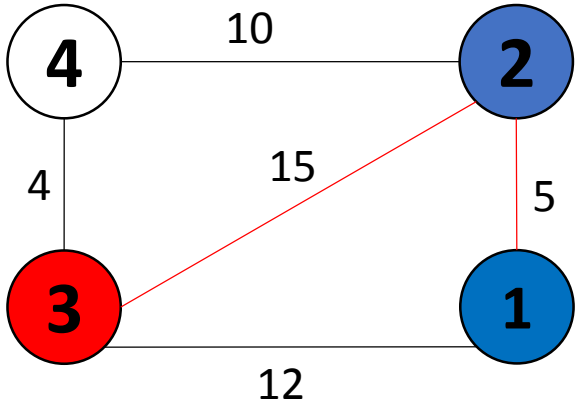


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 20

Melhor solução = inf

numNos = 3

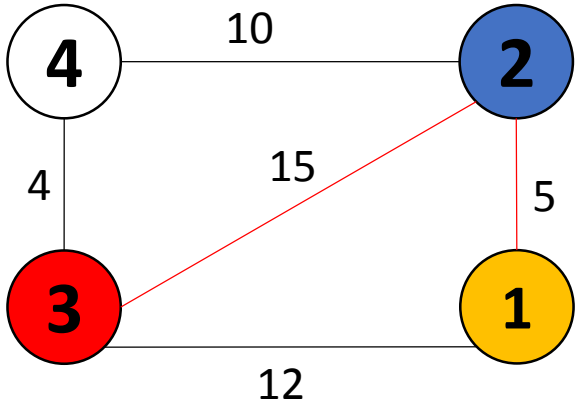


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 20

Melhor solução = inf

numNos = 3

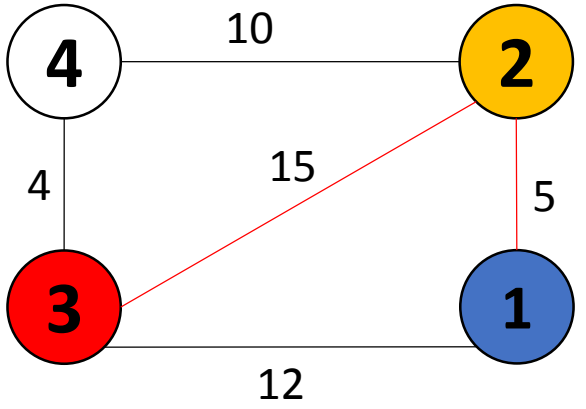


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 20

Melhor solução = inf

numNos = 3

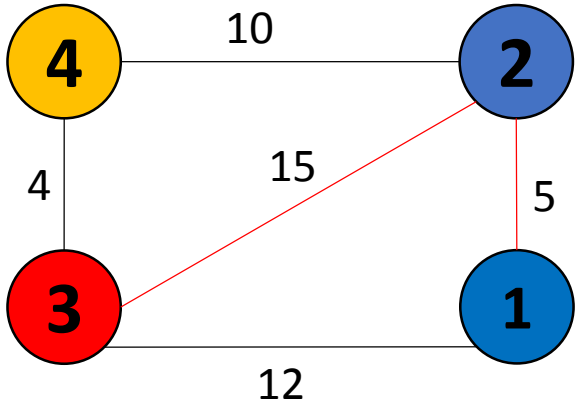


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 20

Melhor solução = inf

numNos = 3

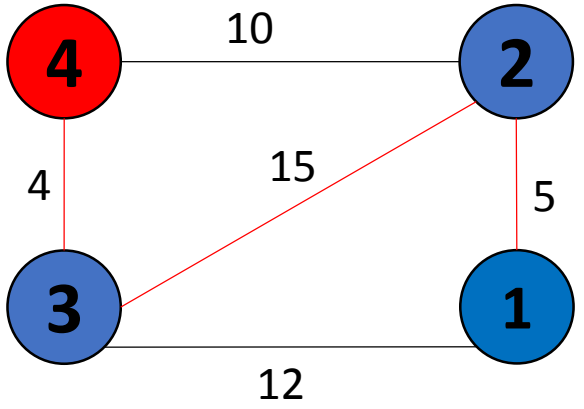


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 24

