Scanpath Similarity Analysis

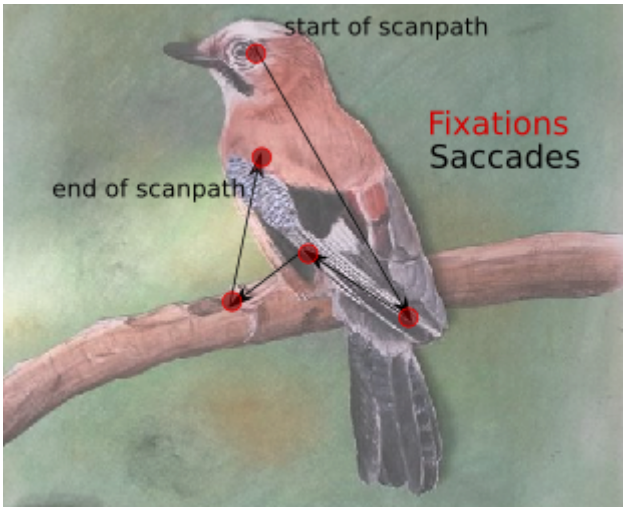
Rules:

1. Use **Python 3.7** or higher in **Google Colab**.
2. Submission should be a single **.zip** file (no other extensions like .rar, .gz) Keep this in mind.
3. Submission should contain
    a. The already provided **utils.py** file and **scanpath_dataset.pickle** file.
    b. Code file in **.ipynb** format. Use one **.ipynb** file for both Part A and Part B questions. **DO NOT** use .py files. We will test your code in Google Colab.
    c. README file describing how to run the code. TAs will run your code to see if they are working.
    d. A write-up in **.pdf** format. It should contain graph plot results at **grid_size (to be explained in point c of Part A) 8x8, 16x16, 32x32 for both Part A and Part B** from the output of your code. Treat this as a mini-report that TAs will evaluate after your code successfully runs.
4. **DO NOT** use built-in functions for cases where you have been asked to implement that exact same functionality.
5. Adhere to the function signature laid out in point d of part A

Assignment Description:

In this assignment, you will work with a scanpath dataset stored in a pickle file.

A scanpath refers to the sequence of eye movements made by an individual's eyes as they explore and visually process a visual stimulus, such as an image, scene, or text. These eye movements are characterized by fixations (brief periods when the eyes are relatively still and focused on a specific point) and saccades (rapid movements of the eyes between fixations).



The dataset contains information about the scanpaths of multiple individuals as they interact with stimulus images.

A brief description of the experiment is as follows:

Part A: Load the dataset, compute the edit distance between the scanpaths of every pair of individuals, and store the results in a 2D list (Edit distance matrix).

Part B: Use Prims MST algorithm to construct an MST from the distance matrix obtained in part A, remove the largest edge in this MST (Thus dividing MST into two connected components), and label nodes 'red'/'blue' depending on their connected component membership using BFS/DFS. Finally, plot the two connected components using the provided helper function.

1. Part A(3+7 Marks):

a. **Load the scanpath dataset:**
Upload files ***scanpath_dataset.pickle***, ***utils.py*** in your collab instance. The following code loads the `'scanpath_dataset.pickle'` file into a Python dictionary `'scanpath_dataset'`.

```
from utils import load_dataset
scanpath_dataset = load_dataset(filename =
'scanpath_dataset.pickle')
```

b. **Extract the following information from the dataset:**

```
# The height of the stimulus images.
h = scanpath_dataset["height"]

# The width of the stimulus images.
w = scanpath_dataset["height"]

# The total number of people whose scanpaths #were
recorded.
total = scanpath_dataset["total_subjects"]

# Retrieve the scanpath of person 0
scanpath = scanpath_dataset[0]

# Retrieve the scanpath of person 1
scanpath = scanpath_dataset[1]
```

The scanpath for each individual (example `scanpath_dataset[0], scanpath_dataset[1], scanpath_dataset[2], ... scanpath_dataset[total -1]`) is a list that contains the sequence of coordinates of fixations in the stimulus image.

```
#print scanpath of person 0
print(scanpath_dataset[0])
```

