Instituto Politécnico
de Viana do Castelo

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Exemplos de Métodos de Resolução de Problemas Usando Inteligência Artificial

Dikstra e Método Hungaro

- **Orientação: Prof. Doutor Jorge Ribeiro**

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

www.ipvc.pt

Ano Letivo: 2022/2023

Índice

1. Introdução

- *Este trabalho tem como objetivo executar um conjunto de técnicas associadas aos métodos de resolução de problemas e a sua relação com as disciplinas de **Matemática Discreta e Investigação Operacional**.*
- *Neste trabalho prático irá abordar-se algoritmo **Dijkstra** e também o **método Húngaro** e mostrar como foram feitas as nossas implementações dos algoritmos e como funcionam.*
- *Para verificação das soluções decidiu-se resolver também os exercicios á mão e colocar fotografia, assim podemos comparar resultados e verificar se o algoritmo está bem implementado.*
- *Primeiramente, iremos abordar o algoritmo Dijkstra, com implementações em Java e com a resolução de três exercicios de Matemática Discreta e mais tarde a implementação do método Húngaro em PHP com a resolução de um exercicio de Investigação Operacional.*

2. Métodos de Resolução de Problemas

Os métodos de resolução de problemas consistem no uso de vários métodos, de uma forma ordenada, com a finalidade de encontrar soluções para problemas específicos. Várias áreas aplicam estes métodos como é o caso da Inteligência Artificial, que pode utilizar várias técnicas ate chegar ao seu objetivo/resultado final.

Os algoritmos que escolhemos e apresentam-se neste documento associados à resolução de problemas, foram o Húngaro e o Dijkstra.

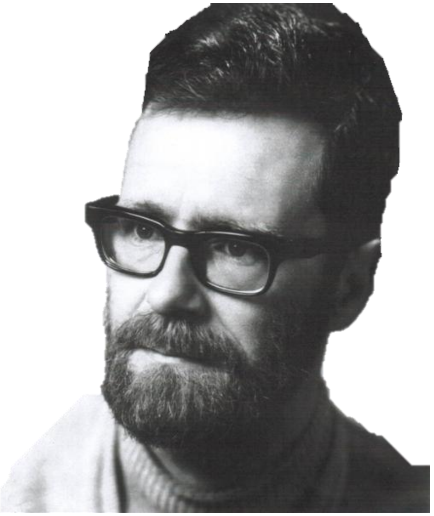
■ 3.1. Dijkstra, o que é?

O Algoritmo de **Dijkstra** é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da procura, este algoritmo calcula o custo mínimo deste vértice para todos os restantes vértices do grafo.

Vamos agora mostrar a implementação do algoritmo nas linguagens de Java e C, mostrando também como funciona. Seguidamente poderá visualizar a resolução de três exercícios e fazer a comparação de resultados.

O que é e quem criou o Algoritmo Dijkstra

O **algoritmo de Dijkstra** foi concebido pelo cientista holandês da computação - ***Edsger Dijkstra***, em 1956 embora só tenha sido publicado em 1959. Este algoritmo soluciona o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de valor não negativo. O algoritmo que serve para resolver o mesmo problema em um grafo com pesos negativos é o algoritmo de Bellman-Ford, que possui maior tempo de execução que o Dijkstra e que vai ser descrito no diapositivo seguinte.



3.3 Dijkstra, Código-Fonte do algoritmo

Vão agora seguir-se alguns prints do código fonte do algoritmo, em que se podem ver as várias classes criadas e também a main onde implementámos um determinado grafo.

```java
public class Graph {
    private Node[] vertices; // stores the nodes of the graph
    private int size; // number of nodes in the graph
    private MinPriorityQueue queue;

    public Graph(int size) {
        this.size = size;
        vertices = new Node[size];
        addNodes();
        queue = new MinPriorityQueue(size);
    }

    public class Node {
        int name;
        int cost;
        Neighbour neighbourList;
        State state;

        Node(int name) {
            this.name = name;
            state = State.NEW;
            cost = Integer.MAX_VALUE;
        }
    }
```

3.3 Dijkstra, Código-Fonte do algoritmo


```java
public class Neighbour {
    int index;
    int weight;
    Neighbour next;

    public Neighbour(int index, Neighbour next, int weight) {
        this.index = index;
        this.next = next;
        this.weight = weight;
    }
}

private void addNodes() {
    for (int i = 1; i <= size; i++) {
        addNode(i);
    }
}

public void addNode(int name) {
    vertices[name - 1] = new Node(name);
}

public void addEdge(int sourceName, int destiName, int weight) {
    int srcIndex = sourceName - 1;
    int destiIndex = destiName - 1;
    Node srcNode = vertices[srcIndex];
    Node destiNode = vertices[destiIndex];
    srcNode.neighbourList = new Neighbour(destiIndex, srcNode.neighbourList, weight);
    // the graph is non directional so if from S, D is reachable then vice
    // versa is also true
    destiNode.neighbourList = new Neighbour(srcIndex, destiNode.neighbourList, weight);
}
```

