一、题目说明

题目79. Word Search，给定一个由字符组成的矩阵，从矩阵中查找一个字符串是否存在。可以连续横、纵找。不能重复使用，难度是Medium。

二、我的解答

惭愧，我写了很久总是有问题，就先看正确的写法，下面是回溯法的代码：

```cpp
class Solution {
public:
    int m,n;
    //左 上 右 下
    int dx[4] = {-1,0,1,0};
    int dy[4] = {0,1,0,-1};
    bool exist(vector<vector<char>>& board,string word){
        if(board.empty()||word.empty()){
            return false;
        }
        m = board.size();
        n = board[0].size();
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(dfs(board,i,j,word,0)){
                    return true;
                }
            }
        }
        return false;
    }
    bool dfs(vector<vector<char>>& board,int x,int y,string&word,int pos){
        if(word[pos]!=board[x][y]){
            return false;
        }
        if(pos == word.size()-1){
            return true;
        }
        board[x][y] = '.';
        for(int i=0;i<4;i++){
            int nx = x + dx[i];
            int ny = y + dy[i];
            if(nx<m && nx>=0 && ny<n && ny>=0){
                if(dfs(board,nx,ny,word,pos+1)){
                    return true;
                }
            }
        }
        board[x][y] = word[pos];
        return false;
    }
};
```

性能：

三、优化措施

我的思路是用 unordered_map<char,vector<vector<int>>> ump; 来存储board中所有字符的出现位置，然后从word的第1个开始起查找，用dfs算法（回溯算法）进行匹配，修改并提交了差不多10次，才成功。

```cpp
class Solution{
    public:
        bool exist(vector<vector<char>>& board,string word){

            if (board.empty() || word.empty()) {
                return false;
            }
            int row = board.size();
            int col = board[0].size();

            if (row * col < word.length()) {
                return false;
            }

            for(int i=0;i<row;i++){
                for(int j=0;j<col;j++){
                    char ch = board[i][j];
                    ump[ch].push_back({i,j});
                }
            }

            if(dfs(board,0,0,0,word)){
                return true;
            }else{
                return false;
            }
        }
        bool dfs(vector<vector<char>>& board,int start,int x,int y,string& word)
{
            char ch = word[start];
            bool matched = false;
            if(ump.count(ch)){
                for(auto current: ump[ch]){
                    int row1 = current[0];
                    int col1 = current[1];
                    //是否相邻
                    if(start==0 || x==row1 && abs(y-col1)==1 || y==col1 &&
abs(x-row1)==1){
                        if(board[row1][col1]!='.'){
                            board[row1][col1] = '.';

                            if(start<word.size()-1){
                                matched = dfs(board,start+1,row1,col1,word);
                                if(matched) return true;
                            }else if(start==word.size()-1){
```

```
                                return true;
                            }

                            board[row1][col1] = ch;
                        }
                    }
                };
            }else{
                return false;
            }

            return false;
        };
    private:
        unordered_map<char,vector<vector<int>>> ump;
};
```

惭愧的是，性能还不如普通的回溯法：

```
Runtime: 764 ms, faster than 5.02% of C++ online submissions for Word Search.
Memory Usage: 169.2 MB, less than 16.18% of C++ online submissions for Word
Search.
```