211. Add and Search Word - Data structure design

题目描述：https://leetcode.com/problems/add-and-search-word-data-structure-design/

> 实现一个word字典，要求有如下两种操作：
>
> ```
> addWord(string word) //在词典中加入该单词
> search(string pattern) //在词典中找到符合该pattern的词
> ```
>
> 其中word只由a-z组成
> patter由a-z和'.'组成，'.' 代表任意字符

解题思路：

回溯+前缀树

代码：

```
class TrieNode {
public:
    vector<TrieNode*> subNode;
    bool freq;
    TrieNode() {
        subNode.resize(26);
        freq = false;
    }
};
class WordDictionary {
public:
    WordDictionary() {
        root = new TrieNode();
    }
    // Adds a word into the data structure.
    void addWord(string word) {
        TrieNode * p = root;
        for(int i = 0; i < word.size(); i++) {
            int k = word.at(i) - 'a';
            if(p->subNode[k] == NULL) {
                p -> subNode[k] = new TrieNode();
            }
            p = p -> subNode[k];
```

```cpp
        }
        p->freq = true;
    }

    // Returns if the word is in the data structure. A word could
    // contain the dot character '.' to represent any one letter.
    bool search(string word) {
        TrieNode* p = findNode(root, word);
        return p != NULL && p -> freq;
    }
    TrieNode* findNode(TrieNode* root, string word) {
        TrieNode * p;
        if(word.size() == 0 || root == NULL) {
            return root;
        }
        if(word.at(0) == '.') {
            for(int i = 0; i < 26; i++) {
                p = findNode(root->subNode[i], word.substr(1));
                if(p != NULL && p->freq) {
                    return p;
                }
            }
            return p;
        }
        return findNode(root->subNode[word.at(0) - 'a'], word.substr(1));
    }
private:
    TrieNode * root;
};

// Your WordDictionary object will be instantiated and called as such:
// WordDictionary wordDictionary;
// wordDictionary.addWord("word");
// wordDictionary.search("pattern");
```