3.3 Dijkstra, Código-Fonte do algoritmo

```java
private void applyDijkstraAlgorithm(Node sourceNode) {
    queue.add(sourceNode);
    sourceNode.state = State.IN_Q;
    sourceNode.cost = 0; // cost of reaching Source from Source Node itself
                         // is 0, for all others we still need to
                         // discover the cost so the cost for them has
                         // been already initialized to Integer.MAX_VALUE
    while (!queue.isEmpty()) {
        Node visitedNode = queue.remove();
        visitedNode.state = State.VISITED;
        Neighbour connectedEdge = visitedNode.neighbourList;
        while (connectedEdge != null) {
            Node neighbour = vertices[connectedEdge.index];
            // adding the not enqued neighbor nodes in the queue
            if (neighbour.state == State.NEW) {
                queue.add(neighbour);
                neighbour.state = State.IN_Q;
            }
            // updating [relaxing] the costs of each non visited neighbor
            // node if its
            // have been made lesser.
            if (neighbour.state != State.VISITED && ((connectedEdge.weight + visitedNode.cost) < neighbour.cost)) {
                neighbour.cost = connectedEdge.weight + visitedNode.cost;
            }
            connectedEdge = connectedEdge.next;
        }
    }
```

3.3 Dijkstra, Código-Fonte do algoritmo

```java
public void computeSortestPathsFrom(int sourceNodeName) {
    for (int i = 0; i < size; i++) {
        if (vertices[i].name == sourceNodeName) {
            applyDijkstraAlgorithm(vertices[i]);
            break;// in this case we need not traverse the nodes which are
                    // not reachable from the source Node
        }
    }
}

public enum State {
    NEW, IN_Q, VISITED
};

public class MinPriorityQueue {
    Node[] queue;
    int maxSize;
    int rear = -1, front = -1;

    MinPriorityQueue(int maxSize) {
        this.maxSize = maxSize;
        queue = new Node[maxSize];
    }

    public void add(Node node) {
        queue[++rear] = node;
    }
```

3.3 Dijkstra, Código-Fonte do algoritmo

```java
public Node remove() {
    Node minValuedNode = null;
    int minValue = Integer.MAX_VALUE;
    int minValueIndex = -1;
    front++;
    for (int i = front; i <= rear; i++) {
        if (queue[i].state == State.IN_Q && queue[i].cost < minValue) {
            minValue = queue[i].cost;
            minValuedNode = queue[i];
            minValueIndex = i;
        }
    }

    swap(front, minValueIndex); // this ensures deletion is still done from front
    queue[front] = null;// lets not hold up unnecessary references in the queue
    return minValuedNode;
}

public void swap(int index1, int index2) {
    Node temp = queue[index1];
    queue[index1] = queue[index2];
    queue[index2] = temp;
}

public boolean isEmpty() {
    return front == rear;
}
```

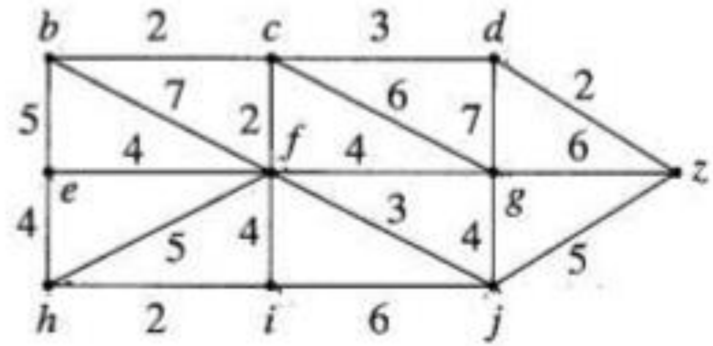
3.3 Dijkstra, Código-Fonte do algoritmo

```
public static void main(String[] args) {
    Graph graph = new Graph(11);
    graph.addEdge(1, 2, 1);
    graph.addEdge(2, 3, 2);
    graph.addEdge(3, 4, 3);
    graph.addEdge(1, 5, 4);
    graph.addEdge(2, 5, 3);
    graph.addEdge(1, 6, 2);
    graph.addEdge(5, 6, 3);
    graph.addEdge(3, 5, 2);
    graph.addEdge(3, 7, 5);
    graph.addEdge(6, 7, 2);
    graph.addEdge(4, 7, 4);
    graph.addEdge(4, 8, 3);
    graph.addEdge(7, 8, 4);
    graph.computeSortestPathsFrom(3);
}
```

3. Dijkstra, Resolução de problemas

Exercício 1 - Enunciado

1) Recorrendo ao algoritmo de Dijkstra, determine o comprimento e o caminho mais curto entre os vértices **b** e **z**.



Exercício 1 – Resolução manual

INTELIGÊNCIA ARTIFICIAL (Ex 1, FICHA 3 MD)

b	c	d	e	f	g	h	i	j	z	temporário
(0, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	{c, d, e, f, g, h, j, i, z}
	(2, b)	(∞, -)	(5, b)	(7, b)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	{d, e, f, g, h, i, j, z}
		(5, c)	(5, b)	(4, c)	(8, c)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	{d, e, g, h, i, j, z}
		(5, c)	(5, b)		(8, c)	(9, f)	(8, f)	(7, f)	(∞, -)	{e, g, h, i, i, z}
			(5, b)		(8, c)	(9, f)	(8, f)	(7, f)	(7, d)	{e, g, h, i, j

