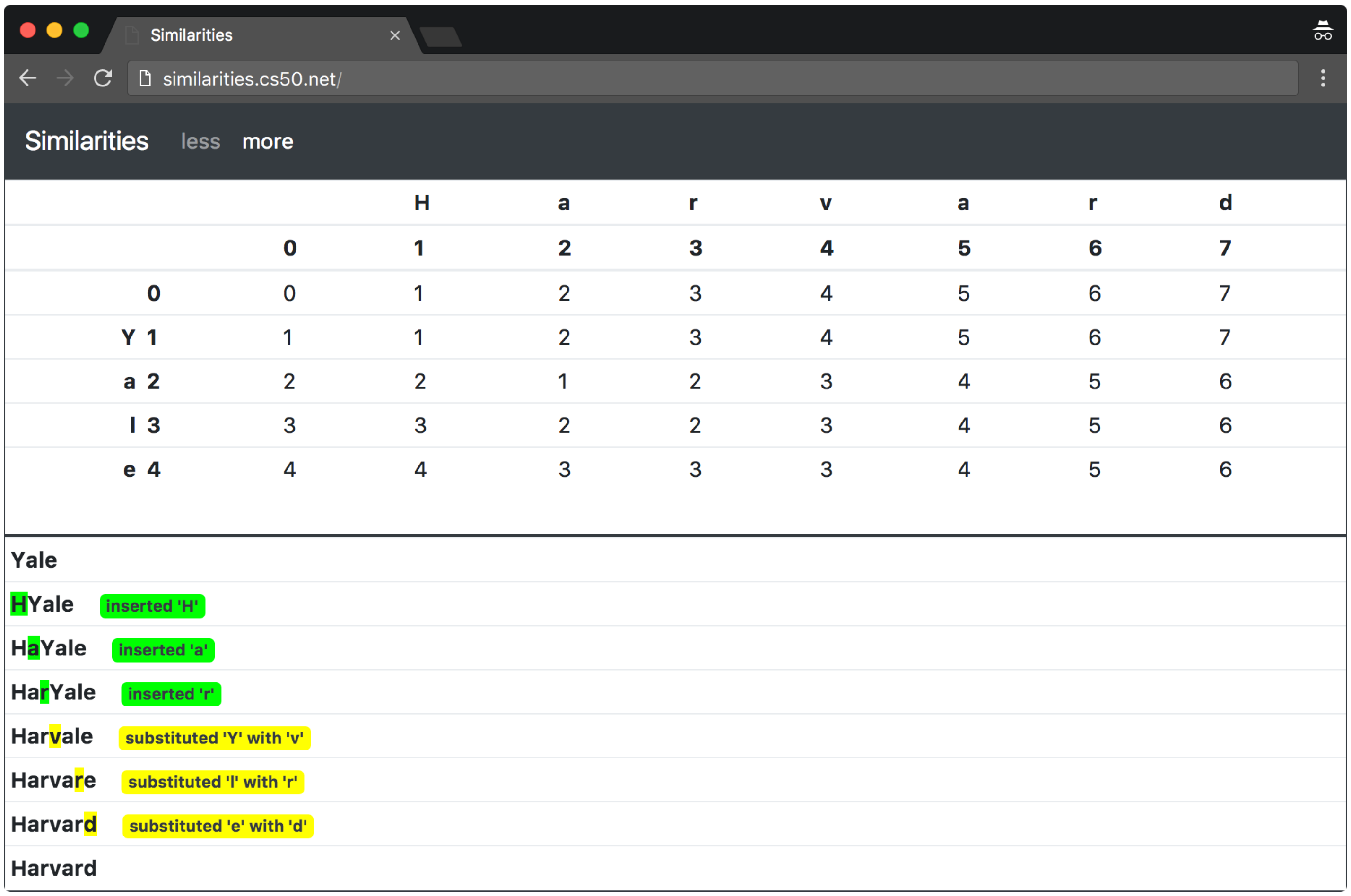
Table of Contents

Similarities

tl;dr

1. Implement a program that measures the edit distance between two strings.
2. Implement a web app that depicts the costs of transforming one string into another, a la the below.



Similarities less more

		H	a	r	v	a	r	d
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
Y 1	1	1	2	3	4	5	6	7
a 2	2	2	1	2	3	4	5	6
l 3	3	3	2	2	3	4	5	6
e 4	4	4	3	3	3	4	5	6

Yale

HYale inserted 'H'

HaYale inserted 'a'

HarYale inserted 'r'

Harvale substituted 'Y' with 'v'

Harvare substituted 'l' with 'r'

Harvard substituted 'e' with 'd'

Harvard

Background

Determining whether two strings are identical is (relatively!) trivial: iterate over the characters in each, checking whether each and every one is identical. But it's non-trivial to quantify just how dissimilar two (non-identical) strings are. And it can be time-consuming, as there are multiple (and often many!) ways to transform one string into the other.

The challenge ahead is to measure the "edit distance" between two strings, the minimal number of additions, deletions, and/or edits necessary to transform one string into the other.

Distribution

Downloading

```
$ wget http://cdn.cs50.net/2017/fall/psets/6/similarities/more/similarities.zip (ht
$ unzip similarities.zip
$ rm similarities.zip
$ cd similarities
$ chmod a+x score
$ ls
application.py  helpers.py  requirements.txt  score*  static/  templates/
```

Understanding

`score`

Open up `score`. Suffice it to say that file's name doesn't end in `.py`, even though the file contains a program written in Python. But that's okay! Notice the "shebang" atop the file:

```
#!/usr/bin/env python3
```

That line tells a computer to interpret (i.e., run) the program using `python3` (aka `python` on CS50 IDE), an interpreter that understands Python 3.

