Graded Project – Advanced Data Structure and Algorithms – Python Language – Youssef ZEMALI – Rayan AL QARAOUI

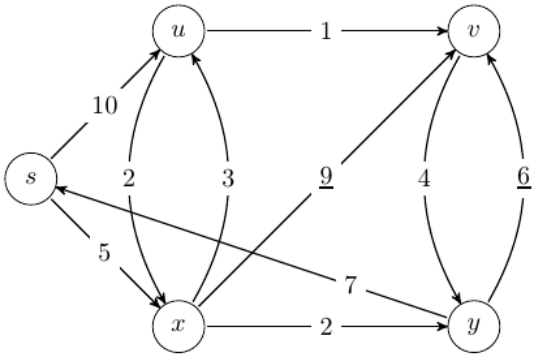
Project Objectives

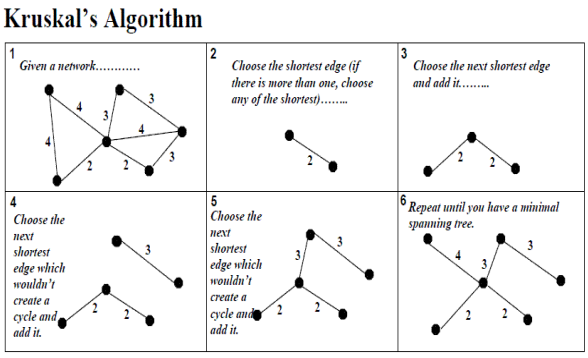
The purpose of this project is to use and put all concepts covered this semester into practice to solve differents kind of problems. In this case, the project is divided in four parts and each part need a different algorithm or method to solve it.

Within the framework of programme, the aim will be:

- Have a good command of python
- Understand algorithmic art, find a strategy to solve a problem
- Use complexity theory, quantify the effectiveness of such a strategy
- Know about theoretical and practical optimizations (how to optimize an algorithm and how to implement it effectively)
- Have a good algorithmic culture

Throughout this project, we will empoy differents kind of algorithms such as Prim, Dijkstra or even Kruskal's method. Those algorithms will help us along this project to deal with an issue.





Adopted Strategy

The strategy that we decided to adopt for simulating the results is based on 3 essentials points:

- Split the program into different classes which contains connected functions
- Use different libraries in PyCharm to plot a graph or use functions
- Multiple-choice Menu

Function Description

This program offers 8 features at menu's display. For each functionality, we will need different functions that we'll make clear. Here is the 8 features :

1. Display Gotham City's Railways
2. Show a connected Graph while minimizing the total amount of distance
3. Show the Shortest path from Gotham City to the others
4. Find a member of Joker team in database with his ID (Linear)
5. Find a member of Joker team in database with his ID (Divide & Conquer)
6. Sort database and find a member of Joker team with his ID
7. Find a member of Joker team in database using BINARY TREE
8. Find a member of Joker team in database using AVL TREE

```
--------------------PROJET ADSA: LE JOKER--------------------

PART 1 : CONNECTING THE PEOPLE
---------------------------------
1)To display the Gotham City's railways.
2)To show a connected graph while minimizing the total amount of distance.
---------------------------------
Part 2: SPREAD THE REVOLUTION
---------------------------------
3)To show the shortest path from to Gotham City (first line/column) to the others.
---------------------------------
Part 3:ORGANIZE THE JOKERS
---------------------------------
4)To find a member of Joker team in database with his ID.(linear algorithm)
5)To find a member of Joker team in database with his ID.(Divide&Conquer algorithm)
6)To sort database and find a member of Joker team with his ID.(Divide&Conquer algorithm)
---------------------------------
Part 4:THE JOKERFATHER
---------------------------------
7)To find a member of Joker team in database using BINARY TREE.
8)To find a member of Joker team in database using AVL TREE.
9)EXIT
---------------------------------
Please choose an option between 1 and 9:
```

