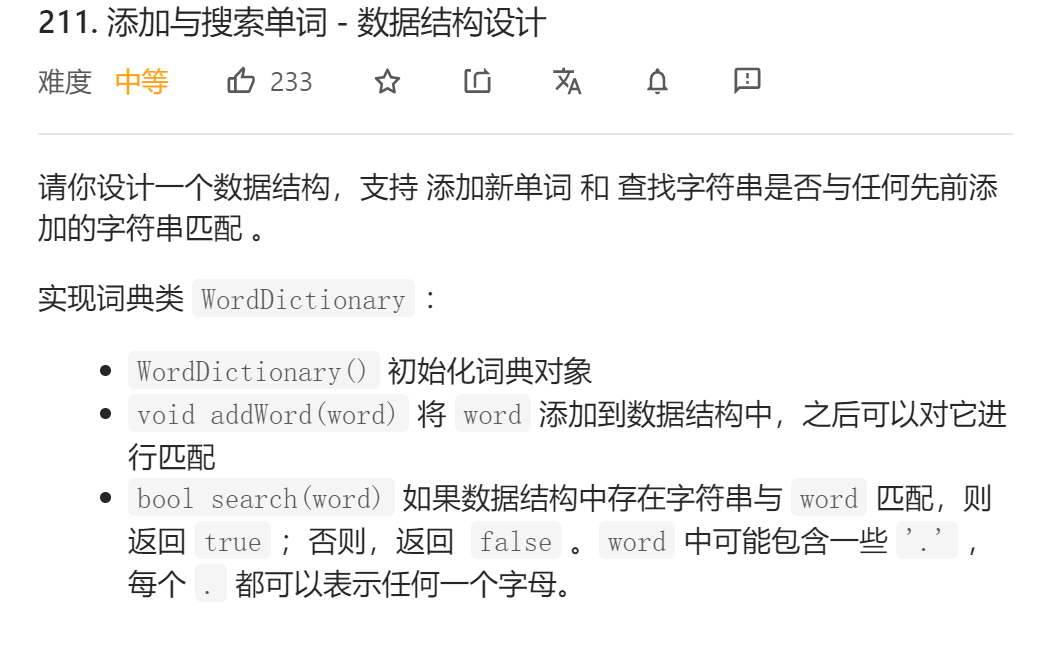
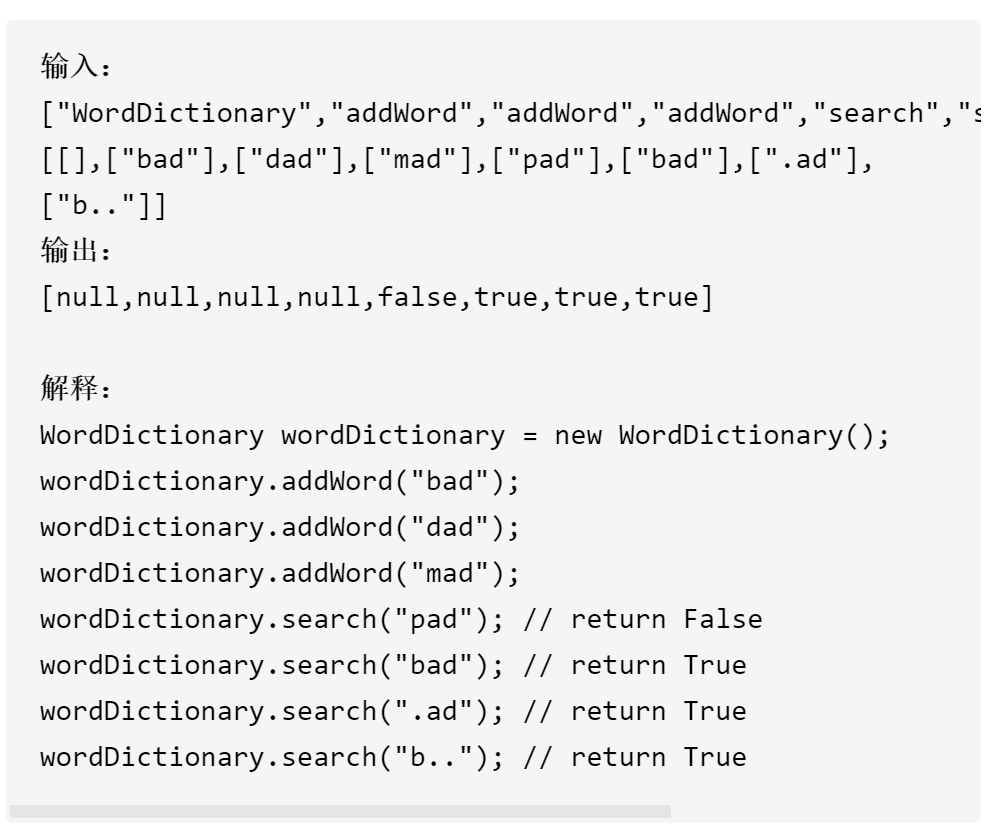
# [211. 添加与搜索单词 - 数据结构设计]