O menor caminho de b para z tem comprimento 7

3. Dijkstra, Resolução de problemas

Exercício 1 – Resolução através do algoritmo

Função main

```java
public static void main(String[] args) {
    Graph graph = new Graph(10);
    graph.addEdge(1, 2, 2);// B vai para c com peso 2
    graph.addEdge(1, 5, 7);// B vai para f com peso 7
    graph.addEdge(1, 4, 2);// B vai para e com peso 5
    graph.addEdge(2, 3, 3);// C vai para d com peso 3
    graph.addEdge(2, 5, 2);// C vai para f com peso 2
    graph.addEdge(2, 6, 5);// C vai para g com peso 6
    graph.addEdge(2, 1, 2);// C vai para b com peso 2
    graph.addEdge(3, 2, 2);// D vai para c com peso 3
    graph.addEdge(3, 6, 2);// D vai para g com peso 7
    graph.addEdge(3, 10, 2);// D vai para z com peso 2
    graph.addEdge(4, 1, 5);// E vai para b com peso 5
    graph.addEdge(4, 7, 4);// E vai para h com peso 4
    graph.addEdge(4, 5, 4);// E vai para f com peso 4
    graph.addEdge(5, 1, 7);// F vai para b com peso 7
    graph.addEdge(5, 2, 2);// F vai para c com peso 2
    graph.addEdge(5, 6, 4);// F vai para g com peso 4
    graph.addEdge(5, 9, 3);// F vai para j com peso 3
    graph.addEdge(5, 8, 4);// F vai para i com peso 4
    graph.addEdge(5, 7, 5);// F vai para h com peso 5
    graph.addEdge(5, 4, 4);// F vai para e com peso 4
    graph.addEdge(6, 3, 7);// G vai para d com peso 7
    graph.addEdge(6, 9, 4);// G vai para j com peso 4
    graph.addEdge(6, 5, 4);// G vai para f com peso 4
    graph.addEdge(6, 2, 6);// G vai para c com peso 6
    graph.addEdge(6, 10, 6);// G vai para z com peso 6
    graph.addEdge(7, 4, 4);// H vai para e com peso 4
    graph.addEdge(7, 5, 5);// H vai para f com peso 5
    graph.addEdge(7, 8, 2);// H vai para i com peso 2
    graph.addEdge(8, 8, 2);// I vai para h com peso 2
    graph.addEdge(8, 5, 4);// I vai para f com peso 4
    graph.addEdge(8, 9, 6);// I vai para j com peso 6
    graph.addEdge(9, 5, 3);// J vai para f com peso 3
    graph.addEdge(9, 8, 6);// J vai para i com peso 6
    graph.addEdge(9, 6, 4);// J vai para g com peso 4
    graph.addEdge(9, 10, 5);// J vai para z com peso 5
    graph.addEdge(10, 3, 2);// Z vai para d com peso 2
    graph.addEdge(10, 6, 6);// Z vai para g com peso 6
    graph.addEdge(10, 9, 5);// Z vai para j com peso 5
```

■ 3. Dijkstra, Resolução de problemas

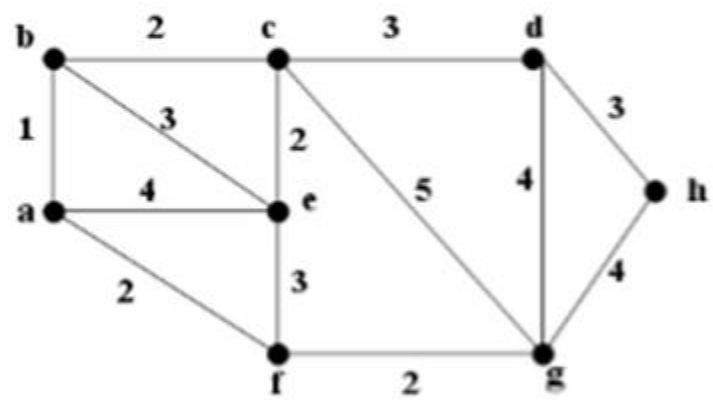
Exercício 1 – Resolução através do algoritmo

Output

```
run:
distance from 1 to 1 is 0
distance from 1 to 2 is 2
distance from 1 to 3 is 5
distance from 1 to 4 is 5
distance from 1 to 5 is 4
distance from 1 to 6 is 8
distance from 1 to 7 is 9
distance from 1 to 8 is 8
distance from 1 to 9 is 7
distance from 1 to 10 is 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

Exercício 2 - Enunciado

2) Recorrendo ao algoritmo de Dijkstra, determine o comprimento e o caminho mais curto entre os vértices **a** e **h**



3. Dijkstra, Resolução de problemas

Exercício 2 – Resolução manual

INTELIGÊNCIA ARTIFICIAL (Ex 2 Ficha 3)

a	b	c	d	e	f	g	h	temporário
(∞,-)	(0,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	{a,c,d,e,f,g,h}
(1,b)		(2,b)	(∞,-)	(3,b)	(∞,-)	(∞,-)	(∞,-)	{c,d,e,f,g,h}
		(2,b)	(∞,-)	(3,b)	(3,a)	(7,c)	(∞,-)	{d,e,f,g,h}
			(5,c)	(3,b)	(3,a)	(7,d)	(∞,-)	{d,f,g,h}
			(5,c)		(3,a)	(7,c)	(∞,-)	{d,g,h}
			(5,c)			(5,f)	(8,d)	{g,h}
						(5,f)	(8,d)	{h}
							(8,d)	

O menor caminho de a para h tem comprimento 8

3. Dijkstra, Resolução de problemas

Exercício 2 – Resolução através do algoritmo

Função main

```
public static void main(String[] args) {
    Graph graph = new Graph(8);
    //1-a ,2-b ,3-c ,4-d, 5-e ,6-f , 7-g , 8-h
    graph.addEdge(1, 2, 1);// A vai para b com peso 1
    graph.addEdge(1, 5, 4);// A vai para e com peso 4
    graph.addEdge(1, 6, 2);// A vai para f com peso 2
    graph.addEdge(2, 1, 1);// B vai para a com peso 1
    graph.addEdge(2, 3, 2);// B vai para c com peso 2
    graph.addEdge(2, 5, 2);// B vai para e com peso 3
    graph.addEdge(3, 1, 2);// C vai para b com peso 2
    graph.addEdge(3, 5, 2);// C vai para e com peso 2
    graph.addEdge(3, 4, 3);// C vai para d com peso 3
    graph.addEdge(3, 7, 5);// C vai para g com peso 5
    graph.addEdge(4, 3, 3);// D vai para c com peso 3
```

