CS 1110 Spring 2021, Assignment 4: Change My View

Updates:

- Thursday Mar 8, 3:45pm Ithaca time: In Fig 2 on page 5, the Post with Gorn numberstring 0.1.1.0.1 (the only one in the tree authored by "E") was missing the last digit (i.e., it mistakenly had 0.1.1.0, which was the Gorn numberstring of its parent Post).

CS 1110 Spring 2021, Assignment 4: Change My View*

The files you'll need are in this zip file:
http://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment4/a4_skeleton.zip . We've provided testing functions that (we hope) are sufficient, so you don't need to submit your own testing code.

Warning: If you are using Python 3.6 or earlier, the testing code might need adjustment.[1]

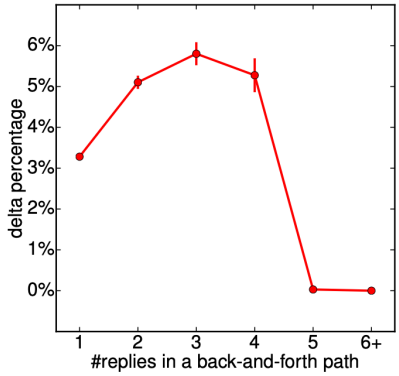
1 Motivation: (How) Do Arguments Change People's Minds?

A 2016 research paper[2] by current and former Cornellians begins,

> Changing someone's opinion is arguably one of the most important challenges of social interaction. The underlying process proves difficult to study: it is hard to know how someone's opinions are formed and whether and how someone's views shift. Fortunately, ChangeMyView, an active community on Reddit, provides a platform where users present their own opinions and reasoning, invite others to contest them, and acknowledge when the ensuing discussions change their original views. In this work, we study these interactions to understand the mechanisms behind persuasion.
>
> We find that persuasive arguments are characterized by interesting patterns of interaction dynamics, such as participant entry-order and degree of back-and-forth exchange.... [and other results too.]

Figure 1 on page 3 depicts part of a real conversation tree from the r/ChangeMyView (CMV) subreddit where such a back-and-forth occurred. In about the center of the figure, there is an extended back-and-forth between the original poster (blue/solid) and someone else (orange/horizontal stripes). On the left-hand side of the figure, we see that the original poster gave the red/vertical-striped post a "delta" (Δ, highlighted by the yellow five-pointed star) to indicate that that post shifted their thinking. But they did not give a "delta" to any of the orange/horizontal-striped posts.

Do back-and-forth exchanges correlate with probability of opinion change? Here is a figure from the paper, based on a large set of CMV conversations. While the overall shape conforms with intuition, the steepness of the decline between 4 and 5 replies is arguably surprising.[3]



Recursion is a natural tool for analyzing patterns like back-and-forths in conversation trees. So, what you're learning in CS1110 can form the basis for cutting-edge research!

*Authors: Lillian Lee. Thanks to Elisabeth Finkel and Brynn Szczesniak for comments and figure help. Errors are my own.

[1]The technical issue is whether dictionary keys are kept in insertion order. Post on Ed Discussions if you cannot use Python 3.7+.

[2]Chenhao Tan, Vlad Niculae, Cristian Danescu-Niculescu-Mizil, Lillian Lee (2016). Winning arguments: Interaction dynamics and persuasion strategies in good-faith online discussions. Proc. of WWW, pp. 613-624. https://chenhaot.com/pages/changemyview.html

[3]One might conclude from this graph, as NPR did, "After three rounds you may as well agree to disagree" (https://www.npr.org/2017/06/29/534916052/change-my-view-on-reddit-helps-people-challenge-their-own-opinions). But there's an alternation explanation: people who go multiple rounds on CMV are not the kind of people who change their minds.