## 题目描述：





## 题目分析：【set+暴力搜索经典超时】

1. 由于'.'可以表示任何的单词，所以用unordered\_set的话有的单词会匹配不到；
2. 并且用unordered\_set的模板的话，空间和时间的消耗会比较大，暴力搜索容易超时；

//暴力搜索经典超时  
class WordDictionary {  
 private:  
 unordered\_set<string>vec;  
 //回溯函数  
 bool backtracking(string&word,int startIndex){  
 //终止条件  
 if(startIndex==word.size()){  
 if(vec.find(word)!=vec.end())return true;  
 else return false;  
 }  
 int count=0;  
 for(int i=startIndex;i<word.size();i++){  
 if(word[i]!='.')continue;  
 count++;  
 for(int j='a';j<='z';j++){  
 word[i]=j;  
 //此处代码可以优化  
 if(backtracking(word,startIndex+1))  
 return true;  
 word[i]='.';  
 }  
 }  
 if(!count){  
 if(vec.find(word)!=vec.end())return true;  
 else return false;  
 }  
 return false;  
 }  
public:  
 /\*\* Initialize your data structure here. \*/  
 WordDictionary() {  
  
 }  
   
 void addWord(string word) {  
 vec.insert(word);  
 }  
   
 //搜索这里需要用到回溯算法  
 bool search(string word) {  
 if(backtracking(word,0))return true;  
 else return false;  
 }  
};

## 前缀树+回溯算法 来解围

1. **在写这个前缀树相关代码的时候出现了很多“回溯算法经典的bug",值得去总结借鉴**
2. **bug1:树结点的指针需要初始化为null，不然会导致指针错误指引引起错误。指针数组的初始化可以用memset(名称，初值，sizeof（名称)）；**

class WordDictionary {  
 private:  
 //前缀树+回溯  
 struct TreeNode{  
 char ch;  
 bool isEnd;  
 TreeNode\*next[26];  
 TreeNode():ch('0'),isEnd(false){}  
 TreeNode(char c,bool p):ch(c),isEnd(p){}  
 };  
 TreeNode\*root;  
 //回溯函数  
 bool backtracking(string&word,int startIndex,TreeNode\*temp){  
 //终止条件  
 if(!temp)return false;  
 if(temp->isEnd&&startIndex==word.size()-1&&(word[startIndex]=='.'||word[startIndex]==temp->ch))return true;  
 if(startIndex==word.size()-1)return false;  
 //if(temp->isEnd&&startIndex==word.size()-1)return true;  
 //else if(startIndex==word.size()-1)return false;  
 /\*  
 if(word[startIndex]=='.'){  
 for(int i=0;i<26;i++){  
 return backtracking(word,startIndex+1,temp->next[i]);  
 }  
 }//else if(word[startIndex]==temp->ch){  
 else{  
 \*/  
 //此处需要处理  
 //if(startIndex<word.size()-1)  
 //return backtracking(word,startIndex+1,temp->next[word[startIndex+1]-'a']);  
 if(word[startIndex+1]=='.'){  
 for(int i=0;i<26;i++){  
 if(backtracking(word,startIndex+1,temp->next[i])){  
 return true;  
 break;  
 }  
 }  
 }else{  
 return backtracking(word,startIndex+1,temp->next[word[startIndex+1]-'a']);  
 }  
   