```
graph.addEdge(4, 7, 4);// D vai para g com peso 4
graph.addEdge(4, 8, 3);// D vai para h com peso 3
graph.addEdge(5, 1, 4);// E vai para a com peso 4
graph.addEdge(5, 2, 3);// E vai para b com peso 3
graph.addEdge(5, 3, 2);// E vai para c com peso 2
graph.addEdge(5, 6, 3);// E vai para f com peso 3
graph.addEdge(6, 1, 2);// F vai para a com peso 2
graph.addEdge(6, 5, 3);// F vai para e com peso 3
graph.addEdge(6, 7, 2);// F vai para g com peso 2
graph.addEdge(7, 3, 4);// G vai para c com peso 5
graph.addEdge(7, 4, 4);// G vai para d com peso 4
graph.addEdge(7, 6, 2);// G vai para f com peso 2
graph.addEdge(7, 8, 4);// G vai para h com peso 4
graph.addEdge(8, 4, 3);// H vai para d com peso 3
graph.addEdge(8, 7, 4);// H vai para g com peso 4
```

3. Dijkstra, Resolução de problemas

Exercício 2 – Resolução através do algoritmo

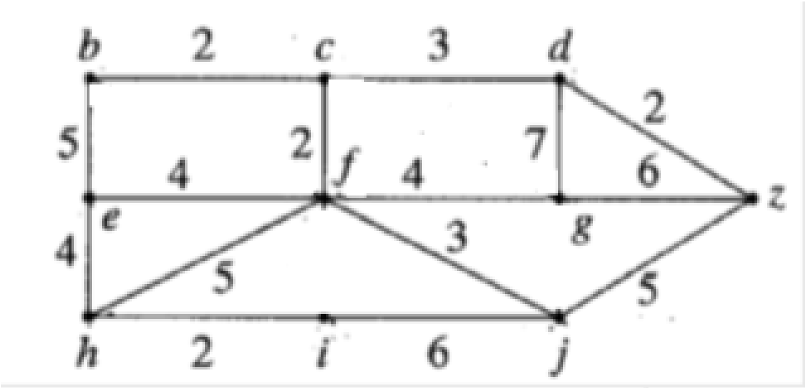
Output

```
run:
distance from 1 to 1 is 0
distance from 1 to 2 is 1
distance from 1 to 3 is 3
distance from 1 to 4 is 6
distance from 1 to 5 is 4
distance from 1 to 6 is 2
distance from 1 to 7 is 4
distance from 1 to 8 is 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Dijkstra, Resolução de problemas

Exercício 3 – Enunciado

Recorrendo ao algoritmo de Dijkstra, determine o comprimento do caminho mais curto, assim como o respectivo caminho do vértice **h** ao vértice **z**.



3. Dijkstra, Resolução de problemas

Exercício 3 – Resolução manual

INTELIGÊNCIA ARTIFICIAL (Ex5 Prova MD 02/06/2014)

b	c	d	e	f	g	h	i	j	z
(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(∞,-)	(0,-)	(∞,-)	(∞,-)	(∞,-)
(∞,-)	(∞,-)	(∞,-)	(4,h)	(5,h)	(∞,-)		(2,h)	(∞,-)	(∞,-)
(∞,-)	(∞,-)	(∞,-)	(4,h)	(5,h)	(∞,-)			(8,i)	(∞,-)
(9,e)	(∞,-)	(∞,-)		(5,h)	(∞,-)			(8,i)	(∞,-)
(9,e)	(7,f)	(∞,-)			(∞,-)			(8,i)	(∞,-)
(9,e)		(10,c)			(9,f)			(8,i)	(∞,-)
(9,e)		(10,c)			(9,f)				(13,j)
		(10,c)			(9,f)				(13,j)
		(10,c)							(12,d)
									(12,d)

O menor caminho de h para z tem comprimento 12

3. Dijkstra, Resolução de problemas

Exercício 3 – Resolução através do algoritmo

```
public static void main(String[] args) {
    Graph graph = new Graph(10);
    graph.addEdge(1, 2, 2);
    graph.addEdge(1, 4, 5);
    graph.addEdge(2, 3, 3);
    graph.addEdge(2, 5, 2);
    graph.addEdge(3, 6, 7);
    graph.addEdge(3, 10, 2);
    graph.addEdge(4, 7, 4);
    graph.addEdge(4, 5, 4);
    graph.addEdge(5, 6, 4);
    graph.addEdge(5, 7, 5);
    graph.addEdge(5, 9, 3);
    graph.addEdge(6, 10, 6);
    graph.addEdge(7, 8, 2);
    graph.addEdge(8, 9, 6);
    graph.addEdge(9, 10, 5);
    graph.computeSortestPathsFrom(1);
}
}
```

■ 3. Dijkstra, Resolução de problemas

Exercício 3 – Resolução através do algoritmo

Output

```
run:
distance from 1 to 1 is 0
distance from 1 to 2 is 2
distance from 1 to 3 is 5
distance from 1 to 4 is 5
distance from 1 to 5 is 4
distance from 1 to 6 is 8
distance from 1 to 7 is 9
distance from 1 to 8 is 11
distance from 1 to 9 is 7
distance from 1 to 10 is 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Húngaro, o que é?

O método Húngaro, denominação dada em homenagem aos pesquisadores húngaros que o desenvolveram, explora eficientemente a estrutura do problema de atribuição, isto é, o Método Húngaro trabalha com qualquer matriz custo n x n, modificando-a em outra matriz mais simples, buscando a melhor alocação possível

Imaginemos que os donos de uma empresa deparam-se com a situação de atribuir empregados a algumas tarefas, certamente o objetivo inicial será de que estes empregados maximizem ao máximo a sua produtividade, neste caso o método húngaro poderá ser utilizado e resolverá a situação, atribuindo os empregados ás tarefas e de forma a que estes maximizem a produtividade ou o lucro.

4 . Húngaro, Código-fonte do algoritmo

Vão agora seguir-se alguns prints do código fonte do algoritmo na linguagem PHP, em que se podem ver as várias classes criadas e o exercício implementado.