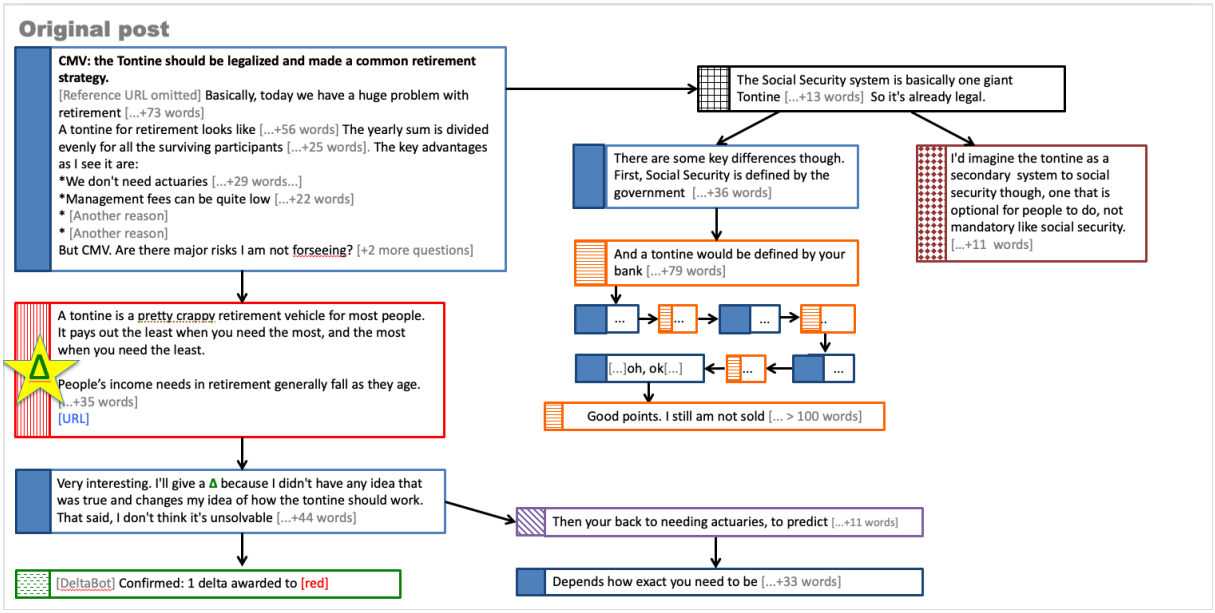


Figure 1: Portion of the conversation tree posted at https://www.reddit.com/r/changemyview/comments/3mzc6u/cmv_the_tontine_should_be_legalized_and_made_a/. Arrows connect a post to a reply to that post. Colors/patterns indicate distinct users; blue/solid is for the *original poster* starting the discussion.

Contents

2 Rules

2.1 New rules

1. A major goal of this assignment is practice with recursion. Hence *we reserve the right to assign no credit for implementations that aren't fundamentally based on recursion*, even if the code fulfills the specification.

2. You *are* allowed to use for-loops[4] as long as recursion is the fundamental basis of your implementation.

2.2 Previous rules that still apply

See Sections 1.1-1.3 of Assignment 1[5], Sections 3.1-3.2 of Assignment 2[6], and Section 2 item 3 of Assignment 3[7].

3 Timeline and Deadlines

(a) If you are partnering:[8] do so well *before* submitting.

(b) By 2pm Ithaca time on Tue Apr 13, submit whatever you have done at that point on to CMS.[9]

(c) By **11:59pm (Ithaca time) on Tue Apr 13**, make your final submission.[10]

4 Representing and Describing Conversation Trees

You'll be writing functions that would help analyze conversation trees, and in particular back-and-forths along paths in conversation trees.[11]

4.1 Tree concepts and main example tree

We'll use Figure 2 on page 5 as a running simplified example of a conversation tree.

The circles represent all Reddit posts that are "reachable" from a given original post by following replies to posts, where replies are indicated by arrows in the diagram. The original post is included in the set of "reachable" posts.

The string inside each circle is the username of the author of that post.

We say that a post is a *leaf* if it has no replies.

Every post in a conversation tree can be considered to be the "original post" of the *subtree* consisting of the posts "reachable" from it.

4.2 Gorn numberstrings: convenient notation for referencing posts in a tree

Gorn numbering is a convenient way to refer to particular posts in a tree. In Figure 2, the Gorn numberstring for each post is to the post's right.

The scheme has a recursive structure:[12]

- The original post has Gorn numberstring `0`.
- Suppose a particular post has Gorn numberstring s. Then, treating the replies to that post as a list, the reply at index i in the reply list has the Gorn numberstring created by concatenating "$.i$" to s.

Looking at Figure 2, for example, the original post (at the top) has Gorn numberstring `0`. There are two replies to the original post; the one authored by "B" has Gorn numberstring `0.0`, and the one by "D" has Gorn numberstring `0.1`. The two replies to the post by "B" have Gorn numberstrings `0.0.0` and `0.0.1`, respectively.