Melhor solução = inf

numNos = 4

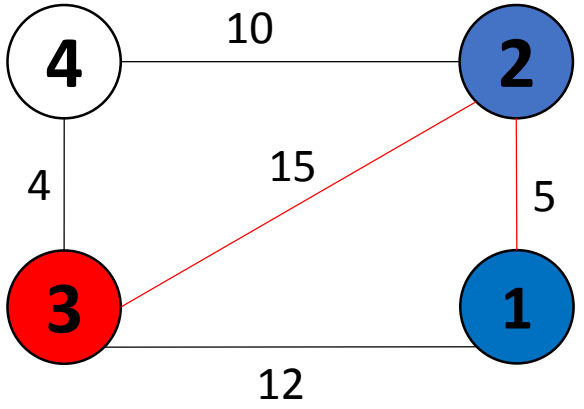


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 20

Melhor solução = inf

numNos = 3

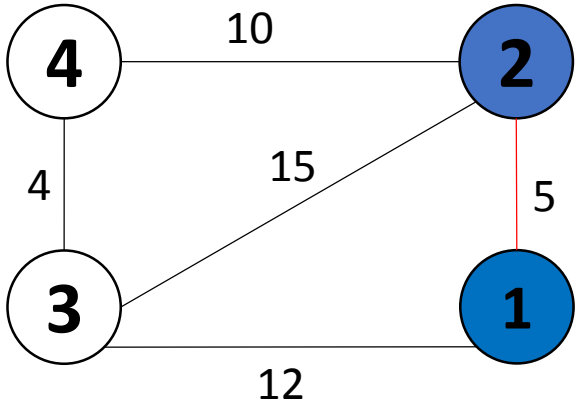


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = inf

numNos = 2

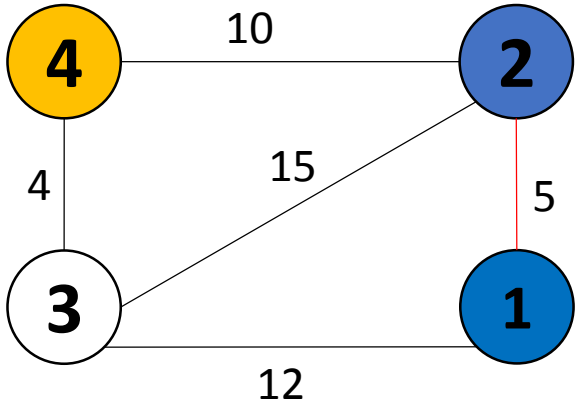


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = inf

numNos = 2

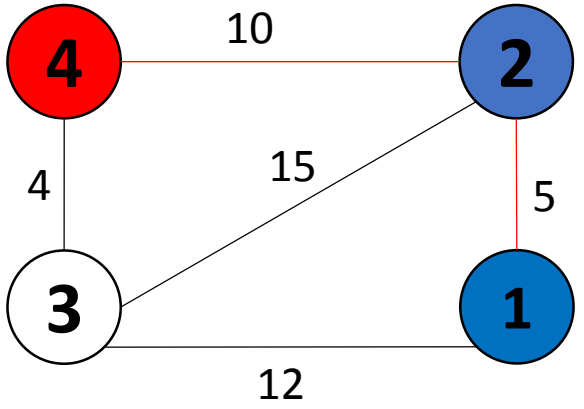


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 15

Melhor solução = inf

numNos = 3

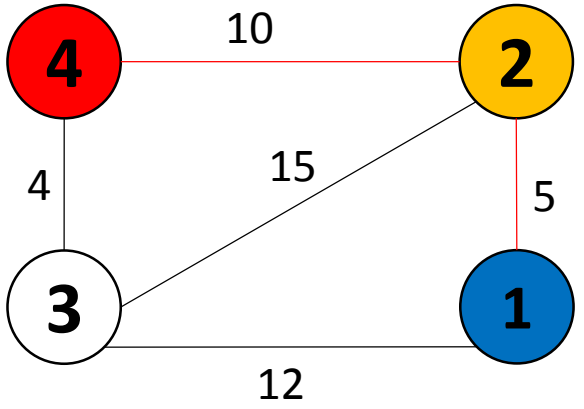


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 15

Melhor solução = inf

