212. Word Search II

Add to List

QuestionEditorial Solution

My Submissions

- Total Accepted: **36894**
- Total Submissions: **164774**
- Difficulty: **Hard**
- Contributors: **Admin**

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

For example,

Given **words** = `["oath","pea","eat","rain"]` and **board** =

```
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
```

Return `["eat","oath"]`.

```
#include<sstream>
#include<iostream>
#include<algorithm>
#include<string>
#include<vector>
#include<map>
#include<stdio.h>
#include<math.h>
using namespace std;

#define ALPHABET_SZ 26
#define INDEX(c) int(c) - int('a')
typedef struct TrieNode
{
	struct TrieNode *children[ALPHABET_SZ];
	int idx;
	bool leaf;
	TrieNode()
	{
		for(int i=0;i<ALPHABET_SZ;++i)
			children[i] = NULL;
		idx = 0;
```

```
            leaf = false;
      }
}Trie;


void insertWords(Trie *root, vector<string> &words, int idx)
{
      int sz = words[idx].size();
      for(int pos=0;pos<sz;pos++)
      {
            int index = INDEX(words[idx][pos]);
            if(root->children[index]==NULL) root->children[index] = new TrieNode();
            root = root->children[index];
      }
      root->idx= idx;
      root->leaf = true;
}

Trie *buildTree(vector<string> &words)
{
      Trie *root = new Trie();
      int len = words.size();
      for(int i=0;i<len;++i)
      {
            insertWords(root,words,i);
      }
      return root;
}

bool searchTrie(Trie *root, string word)
{
      int len = word.size();
      Trie *p  = root;
      for(int i=0;i<len;i++)
      {
            int index = INDEX(word[i]);
            if(p->children[index]==NULL) return false;
            p=p->children[index];
      }
      return p->leaf;
}

void checkwords(vector<vector<char> >&board, int i, int j, int row, int col, Trie
*root,
vector<string> &res, vector<string> &words)
{
      char temp;
      if(board[i][j]=='X') return;
      if(root->children[INDEX(board[i][j])]==NULL) return;
      else{
            temp = board[i][j];
            if(root->children[INDEX(board[i][j])]->leaf == true)
            {
                  res.push_back(words[root->children[INDEX(board[i][j])]->idx]);
                  root->children[INDEX(board[i][j])]->leaf=false;
            }
            // back track
            board[i][j] = 'X';
            if(i>0) checkwords(board,i-1,j,row,col,root-
>children[INDEX(temp)],res,words);
            if(i<row-1) checkwords(board,i+1,j,row,col,root-
>children[INDEX(temp)],res,words);
```

```
            if(j>0) checkwords(board,i,j-1,row,col,root-
>children[INDEX(temp)],res,words);
            if(j<col-1) checkwords(board,i,j+1,row,col,root-
>children[INDEX(temp)],res,words);
            board[i][j] = temp;
      }
}

vector<string> findWords2(vector<vector<char> >&board, vector<string> &words)
{
      vector<string> res;
      int row = board.size();
      if(0==row) return res;
      int col = board[0].size();
      if(0==col) return res;
      int wordCount = words.size();
      if(0==wordCount) return res;
      Trie *root = buildTree(words);
      for(int i=0;i<row;i++)
            for(int j=0;j<col && res.size()<wordCount;j++)
            {
                  checkwords(board,i,j,row,col,root,res,words);
            }
      return res;
}

int main(int argc,char *argv[])
{
      string wordsArr[] = {"oath","pea","eat","rain"};
      vector<string> words;
      /*
            [
                  ['o','a','a','n'],
                  ['e','t','a','e'],
                  ['i','h','k','r'],
                  ['i','f','l','v']
            ]
      */
      char a[][4] = {{'o','a','a','n'},{'e','t','a','e'},
      {'i','h','k','r'},{'i','f','l','v'}};
      vector<vector<char> > board(4,vector<char>(4));
      for(int i=0;i<4;++i)
            for(int j=0;j<4;++j)
            board[i][j] = a[i][j];
      for(int z=0;z<4;++z) words.push_back(wordsArr[z]);
      Trie *trie = buildTree(words);
      int ans = searchTrie(trie,"f");
      cout << ans << endl;
      vector<string> res;
      res = findWords2(board,words);
      for(int i=0;i<res.size();++i) printf("%s ",res[i].c_str());
      return 0;
}
```