DS2030 Data Structures and Algorithms for Data Science
Lab 9

October 28th, 2025

Lab Instructions

- Create a folder named **"DS2030_<RollNo.>"** (all letters in capital) in **"home"** directory. Eg- **DS2030_142402022**
- Name the script files in the given format **"<your_roll_no>_<Name>_Lab7.py"**
- Make sure the **folder, files, classes, functions and attributes** are named as instructed in the lab sheet.
- We will not be able to evaluate your work if the folder is not properly named or is not located in the home directory.
- Make sure to save your progress before leaving the lab.
- Do not shut down the system after completing the lab.
- You are not allowed to share code with your classmates nor allowed to use code from other sources.

Sequence Similarity using Longest Common Subsequence (LCS)

Problem Statement

DNA sequences often contain subsequences that reveal genetic similarity. The **Longest Common Subsequence (LCS)** identifies the longest sequence that appears in both strings in the same order (not necessarily contiguously). A longer LCS indicates a higher degree of similarity between the sequences. In this lab, you will:

- Construct the **LCS Dynamic Programming (DP) Table** for two DNA sequences.
- Recover **one LCS string** using traceback.
- Compute a **similarity score** using the LCS length.
- Generate **all possible LCS strings** (if more than one exists).
- Compare one reference DNA sequence against multiple sequences and **rank them by similarity**.

Tasks

1. **Construct the LCS DP Table**

   Implement:

   ```
   def lcs_table(X, Y):

   ```

   Instructions:

   - Create a $(len(X)+1) \times (len(Y)+1)$ table initialized to 0.

- Fill using the standard recurrence:

$$dp[i][j] = \begin{cases} dp[i-1][j-1] + 1 & \text{if } X[i-1] = Y[j-1] \\ \max(dp[i-1][j],\ dp[i][j-1]) & \text{otherwise} \end{cases}$$

- Do not print the table here.

2. **Print the DP Table**

   Implement:

```
def print_dp_table(dp, X, Y):
```

   Print the table with row and column headers for readability.

3. **Recover One LCS String**

   Implement:

```
def lcs_string(dp, X, Y):
```

   Use traceback from the bottom-right cell:

   - If characters match, move diagonally and record the character.
   - Else move toward the larger of the two adjacent values (up or left).
   - Reverse the collected characters before returning.

4. **Compute the Similarity Score**

$$\text{Similarity Score} = \frac{2 \times \text{LCS Length}}{|X| + |Y|}$$

   Implement:

```
def similarity_score(lcs_len, len_x, len_y):
```

5. **Find *All* LCS Strings**

   Implement:

```
def all_lcs(dp, X, Y):
```

   Use recursive backtracking and return a **set** of all distinct LCS strings.

   Hint: Start recursion from the last cell (i = len(X), j = len(Y)) of the DP table.

6. **Rank Multiple Sequences by Similarity**

   Given one reference sequence X and a list of sequences:

```
def rank_sequences_by_similarity(X, sequence_list):
```

   For each sequence:

   - Compute LCS length with X.
   - Compute similarity score.

   Print sequences sorted from most similar to least similar.

   Only print the final ranking. Do not print DP tables for these sequences.

Clarification on Sequence Roles

Throughout this lab:

- X is the **reference sequence**.
- Y is the sequence used to construct the DP table, extract the LCS length, recover one LCS string, find all LCS strings, and compute the similarity score.
- `sequence_list` is a list of additional sequences that will each be compared to X (not to Y) for similarity ranking.

Thus:

- **All LCS strings** are computed between X and Y.
- **Similarity Rankings** compare X to every sequence in `sequence_list`.

Instructions

- Do **not** modify function names: `lcs_table`, `lcs_string`, `similarity_score`.
- Use only standard Python (no external libraries).
- Print the DP table in a readable grid format.
- Include appropriate comments to document the code you have implemented.
- **Make sure to save the file before final submission to avoid discrepancies in the file used for evaluation.**

