SSAD Assignment-1

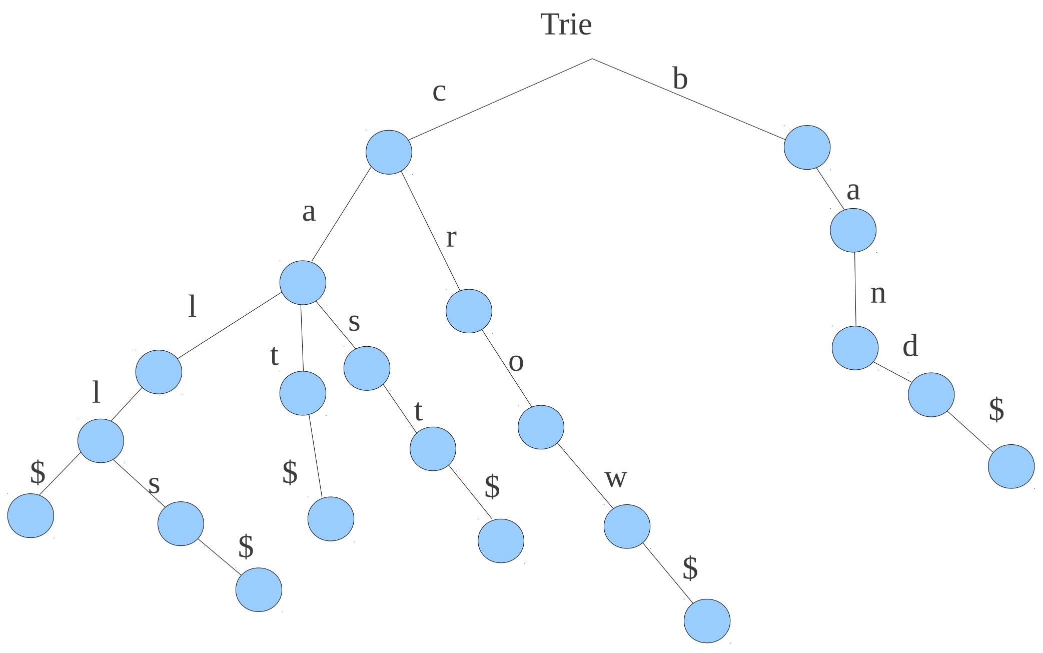
Due time : August 17, 6:00 PM.

Concept :

Trie , or prefix tree, is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. The basic idea is to share the storage of common prefixes.

See, http://en.wikipedia.org/wiki/Trie , for details.

Example : A Trie which includes a set of words {cat, call, cast, crow, calls, band}
We end each word with a "$" symbol, to make sure that the Trie correctly handles the case when one word is a prefix of another word , example: call and calls.



Assignment :

PART 1 : Write a python program to construct a trie and support the following functions:

1) insert(trie, word):

   Inserts 'word' into trie and returns number of duplicates of 'word' present in trie. (including the newly inserted).
   Example, consider the above mentioned trie having words{cat, call, cast, crow, calls, band} as already constructed. Now :

   ```
   Input:
   2                              //No. of instructions
   insert calls                   //instruction-1
   insert bat                     //instruction-2

   Output:
   2                              //case-1 output
   1                              //case-2 output
   ```

2) search(trie, word):

   if 'word' is in trie : return “true” and number of instances of 'word' in trie.
   Else return “false” and list of tuples of all the words and there instance count (duplicate count) in sorted order which have largest matched prefix with the input word.
   For example, considering above mentioned trie as already constructed:

   ```
   Input:
   4                              //No. of instructions for the program.
   search calling                 //instruction -1
   search call                    //instruction -2
   insert call                    //instruction-3
   search calling                 //instruction-4

   Output:
   false call 1 calls 1                   //case-1
   true 1                                 //case-2
   2                                      //case-3
   false call 2 calls 1                   //case-4
   ```

   Explanation of case-1 : 'calling' is not found so 'false'. Now, call and calls each have longest common prefix 'call' with 'calling'. So, both with their number of instances are returned. 'call' comes first to 'calls' due to sorted order.

3) remove(trie, word):
   if 'word' is in trie, remove single instance of 'word' and print the number of remaining duplicates of the 'word'.
   Else print "-1"
   Example, consider the above mentioned trie having words{cat, call, cast, crow, calls, band} as already constructed. Now :

```
Input:
6                        //No. of instructions
insert calls             //instruction-1
remove calls             //instruction-2
remove calls             //instruction-3
remove calls             //instruction-4
remove calls             //instruction-5
insert calls             //instruction-6

Output:
2                        //case-1 output
1                        //case-2 output
0                        //case-3 output
-1                       //case-4 output
-1                       //case-5 output
1                        //case-6 ouput
```

4) ptrie(trie) :

Prints trie with indentations. The print of the trie starts with a single line of root, and each level adds an indentation of | . The entries in print should be in lexicographical order, which implies that the output is unique. Duplicates shouldn't be printed.

Example, consider the above mentioned trie having words{cat, call, cast, crow, calls, band} as already constructed. Now :

```
Input:
1
ptrie

Output:
root
| b
| | a
| | | n
| | | | d
| | | | | $
| c
| | a
| | | l
| | | | l
| | | | | $
| | | | | s
| | | | | | $
| | | s
| | | | t
| | | | | $
| | r
| | | o
| | | | w
| | | | | $
```

Note:

1. The special symbol '$' "MAY BE" present in the internal representation to mark the end of string.
2. "$" is not given as a part of input "word" argument.

Input/Output Specifications :

Input : First line contains 'n' (n<10^5) number of instructions. Following n lines contain one instruction per line. Each instruction can be of following types :
1) insert <word>
2) search <word>
3) remove <word>
4) ptrie

Example Input:

```
8
insert balloon
search ball
insert ball
search ball
insert bally
search ba
remove ball
ptrie
```

Output :

```
1
false balloon 1
1
true 1
1
false ball 1 balloon 1 bally 1
0
root
| b
| | a
| | | l
| | | | l
| | | | | o
| | | | | | o
| | | | | | | n
| | | | | | | | $
| | | | | y
| | | | | | $
```

Part 2 :

Write a python program to print the number of distinct (unique) subtrings of a given string. You should strictly use “Trie” ,otherwise would be awarded zero in this part.

Input : First line contains 't' (<100) number of test-cases. Each test case consists of two lines. First line of each testcase contains the number of characters, n (< 10^5) in the string 's'. Next line contains the string 's'.

Output: A single line containing number of unique substrings of string 's'.

```
Input:
3
4
abba
3
aaa
3
cat
Output:
8
3
6
```

---------------------------------------End-Of-Assignment---