126. Word Ladder II

Total Accepted: **45212** Total Submissions: **329392** Difficulty: **Hard**

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find all shortest transformation sequence(s)

from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time

2. Each intermediate word must exist in the word list

For example,

Given:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

Return

[

["hit","hot","dot","dog","cog"],

["hit","hot","lot","log","cog"]

]

Note:

- All words have the same length.

- All words contain only lowercase alphabetic characters.

```
//C++
//zzw
class Solution {
public:
    vector<vector<string>> result;
    vector<string> path;
    unordered_map<string,vector<string> >myMap;
    void getChildren(string s,unordered_set<string> &next,unordered_set<string>
&wordList)
    {
        for(int i=0;i<s.length();i++)
        {
            string temp = s;
            for(int j=0;j<26;j++)
            {
                temp[i] = j + 'a';
                if(wordList.count(temp)>0)
                {
                    next.insert(temp);
                    myMap[temp].push_back(s);
                }
            }
```

```
        }
    }

    void printPath(string beginWord,string endWord)
    {
        path.push_back(endWord);
        if(beginWord == endWord)
        {
            reverse(path.begin(),path.end());
            result.push_back(path);
            reverse(path.begin(),path.end());
        }else if (myMap.count(endWord)>0){
            vector<string> v = myMap[endWord];
            for(int i=0;i<v.size();++i)
            {
                printPath(beginWord,v[i]);
            }
        }
        path.pop_back();
    }

    vector<vector<string>> findLadders(string beginWord, string endWord,
unordered_set<string> &wordList) {
        wordList.insert(beginWord);
        wordList.insert(endWord);
        unordered_set<string> current;
        unordered_set<string> next;
        current.insert(beginWord);
        while(current.size()>0)
        {
            if(current.count(endWord)>0)
            {
                printPath(beginWord,endWord);
                return result;
            }
            unordered_set<string>::iterator it;
            for(it=current.begin();it!=current.end();++it)
            {
                wordList.erase(*it);
            }
            for(it=current.begin();it!=current.end();++it)
            {
                string s = *it;
                getChildren(s,next,wordList);
            }
            current.clear();
            current = next;
            next.clear();
        }

        return result;
    }
};
```