1. Word Ladder II

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 transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

**Note:**

* Return an empty list if there is no such transformation sequence.
* All words have the same length.
* All words contain only lowercase alphabetic characters.
* You may assume no duplicates in the word list.
* You may assume *beginWord* and *endWord* are non-empty and are not the same.

**Example 1:**

Input:  
beginWord = "hit",  
endWord = "cog",  
wordList = ["hot","dot","dog","lot","log","cog"]  
  
Output:  
[  
 ["hit","hot","dot","dog","cog"],  
 ["hit","hot","lot","log","cog"]  
]

**Example 2:**

Input:  
beginWord = "hit"  
endWord = "cog"  
wordList = ["hot","dot","dog","lot","log"]  
  
Output: []  
  
Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

**解** 最短路问题，dijkstra算法

* Dijkstra() {  
   初始化;  
   for(循环n次) {  
   u = 使dis[u]最小的还未被访问的顶点的编号;  
   记u为确定值;  
   for(从u除法能到达的所有顶点v){  
   for(v未被访问 && 以u为中介点使s到顶点v的最短距离更优)  
   优化dis[v];  
   }  
   }  
   }

class Solution {  
public:  
 vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wordList) {  
 wordList.push\_back(beginWord);  
 int n = wordList.size();  
 vector<vector<int>>g(n);  
 int e = -1;  
 for(int i = 0; i < n; ++i){  
 if(wordList[i] == endWord)e = i;  
 for(int j = i + 1; j < n;++j){  
 if(isNeighbor(wordList[i], wordList[j])){  
 g[i].push\_back(j);  
 g[j].push\_back(i);  
 }  
 }  
 }  
 vector<vector<int>> pre = dij(g, n - 1);  
   
 vector<string> tmp;  
 vector<vector<string>> path;  
 if(e!=-1)dfs(wordList, n-1, e, tmp, path, pre);  
 return path;  
   
 }  
   
 vector<vector<int>> dij(vector<vector<int>>&g, int s){  
 int n = g.size();  
 vector<int>dist(n, INT\_MAX);  
 vector<vector<int>> pre(n);  
 vector<bool>vis(n, false);  
   
 dist[s] = 0;  
 for(int i = 0; i < n; ++i){  
 int u = -1, mind = INT\_MAX;  
 for(int j = 0; j < n; ++j){  
 if(!vis[j] && dist[j] < mind){  
 u = j;  
 mind = dist[j];  
 }  
 }  
 if(u == -1)break;  
 vis[u] = true;  
 for(int v : g[u]){  
 if(!vis[v]){  
 if(dist[u] + 1 < dist[v]){  
 pre[v].clear();  
 pre[v].push\_back(u);  
 dist[v] = dist[u] + 1;  
 }else if(dist[u] + 1 == dist[v]){  
 pre[v].push\_back(u);  
 }  
 }  
 }  
 }  
 return pre;  
 }  
 void dfs(vector<string>& w, int s, int e,   
 vector<string>& tmp, vector<vector<string>>& path,  
 vector<vector<int>>& pre){  
 tmp.push\_back(w[e]); // 在此处加入路径  
 if(e == s){  
 reverse(tmp.begin(), tmp.end());  
 path.push\_back(tmp);  
 reverse(tmp.begin(), tmp.end());  
 tmp.pop\_back();  
 return;  
 }  
 for(int u : pre[e]){  
 dfs(w, s, u, tmp, path, pre);  
 }  
 tmp.pop\_back(); // 退出最后一个  
 }  
   
 bool isNeighbor(const string& s1, const string& s2){  
 bool flag = false;  
 if(s1.size() != s2.size())return flag;  
 for(int i = 0; i < s1.size(); ++i){  
 if(s1[i] != s2[i]){  
 if(flag)return false;  
 flag = true;  
 }  
 }  
 return flag;  
 }  
};