numNos = 3

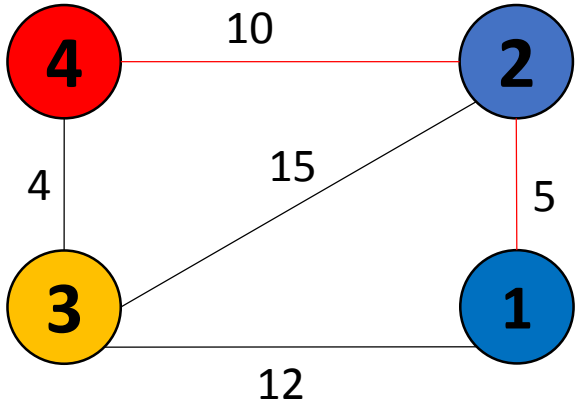


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 15

Melhor solução = inf

numNos = 3

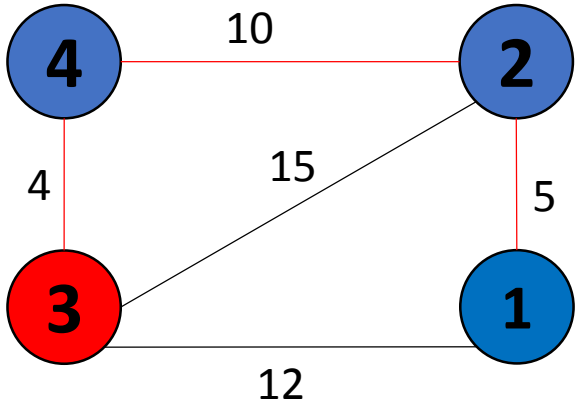


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 19

Melhor solução = inf

numNos = 4

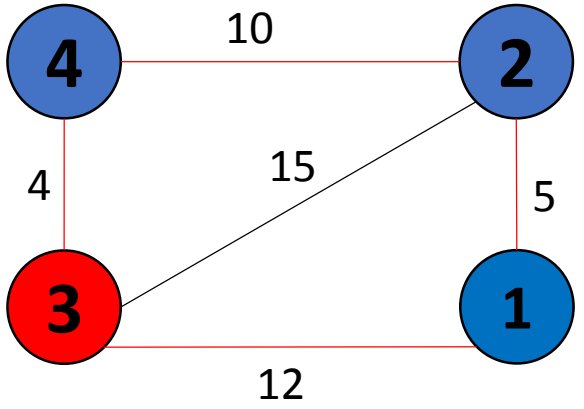


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 31

Melhor solução = inf

numNos = 4

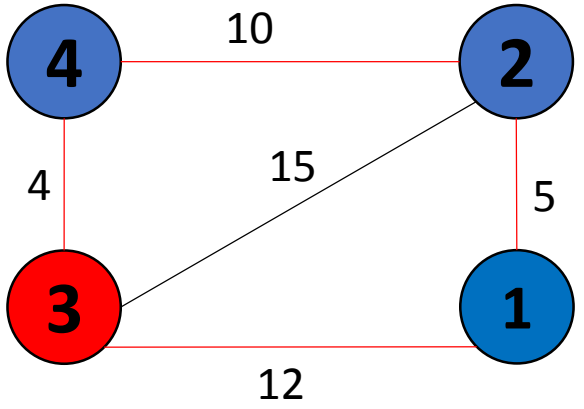


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 31

Melhor solução = 31

numNos = 4

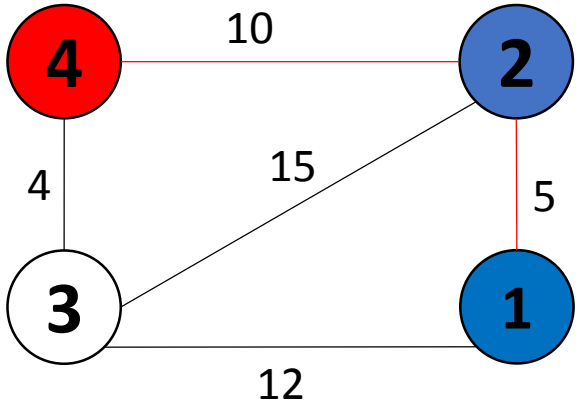


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 15