Part 1 : CONNECTING THE PEOPLE

1. Display Gotham City's Railways

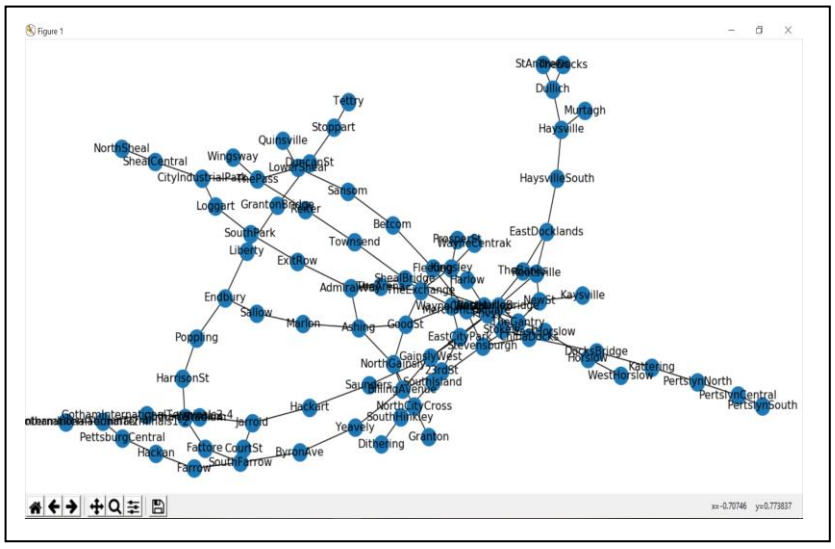
1. Create and display a graph representing the Gotham City's railways and subways below (10%)

This feature allows the user to visualize the whole Gotham City's Railways and to explore it. It requires a ".csv" file which contains all the stations and the weight represented by the "distance" stem from the City's Railways.

- **Plot Function**

```
def __plotnetwork__(self):
```

This function is in the class Graph, which contains all of it's related to it. It only takes a "self" in argument to access the attributes and methods of the class and return nothing. This function read a ".csv" file with the function "read_csv" include in "pandas" library, use it into a variable and assign the data thanks to a "for … in …" to a graph "g" created with "networkx" library. Finally, the graph is plotted by a "draw" function include in "networkx" library.



2. **Show and display a solution of the connected network (TO DO)**

2. Which kind of algorithm create a connected graph while minimizing the total amount of distance? Show the algorithm? What is its complexity? (5%)

We have to solve a Minimum Spanning Tree problem. The objective is to find the solution to cover all stations of Gotham City, being the most efficient, that is to say by having the minimum weight.
For this, we can use Prim's algorithm or Kruskal's algorithm.
The complexity of Prim's algorithm is in $O(V^2)$ with V the number of vertexes.

3.Show and display a solution of the connected network. (10%)

Part 2 : SPREAD THE REVOLUTION

3. **Show the Shortest path from Gotham City to the others**
   1. *Which kind of algorithm create a connected graph while minimizing the distance to a unique city? What is its complexity? (5%)*

Here, we have to use a Shortest Path Algorithm. Thus, we decided to use Dijkstra because there are no negative weights (distances) in our diagram. This algorithm has a complexity in O((a + n)log(n)) with n the number of nodes, and has the number of arcs: it is a polynomial complexity.

$$O(a + nlogn)\ avec\ a\ arcs\ et\ n\ noeuds$$

2. Show and display a solution of the proposed algorithm. (10%)

This functionality retrieves the matrix of the Project n°2 which link Gotham City to the others in order to present the best path by using the shortest. To be able to suggest a pathway, we used Dijkstra's algorithm.