In general, the more "dots" in a Gorn numberstring, the deeper the post is in the tree. And the bigger a "digit" is in a Gorn numberstring is, the farther to the right it is.

[4] But not while-loops, if you even know what those are.

[5] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment1/a1.pdf

[6] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment2/a2.pdf

[7] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment3/a3.pdf

[8] Reminder: Both parties need to act on CMS in order for the grouping to take effect. See the "How to form a group" instructions at https://www.cs.cornell.edu/courses/cs1110/2021sp/resources/cms.html .

[9] It is OK if you haven't finished the assignment yet; the 2pm checkpoint provides you a chance to alert us if any problems arise, and us to alert you if your submission seems to be missing and of the deadline that day. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason. There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy. Of course, if some special circumstances arise, contact the instructor(s) immediately.

[10] And, as usual, perform steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of https://www.cs.cornell.edu/courses/cs1110/2021sp/resources/cms.html .

[11] But to keep the scope of this assignment manageable, we will not be implementing a full application like we did in A3.

[12] As do many concepts regarding trees.

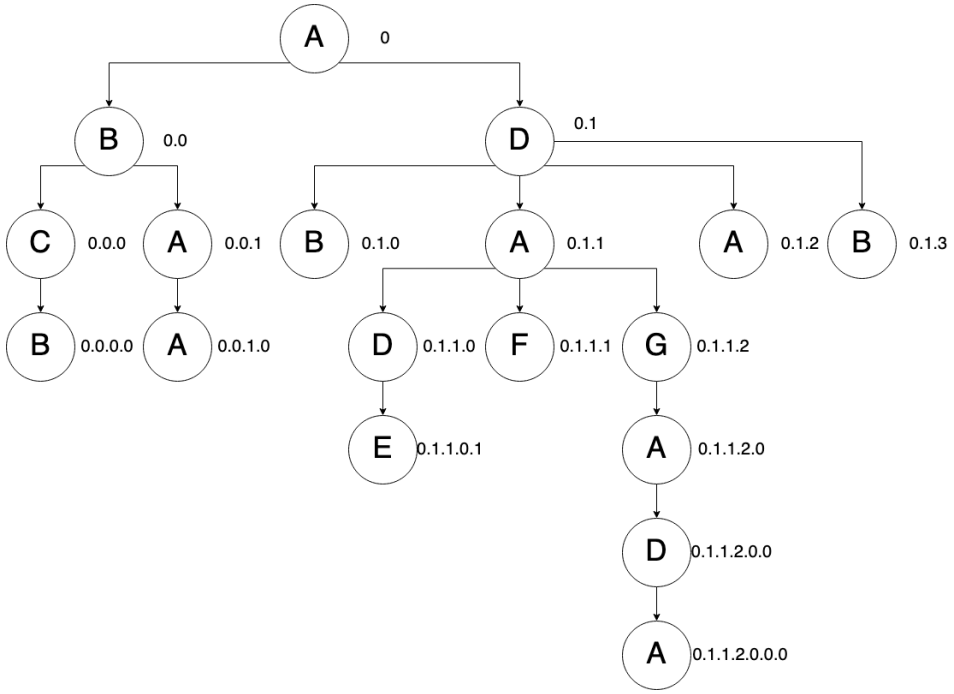


Figure 2: Main example tree. The function `main_example_tree()` in file `a4_test.py` creates this tree for you. The original post is at the top. Each post has its author's username inside it. The Gorn numberstring for each post is to the post's right.

4.3 Class Post and function `main_example_tree()`

File `post.py` defines class Post for representing Reddit posts. But you don't need to look inside that file; all you need to know about a Post object's three attributes is that they are:

```
tag [non-empty str]:  A tag string distinct from all other Posts

author [non-empty str]:   username of the author of this Post

replies [list of Posts, possibly empty]: direct replies to this Post, ordered by tag
```

The function `main_example_tree()` in file `a4_test.py` creates our main example tree from Figure 2 for you. This function sets the `tag` attribute for each post to its the Gorn numberstring but with the periods removed.

5 Task 1: A Recursive Implementation of `num_user_posts()`

One sign that a (long) back-and-forth could be happening is a user making multiple posts in the tree.