Melhor solução = 31

numNos = 3

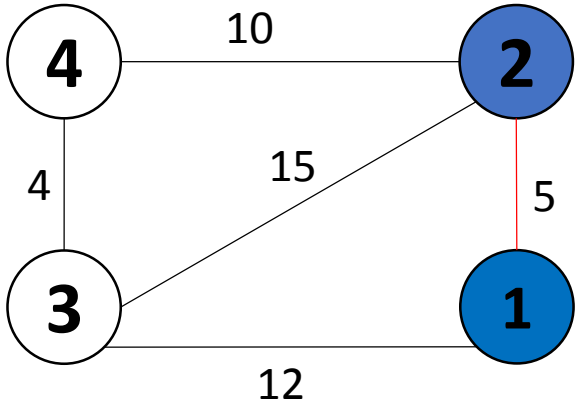


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 5

Melhor solução = 31

numNos = 2

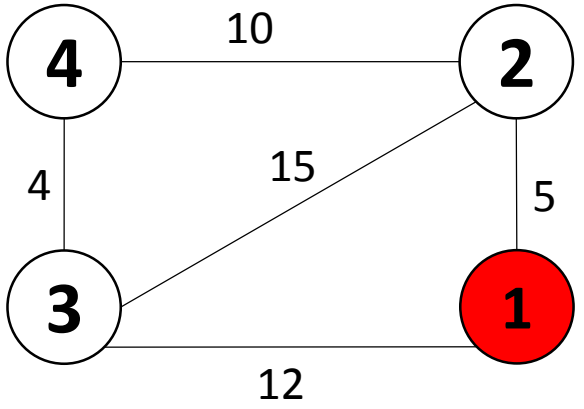


```java
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 0

Melhor solução = 31

numNos = 1

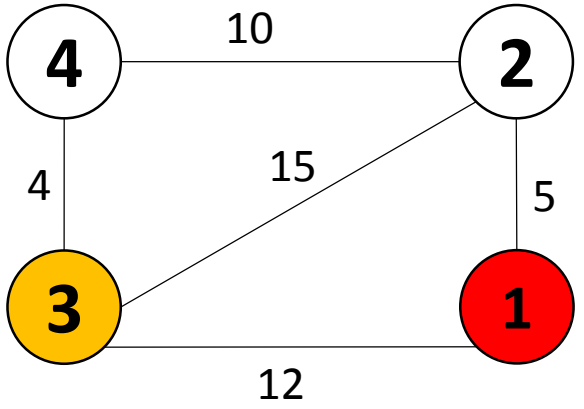


```
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 0

Melhor solução = 31

numNos = 1

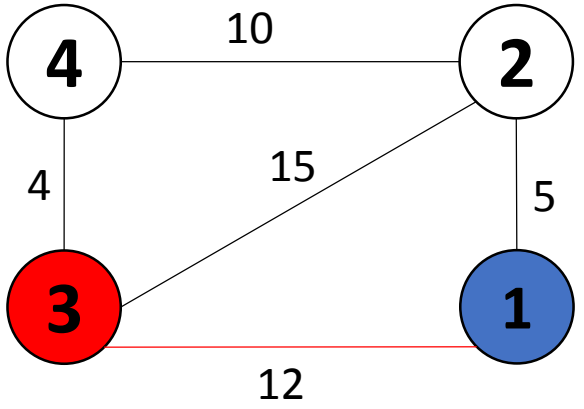


```
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

Solução atual = 12

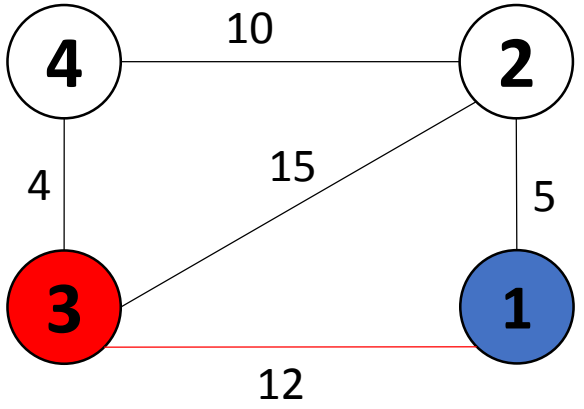
Melhor solução = 31

numNos = 1



