208. 实现 Trie (前缀树)

入选理由

1. 学习完讲义了，那么就动手撸一个吧。后面的题都建立在手撸前缀树的前提上哦~

标签

- 前缀树

难度

- 中等

题目地址（实现 Trie (前缀树)）

https://leetcode-cn.com/problems/implement-trie-prefix-tree

题目描述

实现一个 Trie (前缀树)，包含 insert, search, 和 startsWith 这三个操作。

示例:

Trie trie = new Trie();

trie.insert("apple"); trie.search("apple"); // 返回 true trie.search("app"); // 返回 false trie.startsWith("app"); // 返回 true1 trie.insert("app"); trie.search("app"); // 返回 true 说明:

你可以假设所有的输入都是由小写字母 a-z 构成的。 保证所有输入均为非空字符串。

前置知识

- 树
- Trie

思路

大家是否看完了讲义呢，看完了正准备自己动手实现的话，这个题正合适，由于这个题已经说让我们实现一个 Trie，我们也就别想啥其他操作了，老老实实实现一个 Trie 就好，插入和查找及其时间复杂度分析我在讲义里写清楚啦，有啥不清楚的可以回过头看下讲义，这里我贴个我自己实现的 Java 和 Python(两年前写的，直接粘过来 😄)版本的代码吧

代码

代码支持：Java，Python

Java Code:

```
class Trie {

    TrieNode root;

    public Trie() {

        root = new TrieNode();
    }

    public void insert(String word) {

        TrieNode node = root;

        for (int i = 0; i < word.length(); i++) {

            if (node.children[word.charAt(i) - 'a'] == null)
                node.children[word.charAt(i) - 'a'] = new TrieNode();

            node = node.children[word.charAt(i) - 'a'];
            node.preCount++;
        }

        node.count++;
    }

    public boolean search(String word) {

        TrieNode node = root;

        for (int i = 0; i < word.length(); i++) {

            if (node.children[word.charAt(i) - 'a'] == null)
                return false;

            node = node.children[word.charAt(i) - 'a'];
        }

        return node.count > 0;
    }

    public boolean startsWith(String prefix) {
```

```
        TrieNode node = root;

        for (int i = 0; i < prefix.length(); i++) {

            if (node.children[prefix.charAt(i) - 'a'] == null)
                return false;
            node = node.children[prefix.charAt(i) - 'a'];
        }

        return node.preCount > 0;
    }

    private class TrieNode {

        int count; //表示以该处节点构成的串的个数
        int preCount; //表示以该处节点构成的前缀的字串的个数
        TrieNode[] children;

        TrieNode() {

            children = new TrieNode[26];
            count = 0;
            preCount = 0;
        }
    }
}
```

Python Code:

> 这里我 children 用的字典，因为我不太喜欢 python 里的 ord，chr，用起来嫌乱，大家可以用 ord,chr 来实现 children

```
class TrieNode:
    def __init__(self):
        self.count = 0
        self.preCount = 0
        self.children = {}

class Trie:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = TrieNode()

    def insert(self, word):
        """
        Inserts a word into the trie.
        :type word: str
```

```
        :rtype: void
        """
        node = self.root
        for ch in word:
            if ch not in node.children:
                node.children[ch] = TrieNode()
            node = node.children[ch]
            node.preCount += 1
        node.count += 1

    def search(self, word):
        """
        Returns if the word is in the trie.
        :type word: str
        :rtype: bool
        """
        node = self.root
        for ch in word:
            if ch not in node.children:
                return False
            node = node.children[ch]
        return node.count > 0

    def startsWith(self, prefix):
        """
        Returns if there is any word in the trie that starts with the given prefix.
        :type prefix: str
        :rtype: bool
        """
        node = self.root
        for ch in prefix:
            if ch not in node.children:
                return False
            node = node.children[ch]
        return node.preCount > 0
```

JS Code：

```
var Trie = function () {
  this.root = {};
};

/**
 * Inserts a word into the trie.
 * @param {string} word
 * @return {void}
 */
Trie.prototype.insert = function (word) {
```

```
  let node = this.root;
  for (const c of word) {
    if (!node[c]) node[c] = {};
    node = node[c];
  }
  node.isEnd = true;
};

/**
 * Returns if the word is in the trie.
 * @param {string} word
 * @return {boolean}
 */
Trie.prototype.search = function (word, node = this.root) {
  for (const c of word) {
    if (!node[c]) return false;
    node = node[c];
  }

  return node.isEnd === true;
};

/**
 * Returns if there is any word in the trie that starts with the given prefix.
 * @param {string} prefix
 * @return {boolean}
 */
Trie.prototype.startsWith = function (prefix, node = this.root) {
  for (const c of prefix) {
    if (!node[c]) return false;
    node = node[c];
  }
  return true;
};
```

CPP Code:

```
class TrieNode {
  public:
    TrieNode *child[26];
    bool isWord;
    // 初始化
    TrieNode(): isWord(false){
        for (auto &c: child) c = nullptr;
    }
};


class Trie {
```

```
private:
    TrieNode* root;

public:
    /** Initialize your data structure here. */
    Trie() {
        root = new TrieNode();
    }

    /** Inserts a word into the trie. */
    void insert(string word) {
        TrieNode *p = root;

        for (auto a: word) {
            int index = a - 'a';
            if (!p->child[index]) p->child[index] = new TrieNode();
            p = p->child[index];
        }
        p->isWord = true;
    }

    /** Returns if the word is in the trie. */
    bool search(string word) {
        TrieNode *p = root;

        for (auto a: word) {
            int index = a - 'a';
            if (!p->child[index]) return false;
            p = p->child[index];
        }
        return p->isWord;
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    bool startsWith(string prefix) {
        TrieNode *p = root;
        for (auto a : prefix) {
            int index = a - 'a';
            if (!p->child[index]) return false;
            p = p->child[index];
        }
        return true;
    }
};
```

复杂度分析

令 n 为待操作的字符串的长度。

- 时间复杂度：创建 Trie:$O(1)$, 其余操作为 $O(n)$。

- 空间复杂度：最坏情况下没有任何前缀，此时空间复杂度为所有操作的单词所占的空间，具体来说就是 $O(字符集大小 * 单词总字符数)$。不过随着前缀相同的单词增多，效率会变好。（也就是说如果字符串中的相同前缀比较多，则性能优化明显）

上面的是我常用的板子，直接拿来放到这个题上用就可以了，操作也都挺直观的，其中的 startsWith 操作我在讲义里倒是没写，但是大家看一下，startsWith 的操作逻辑是不是和 search 几乎相同。

自己动手实现好、优化好的 Trie 保存好当作以后的 Trie 板子岂不是美滋滋，不过我这两天的题大家还是从头自己敲吧，等以后再活用板子。

上一页 下一页