- **Dijksta's Algorithm**

You can check this function in the code:

```
def dijkstra(matrice, start):
```

3. Show and display a solution of the problem. (5%)

"Dijkstra's Algorithm" picks a matrix and "start" the origin up for attributes and detect the shortest path. The origin is represented by a number and the start-up is made with a variable regarded as infinite which embody the upper-bound for the longest path. "S_connu" is used to hold for each top the shortest path and "S_inconnu" join all temporary distance from a top. Secondly, in the research's part, we won't stop until it still points in "S_inconnu" and pick up the shortest.

```
[2.3, 9.5, 11.1, 9.2, 11.2, 0, 5.6, 12.1, 7.7, 8.5]
[5.1, 10.1, 8.1, 9.5, 8.5, 5.6, 0, 9.1, 8.3, 9.3]
[10.2, 12.8, 1.1, 12.0, 1.0, 12.1, 9.1, 0, 11.4, 12.4]
[6.1, 2.0, 10.5, 1.6, 10.6, 7.7, 8.3, 11.4, 0, 1.1]
[7.0, 1.0, 11.5, 1.1, 11.6, 8.5, 9.3, 12.4, 1.1, 0.0]

Plus court chemins trouvé grace a Dijkstra de
longueur 2.3 : 0 -> 5
longueur 5.1 : 0 -> 6
longueur 6.1 : 0 -> 8
longueur 7.0 : 0 -> 9
longueur 7.699999999999999 : 0 -> 8 -> 3
longueur 8.0 : 0 -> 9 -> 1
longueur 9.2 : 0 -> 2
longueur 9.3 : 0 -> 4
longueur 10.2 : 0 -> 7
```

As we can see, Joker can send just 7 people because a man can pass by 8 and 2 in the same time and another can pass by 9 and 1. It will be better to do that than send 10 people in each destination.

Part 3 : ORGANIZE THE JOKER

1. Suggest and show a greedy method to find an ID in a (linear) database. What is its complexity? (5%)

- **4 Find a member of Joker team in database with his ID (Linear)**

```
def recherche(l, id):
```

First and foremost, to use this function we need to open a file ".txt" which contains data and allocate it to a variable "x". This variable will be split into another one "y" to finally add it to a tab while "x" isn't correctly over, the tab will be used as an attribute in "recherche". This function is very straightforward inasmuch as it keeps hold of the first element of the tab and try to associate it to the id's attribute.

The worst-case complexity of linear search is O(n).

2. Suggest and show a divide and conquer method to find an ID in a (linear) database. What is its complexity? (5%)

- **5 Find a member of Joker team in database with his ID (Divide & Conquer)**

```
def recherche_dicho(l, id):
```

First and foremost, to use this function we need to open a file ".txt" which contains data and allocate it to a variable "x". This variable will be split into another one "y" to finally add it to a tab while "x" isn't correctly over, the tab will be used as an attribute in "recherche_dicho". This function searches a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found, or the interval is empty.

The complexity of Binary Search algorithm is O(logn).

3. Suggest and show a method that sort the (linear) database and find an ID. What is its complexity? (5%)

- **6 Sort database and find a member of Joker team with his ID**

```
def tri_insertion(l):
```

First and foremost, to use this function we need to open a file ".txt" which contains data and allocate it to a variable "x". This variable will be split into another one "y" to finally add it to a tab while "x" isn't correctly over, the tab will be used as an attribute in "tri_insertion". Also, before to use the last, we decided to shuffle the data inside the tab and print it, this is what we obtain:

```
8)To find a member of Joker team in database using AVL TREE.
9)EXIT
--------------------------------
Please choose an option between 1 and 9: 6
[[1, 'Leonardo_DiCaprio'], [2, 'Johnny_Depp'], [3, 'Chris_Pratt'], [4, 'Matt_Damon'], [5
[[32, 'Jim_Carrey'], [51, 'Ben_Foster'], [113, 'Kevin_Pollak'], [120, 'Guy_Pearce'], [73
```