```
class HungarianBipatiteMatching {
  public $costMatrix = array();
  public $rows = 0;
  public $cols = 0;
  public $dim = 0;
  public $labelByWorker = array();
  public $labelByJob =array();
  public $minSlackWorkerByJob=array();
  public $minSlackValueByJob=array();
  public $matchJobByWorker=array();
  public $matchWorkerByJob=array();
  public $parentWorkerByCommittedJob=array();
  public $committedWorkers=array();
 public function computeInitialFeasibleSolution() {
       for ($j = 0; $j < $this->dim; $j++) {
           $this->labelByJob[$j] = INF;
       }

       for ($w = 0; $w < $this->dim; $w++) {
           for ($j = 0; $j < $this->dim; $j++) {
               if ($this->costMatrix[$w][$j] < $this->labelByJob[$j]) {
                   $this->labelByJob[$j] = $this->costMatrix[$w][$j];
               }
           }
       }
   }
```

4 . Húngaro, Código-fonte do algoritmo

```
public function HungarianBipatiteMatching($intMatrix) {
  $this->rows = sizeof($intMatrix);

  $this->cols = sizeof($intMatrix[0]);
  $this->dim = max($this->rows,$this->cols);
  for($i = 0;$i<$this->dim;$i++) {
      $costMatrix[$i] = array_fill(0,$this->dim,0);
  }
  for ($w = 0; $w < $this->dim; $w++) {
          if ($w < sizeof($intMatrix)){
              if (sizeof($intMatrix[$w]) != $this->cols){
                  throw new InvalidArgumentException("Irregular cost matrix");
              }

              $this->costMatrix[$w] = $this->arrayCopyOf($intMatrix[$w],$this->dim);
          }
          else {
             $this->costMatrix[$w] = array();
              for($i = 0;$i<$this->dim;$i++){
                                                  $this->costMatrix[$w][] = 0;
              }
          }
      }
      for($i = 0;$i<$this->dim;$i++) {
           $this->labelByWorker[] = 0;
           $this->labelByJob[] = 0;
           $this->minSlackWorkerByJob[] = 0;
           $this->minSlackValueByJob[] = 0;
           $this->parentWorkerByCommittedJob[] = 0;
           $this->matchJobByWorker[] = 0;
           $this->matchWorkerByJob[] = 0;
      }

      $this->committedWorkers = array_fill(0, $this->dim, false);
      $this->matchJobByWorker = array_fill(0,$this->dim,-1);
      $this->matchWorkerByJob = array_fill(0,$this->dim,-1);
}
```

4 . Húngaro, Código-fonte do algoritmo

```
public function execute() {
    $this->reduce();
    $this->computeInitialFeasibleSolution();
    $this->greedyMatch();
    $w = $this->fetchUnmatchedWorker();

    while ($w < $this->dim) {
        $this->initializePhase($w);
        $this->executePhase();
        $w = $this->fetchUnmatchedWorker();
    }

    $result = $this->arrayCopyOf($this->matchJobByWorker, $this->rows);

    for ($w = 0; $w < sizeof($result); $w++){
        if ($result[$w] >= $this->cols){
            $result[$w] = -1;
        }
    }
    return $result;
}
```

4 . Húngaro, Código-fonte do algoritmo

```
protected function executePhase() {
    while (true)
    {
        $minSlackWorker = -1;
        $minSlackJob = -1;

        $minSlackValue = INF;

        for ($j = 0; $j < $this->dim; $j++)
        {
            if ($this->parentWorkerByCommittedJob[$j] == -1)
            {
                if ($this->minSlackValueByJob[$j] < $minSlackValue)
                {
                    $minSlackValue = $this->minSlackValueByJob[$j];
                    $minSlackWorker = $this->minSlackWorkerByJob[$j];
                    $minSlackJob = $j;
                }
            }
        }

        if ($minSlackValue > 0)
        {
            $this->updateLabeling($minSlackValue);
        }

        $this->parentWorkerByCommittedJob[$minSlackJob] = $minSlackWorker;
```

4 . Húngaro, Código-fonte do algoritmo

```
            if ($this->matchWorkerByJob[$minSlackJob] == -1)
            {
                $committedJob = $minSlackJob;
                $parentWorker = $this->parentWorkerByCommittedJob[$committedJob];

                while (true)
                {
                    $temp = $this->matchJobByWorker[$parentWorker];
                    $this->match($parentWorker, $committedJob);
                    $committedJob = $temp;

                    if ($committedJob == -1)
                    {
                        break;
                    }

                    $parentWorker = $this->parentWorkerByCommittedJob[$committedJob];
                }

                return;
            }
            else
            {
                $worker = $this->matchWorkerByJob[$minSlackJob];
                $this->committedWorkers[$worker] = true;

                for ($j = 0; $j < $this->dim; $j++)
                {
                    if ($this->parentWorkerByCommittedJob[$j] == -1)
                    {
                        $slack = $this->costMatrix[$worker][$j]
                                - $this->labelByWorker[$worker] - $this->labelByJob[$j];

