Due : December 3rd, 2017

Overview

This assignment consists of two parts: implementing a Trie and implementing Knuth-Morris-Pratt.

General Notes

- Do not change any method or class signatures. You should only edit inside of the functions. If your code changes any class or method names or signatures, you will receive an automatic 0. You should not implement any other functions or instance variables besides the ones that are provided, unless explicitly allowed.
- If you are using Eclipse, be sure to remove the line declaring your code a package when you submit your code. Failure to do so could result in you receiving a 0 if the autograder fails.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- Make sure that your code does not print out anything (there should be no System.out.println in your code). You will receive an automatic 0 if your code outputs something to STDOUT during the tests.

Trie

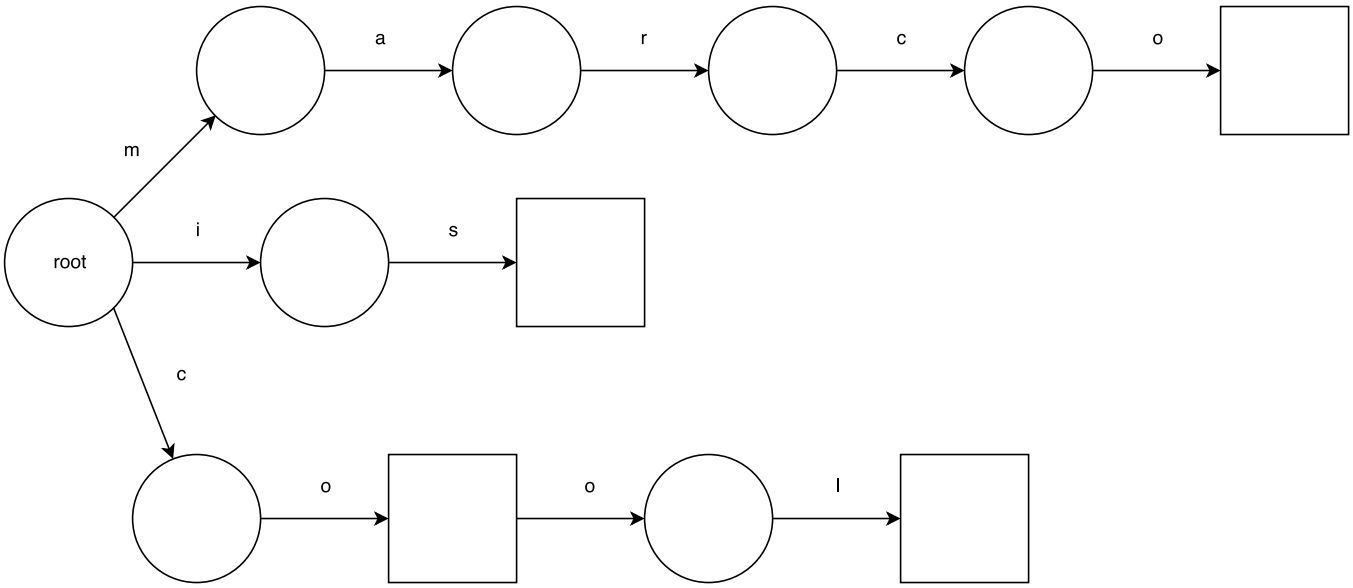
As you learned in the lecture, a trie is a useful data structure for operations on strings. The standard trie from the lecture has each external node as an "output" node, but we will use a variant where any node can be an output node. A path from the root to an output node corresponds to a word that has been inserted into the trie.

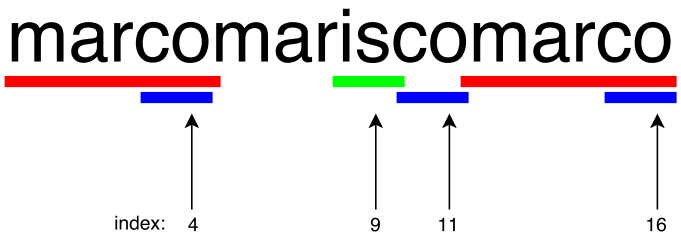
Definition

For this assignment, you will implement a trie that can perform the following major operations:

- `insert` - insert a string into the trie
- `remove` - remove a string from the trie
- `contains` - determines if a string is in the trie
- `getMatchesInString` - determine all indices of a string where a word in the trie ends
- `countMatchesInString` - count the number of substrings that match a word in the trie

Here is an example to help you (square nodes are output nodes). This trie could be constructed by calling `insert`("marco"); `insert`("cool"); `insert`("co"); `insert`("is");.





In the above example, `getMatchesInString` would return a list of $\{4, 9, 11, 16\}$. However, `countMatchesInString` would return 6.