   
 return false;  
 }  
public:  
 /\*\* Initialize your data structure here. \*/  
 WordDictionary() {  
 root=new TreeNode();  
 memset(root->next, NULL, sizeof(root->next));  
 }  
   
 void addWord(string word) {  
 TreeNode\*temp=root;  
 for(auto tmp:word){  
 if(temp->next[tmp-'a'])temp=temp->next[tmp-'a'];  
 else{  
 TreeNode\*New=new TreeNode(tmp,false);  
 //后续指针的初始化  
 memset(New->next, NULL, sizeof(New->next));  
 temp->next[tmp-'a']=New;  
 temp=temp->next[tmp-'a'];  
 }  
 //temp->isEnd=true;  
 /\*  
 TreeNode\*New=new TreeNode(tmp,false);  
 //后续指针的初始化  
 memset(New->next, NULL, sizeof(New->next));  
 temp->next[tmp-'a']=New;  
 temp=temp->next[tmp-'a'];  
 \*/  
 }  
 temp->isEnd=true;  
 }  
   
 bool search(string word) {  
 //此处查找需要回溯  
 if(word[0]=='.'){  
 for(int i=0;i<26;i++){  
 if(backtracking(word,0,root->next[i])){  
 return true;  
 break;  
 }  
 //return true;  
 }  
 }else{  
 return backtracking(word,0,root->next[word[0]-'a']);  
 }  
 return false;  
 }  
};  
  
/\*\*  
 \* Your WordDictionary object will be instantiated and called as such:  
 \* WordDictionary\* obj = new WordDictionary();  
 \* obj->addWord(word);  
 \* bool param\_2 = obj->search(word);  
 \*/

## 修改优化后的代码：

class WordDictionary {  
 private:  
 //前缀树+回溯  
 struct TreeNode{  
 char ch;  
 bool isEnd;  
 TreeNode\*next[26];  
 TreeNode():ch('0'),isEnd(false){}  
 TreeNode(char c,bool p):ch(c),isEnd(p){}  
 };  
 //前缀树的根指针  
 TreeNode\*root;  
 //回溯函数  
 bool backtracking(string&word,int startIndex,TreeNode\*temp){  
 //终止条件  
 if(!temp)return false;  
 if(startIndex==word.size()-1){  
 if(temp->isEnd){  
 return true;  
 }else return false;  
 }  
 //回溯  
 if(word[startIndex+1]=='.'){  
 for(int i=0;i<26;i++){  
 if(backtracking(word,startIndex+1,temp->next[i])){  
 return true;  
 break;  
 }  
 }  
 }else{  
 return backtracking(word,startIndex+1,temp->next[word[startIndex+1]-'a']);  
 }  
 //在该层次未找到答案  
 return false;  
 }  
public:  
 /\*\* Initialize your data structure here. \*/  
 WordDictionary() {  
 //初始化根节点  
 root=new TreeNode();  
 memset(root->next, NULL, sizeof(root->next));  
 }  
   
 void addWord(string word) {  
 TreeNode\*temp=root;  
 for(auto tmp:word){  
 if(temp->next[tmp-'a'])temp=temp->next[tmp-'a'];  
 else{  
 TreeNode\*New=new TreeNode(tmp,false);  
 memset(New->next, NULL, sizeof(New->next));  
 temp->next[tmp-'a']=New;  
 temp=temp->next[tmp-'a'];  
 }  
 }  
 temp->isEnd=true;//最后一个结点的isEnd设置未true,表示是单词的结尾  
 }  
   
 bool search(string word) {  
 //此处查找需要回溯  
 if(word[0]=='.'){  
 for(int i=0;i<26;i++){  
 if(backtracking(word,0,root->next[i])){  
 return true;  
 break;  
 }  
 }  
 }else{  
 return backtracking(word,0,root->next[word[0]-'a']);  
 }  
 return false;  
 }  
};