                        if ($this->minSlackValueByJob[$j] > $slack)
                        {
                            $this->minSlackValueByJob[$j] = $slack;
                            $this->minSlackWorkerByJob[$j] = $worker;
                        }
                    }
                }
            }
        }
    }
```

■ 4 . Húngaro, Código-fonte do algoritmo

```
protected function fetchUnmatchedWorker()
{
    $w;

    for ($w = 0; $w < $this->dim; $w++)
    {
        if ($this->matchJobByWorker[$w] == -1)
        {
            break;
        }
    }

    return $w;
}
protected function greedyMatch()
{
    for ($w = 0; $w < $this->dim; $w++)
    {
        for ($j = 0; $j < $this->dim; $j++)
        {
            if ($this->matchJobByWorker[$w] == -1
                    && $this->matchWorkerByJob[$j] == -1
                    && $this->costMatrix[$w][$j] - $this->labelByWorker[$w] - $this->labelByJob[$j] == 0)
            {
                $this->match($w, $j);
            }
        }
    }
}
```

4 . Húngaro, Código-fonte do algoritmo


```
protected function initializePhase($w)
{
    $this->committedWorkers = array_fill(0,sizeof($this->committedWorkers),false);
    //Arrays.fill(committedWorkers, false);
    $this->parentWorkerByCommittedJob = array_fill(0,sizeof($this->parentWorkerByCommittedJob),-1);
    //Arrays.fill(parentWorkerByCommittedJob, -1);

    $this->committedWorkers[$w] = true;

    for ($j = 0; $j < $this->dim; $j++)
    {
        $this->minSlackValueByJob[$j] = $this->costMatrix[$w][$j] - $this->labelByWorker[$w]
                - $this->labelByJob[$j];

        $this->minSlackWorkerByJob[$j] = $w;
    }
}

protected function match($w, $j)
{
    $this->matchJobByWorker[$w] = $j;
    $this->matchWorkerByJob[$j] = $w;
}
```

4 . Húngaro, Código-fonte do algoritmo

```
protected function reduce(){
    for ($w = 0; $w < $this->dim; $w++){
       $min = INF;

       for ($j = 0; $j < $this->dim; $j++){
            if ($this->costMatrix[$w][$j] < $min){
                $min = $this->costMatrix[$w][$j];
            }
       }
       for ($j = 0; $j < $this->dim; $j++){
            $this->costMatrix[$w][$j] -= $min;
       }
    }
    $min = array_fill(0,$this->dim,0); //ALERT

    for ($j = 0; $j < $this->dim; $j++){
        $min[$j] = INF;
    }
    for ($w = 0; $w < $this->dim; $w++)
    {
        for ($j = 0; $j < $this->dim; $j++)
        {
            if ($this->costMatrix[$w][$j] < $min[$j])
            {
                $min[$j] = $this->costMatrix[$w][$j];
            }
        }
    }
    for ($w = 0; $w < $this->dim; $w++)
    {
        for ($j = 0; $j < $this->dim; $j++)
        {
            $this->costMatrix[$w][$j] -= $min[$j];
        }
    }
}
```

4 . Húngaro, Código-fonte do algoritmo

```
protected function updateLabeling($slack)
{
    for ($w = 0; $w < $this->dim; $w++)
    {
        if ($this->committedWorkers[$w])
        {
            $this->labelByWorker[$w] += $slack;
        }
    }

    for ($j = 0; $j < $this->dim; $j++)
    {
        if ($this->parentWorkerByCommittedJob[$j] != -1)
        {
            $this->labelByJob[$j] -= $slack;
        }
        else
        {
            $this->minSlackValueByJob[$j] -= $slack;
        }
    }
}
```

4 . Húngaro, Código-fonte do algoritmo

```
public function arrayCopyOf($array, $size) { // Java API port
    $tmp = array();

    foreach($array as $arr) {
        $tmp[] = $arr;
    }

    if(sizeof($array) < $size) {
       for($i = 0; $i < $size-sizeof($array); $i++) {
            $tmp[]=0;
       }
    }

    return $tmp;
}
```

```
$m = [

[25, 31, 35],
[24, 17, 16],
[15, 23, 18]


];
```

4 . Húngaro, Código-fonte do algoritmo

```
$hungarian = new HungarianBipatiteMatching($m);
$result = $hungarian->execute();
$tam = 3;
$total = 0;
//print_r($result);
for ($x = 0; $x < $tam; $x++) {
    echo $m[$x][$result[$x]] . '<br>';
        $total += $m[$x][$result[$x]];
}

echo $total;