The data is accurately blended, henceforth we'll use "tri_insertion"'s function and observe if data goes back to normal.

```
8)To find a member of Joker team in database using AVL TREE.
9)EXIT
--------------------------------
Please choose an option between 1 and 9: 6
[[1, 'Leonardo_DiCaprio'], [2, 'Johnny_Depp'], [3, 'Chris_Pratt'], [4, 'Matt_Damon'], [5,
[[32, 'Jim_Carrey'], [51, 'Ben_Foster'], [113, 'Kevin_Pollak'], [120, 'Guy_Pearce'], [73,
[[1, 'Leonardo_DiCaprio'], [2, 'Johnny_Depp'], [3, 'Chris_Pratt'], [4, 'Matt_Damon'], [5,
```

Now it's correctly running. In this algorithm, we select the first unsorted element then swap other elements to the right to create the correct position and shift the unsorted element to finally advance the marker to the right one element and now we can use a Divide&Conquer algorithm to find ID which is rechercher_dichotomique.

4.Show and display the three methods with an example of database of your choice (at least 50 data). (10%)

```
[[1, 'Leonardo_DiCaprio'], [2, 'Johnny_Depp'], [3, 'Chris_Pratt'], [4, 'Matt_Damon'],
Please choose an ID between 1 and 129 of the joker team:(Linear search) 4
[4, 'Matt_Damon']
```

```
[[1, 'Leonardo_DiCaprio'], [2, 'Johnny_Depp'], [3, 'Chris_Pratt'], [4, 'Matt_Damon'], [5, 'Chris_Eva
Please choose an ID between 1 and 129 of the joker team:(Divide&Conquer search dicho) 10
[10, 'Denzel_Washington']
```

Part 4 : THE JOKERFATHER

7.Find a member of Joker team in database using Binary Tree

1. Suggest and show a method to manage a database in a binary tree; present a method to find any value in the tree (show both complexity). Show the methods with your own example (at least 50 data). (5%)

In this section we'll create a tree and a tab where we going to stock all the data from the file "acteurs.txt" split earlier. Then, the function "insert" will behave in order to integrate data into the tree and "inorder"'s function will display all nodes at screen. Finally, the function "search" will find value wanted.

- **"Insert" Function**

```
def insert(root, node):
```

This function is in the class Node, which contains all of it's related to it. It takes "root" and "node" as arguments. This function allows to the user to add a node to the binary tree and if it is empty, therefore the tree will be created with this first node. The function verifies in a first time if the tree is non-existent, if not, the root's ID is compared to node's one. If the root's one is smaller, then the function will shift in the right node. In the event of the right node doesn't exist, it will be created, otherwise the function calls it-self with "right.node" instead of all tree. Within the scope of root's one is bigger, it's the same procedure, except "left.node" instead of the right's one. Therefore, insertion in binary tree has worst case complexity of O(n).

- **"Inorder" function**

```
def inorder(root):
```

This function is in the class Node, which contains all of it's related to it. It takes "root" as argument and allows the user to explore all the tree by displaying it at screen. It's a recursive function which call itself at each node and show the "ID" and "name" enclosed. It firstly starts with the left node because the smallest "ID" is always in the left part of a binary tree.

- **"Search" function**

```
def search(root, value):
```

This function is recursive, and it is in the class Node which contains all of it's related to it. It takes "root" and "value" as arguments and allows the user to find the value seeked. First, the function check if "root" is not empty or even if the first node is not directly equal to the value. Otherwise, the ID is verified and if it's smaller than the value, then the function call to itself with the "right.node" instead of all tree. In the other hand, if it's bigger, the procedure is the same with "left.hand" in place of "right.node". Therefore, searching in binary tree has worst case complexity of O(n).

```
127:Zac_Efron
128:Jason_Statham
129:Jeremy_Renner


Please choose an ID between 1 and 129 of the joker team: (Binary Search Tree)45
45:Michael_Shannon
```

2. Suggest and show a method to improve the database management thanks to the works of the soviets Adelson-Velsky and Landis with the previous database. (10%)