This motivates the following function for you to implement.

```
def num_user_posts(post, u):
    """Returns: The number of Posts whose listed author is u, from among all
    Posts "reachable" from `post` via `replies` attributes.
    The "reachable" Posts are:
        * the Post `post` itself
        * the Posts in post.replies
        * the Posts in the replies list of the Posts in post.replies ... and so on.

    Precondition:
        post: a Post object.
        u: a non-empty string (meant to be a username).
    """
```

We've written a test function `test_num_user_posts()` for you in `a4_test.py`. It's complicated,[13] but you can understand it as follows.

It contains lines like these:

```
print("\tTesting 0.1 in main example tree")
post = main_dict["0.1"]
true_num =  {"A": 4, "B": 2, "C": 0, "D": 3, "E": 1, "F": 1, "G": 1}
```

What this means is that for the subtree of our main example tree starting at post `0.1`, there are 4 posts authored by user "A", 2 by user "B", and so on. For the full main example tree, the number of posts by each user is given by this dictionary:

`true_num = {"A": 7, "B": 4, "C": 1, "D": 3, "E": 1, "F": 1, "G": 1}`

See the testing code for other examples/testcases.

6 Task 2: A Recursive Implementation of `user_paths()`

We might be interested in conversation *paths* — chains of replies starting at a particular Post and continuing until one hits a leaf Post — since back-and-forths occur along such paths.

So, one way to look for back-and-forths is to have an explicit listing of all the paths in a conversation, as strings, because then we could search for particular patterns in those strings. This motivates the following function for you to implement.

```
def user_paths(post):
    """Returns: list of strings, one for each reply-path starting from `post`
    and going all the way down to a leaf post (one with no replies to it).
    Each string is a comma-separated sequence of authors along the path, in order.

    If `post` has no replies, the only string to be included is the one
    containing only the author of `post`.

    Preconditions: `post` is a Post object.
    """
```

We've written a test function `test_user_paths()` for you in `a4_test.py`. Again, it's complicated, but you can understand it as follows.

It contains lines like these:

```
print("\tTesting on main example '0'")
true_paths = ['A, B, C, B',
              'A, B, A, A',
              'A, D, B',
              'A, D, A, D, E',
              'A, D, A, F',
              'A, D, A, G, A, D, A',
              'A, D, A',
              'A, D, B']
```

gives is the list of the strings that your code should output if it were given our main example tree's original post as input. (It' fine for your code to output the same strings in a different order.)

As another example, these lines show the three strings your code should output (although perhaps in a different order) if it were given the Post at `0.1.1` as input.

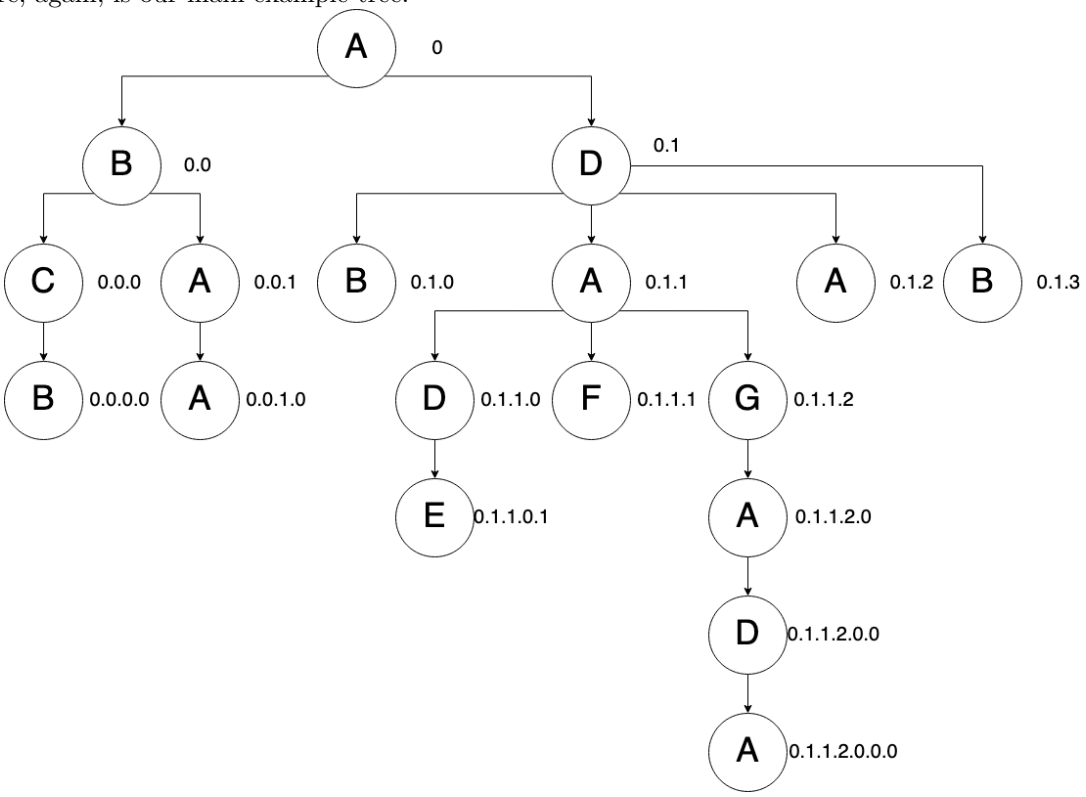
[13] If you have plenty of time, it would be educational to try to understand the testing code, which features dictionaries quite a bit. But it is definitely OK if you don't have time for such an exercise.

```
print("\tTesting on '0.1.1', three paths out")
    result = a4.user_paths(main_dict["0.1.1"])
    true_paths = ['A, D, E',
                  'A, F',
                  'A, G, A, D, A']
```

See the testing code for other examples/testcases.

7 Task 3: A Recursive Implementation of `bnf_starts_here()`

Here, again, is our main example tree:



We observe that there are exactly 7 back-and-forths of (exactly) length 3:
`0` → `0.0` → `0.0.1` between authors "A" and "B"
`0` → `0.1` → `0.1.2` between "A" and "D"
`0.1.1.2.0` → `0.1.1.2.0.0` → `0.1.1.2.0.0.0` between "A" and "D"
`0.0` → `0.0.0` → `0.0.0.0` between "B" and "C"
`0.1` → `0.1.1` → `0.1.1.0` between "D" and "A"
`0.1.1` → `0.1.1.2` → `0.1.1.2.0` between "A" and "G"

But we **don't** want to say that there is a length-3 back-and-forth starting at `0` between "A" and "D". This is because the "ADA"-labeled sequence `0` → `0.1` → `0.1.1` *can be continued* with the "D"-authored post `0.1.1.0`; so we'd rather say instead that there is a length-**4** back-and-forth starting at `0` between "A" and "D".

On the other hand, to make this assignment easier for you, we **do** say that there's a back-and-forth of length 3 between "D" and "A" starting at `0.1`, *even though* this back-and-forth is part of a longer one between "A" and "D" starting "one post above", at `0`.

There are also length-2 back-and-forths in the figure, such as `0.0` → `0.0.1` between "B" and "A" (where `0.0.1`) is not a leaf), and `0.1` → `0.1.3` between "D" and "B" (where `0.1.3` is a leaf.)

But similarly to before, we would **not** say `0` → `0.0` is an back-and-forth of length 2 between "A" and "B", because we'd say that actually there's a back-and-forth of length **3** between those two authors starting at `0`, namely, `0` → `0.0` → `0.0.1`.

7.1 All $k > 2$-length back-and-forths in our main example

They are all listed in file `a4_test.py` from the A4 zipfile, lines 226-249 — just focus on the strings (in red) below:

```
true_list = [
             # length-2 back-and-forths
             ["B", "A", gnlist2plist(["0.0", "0.0.1"], main_dict)],
             ["C", "B", gnlist2plist(["0.0.0", "0.0.0.0"], main_dict)],
             ["D", "E", gnlist2plist(["0.1.1.0", "0.1.1.0.0"], main_dict)],
             ["A", "D", gnlist2plist(["0.1.1", "0.1.1.0"], main_dict)],
             ["A", "F", gnlist2plist(["0.1.1", "0.1.1.1"], main_dict)],
             ["D", "A", gnlist2plist(["0.1.1.2.0.0", "0.1.1.2.0.0.0"], main_dict)],
             ["G", "A", gnlist2plist(["0.1.1.2", "0.1.1.2.0"], main_dict)],
             ["D", "B", gnlist2plist(["0.1", "0.1.0"], main_dict)],
             ["D", "A", gnlist2plist(["0.1", "0.1.2"], main_dict)],
             ["D", "B", gnlist2plist(["0.1", "0.1.3"], main_dict)],

             # length-3 back-and-forths
             ["A", "B", gnlist2plist(["0", "0.0", "0.0.1"], main_dict)],
             ["A", "D", gnlist2plist(["0", "0.1", "0.1.2"], main_dict)],
             ["A", "D", gnlist2plist(["0.1.1.2.0", "0.1.1.2.0.0","0.1.1.2.0.0.0"],
                                     main_dict)],
             ["B", "C", gnlist2plist(["0.0", "0.0.0", "0.0.0.0"], main_dict)],
             ["D", "A", gnlist2plist(["0.1", "0.1.1", "0.1.1.0"], main_dict)],
             ["A", "G", gnlist2plist(["0.1.1", "0.1.1.2", "0.1.1.2.0"], main_dict)],
             # length-4 back-and-forth
             ["A", "D", gnlist2plist(["0", "0.1", "0.1.1", "0.1.1.0"], main_dict)],
            ]
```

7.1.1 More formal, recursive definition

Starting at a particular post P with author x, we say that a back-and-forth of length $k \geq 1$ between x and (different) author y starts at P if (and only if) there is a reply chain starting at P where the author order is x, y, x, y, ..., and where if there is a reply at the kth item in the back-and-forth, it would *not* constitute a continuation of the back-and-forth — in other words, the back-and-forth cannot be extended beyond k.

We can re-formulate this definition recursively!

A back-and-forth of length k between user u_1 and (different) user u_2 starts at a post P *if (and only if)* P was written by u_1 and ...

- Case: $k = 1$: ... there is no reply to P written by u_2. (This includes the case where P has no replies at all).[14]
- Case: $k \geq 2$: ... there is a back-and-forth of length $k - 1$ between u_2 and u_1 starting at some reply to P.

7.1.2 Function specification

Here is the specification for the function you are to implement using recursion effectively.

```
def bnf_starts_here(post, u1, u2, k):
    """If, ignoring the parent of `post`, a back-and-forth of length k (and not
    longer) between users u1 and u2 starts at Post `post`,
    returns a list of Post objects in that back-and-forth, in top-down order.
    (If there is more than one such back-and-forth, this function returns one of
    them.)

    Otherwise, returns False.

    Preconditions:
```

[14] Contrariwise, no back-and-forth of length $k = 1$ between u_1 and u_2 starts at P if P wasn't written by u_1 or if there's a reply to P written by u_2.

```
    post: a Post object.
    u1 and u2: usernames: non-empty strings that are distinct.
    k: int >= 1."""
```

We've written a test function `test_bnf_starts_here()` for you in `a4_test.py`. It looks complicated, but you should look at the lines that set up `true_list`: these lines show you all the back-and-forths in our main example tree.

7.1.3 Optional: If you want to do additional testing

If you want to add additional testing — such as if you are having trouble getting any of the test cases to pass — you may want to add your own small-scale tests.

Here's an example of what we recommend: set up your own test function, like this:

```
def my_own_private_testfn():
    print("**** Running my own private test function!")

    [main_tree, main_dict] = main_example_tree()

    result = bnf_starts_here(main_dict["0"], "A", "D", 2) # replace with whatever test you like
    if result:
        print("\tHere's the length-2 back-and-forth I found at 0 for A and D") # change as appropriate
        for p in result:
            print(p.tag + " by " + p.author)
    else:
        print("No such back-and-forth here!")
    print("**** Done with my own private test function!")
```

Then add a call to this test function early in the testing code, maybe right after line 337:

```
337 if __name__ == '__main__':
338     my_own_private_test_fn()  #<-- you added this
339
340     test_num_user_posts()
341     test_user_paths()
342     test_bnf_starts_here()
```

8 Advice (Beyond What We Mentioned in the Previous Assignment)

If you find yourself making modification after modification but not making progress, *save your work*, but try starting over, on a blank sheet of paper. A fresh start can sometimes work wonders.

9 Code Cleanup and Pre-Submission Checklist

See the relevant sections in Assignment 1[15].

[15] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment1/a1.pdf