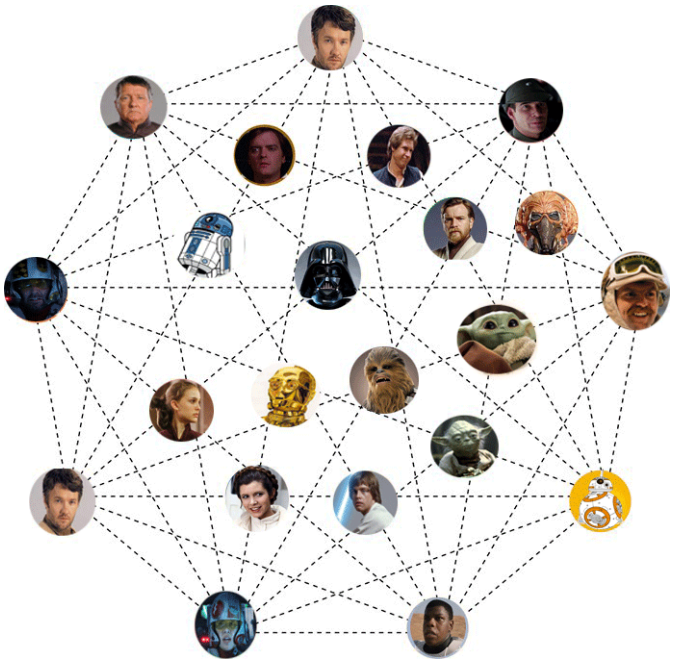
Universidad Castilla La Mancha

Ingeniería informática

Non linear data structure

Data Structure laboratory



Authors:
Andrés González Varela
María Jesús Dueñas Recuero

December 2021

Contents

1 Non linear data structure - Graphs

1.1 Problem description

We have at our disposal two CSV files with information that represent a network of relationships betweencharacters from the Star Wars saga, corresponding to the interactions they have had throughout the 9 cinematic episodes. The starwars-pers.csv file contains the data for each character (identifier, name and aninteger indicating the total number of references to the character in the saga), and the starwars-links.csv file contains, for each pair of characters, the number of interactions between the two.

The objective of this practical work is to implement a Java program that processes the indicated files and responds to the following requirements:

a) Build the corresponding graph and **show the number of characters**, **the total number of relationships between characters**, **the character who has more relationships with others**, and **the pair of characters that maintain the highest level of interaction with each other**.

b) Calculate if there are **subsets of characters that are not related to each other** (example: A, B and Chave relationships between them, and the same happens between D, E and F, but between the vertices of the first group and those of the second there are no relationships). It will only be necessary to indicate if they exist or not.

c) Since you want to send a **holo-message secretly through trusted intermediaries**, given two characters read by keyboard (sender and recipient), indicate the shortest sequence of trusted2 people that allows you to reach the recipient to transfer the message.

1.2 Approach

The main idea for this project is to establish the connections that are represented in the excels: **starwar-pers** and **starwars-links**.And to learn the main knowledge of graphs:How to create a graph, its vertices, edges; the development of dfs or bfs and even the vertices with more unions with other vertices,etc.

1.3 Classes in the program

1.3.1 ReadFile Class

This class is in charge of reading the files provided on the campus.

The first file we have read is the **starwars-pers** file, since this is where we will collect the character information and in the future where we will create the vertices for the graph.To realise the above version we have the **readCharacters()**method.

In this method the main idea is to read the excel where contains the information of the characters in this way as we are reading the file storing the information in an array called tokens[], in such a way that inside the while we read the line and separating them the information by ";", being:

- **tokens[0]** which stores the **id** of the character.
- **tokens[1]** which stores the **name** of the character.
- **tokens[2]** which stores **value** the numbers of iterations along of the films.

As we get the character object we create the character decorator element and add it to the graph.

1.3.2 Character class

This class is the one that we will use for create Character objects that we will use for creating the DecoratedElements for the vertex of the class. The class has 3 variables: ID, name, value.

- **ID:**Which will store the ID that references the character.
- **name:**Stores the name of the character, for example, Darth Vader.
- **value:**Value refers to the number of references to the character in the saga.

