Math 560 Project 4: Edit Distance Spring 2024

Due: Wednesday, April 17th

In this project, you will implement the Edit Distance algorithm for two strings to determine the number of edits required to convert one string into the other. You will be required to implement your solution in its respective location in the provided `project4.py` file.

Contents

1 Problem Statement

Your primary task in this project is to implement the function `ED` in `project4.py` so that it correctly performs the Edit Distance algorithm using dynamic programming.

1.1 `ED` Function

The function `ED` will take in two strings, `src` and `dest`, and return two outputs:

- The number of edits required to convert `src` into `dest`, where edits can be insertions, deletions, or substitutions.
- The list of edits to perform. This list must have a very specific format, discussed below.

You are required to implement a dynamic programming approach to this problem, and then reconstruct an optimal set of edits using your DP table. You may set up the table in any acceptable fashion and fill the table in any acceptable order. While there may be more than one optimal set of edits, your function is only required to return one of them.

Hint: carefully watch the indices you use for the edits (see below for discussion of formatting).

Note: the `ED` function takes in an optional input `prob`. See the discussion of the provided tests for what to do with this input.

1.2 Formatting the Edit Output

The format of the edits output by the `ED` must follow a few specific guidelines:

- The output should be a list of edits.
- The edits should be provided in order **starting from the end of the string**.
- Each edit in the list should be a 3-tuple, with components:

  0. One of the following strings: `‘insert’`, `‘delete’`, `‘sub’`, `‘match’`.
  1. The char to be inserted, deleted, substituted, or matched (for substitution, note that the letter should be the new letter that replaces the old one).
  2. The index of this edit/match **in the original string**.

As an example, consider the case where `src = spam` and `dest = pims`. An optimal set of edits would be to insert the ‘s’ at the end, match the ‘m’, replace the ‘a’ with the ‘i’, match the ‘p’, and delete the ‘s’ at the beginning. This would be encoded as the list:

```
[ (‘insert’, ‘s’, 4), (‘match’, ‘m’, 3), (‘sub’, ‘i’, 2), (‘match’, ‘p’, 1), (‘delete’, ‘s’, 0) ]
```

Another example is `src = libate` and `dest = flub`. An optimal set of edits would be to delete the ‘e’, delete the ‘t’, delete the ‘a’, match the ‘b’, replace the ‘i’ with ‘u’, match the ‘l’, and insert an ‘f’. This would be encoded as the list:

```
[ (‘delete’, ‘e’, 5), (‘delete’, ‘t’, 4), (‘delete’, ‘a’, 3), (‘match’, ‘b’, 2),
  (‘sub’, ‘u’, 1), (‘match’, ‘l’, 0), (‘insert’, ‘f’, 0) ]
```

2 Provided Tests

2.1 Test Cases

A number of simple tests have been provided in the `edTests` function in the `p4tests.py` file. The tests are:

1. `‘spam’` to `‘pims’`
2. `‘libate’` to `‘flub’`
3. `‘’` to `‘abc’`
4. `‘abc’` to `‘’`
5. `‘aaa’` to `‘bbb’`

The first two tests are simple examples we saw in class. The last three tests are meant to provide special cases for your function to consider: all insertions, all deletions, and all substitutions. These should help identify specific bugs in your algorithm.

Note that the `edTests` function is called in the main function of `project4.py`. If you set its `verbosity` input to `True`, it will print more informative information beyond simply whether the tests passed or failed.

You should also note the makeEdits function in p4tests.py that takes in a source string and a list of properly formatted edits, and actually performs the edits. This is how your output will be tested.

2.2 Genomes and Strings

There are four other functions included in the p4tests.py file that involve a few interesting uses of the edit distance function. Two of these functions are simple helper functions: getRandGenomes will return two random strings made up of chars 'A', 'G', 'T', 'C'; getRandStrings will return two random strings made up of the 26 chars of the English alphabet.

The other two functions, compareGenomes and compareRandStrings perform some more interesting tasks. They each perform some number of trials (input into the functions), where they generate two random strings for each trial and calculate the edit distance. They track the average edit distance and report the results.

The main function of project4.py calls both of these functions to perform 30 trials with strings of length 300, and prints the results. For these tests, you should see that the genomes give an average error around 0.54, while the random strings give an average error around 0.9.

Now, as you might hope, this topic of average edit distance has been studied. Unfortunately, I have not been able to find a good resource examining our more standard edit distance problem. But, I have found a resource for a closely related problem, approximate string matching (ASM)[1].

For the ASM problem, we ignore any insertions that would need to occur to the left of the first edit. This means that the only difference between ASM and ED is in one of the base cases: the first row of the table should be filled with 0s for the ASM problem.

You should alter your ED function: when the optional input prob is set to 'ASM', your function should fill the first row of the table with 0s at the start of the algorithm.

Without going into too much detail, the approximation for the error in ASM is $1 - 1/\sqrt{\sigma}$, where σ is the number of characters in the alphabet. For our genome tests, this would be 1/2, and for the random strings of English characters, this would be about 0.8. The main function of project4.py will run the necessary tests and print these values as well.

[1] http://www.eecs.ucf.edu/ dcm/Teaching/COT4810-Spring2011/Literature/ApproximativeStringMatching.pdf

3 Submission

You **must** submit your `project4.py` code online on Gradescope. If you worked with a partner, only submit one version of your completed project and indicate clearly the names and NetIDs of both partners.

4 Pair Programming

You are allowed to work in pairs for this project. If you elect to work with a partner:

- You should submit only one version of your final code. Have one partner upload the code on their Gradescope site. Make sure that both partner's names are clearly indicated at the top of the code.
- When work is being done on this project, both partners are required to be physically present at the machine in question, with one partner typing ('driving') and the other partner watching ('navigating').
- You should split your time equally between driving and navigating to ensure that both partners spend time coding.
- You are allowed to discuss this project with other students in the class, but the code you submit must be written by you and your partner alone.

5 Style Points

Part of your grade for this project will be 'style points'. The idea here is that the code you turn in must be well commented and readable. A reasonable user aught to be able to read through your provided code and be able to understand what it is you have done, and how your functions work. This means that a grader should be able to read over your code and tell that your algorithms are implemented correctly.

The guidelines for these 'style points':

- Your program file should have a header stating the name of the program, the author(s), and the date.
- All functions need a comment stating: the name of the function, what the function does, what the inputs are, and what the outputs are.
- Every major block of code should have a comment explaining what the block of code is doing. This need not be every line, and it is left to your discretion to determine how frequently you place these comments. However, you should have enough comments to clearly explain the behavior of your code.
- Please limit yourself to 80 characters in a line of code. In python, you can use the symbol \ to indicate a line break/continuation.