?>
```

4. Húngaro, Resolução do problema

Exercício - Enunciado

Exemplo:

Numa fábrica foram existem 3 tarefas para serem executadas e existem 3 empregados disponíveis para as realizar, com tempos de execução distintos. O objectivo da Direcção da fábrica é estabelecer uma afetação empregado-tarefa recíproca e exclusiva, que envolva um tempo mínimo de execução dessas tarefas. Os tempos de execução são os seguintes :

		Tarefa		
		1	2	3
Empregado	1	25	31	35
	2	24	17	16
	3	15	23	18

4. Húngaro, Resolução do problema

Exercício – Resolução manual

É neste facto que se apoia o *algoritmo Húngaro*, que consiste nos seguintes passos :

Passo 1. ***Aos elementos de cada linha da matriz de custos, subtrair o mínimo dessa linha.***
Na matriz resultante, aos elementos de cada coluna, subtrair o mínimo dessa coluna.

25	31	35
24	17	16
15	23	18

→

0	6	10
8	1	0
0	8	3

→

0	5	10
8	0	0
0	7	3

4. Húngaro, Resolução do problema

Exercício – Resolução manual

Passo 2. ***Tomar uma das linhas/colunas com menor nº de zeros, enquadrar um deles (aquele que cortar menos zeros) e cortar todos os restantes dessa linha e dessa coluna. Prosseguir até que todos os zeros estejam cortados.***
Se houver n zeros enquadrados, tem-se a solução óptima; caso contrário prosseguir.

$\boxed{0}$	5	10
8	$\boxed{0}$	$\cancel{0}$
$\cancel{0}$	7	3

Como o nº de zeros enquadrados não é 3, continuar

4. Húngaro, Resolução do problema

Exercício – Resolução manual

Passo 3. ***Cobrir os zeros enquadrados com o menor nº possível de traços :***

1. ***assinalar (com √) as linhas que não contêm zeros enquadrados;***

2. ***assinalar as colunas com pelo menos um zero cortado nas linhas assinaladas;***

3. ***assinalar as linhas com um zero enquadrado nas colunas assinaladas;***

4. ***repetir 2 e 3 até não ser possível assinalar mais linhas ou colunas;***

5. ***traçar as linhas não assinaladas e as colunas assinaladas.***

$\boxed{0}$	5	10	
8	$\boxed{0}$	~~0~~	
~~0~~	7	3	√

4. Húngaro, Resolução do problema

Exercício – Resolução manual

***Passo 4.** Determinar o menor elemento da sub-matriz constituída pelos elementos não traçados; subtrair esse elemento aos elementos dessa sub-matriz e adicioná-lo aos elementos na intersecção de dois traços. Voltar ao passo 2.*

$\boxed{0}$	5	10
8	$\boxed{0}$	~~0~~
~~0~~	7	**3**

→

$\boxed{0}$	2	7
11	$\boxed{0}$	~~0~~
~~0~~	4	$\boxed{0}$

Três zeros enquadrados, solução ótima encontrada

Empregado 1 – Máquina 1
Empregado 2 – Máquina 2
Empregado 3 – Máquina 3

Z=25+17+18=60 u. t

4. Húngaro, Resolução do problema

Exercício – Resolução através do algoritmo

```
$m = [

[25, 31, 35],
[24, 17, 16],
[15, 23, 18]


];


$hungarian = new HungarianBipatiteMatching($m);
$result = $hungarian->execute();
$tam = 3;
$total = 0;
//print_r($result);
for ($x = 0; $x < $tam; $x++) {
    echo $m[$x][$result[$x]] . '<br>';
        $total += $m[$x][$result[$x]];
}

echo $total;


?>
```

4. Húngaro, Resolução do problema

Ao correr no xampp, este irá ser o output

25
17
18
60

Este output significa que os zeros enquadrados vão aparecer onde os valores são 25,17 e 18 na matriz inicial. A solução do problema será 60 tal como na resolução analítica feita anteriormente $Z=25+17+18=60$ u. t

5. Algoritmo Bellman-ford

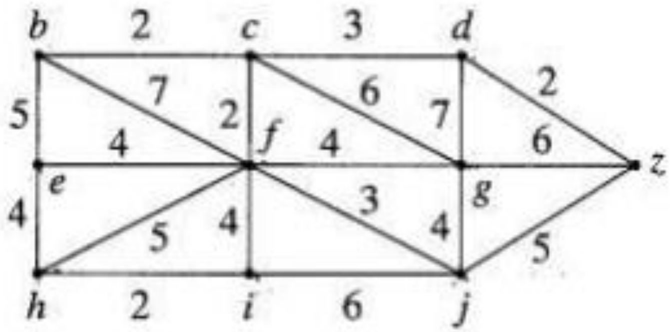
Algoritmo de Bellman-Ford

O **Algoritmo de Bellman-Ford** foi criado por ***Richard Bellman*** e ***Lester Ford, Jr***. Tal como o algoritmo de Dijkstra, o algoritmo de Bellman-Ford procura os caminhos mais curtos de um grafo mas é mais lento no que toca a fazer essa procura, sendo assim o algoritmo de Dijkstra mais viável. Este algoritmo é maioritariamente usado para quando os caminhos do grafo são negativos pois, como já mencionado, não podem ser usados no algoritmo de Dijkstra.

■ 5. Algoritmo Bellman-ford

Exercício 2 Bellman-Fo

1) Recorrendo ao algoritmo de Dijkstra, determine o comprimento e o caminho mais curto entre os vértices **b** e **z**.



5. Algoritmo Bellman-ford

Resolução Exercício 1 Bellman-Ford

Vértice B como fonte:

B	C	D	E	F	G	Z	H	I	J
0	2	∞	5	7	∞	∞	∞	∞	∞
B	B		B	B					

Vértice C como fonte:

B	C	D	E	F	G	Z	H	I	J
0	2	5	5	4	8	∞	∞	∞	∞
B	B	C	B	C	C				

5. Algoritmo Bellman-ford

Vértice F como fonte:

B	C	D	E	F	G	Z	H	I	J
0	2	5	5	4	8	∞	9	8	7
B	B	C	B	C	C		F	F	F

Vértice D como fonte:

B	C	D	E	F	G	Z	H	I	J
0	2	5	5	7	8	7	9	8	7
B	B	C	B	C	C	D	F	F	F

■ 5. Algoritmo Bellman-ford

Resolução Exercício 1 Bellman-Ford (Java)

Legenda:

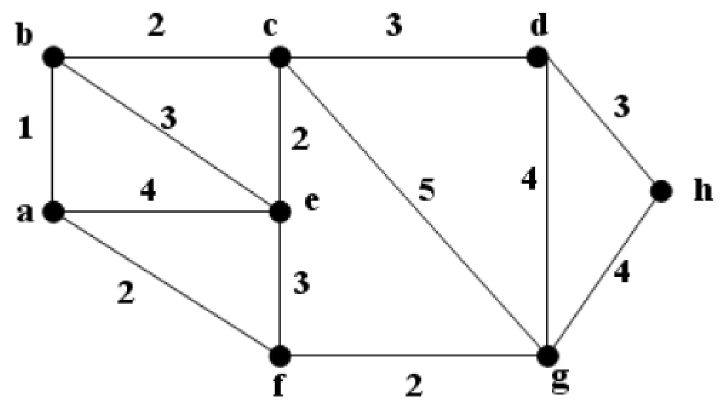
```
B - 1
C - 2
D - 3
E - 4
F - 5
G - 6
Z - 7
H - 8
I - 9
J - 10
```