Notice how the file defines a function called `main` and calls that function toward the bottom of the file. Defining `main` isn't strictly necessary in Python, but it's not uncommon.

Notice how `score` uses Python's argparse (https://docs.python.org/3/library/argparse.html) module in order to parse two command-line arguments, `FILE1` and `FILE2`, the files to compare. The program then tries to read the contents of those files into strings, `file1` and `file2`. If something goes wrong, as indicated by an `IOError`, the "exception" is caught. See https://docs.python.org/3/tutorial/errors.html (https://docs.python.org/3/tutorial/errors.html) for more on exceptions.

Finally, the program passes those strings to `distances`, a function we'll soon see, and ultimately prints the edit distance between the two files!

`helpers.py`

Open up `helpers.py`. Ah, the familiar `TODO`. Declared in this file is a function called `distances` that takes two strings as arguments, `a` and `b`, and is supposed to return (via a matrix of costs) the edit distance between one and the other. At the moment, though, it simply returns an empty two-dimensional `list`!

This file also defines an "enumeration" (i.e., `Enum`) that essentially defines three constants, each of which represents an operation via which a string might be transformed into another: `Operation.DELETED`, `Operation.INSERTED`, and `Operation.SUBSTITUTED`.

`application.py`

Open up `application.py`. This file implements a web application that, ultimately, will allow you to visualize the edit distance between two strings as well as the operations necessary to transform one into the other at minimal cost. No need to understand the entirety of this file, but notice how `score` infers from the matrix returned by `distances` the sequence of operations that yield that minimal cost.

`templates/layout.html`

Open up `templates/layout.html`. In this file is a template for the web application's overall layout. Odds are you'll recognize a few of the HTML tags therein and notice a few new ones. Notice, in particular, how the template uses Bootstrap, a popular library. In fact, we based this template on their own starter template (http://getbootstrap.com/docs/4.0/getting-started/introduction/).

`templates/index.html`

Open up `templates/index.html`. Ah, another `TODO`. Notice how this template "extends" `layout.html`, which is to say that `layout.html` is the "mold" from which `index.html` itself will be made. The `block` defined in `index.html` will effectively get plugged into the placeholder for `block` in `layout.html`.

Ultimately, this file will contain the form via which users will be able to submit two strings to your web application for comparison.

`templates/score.html`

Open up `templates/score.html`. We took the liberty of implementing this file for you. Thanks to its use of some CSS (particularly a class called `row`), it ensures that `matrix.html` will fill the top half of a browser's viewport and that `log.html` will fill the bottom half of the same.

`templates/matrix.html`

Open up `templates/matrix.html`. Ah, another `TODO`. It's via this file that you'll need to generate an HTML table that depicts the costs via which one string can be transformed into another.

`templates/log.html`

Open up `templates/log.html`. Phew, looks like we implemented this file for you. Indeed, it's via this file that your web app will generate an HTML table that summarizes the operations via which one string can be transformed into another.

`templates/error.html`

Open up `templates/error.html`. In this file is a template with which any HTTP errors will be displayed. It happens to use Bootstrap's Jumbotron (https://getbootstrap.com/docs/4.0/components/jumbotron/) feature.

`static/styles.css`

Open up `static/styles.css`. In this file are some CSS properties that collectively implement your web application's user interface. Essentially, they modify some of Bootstrap's own defaults.

`requirements.txt`

Open up `requirements.txt` (without changing it, though you can later if you'd like). This file specifies the libraries, one per line, on which all of this functionality depends.

Specification

`helpers.py`

`distances`

Implement `distances` in such a way that, given two strings, `a` and `b`, it calculates the edit distance from `a` to `b`, returning (as a `list` of `lists`) the matrix of operational costs incurred along the way. Treat the matrix's top-left corner as `[0][0]` and the matrix's bottom-right corner as `[len(a)][len(b)]`. Stored in each element of the matrix should be a `tuple`, `(cost, operation)`, where `cost` is an `int` and `operation` is an `Operation`.

`templates/index.html`

Implement `templates/index.html` in such a way that it contains an HTML form via which a user can submit:

- a string called `string1`
- a string called `string2`

You're welcome to look at the HTML of the staff's solution as needed, but do try to figure out the right syntax on your own first, as via https://www.google.com/search?q=html+forms! (https://www.google.com/search?q=html+forms!)

`templates/matrix.html`

Implement `templates/matrix.html` in such a way that it generates, using Jinja2 (http://jinja.pocoo.org/), a visualization of a matrix returned by `distances` (given some `a` and `b`) via an HTML table. In each cell of the table should be only a cost, not an operation. Along the lefmost column should be the characters from `a`, each in its own cell (and row); along the topmost row should be the characters from `b`, each in its cell (and column).

Walkthroughs

Testing

To test your implementation of `distances` via the command line, execute `score` as follows, where `FILE1` and `FILE2` are any two text files:

```
./score FILE1 FILE2
```

To test your implementations via a web app, execute

```
flask run
```

and then visit the outputted URL.

See http://cdn.cs50.net/2017/fall/psets/6/similarities/inputs/ (http://cdn.cs50.net/2017/fall/psets/6/similarities/inputs/) for sample inputs, though be sure to test with some of your own!

`check50`

```
check50 cs50/2018/x/similarities/more
```

`style50`

```
style50 helpers.py
```

Staff's Solution

CLI

```
~cs50/pset6/more/score
```

Web

http://similarities.cs50.net/more (http://similarities.cs50.net/more)