8.Find a member of Joker team in database using AVL Tree

This part of the project concerns the AVL Tree, the distinctive feature is on the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. We will apply this to Jokefather's part. Firstly, when we select the choice 8 into the menu, a "AVL_Tree" instance is created and a null variable "AVL" too. Secondly, we need to open a file ".txt" which contains data and allocate it to a variable "x". This variable will be split into another one "y" to finally add it to a tab while "x" isn't correctly over, the tab will be used after with "for … in …" method to fill up the instance and "AVL" at the same time. Then, the function "preOrderAVL" will display the tree in a first time and secondly the function "inorderAVL" will organise the tree. Finally, we will use the function "Search" to find for example ID=86.

- **"PreOrder" Function**

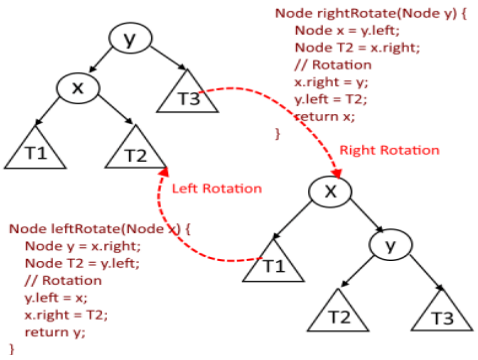
```
def preOrder(self, root):
```

This function is in the class "AVL_Tree", which contains all of it's related to it. It takes "root" as argument and allows the user to explore all the tree by displaying it at screen. At the beginning, the function check if root really exist otherwise it returns nothing. Conversely, it's a recursive function which call itself at each node and show the "ID" and "name" enclosed. It firstly starts with the left node because the smallest "ID" is always in the left part of a binary tree.

- **“Insert” Function**

```
def insert(self, root, valeur, nom):
```

This function allows the insertion of a node by respecting “AVL” conditions known as self-balancing binary search tree. In AVL Tree, the heights of child subtrees at any node differ by at most 1. At any time if height difference becomes greater than 1 then tree balancing is done to restore its property. Search, Insertion and deletion, all operations takes O(logn) time since the tree is balanced.



Firstly, to deal with, we must use recursion to check whether they are still balanced or not, then when a new node is added, its parent’s node height gets increased by 1.

At every node the balance factor will be checked:

- If **balance factor =1** means tree is balanced at that node
- If **balance factor >1** means tree is not balanced at that node, left height is more that the right height so that means we need rotation.
- If **balance factor <-1** means tree is not balanced at that node, right height is more that the left height so that means we need rotation.

Say the current node which we are checking is X and If new node is less than the X.left then it will be **Left-Left case**, and if new node is greater than the X.left then it will be **Left-Right case**.

- **“InOrderAVL” function**

```
def inorderAVL(self, root):
```

This function is in the class “AVL_Tree”, which contains all of it’s related to it. It takes “root” as argument and allows the user to explore all the tree in non-decreasing order. At the beginning, the function check if root really exist otherwise it returns nothing. Conversely, it’s a recursive function which call itself at each node and show the “ID” and “name” enclosed. It firstly traverse the left subtree, visit the root and traverse the right subtree.

```
Preorder traversal of the constructed AVL tree is

64:Bradley_Cooper 32:Jim_Carrey 16:Benedict_Cumberbatch 8:Robert_Downey_Jr 4:Matt_Damon 2:Johnny_

Inorder traversal of the constructed AVL tree is

1:Leonardo_DiCaprio
2:Johnny_Depp
3:Chris_Pratt
4:Matt_Damon
```

First and foremost, to use this function we need to open a file “.txt” which contains data and allocate it to a variable “x”. This variable will be split into another one “y” to finally add it to a tab while “x” isn’t correctly over, the tab will be used as an attribute in “recherche”. This function is very straightforward inasmuch as it keeps hold of the first element of the tab and try to associate it to the id’s attribute. The worst-case complexity of linear search is O(n).

Finally, we will use the function “Search” to find for example ID=86.