This class also has all the getters and setters of the variables and the constructor to build the characters.

1.3.3 Links class

This class is the one that we will use for create Links objects that we will use for creating the DecoratedElements for the edges of the graph.The class has 3 variables: sourceID, targetID, weight.

- **sourceID:** which will store the ID of the source Vertex.
- **targetID:** which will store the ID of the target Vertex.
- **weight:** the weigth of the Edge or relation.

This class also has all the getters and setters of the variables and the constructor to build a link.

1.3.4 DecoratedElement class

This class is used to structure and build the DecoratedElement objects that we will insert in the Vertex of the graph and also the Edges. The DecoratedElement is composed by and ID and a Character element. The variables are:

- **ID:** Which will store the ID of the Vertex.
- **Character:** A character object with its own variables.
- **Visited:**a boolean that tells us if the Verte has been visited.
- **Parent:**The Vertex that parents the Vertex.
- **Distance:**Distance in Vertex from the original node.

This class also has all the getters and setters of the variables and the constructor to build the DecorateElements.

1.3.5 Objectives class

This class will be used to store all the methods to solve the problems of the task. We have four methods:

- **moreRelations:**This method will receive the graph by reference. We will create two Iterators to travel the Graph node by node and Edge by Edge. Using the method incidentEdges we will take all the Edges of a specified Vertex. We will count the number of edges that has each node and then print the result. We will compare the number of edges and store the Character that has more Edges in a Stack.
- **moreInteractions:**This method will receive the graph by reference. We use a very similar method to the moreRelations method. We create the Iterators and use them to travel the graph. When we find an Edge that has more weight we will store the Node that we are looking at an its opposite with that Edge.
- **dfs:**is a search algorithm for which it traverses the nodes of a graph. It works by expanding each of the nodes that it locates, in a recurrent manner (from the parent node to the child node). When there are no more nodes to visit on that path, it returns to the predecessor node, so that it repeats the same process with each of the node's neighbours. It should be noted that if the node is found before all nodes have been traversed, the search is finished.
- **subsets:** this method is where we will traverse the graph and check that all vertices are connected. If they are connected it is a connected graph.

1.3.6 ShorthestPath Class

This class contains the two methods that are necessary for the third section of the problema. We have two methods: **BFS** and **holo**.

- **BFS:** this method will be used to obtain path using a BFS strategy. First we introduce the first vertex in a queue. After this we will check all the vertex and in case the relation between the vertex in the queue and the one we are checking is greater than 8 we introduce it in the queue and we change the vertex variable Visited to "True" and set its father as the first vertex. When we reach the end of the queue we have done this with all the vertex.
- **Holo:** in this method the user will give us two characters he want to tries to try his path with the BFS method. First you check both vertex are in the graph. Then if they are valid vertexs we call the BFS method. If there is a usable path we receive the final vertex with it parent. Then you start coming back parent by parent and like that we obtain the path to send the message.

1.3.7 Main class

The only purpose of this class is to execute the program and we do this through a menu where we create objects of all the classes and call the methods in order to fullfil the assessment. First we ask for the direction of the files and the create the graph. Then, after this we will deploy the menu with the options.

- **Option 1:** This will give the solution to the first assessment.
- **Option 2:** This will give the solution to the second assessment.
- **Option 3:** This will give the solution to the third assessment.
- **Option 4:** Stop the execution of the program

1.4 Technical requirements

1.4.1 Generation bat

In order to be able to run the programme on an executable we have followed the following steps:

- Create an executable jar of the developed project.
- In a text editor we have written with extension.bat,which we have called Start as if it were the demo of an application.

```
java -jar nameOfTheJarExecutable.jar
PAUSE;
```

1.4.2 Run program

To be able to execute this bat the only thing that is needed is to make double click in the file that puts "Start", later, I will ask you the route of the files of which will show a menu where the user could select which option ypu want to show the result.

To run in any IDE it is necessary to follow the following steps:

1. Download and unzip the file containing the code.
2. Create a project in the IDE of use
3. Right click on the created project and click on the import option.
4. Click on the file system option
5. Select the folder with the code to execute
6. Add the package if necessary in each class
7. Run the program