Starter Code

```
def lcs_table(X, Y):
    """
    Construct the DP table for Longest Common Subsequence (LCS).

    Parameters:
        X (str): First sequence
        Y (str): Second sequence

    Returns:
        dp (list of lists): The completed DP table, without printing it.

    Instructions:
    1. Create a 2D list dp with dimensions (len(X)+1) x (len(Y)+1), and initialize all
     cells to 0.
    2. Fill the dp table using the standard LCS recurrence:
         If X[i-1] == Y[j-1]:
             dp[i][j] = dp[i-1][j-1] + 1
         Else:
             dp[i][j] = max(dp[i-1][j], dp[i][j-1])
    3. Do NOT print the table here. Just return it.
    """
    pass


def print_dp_table(dp, X, Y):
    """
    Print the DP table in a readable, labeled grid format.

    Instructions:
```

```
    - Print column headers (characters of Y).
    - Print row headers (characters of X).
    - Ensure numbers align consistently for readability.
    """
    pass


def lcs_string(dp, X, Y):
    """
    Recover ONE LCS string from the DP table using traceback.

    Returns:
        A single valid LCS string.

    Instructions:
    1. Start from the bottom-right corner of the dp table.
    2. While both i > 0 and j > 0:
         - If characters match, append that character to the result and move
    diagonally (i-1, j-1).
         - Else move in the direction of the larger dp value (either upward or
    leftward).
    3. Reverse the collected characters before returning, since traceback builds it
    backwards.
    """
    pass


def similarity_score(lcs_len, len_x, len_y):
    """
    Compute similarity score between two sequences:

        Score = (2 * LCS Length) / (len(X) + len(Y))

    Returns a value between 0 and 1.

    Instructions:
    - Ensure float division is used.
    - Round only when printing, not here.
    """
    pass


def all_lcs(dp, X, Y):
    """
    Return a set of ALL distinct LCS strings (not just one).

    Instructions:
    1. Use recursion + backtracking starting from dp[len(X)][len(Y)].
    2. If characters match     include the character and move diagonally.
    3. If characters do not match:
         - Explore upwards if dp[i-1][j] is equal to dp[i][j].
         - Explore leftwards if dp[i][j-1] is equal to dp[i][j].
    4. Use a Python set to avoid duplicate LCS strings.
    5. This function should NOT print anything; only return the set.
    """
    pass


def rank_sequences_by_similarity(X, sequence_list):
    """
    Rank multiple sequences by similarity to a reference sequence X.

    Instructions:
    1. For each sequence in sequence_list:
         - Compute its DP table with X.
         - Recover one LCS length.
```

```
        - Compute similarity score.
    2. Store results in a (sequence, score) list.
    3. Sort the list in descending order of similarity score.
    4. Print ranking in a clean, numbered format.

    Note:
    - Do NOT print any DP tables here.
    - Only print the final ranking.
    """
    pass
```

Test Function

```
def main():
    # Given sequences for primary comparison
    X = "ACGTACGTGAC"
    Y = "TGCATCGTAC"

    # Additional sequences for similarity ranking
    sequence_list = ["TGCATCGTAC", "ACGACG", "GTCAGTCA"]

    print("Sequence X:", X)
    print("Sequence Y:", Y)

    # Step 1: Construct DP table and print it
    dp = lcs_table(X, Y)
    print_dp_table(dp, X, Y)

    # Step 2: Recover one LCS string
    one = lcs_string(dp, X, Y)
    print("\nLCS Length:", len(one))
    print("One LCS:", one)

    # Step 3: Compute similarity score
    score = similarity_score(len(one), len(X), len(Y))
    print("Similarity Score:", round(score, 3))

    # Step 4: Print ALL LCS strings
    print("\nAll LCS Strings:")
    all_strings = all_lcs(dp, X, Y)
    for s in sorted(all_strings):
        print(s)

    # Step 5: Rank additional sequences
    print("\nSimilarity Rankings:")
    rank_sequences_by_similarity(X, sequence_list)


if __name__ == "__main__":
    main()
```

Sample Output

```
Sequence X: ACGTACGTGAC
Sequence Y: TGCATCGTAC

DP Table (X vs Y):
       T  G  C  A  T  C  G  T  A  C
     0  0  0  0  0  0  0  0  0  0  0
A    0  0  0  0  1  1  1  1  1  1  1
C    0  0  0  1  1  1  2  2  2  2  2
G    0  0  1  1  1  1  2  3  3  3  3
T    0  1  1  1  1  2  2  3  4  4  4
A    0  1  1  1  2  2  2  3  4  5  5
C    0  1  1  2  2  2  3  3  4  5  6
G    0  1  2  2  2  2  3  4  4  5  6
T    0  1  2  2  2  3  3  4  5  5  6
G    0  1  2  2  2  3  3  4  5  5  6
A    0  1  2  2  3  3  3  4  5  6  6
C    0  1  2  3  3  3  4  4  5  6  7

LCS Length: 7
One LCS: ATCGTAC
Similarity Score: 0.667

All LCS strings:
ATCGTAC
CACGTAC
CTCGTAC
GACGTAC
GTCGTAC
TACGTAC

Similarity Rankings:
1. ACGACG        Score = 0.706
2. TGCATCGTAC    Score = 0.667
3. GTCAGTCA      Score = 0.632
```