```
Enter the number of vertices
10
Enter the adjacency matrix
0 2 0 5 2 0 0 0 0 0
2 0 3 0 2 6 0 0 0 0
0 3 0 0 0 7 2 0 0 0
5 0 0 0 4 0 0 4 0 0
7 2 0 4 0 4 0 5 4 3
0 6 7 0 4 0 6 0 0 4
0 0 2 0 0 6 0 0 0 5
0 0 0 4 5 0 0 0 2 0
0 0 0 0 4 0 0 2 0 6
0 0 0 0 3 4 5 0 6 0
Enter the source vertex
1
Enter the destination vertex:
7
distance of source  1 to 7 is 7
BUILD SUCCESSFUL (total time: 40 seconds)
```

5. Algoritmo Bellman-ford

Exercício 2 Bellman-Ford

2) Recorrendo ao algoritmo de Dijkstra, determine o comprimento e o caminho mais curto entre os vértices **a** e **h** e entre o vértice **b e** qualquer vértice do grafo.



5. Algoritmo Bellman-ford

Resolução Exercício 2 Bellman-Ford

Vértice A como fonte:

A	B	C	D	E	F	G	H
0	1	∞	∞	4	2	∞	∞
A	A			A	A		

Vértice B como fonte:

A	B	C	D	E	F	G	H
0	1	3	∞	4	2	∞	∞
A	A	B		A	A		

5. Algoritmo Bellman-ford

Resolução Exercício 2 Bellman-Ford

Vértice F como fonte:

A	B	C	D	E	F	G	H
0	1	3	∞	4	2	4	∞
A	A	B		A	A	F	

Vértice C como fonte:

A	B	C	D	E	F	G	H
0	1	3	6	4	2	4	∞
A	A	B	C	A	A	F	

5. Algoritmo Bellman-ford

Resolução Exercício 2 Bellman-Ford

Vértice G como fonte:

A	B	C	D	E	F	G	H
0	1	3	6	4	2	4	8
A	A	B	C	A	A	F	G

5. Algoritmo Bellman-ford

Resolução Exercício 2 Bellman-Ford (Java)

Legenda:

B	-	1
C	-	2
D	-	3
A	-	4
E	-	5
H	-	6
F	-	7
G	-	8

```
run:
Enter the number of vertices
8
Enter the adjacency matrix
0 2 0 1 3 0 0 0
2 0 3 0 2 0 0 5
0 3 0 0 0 3 0 4
1 0 0 0 4 0 2 0
3 2 0 4 0 0 3 0
0 0 3 0 0 0 0 4
0 0 0 2 3 0 0 2
0 5 4 0 0 4 2 0
Enter the source vertex
4
Enter the destination vertex:
6
distance of source  4 to 6 is 8
BUILD SUCCESSFUL (total time: 1 minute 3 seconds)
```

6. Método Vogel

O método de Vogel é um procedimento iterativo para calcular uma solução viável básica de um problema de transporte.

Este método é preferido em relação aos outros dois métodos usados (Método do Canto Noroeste e Método do Mínimo da Matriz de Custos), porque a solução inicial obtida por este método é ótima ou muito próxima da solução ótima.

6. Método Vogel

Exemplo do método de Vogel

	Destination				
Origin	1	2	3	4	Supply
1	20	22	17	4	120
2	24	37	9	7	70
3	32	37	20	15	50
Demand	60	40	30	110	240

```
run:
[0, 10, 0, 110]
[60, 10, 0, 0]
[0, 20, 30, 0]
Total cost: 3810
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Referências

- ***Ficha 3 Matemática Discreta 2015/2016****, local de onde foram retirados os dois primeiros exercícios*
- ***Exame de Matemática Discreta 02/06/2014,*** *sitio de onde retiramos o exemplo 3 de Dijkstra*
- ***PowerPoint de Afetação de Investigação Operacional 2015/2016****, local de onde foi tirado o exemplo utilizado no método Húngaro.*
- *Site de onde foi retirado o código fonte do algoritmo de Dijkstra -* ***http://krishnalearnings.blogspot.pt/2015/07/implementation-in-java-for-dijkstras.html***
- *Local de onde retiramos o código fonte do método húngaro em PHP –* ***http://stackoverflow.com/questions/30892659/hungarian-algorithm-in-php-with-multiple-assignments***
- ***(JAVA-Dijkstra) - http://www.pracspedia.com/CN/dijkstra.html***
- ***(Linguagem C - Dijkstra) - http://www.thecrazyprogrammer.com/2014/03/dijkstra-algorithm-for-finding-shortest-path-of-a-graph.html***
- ***(JAVA- Hungaro) - http://www.sanfoundry.com/java-program-implement-hungarian-algorithm-bipartite-matching/***