**Assignment 3. Matching with Signature Data**

**Student ID:** m5281022 **Name:** LAI HUI SHAN

**Theoretical Background**

The task involves calculating the distance between two characters represented by their signature data using Linear and Dynamic Programming (DP) Matching techniques. These techniques are crucial for understanding the geometric variances and aligning sequences with minimal error.

**Linear Matching**

This process calculates the direct distance between points of corresponding strokes in two signatures. It assumes that each stroke's points in one signature directly correspond to the same order points in the other signature, making it simpler but less flexible than DP matching.

**Dynamic Programming Matching**

DP Matching provides a more sophisticated approach by optimizing the point-to-point correspondence between strokes to minimize the overall distance. It handles discrepancies in the number of points per stroke more effectively, allowing for non-linear matching which is crucial for accurately comparing more variable signatures.

**Algorithm Implementation**

Two distinct algorithms were implemented: one for linear matching and another for DP matching, detailed as follows.

1. **Linear Distance Calculation**

* Iterate over each stroke in the two signatures.
* For each stroke, compute the Euclidean distance between corresponding points.
* Sum these distances to get the total linear distance.

1. **Dynamic Programming Distance Calculation**

* For each stroke, construct a DP table that represents the cost of matching each point in one stroke to every point in another.
* Use recursion to fill the table with the minimum matching cost based on neighboring cells.
* The final cell of the table gives the total minimum cost for that stroke pair, summed across all strokes.

**Visualization**

Post-calculation, the matching is visualized by plotting both signatures in a 2D space and drawing lines between matched points. This visualization helps in understanding the effectiveness of the matching algorithms.

**Programming Language Used:** Python

**Required Libraries:**

* numpy
* matplotlib

**How to Run the Program:**

1. **Environment Setup:**
   * Ensure Python 3.x is installed on your system.
   * Install required libraries using the commands:

pip install numpy matplotlib

1. **Executing the Script:**
   * Save the code in a file named **signature\_matching.py**.
   * Open a command line or terminal window.
   * Navigate to the directory where the script is saved.
   * Before running the script, ensure that the SDT files are placed within the same directory as the Python script.
   * Run the script using the command:

python signature\_matching.py

**Code Explanation:**

* **parse\_sdt\_file:** Reads an SDT file and extracts stroke data grouped by each stroke's x, y coordinates and other attributes.
* **calculate\_distance:** Computes the linear distance between two signatures using Euclidean distances between corresponding stroke points.
* **dp\_matching:** Applies the DP algorithm to find the minimal cost matching between two strokes.
* **calculate\_dp\_distance:** Calculates the total DP matching distance by applying dp\_matching to each stroke pair in the signatures.
* **plot\_matching:** Visualizes the correspondence between two signatures by plotting matched strokes and connecting points.

**Example Output:**

The examples below illustrate the results of applying linear and DP matching algorithms to signature data, showcasing the precision of the matching techniques.

A graph of a line graph

Description automatically generated with medium confidence

Figure 1: Linear Matching of Two Characters

This figure, created from files **ref.sdt** and **test.sdt**, confirms that the code functions properly by showing the matching of two characters shaped like an "S". The distances computed are **39.2** for linear matching and **1645.6** for DP matching.

A graph of lines and numbers

Description automatically generated with medium confidence

Figure 2: Linear Matching of Two Signatures

This visualization uses **001.001.000.sdt** and **001.001.001.sdt** to display the complexities in matching actual signatures. It contrasts the methods with a linear matching distance of **1629.7** and a DP matching distance of **39185.8**.

|  |
| --- |
| **Source Code:** |
| import numpy as np import matplotlib.pyplot as plt import tkinter as tk from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  # Function to parse SDT files containing signature data def parse\_sdt\_file(file\_path):  data = [] # Initialize a list to store all strokes  current\_stroke = [] # Initialize a list to store the current stroke  with open(file\_path, 'r') as file:  next(file) # Skip the first line (usually header or metadata)  for line in file:  parts = line.strip().split()  if parts[0] == '-1': # Stroke delimiter  if current\_stroke: # Check if there is data in the current stroke  data.append(current\_stroke)  current\_stroke = [] # Reset current stroke for the next one  else:  # Convert coordinates and other data to integers and store them  x, y, pressure, direction, altitude, time = map(int, parts)  current\_stroke.append((x, y, pressure, direction, altitude, time))  if current\_stroke:  data.append(current\_stroke)  return data  # Function to calculate distance between two signatures using linear matching def calculate\_distance(signature1, signature2):  total\_distance = 0  N = len(signature1) # Assume both signatures have the same number of strokes  for i in range(N):  stroke1 = signature1[i]  stroke2 = signature2[i]  stroke\_distance = 0  n = min(len(stroke1), len(stroke2)) # Use the smaller number of points to avoid index errors  for j in range(n):  x1, y1 = stroke1[j][:2]  x2, y2 = stroke2[j][:2]  stroke\_distance += np.sqrt((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2)  average\_distance = stroke\_distance / n  total\_distance += average\_distance  return total\_distance  # Function to calculate distance using Dynamic Programming (DP) matching def dp\_matching(stroke1, stroke2):  n = len(stroke1)  m = len(stroke2)  dp = np.full((n + 1, m + 1), float('inf')) # Initialize DP table with infinite values  dp[0, 0] = 0  for i in range(1, n + 1):  for j in range(1, m + 1):  cost = np.sqrt((stroke1[i - 1][0] - stroke2[j - 1][0]) \*\* 2 + (stroke1[i - 1][1] - stroke2[j - 1][1]) \*\* 2)  dp[i, j] = min(dp[i - 1, j - 1] + cost, dp[i - 1, j] + cost, dp[i, j - 1] + cost)  return dp[n, m]  # Function to calculate DP matching distance for two signatures def calculate\_dp\_distance(signature1, signature2):  total\_distance = 0  N = len(signature1)  for i in range(N):  stroke1 = signature1[i]  stroke2 = signature2[i]  stroke\_distance = dp\_matching(stroke1, stroke2)  total\_distance += stroke\_distance  return total\_distance  # Function to plot the matching of two signatures def plot\_matching(signature1, signature2):  fig, ax = plt.subplots(figsize=(10, 6))  scale = 1.0 # Scaling factor for visualization  for stroke1, stroke2 in zip(signature1, signature2):  x1, y1 = zip(\*[(x \* scale, y \* scale) for x, y, \_, \_, \_, \_ in stroke1])  x2, y2 = zip(\*[(x \* scale, y \* scale) for x, y, \_, \_, \_, \_ in stroke2])  ax.plot(x1, y1, 'r-') # Plot stroke1  ax.plot(x2, y2, 'b-') # Plot stroke2  # Draw matching lines between points  for (x1i, y1i), (x2i, y2i) in zip(zip(x1, y1), zip(x2, y2)):  ax.plot([x1i, x2i], [y1i, y2i], 'grey', linestyle='--', linewidth=0.3)  ax.set\_title('Linear Matching of Two Signatures')  ax.legend(['Signature 1', 'Signature 2', 'Matching'])  plt.show()  # Main function def main():  signature\_data1 = parse\_sdt\_file('001.001.000.sdt')  signature\_data2 = parse\_sdt\_file('001.001.001.sdt')  distance = calculate\_distance(signature\_data1, signature\_data2)  dp\_distance = calculate\_dp\_distance(signature\_data1, signature\_data2)  print("Linear Matching Calculated Distance:", distance)  print("DP Matching Calculated Distance:", dp\_distance)  plot\_matching(signature\_data1, signature\_data2)  if \_\_name\_\_ == '\_\_main\_\_':  main() |