CMP-5014Y Data Structures and Algorithms

100172935

April 25, 2018

1 Form a dictionary

Algorithm 1 formDictionary algorithm

Input: List of String words

Output: SortedMap treeMap

```
TrieNode $currentNode \leftarrow root$
for String $word$ in $words$ do
    if $treeMap$ does not contain $word$ then
        Add word,key to treeMap
    else
        Add word, key by n+1 to treeMap
    end if
end for
return $treeMap$
```

1.1 Fundamental Operation

The fundamental operation for the algorithm is Add word,key to treeMap and Add word, key by n+1 to treeMap.

1.2 Run time complexity function

$$\sum_{i=1}^{n} log(i-1)1$$

1.3 Worst case scenario

Worst case scenario is that the words that have been added to the treeMap are new words.

2 Trie data structure

2.1 Add method for adding a key to the trie

Algorithm 2 add algorithm

Input: String key

Output: true if key was successfully added to the trie, false otherwise

```
TrieNode $currentNode$ ← $root$
for every letter $current$ in $key$ do
    TrieNode $child$ ←$currentNode$.getNode($current$)
    if $child$ is not equal null then
        $currentNode$.setNode($current$)
        $child$ ←$currentNode$.getNode($current$)
    end if
    $currentNode$ ← $child$
end for
if $currentNode$.isComplete() = true then
    return false
end if
$currentNode$.setComplete()←true
return true
```

2.2 Contains method to check whether the word that is passed is a full word and not prefix

Algorithm 3 contains algorithm

Input: String key

Output: true if the whole word was in the trie, false otherwise

```
TrieNode $currentNode$ ← $root$
for every letter c in $key$ do
    if $currentNode$.getNode($c$) is equal null then
        return false
    else
        $currentNode$ ← $currentNode$.getNode($c$)
    end if
end for
return $currentNode$.isComplete()
```

2.3 Output by Breadth First Search Method

Algorithm 4 outputBreadthFirstSearch algorithm

Input: No Input

Output: String result

```
String $result$ ← empty String
Queue $nodes$ ← empty LinkedList
$nodes$.add($root$)
while $nodes$.isEmpty() is equal false do
    TrieNode $temp$ ← $nodes$.poll()
    append $result$ with $temp$.getChar()
    for each offspring $nodein$ $temp$.getOffSpring() do
        if $node$ is not equal null then
            $nodes$.add($node$)
        end if
    end for
end while
return result
```

2.4 Depth First Search Method

Algorithm 5 DepthFirsSearch algorithm

Input: Trienode trienode

Output: result

```
String *result* ← empty String
Queue *nodes* ← empty LinkedList
**for** each offspring *nodein* *trienode*.getOffSpring() **do**
    **if** *node* is not equal null **then**
        append *result* with depthFirstSearch(*node*)
    **end if**
**end for**
append *result* with *trienode*.getChar()
```

2.5 Output by Depth First Search Method

Algorithm 6 OutputDepthFirsSearch algorithm

Output: result

```
String *result* ← empty String
**if** *root* is not equal null **then**
    append *result* with depthFirstSearch(*root*)
**end if**
**return** result
```

2.6 get SubTrie Method to return a trie rooted at the prefix

Algorithm 7 getSubTrie algorithm

Input: String prefix
Output: Trie result

```
TrieNode $currentNode$ ← $root$
Trie $result$ ←new Trie()
**for** every prefix $i$ in *prefix*.lenght() **do**
    int *index*←prefix.$charAt$(i) - 'a'
    **if** $currentNode$.getNode($prefix$.charAt($i$) not equal null **then**
        $result$.root ← $currentNode$.getNode($prefix$.charAt($i$)
    **end if**
    $currentNode$ ← $currentNode.offspring[index]$
**end for**
**return** result
```

2.7 get AllWords function to get the all the words in the trie

Algorithm 8 getAllWords function algorithm

Input: String prefix, TrieNode trienode, List of String Nodes

```
**for** each offspring *temp* in *trienode*.getOffspring() **do**
    **if** *temp* is not equal null **then**
        String $prefix2$ ← $prefix$ + *temp*.getChar()
        getAllWords($prefix2$, *temp*, *nodes*)
    **end if**
**end for**
**if** *trienode*.isComplete() **then**
    *nodes*.add($prefix$)
**end if**
```

2.8 get AllWords function to return the all the words in the trie

Algorithm 9 getAllWords algorithm

```
Output: List of Strings output
List of Strings output ←new LinkedList
getAllWords("",root,output)
return  output
```

3 Word Auto Completion

3.1 Auto Competion program

Algorithm 10 AutoCompletion algorithm

```
ArrayList of Strings LotrQueries ←a list of prefixes
List of Strings lotr ←new ArrayList
Trie wordstrie ←a trie of all words
for each prefix i in LotrQueries.size() do
    lotr.add(LotrQueries.get(i))
    temp ←wordstrie.getSubTrie(lotr.get(i))
    List of Strings list ← temp.getAllWords()
    prefix ← lotr.get(i)
    for each word j in list.size() do
        auto ← prefix + list.get(j)
        for every entry of Map of String and Integer in words.entrySet() do
            if auto.equals(entry.getKey()) then
                storeAuto.put(entry.getKey(),entry.getValue())
            end if
        end for
    end for
end for
```

3.2 AutoCompletion output

Word	Probability
able	0.14285714285714285
abominable	0.047619047619047616
about	0.8095238095238095
frodo	0.4909090909090909
from	0.43636363636363634
front	0.07272727272727272
go	0.7647058823529411
goblins	0.058823529411764705
goes	0.17647058823529413
grasp	0.07692307692307693
grass	0.7692307692307693
grasses	0.15384615384615385
merely	0.02631578947368421
merrily	0.02631578947368421
merry	0.9473684210526315
sam	1.0
the	0.8471454880294659
their	0.06077348066298342
them	0.09208103130755065