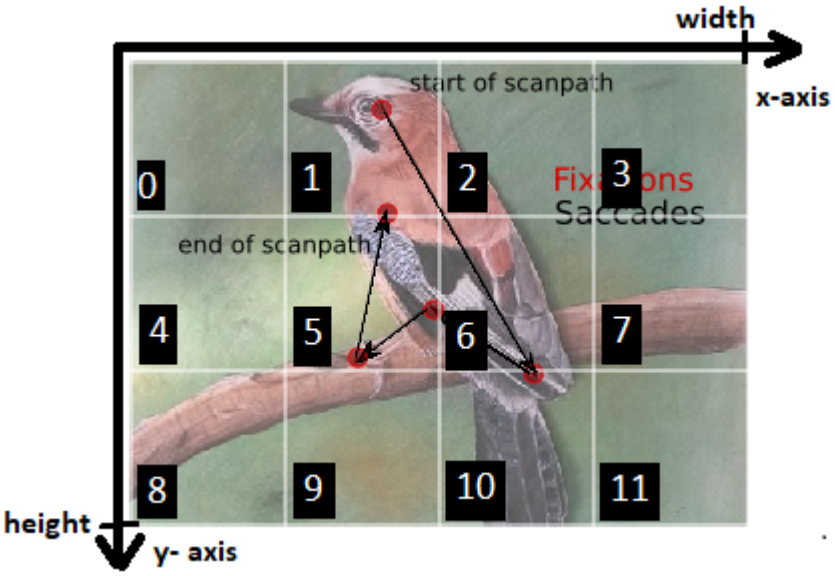
Gives output:

```
[ [504.0, 393.0], [492.0, 396.0], [508.0, 355.0],[504.0, 361.0],
[504.0, 246.0], [476.0, 326.0], [480.0, 342.0], [515.0, 457.0],
[515.0, 470.0], [716.0, 355.0], [492.0, 102.0], [500.0, 86.0],
[504.0, 64.0], [508.0, 44.0], [890.0, 358.0], [575.0, 438.0],
[582.0, 451.0], [543.0, 585.0], [508.0, 432.0]]
```

Thus, in this list,
`[504.0, 393.0]` is the first fixation point with x-coordinate (i.e., along the width) = 504.0 pixels and y-coordinate (i.e., along the height)=393.0 pixels,
`[492.0, 396.0]` is the second fixation point with x-coordinate (i.e., along the width) = 492.0 pixels and y-coordinate (i.e., along the height)=396.0 pixels
, and so on.

c. Conversion of 2D coordinate into a unique index:

Recall that the edit distance is computed between two sequence character strings (like **'Peter'** and **'Parker'**) or equivalently between two arrays of integers. In contrast, a scanpath is a sequence of 2D coordinates of fixations, and we are required to compute the edit distance between two scanpaths. Thus, we must convert each 2D coordinate within a scanpath to a unique index. This is done by discretizing the image into grids, each of size **grid_sizeXgrid_size**, and assigning a unique index to each grid cell. The fixation point within a specific grid cell inherits that grid cell's index.



In the above image, the scanpath is re-encoded as **[1, 10, 5, 5, 1]**

d. Implement the function to compute the edit distance between each pair of two scanpaths. **The function must adhere to the following signature**:

```
def get_all_pair_edit_distances(
    scanpath_dataset,
    grid_size):
    total = scanpath_dataset['total_subjects']

    # Create 2D list total x total
    # to record all pairs of distances
    distance_matrix = []
    for rows in range(total):
        distance_matrix.append([0]*total)

    #Your code here to fill distance_matrix


    # Your code ends here
    return distance_matrix
```

In the above code signature, `scanpath_dataset` is the dataset dictionary, and `grid_size` is the grid size discussed in point **c**. You may test your code on 1x1, 8x8, 16x16, and 32x32 grid sizes.

Possible reasons for disqualification of your solution:

- **Importing any library.** There shouldn't be any import statement apart from

```
from utils import load_dataset, plot_Graph
```

- **Deviating from function signature as recommended in point d.** (This is enforced to make the job of evaluators easy; 160+ submissions are a lot!!)

- **Your code will be evaluated on a different scanpath dataset** (i.e., stimulus height, width, total_subjects, and scanpaths will be different)**. We expect that the solution provided works for any 'held-out' dataset. This accounts for 70% of the marks of part A. Therefore, you must not hardcode any parameters specific to this dataset.**

2. Part B (10 Marks):

a. Use the distance matrix obtained in part A to get an MST using Prim's/Kruskal algorithm. The MST may be recorded as a list of edges in (u,v,w) format as in the code snippet.
b. Remove the largest edge in MST, this will generate two connected components.
c. Using BFS/DFS, Label nodes in the first connected components as 'blue' and in the second connected component as 'red'.
d. Use the following code snippet to plot your graph

```
from utils import load_dataset, plot_Graph

# Test drawing the connected components
# Imagine the MST is recorded as edge_list,
# each edge in edge_list follows format
# (u,v,w):u= source node, v= destination node
```

```
#          w= weight of the edge
mst_edge_list = [
        (2, 4, 5),
        (0, 3, 5),
        (1, 4, 7) ,
        (3, 5, 6),
        (4, 6, 9)
    ]

# Labels of Nodes recorded in the list
# 'red' represents Connected component 1
# 'blue' represents Connected component 2
# node 0 is red, node 1 is blue, node 2 is blue
node_labels =
['red','blue','blue','red','blue','red','blue']

#plot the graph
plot_Graph(mst_edge_list ,node_labels)
```

The above code plots the following graph: