Programming Assignment (Dictionary lookup and autocomplete using Tries)

Important Information

Assignment Out
Sunday, Feb 22, 2015.

Assignment Due
Tuesday, Mar 10, 2015.

Download
Downlod a zip file.

Submission
Upload a zip file to moodle. Zip file should be called **firstname_lastname.zip** Do not press submit for grading on moodle unless you are really ready to submit.

Trie

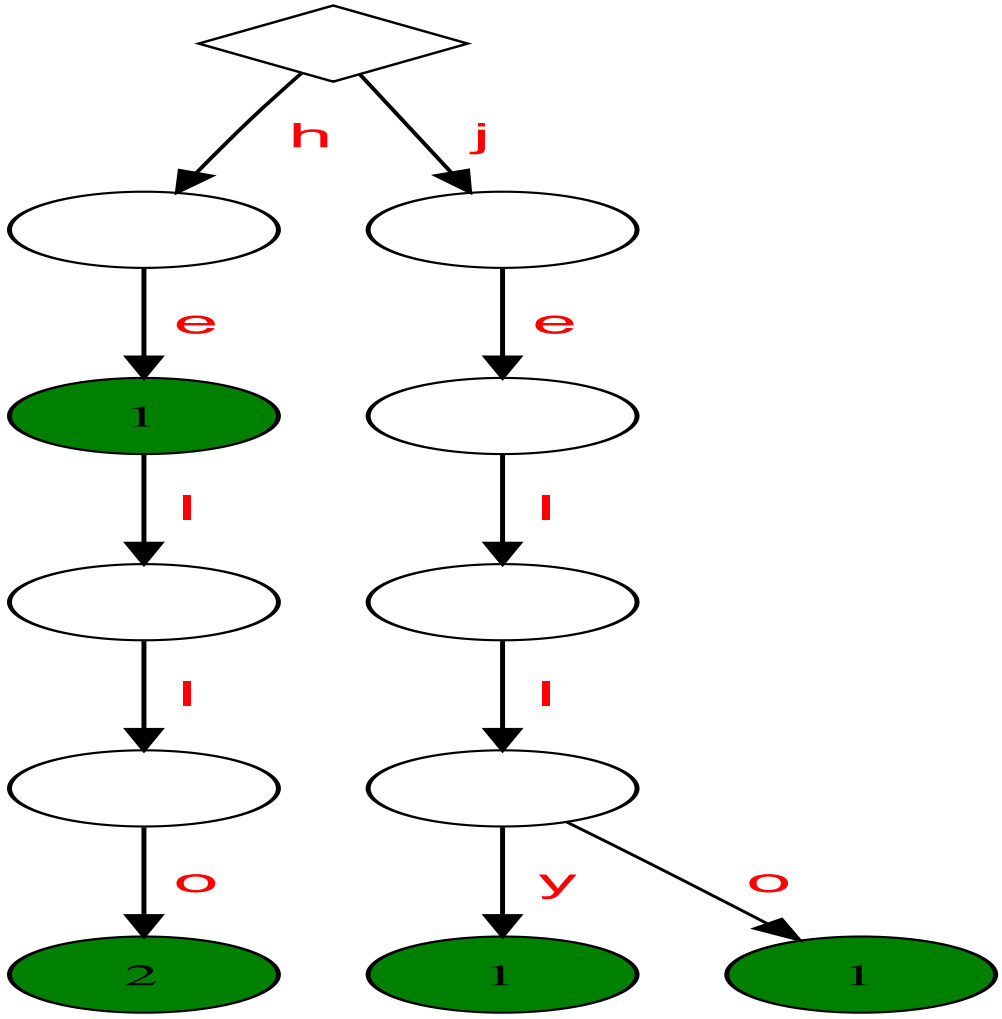
A trie is a special *rooted tree*. that allows us to store a set of words by sharing their common prefixes.

Example # 1

Consider the following list of five words:

hello, jello, he, hello, jelly

The trie data structure for this list above is given below:



A **trie node** can be of three types:

1. Root of the trie (shown as a diamond)
2. Internal non-terminal node of a trie (white ovals)
3. Word-ending node of a trie (green ovals).

Note that a word-ending node need not be a leaf. Also, such nodes are associated with a number that represents the frequency of the corresponding word.

Each **trie edge** is labelled with an alphabet.

Reading the words in the dictionary.

Starting from the root (diamond node), as we traverse a path along the trie, the letters along a path form a word. Whenever we encounter a word-ending node, we can stop and read off a valid word in the dictionary.

For instance, we note the correspondence between the list of words:

```
hello, hello, jello, he, jelly
```

and the paths in the trie.

Note

The word **he** is a prefix of the word **hello**. Therefore, in the trie, starting from the root, we see that a word-ending node corresponding to **he** is extended by the suffix “llo” to form a word-ending node for **hello**.

The frequency of occurrence of a word is the number associated with its corresponding word-ending node. Notice how the frequency of “he” is 1 but the frequency of “hello” is 2.

Finally, two words with a common prefix such as “jelly” and “jello” will share a common path corresponding to the prefix “jell” in the trie.

Trie Data Structure: Specification

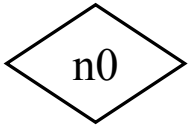
A trie (or a prefix tree) data structure allows us to store a set of strings from a dictionary along with a frequency count for each strings. The following operations should be supported:

Operation	Description	Return Type
addWord(w)	Insert a word w into the trie	None
lookupWord(w)	Return the frequency of w in the trie (0 if not present)	**int**
autoComplete(w)	Return a list of words in the trie along with their frequencies so that w is a prefix of each word in the list	**list** of (**str**,**int**)

Example

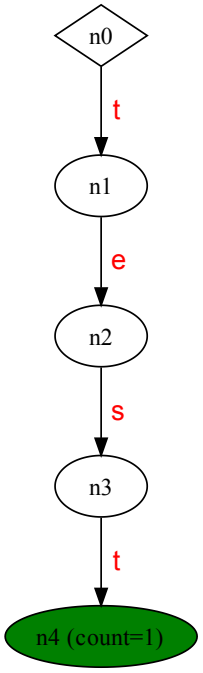
We will illustrate the process of constructing a trie using an example.

At the beginning the empty trie is simply a root node:

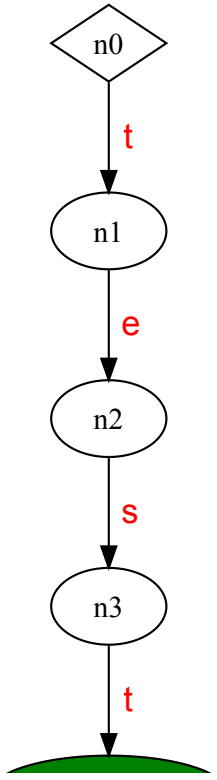


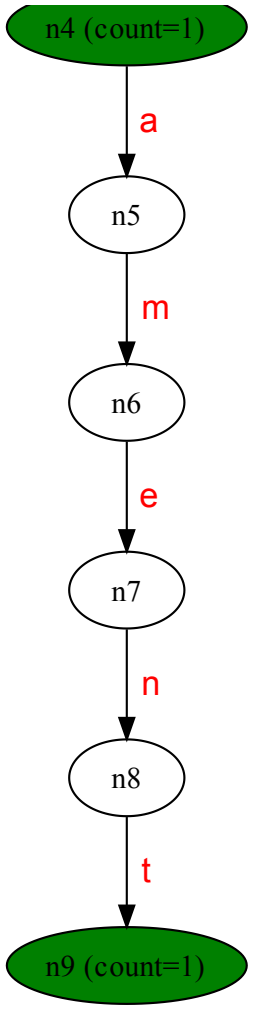
Suppose we add the strings in the following order:

'test','testament','testing','ping','pin','pink','pine','pint','testing','pinetree'

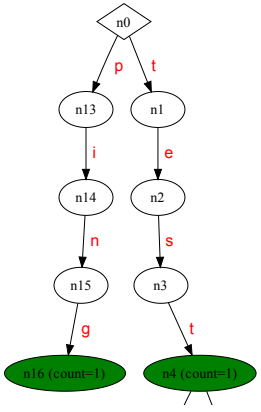


1: **addWord**('test')

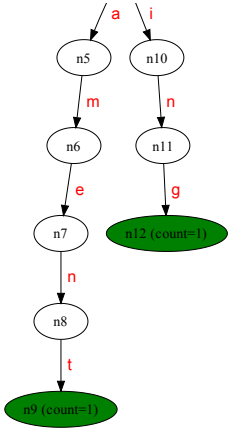




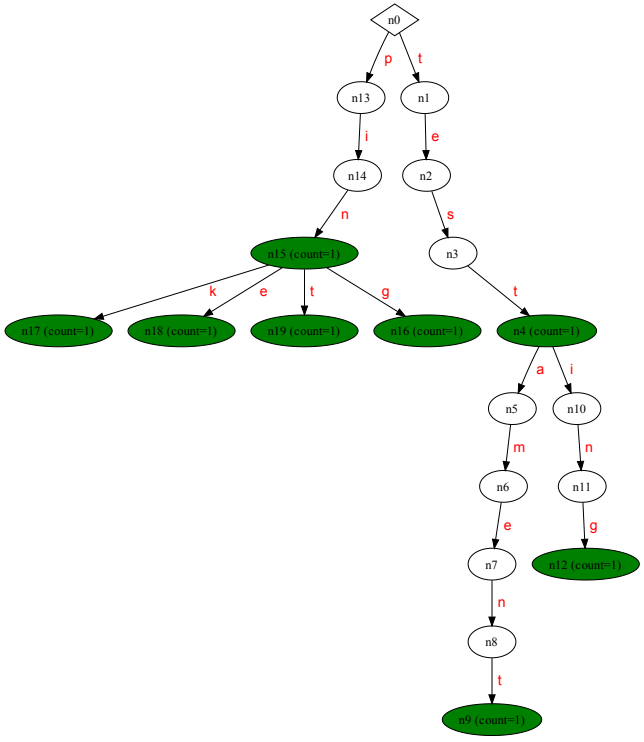
2: **addWord**('testament')



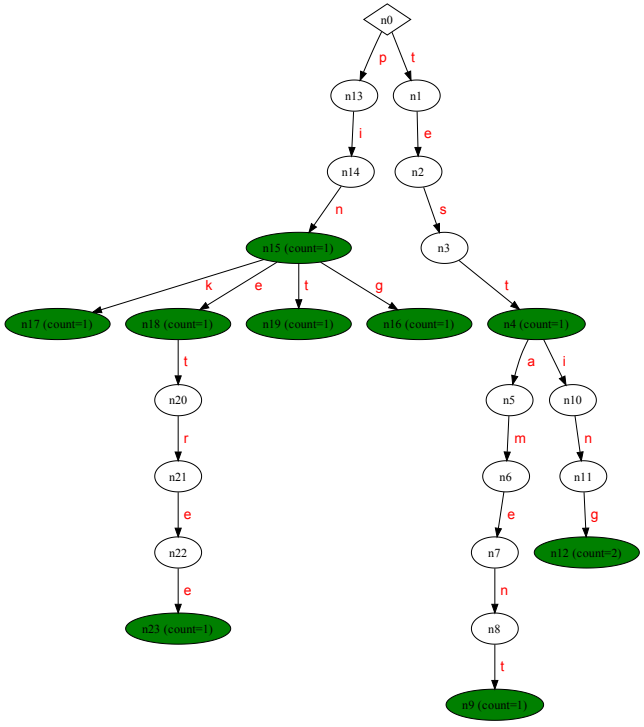
3: **addword**('testing')



4: **addWord**('ping')



5: **addWord**('pin')
6: **addWord**('pink')
7: **addWord**('pine')
8: **addWord**('pint')

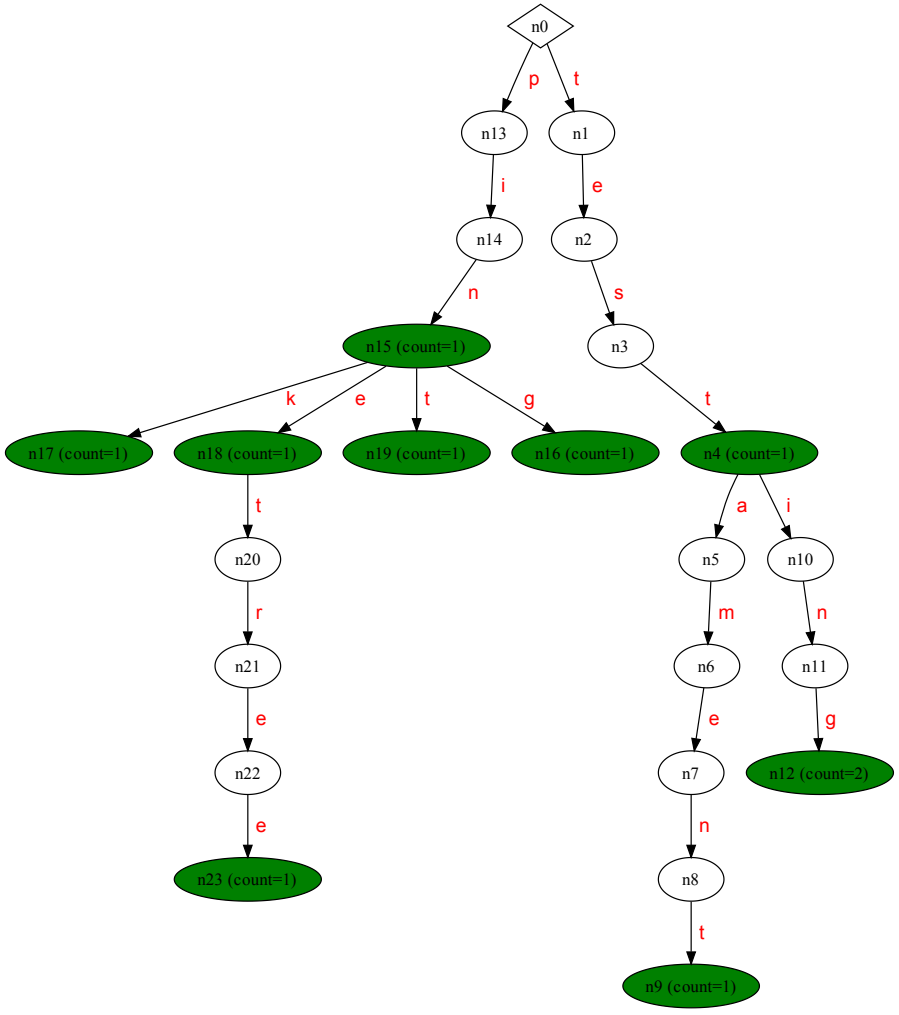


9: **addWord**('testing')
10: **addWord**('pinetree')

Lookup of a Word

The operation **lookupWord** allows us to count the frequency of a word in a trie. If a word does not belong to a trie, then the frequency is 0.

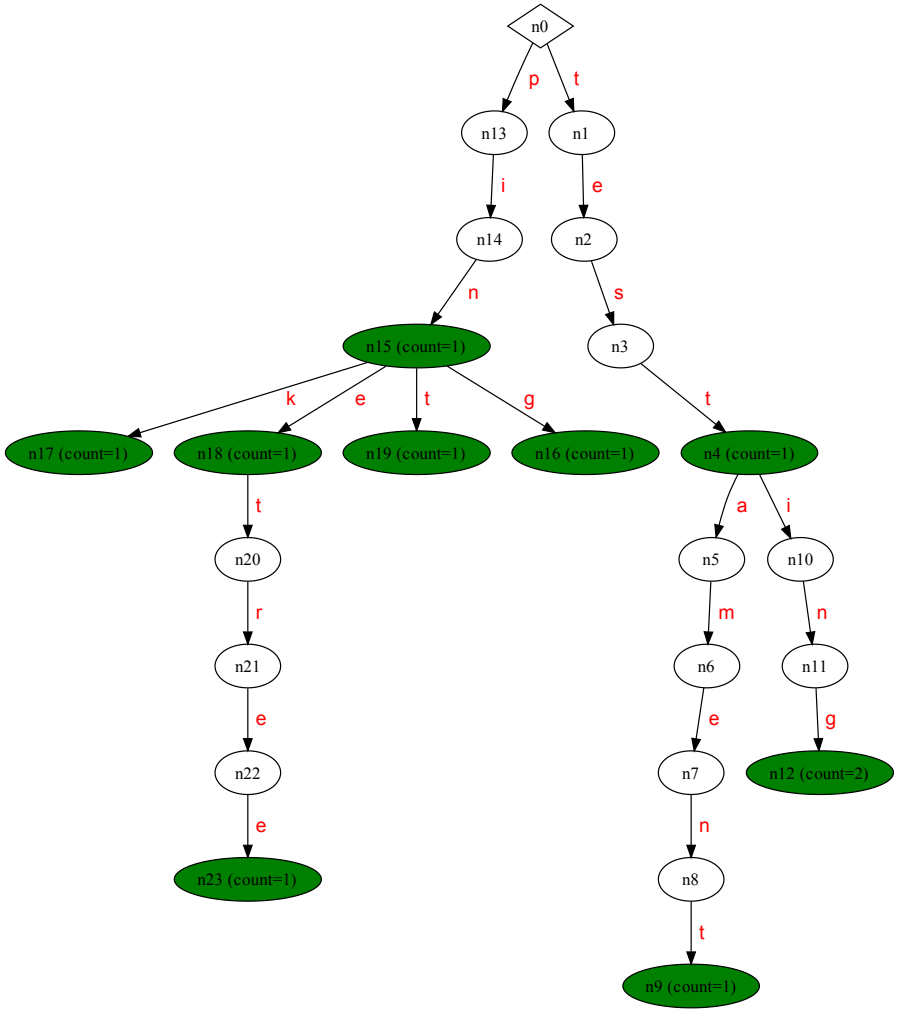
For instance, consider the trie below:



lookupWord('test') = 1
lookupWord('testing') = 2
lookupWord('tesla') = 0
lookupWord('tom') = 0
lookupWord('pinetree') = 1

Autocomplete

The operation **autoComplete** provides a list of "completions" of a given string along with their frequencies.



autoComplete('pi') = [('pin',1), ('pink',1), ('pine',1),('pinetree',1),('pint',1), ('ping',1)]
autoComplete('tom') = [ ] # emtpy list because no word with prefix tom belongs to the trie
autoComplete('testi') = [('testing',2)]

Trie Node

A trie node structure has the following attributes:

- **isRoot** A Boolean flag denoting if the node is a root.
- **isWordEnd** A Boolean flag denoting if the node is the word ending node.
- **count** A count field for frequency count for a word ending node.
- **next**: A dictionary that maps from each of the alphabets 'a' - 'z' to the child

node corresponding to the alphabet.

Assignment Instructions

The downloadable zip file for the assignment has the following contents:

- **trieClass.py** This is a skeleton class file for data structuring a trie. You are asked to implement the functions **addWord**, **lookupWord** and **autoComplete**.

Warning

Do not change the name of the file, the class name or the name of the three methods **addWord**, **lookupWord** and **autoComplete**. Doing so will break the grading script and cause deductions. .

- **trieTester.py** This is the autograding code that will be used to grade your assignment.
- **test1.spec** to **test9.spec** different test cases that will insert and lookup words in the trie. You have to pass each of these tests to be successful.

Running a test

To run a test say “test1.spec”, call the python script

```
python trieTester.py test1
```

This will print a bunch of messages including failure messages and tell you whether the test was passed by your code or not.

```
bash-3.2$ python trieTester.py test1
```

```
Running test1
Insert: hello
Insert: hello
Insert: he
Insert: jello
Insert: jelly
Lookup: hello
Lookup: hell
Lookup: howdy
Autocomplete: he
Autocomplete: je
test1 passed
```

Upload Instructions

You are asked to upload a zip file that contains your code with the file **trieClass.py** containing the class for the trie node as provided in your skeleton. Any extra files you have created for implementing the functionalities can be included. Please do not include anything else.

Uploaded filename should be called firstname_lastname.zip. Do not “submit for grading” on moodle unless your assignment is really final. Reopening submissions causes enormous problems on moodle.