UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 7

Release: Thursday 6 April 2016
Deadline: Friday 7 April 2016, 18:00

Instructions

Complete the tasks below. Certain classes have been provided for you in the *files* sub-folder of the practical download. Modify the given main class to test the functionality of your work. Remember to test "corner" cases. Upload **only** the given source files with your changes in a zip archive before the deadline. Please comment your name **and** student number in at the top of each file.

Tries [30]

Introduction

Tries are trees that use parts of the key to guide navigation within the tree. Every node in a trie contains either an entire key (in which case it is a leaf node) or it contains a set of pointers to subtries (in which case it is a non-leaf node). For simplicity's sake, suppose that a trie only needs to be able to store keys consisting of the characters {`Q,W,E,R,T,Y`}. If the pointers array of a node at level i has en entry at index k, then the i_{th} letter in the key being processed is the k_{th} letter in {`#,Q,W,E,R,T,Y`}. Note that the character '#' is used to represent the end of a key.

For further explanation of how tries work, you are referred to page 377 of the textbook.

For this practical, you are required to implement a trie. The trie must support keys that consist of characters from a fixed charset (like {`Q,W,E,R,T,Y`} in the example above). You do not need to perform any compression on the trie.

You have been provided with a `Node` which defines the nodes of the trie. You may not edit this class. The class has two constructors, one for a leaf node instance and one for a non-leaf node instance. Observe that the `ptrs` array of leaf nodes will be empty and that the `key` field of non-leaf nodes will be empty. Consult the comments in the source code if you are unsure what a given `Node` member variable should be used for.

You have also been provided with the basic structure and some helper functions for the `Trie` class. First a brief explanation of the functions that have been implemented for you:

`Trie(char[] _letters)`
: Initializes the trie structure. The given set of letters are stored so that the order of pointers in the nodes is known. Notice that the end-of-word character '#' should not be specified in the input array; it will be added automatically. The 0_{th} index of a `ptrs` array will correspond to the end-of-word character.

`String nodeToString(Node node)`
: Returns a string representation of the given node. If the node is a leaf, then the result is just its key. If the node is a non-leaf, then this returns a set of pairs, such as '(#,1) (Q,0) (W,1) (E,1) (R,0) (T,1) (Y,1)'. This indicates that the pointers corresponding to

the characters '#','W','E','T' and 'Y' are not null. Also, that the pointer corresponding to the character 'Q' and 'R' are null.

`print()`
: This prints the tree by means of a breadth-first traversal.

`int index(char c)`
: This is a helper function that may be useful for interaction with `ptrs` arrays. This function will return the index in a `ptrs` array that corresponds to the character 'c' .

`char character(int i)`
: This is a helper function that may be useful for interaction with `ptrs` arrays. This function will return the character corresponding to an index *i* in a `ptrs` array.

`insert(String word)`
: Inserts the given word into the tree, creating the necessary non-leaf nodes to structure it.

Task

You are required to implement one function, which is described below:

`void delete(String key)` **[30]**
: Searches for and deletes the given key. You must also delete any non-leaf nodes which become redundant after the key has been deleted. A redundant non-leaf node is one which contains a single leaf node as its child, since that child can simply become the child of the non-leaf's parent, cutting out the middle. In the example in main, RTRRR is deleted, but it's parent contains 2 children, another non-leaf node and RTE, thus this parent is kept in place. The second delete targets WRETE however, leaving its parent with only a single child WRWW. This parent is then deleted, and WRWW is moved to the 'R' branch of its grandparent. If the key cannot be found, do nothing.

You may notice the presence of a `Queue` class, this is used for breadth-first traversal in the trie's `print` function. You may not edit this class. You may use this class to keep track of the path used to reach the key to be deleted, though keep in mind that the queue is First-In-First-Out. You can not edit the queue class, but you can use it in such a way that you effectively have a stack. You are also advised to use recursion.

Submission

Submit your source files on the CS Website. Place all the files in a zip archive named as uXXXXXXXX.zip where XXXXXXXX is your student number. Please make use of the submission slot that corresponds to your practical session (Practical 7 Friday). Submit your work before the deadline. No late submissions will be accepted.