Implementation Notes

- You are not allowed to add any instance variables or methods.
- Strings are indexed from 0.
- Words in the trie and test strings will only consist of letters from $a - z$.
- We will never insert the empty string (a string of length 0) into the trie. The test string will never be empty.
- A trie should always have at least one node (the root node should always exist). When you clear the trie, you should still have one node initialized as the root.
- For each node in the trie, you have access to a hash map with character keys and node values. You can use this to take transitions between nodes in the trie.
  - To insert a value v with key k into a map, you can use `yourMap.put(k,v);`
  - To get the value stored with key k, you can use `yourMap.get(k);`
    - If the result of this is `null`, the the key does not exist in the map.
  - https://beginnersbook.com/2013/12/hashmap-in-java-with-example/
- A hash set is a special type of list where you cannot have duplicate values. If you try to insert multiple duplicate values, only one will be stored in the set.
  - To insert a value v into a set, you can use `yourSet.add(v);`
  - To iterate over a set, you can use an iterator or a for each loop.
    - https://beginnersbook.com/2014/08/how-to-iterate-over-a-sethashset/
    - Your set needs to be of type `HashSet<TrieNode>` so that it stores TrieNodes.

Directions

- Write your name and student ID number in the comment at the top of the Trie.java file.
- Implement all of the required functions in the Trie class.
- You are not allowed to change the format of this class. You must only use the methods and instance variables provided.

- Pay careful attention to the required return types.

KMP

In the lecture, you learned about the KMP pattern matching algorithm. Here you will implement a slight extension of the algorithm that includes some *wildcards*. Wildcard characters are characters that match more than one different character. In this we will use two types (note that we will only input lowercase strings as test data):

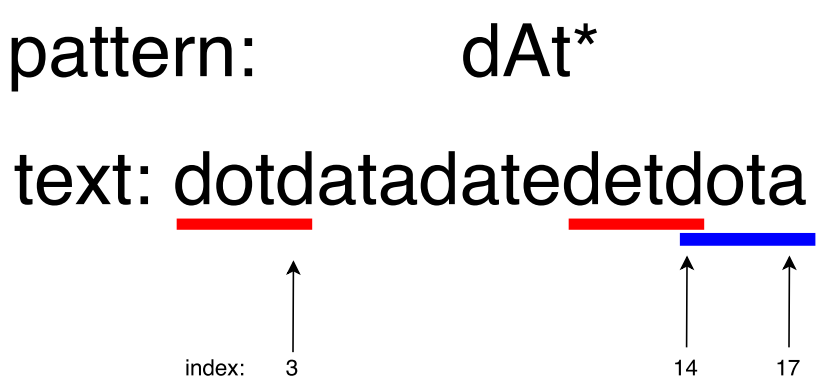
- $*$ - matches *any* single character
- capital letter - matches any character *other than* the lowercase version of the letter
  - For example "B" matches any character **except** "b", and "*" matches any character in the alphabet.

Definition

For this assignment, you will implement a KMP algorithm that can perform the following major operations:

- `match` - return true if the input string matches the pattern exactly
- `getMatchesInString` - determine all indices of a string where a substring matching the pattern ends.
- `countMatchesInString` - count the number of substrings that match the pattern

Here is an example to help you.



In the above example, we are matching all strings that begin with a "d", the second character is not "a", the third character is a "t", and the fourth character is any lowercase letter. Here, `getMatchesInString` would return a list $\{3, 14, 17\}$ and `countMatchesInString` would return 3.

Implementation Notes

- You are not allowed to add any instance variables or methods.
- Strings are indexed from 0.
- The pattern will consist of characters from $A-Z, a-z$, and $*$. The test strings will only contain characters from $a-z$.
- The pattern and input text will never be empty.

Directions

- Write your name and student ID number in the comment at the top of the KMP.java file.
- Implement all of the required functions in the KMP class.
- You are not allowed to change the format of this class. You must only use the methods and instance variables provided.
- Pay careful attention to the required return types.

General Directions

- Write your name and student ID number in the comment at the top of the java files.
- Implement all of the required functions in the java files.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.

Submission Procedure

To submit your homework, simply drag and drop each file individually into the attachment box on the YSCEC assignment page. You should **NOT** zip them or place them in any folder. For this assignment, you should submit only the following files:

- Trie.java
- KMP.java
- ⟨student_id⟩.txt

Inside of the ⟨student_id⟩.txt file, you should include the following text, and write your name at the bottom.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, `2017123456.txt` is correct while something like `2017123456_hw1.txt` will receive a 0.

Testing

We have provided a small test suite for you to check your code. You can test your code by compiling and running the tester program. You will always get a notification if you fail any of the tests, but you may not get a notification if you passed.

Note that the test suite we use to grade your code will be much more rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.