```
private static void caixeiroViajanteAux(int i) {
    if (valorSolucaoAtual > valorMelhorSolucao) return;
    if (i==numNos){
      int dist = matrizDistancia[tempSolucao[i-1]][tempSolucao[0]] ;
      if (dist < INFINITO && valorSolucaoAtual + dist < valorMelhorSolucao) {
        valorMelhorSolucao = valorSolucaoAtual + dist;
        melhorSolucao = tempSolucao.clone();
      }
      return;
    }
    int ultimo = tempSolucao[i-1];
    for (int t=0;t<numNos;t++){
      if (!visitados[t] && matrizDistancia[ultimo][t] < INFINITO){
        visitados[t] = true;
        tempSolucao[i] = t;
        valorSolucaoAtual += matrizDistancia[ultimo][t];
        caixeiroViajanteAux(i+1);
        valorSolucaoAtual -= matrizDistancia[ultimo][t];
        visitados[t] = false;
      }
    }
```

A recursão continua procurando todos os demais ciclos (neste caso só há mais um ciclo: 1, 3, 4, 2 com o mesmo custo: 31.



Algoritmos de Distância – grafos ponderados

- Dijsktra
- Bellman–Ford
- Floyd-Warshall

Algoritmos de Distância – grafos ponderados

- Dijsktra: complexidade O(|V|*log|V|+|E|)
  - Distância de um nó para todos
  - Arestas com peso não-negativo

- Bellman–Ford: complexidade O(|V|*|E|)
  - Distância de um nó para todos
  - Arestas podem ter pesos negativos (não pode haver ciclo negativo)

- Floyd-Warshall: complexidade O(|V^3|)
  - Distância de todos para todos

Dijsktra

- Dijsktra: complexidade O(|V|*log|V|+|E|)
  - Distância de um nó para todos
  - Arestas com peso não-negativo

Apresentação em conjunto de slides à parte.

Bellman–Ford

- Bellman–Ford: complexidade O(|V|*|E|)
  - Distância um para todos
  - Arestas podem ter pesos negativos (não pode haver ciclo negativo)

Bellman–Ford

```
function BellmanFord(list vertices, list edges, vertex source)
    ::distance[],predecessor[]
    for each vertex v in vertices:    // INICIALIZAÇÃO
        if v is source then distance[v] := 0
        else distance[v] := inf
        predecessor[v] := null

    for i from 1 to size(vertices)-1: // RELAXAMENTO DAS ARESTAS
        for each edge (u, v) with weight w in edges:
            if distance[u] + w < distance[v]:
                distance[v] := distance[u] + w
                predecessor[v] := u

    for each edge (u, v) with weight w in edges: // VERIFICAÇÃO DE CICLOS NEGATIVOS
        if distance[u] + w < distance[v]:
            error "Graph contains a negative-weight cycle"

    return distance[], predecessor[]
```

Floyd-Warshall

- Floyd-Warshall: complexidade $O(|V^3|)$
  - Distância de todos para todos

Floyd-Warshall

```
ROTINA fw(Inteiro[1..n,1..n] grafo)
    # Inicialização
    VAR Inteiro[1..n,1..n] dist := grafo
    VAR Inteiro[1..n,1..n] pred
    PARA i DE 1 A n
        PARA j DE 1 A n
            SE dist[i,j] < Infinito ENTÃO
                pred[i,j] := i
    # Laço principal do algoritmo
    PARA k DE 1 A n
        PARA i DE 1 A n
            PARA j DE 1 A n
                SE dist[i,j] > dist[i,k] + dist[k,j] ENTÃO
                    dist[i,j] = dist[i,k] + dist[k,j]
                    pred[i,j] = pred[k,j]
    RETORNE dist
```

Grafos – Outras Implementações

Prof. Luciano Antonio Digiampietri