

//52ms

int ladderLength(string beginWord, string endWord, vector<string>& wordList)

{

unordered\_set<string> s1;

unordered\_set<string> s2;

unordered\_set<string> dict(wordList.begin(),wordList.end());

if(!dict.count(endWord)) return 0;

int len=beginWord.size();

int ans=0;

s1.insert(beginWord);

s2.insert(endWord);

while(!s1.empty() && !s2.empty()){

ans++;

if(s1.size()>s2.size()){

swap(s1,s2);

}

unordered\_set<string> cur;

for(string w:s1){

for(int i=0;i<len;i++){

char temp=w[i];

for(char x='a';x<='z';x++){

w[i]=x;

if(s2.count(w)){

return ans+1;

}

if(!dict.count(w))continue;

dict.erase(w);

cur.insert(w);

}

w[i]=temp;

}

}

s1=cur;

}

return 0;

}

};

//mine 168ms

int ladderLength(string beginWord, string endWord, vector<string>& wordList) {

unordered\_set<string> s(wordList.begin(),wordList.end());

if(s.find(endWord)==s.end()) return 0;

else

{

int l=beginWord.length();

queue<string> que;

que.push(beginWord);

int count=1,size=1;

while(!que.empty())

{

while(size>0)

{

string c=que.front();

que.pop();

for(int i=0;i<l;i++)

{

char ch=c[i];

for(char x='a';x<='z';x++)

{

c[i]=x;

auto it=s.find(c);

if(it!=s.end())

{

if(c==endWord) return count+1;

else{ que.push(c);

s.erase(it);}

}

}

c[i]=ch;

}

size--;

}

size=que.size();

count++;

}

}

return 0;

}

//24ms

int BBFS(unordered\_set<string> &dictionary,

int length,

unordered\_set<string> &beginSet,

unordered\_set<string> &endSet,

unordered\_set<string> &workingSet)

{

workingSet.clear();

int n = beginSet.begin()->size();

for(auto &i : beginSet)

{

dictionary.erase(i);

}

for(auto &word : beginSet)

{

for(int i=0; i<n; ++i)

{

string newWord = word;

for(char c='a'; c<='z'; ++c)

{

newWord[i] = c;

if(dictionary.count(newWord))

{

if(endSet.count(newWord))

{

return length+1;

}

workingSet.insert(newWord);

}

}

}

}

if(workingSet.size() == 0)

{

return 0;

}

if(endSet.size() < workingSet.size())

{

return BBFS(dictionary, length+1, endSet, workingSet, beginSet);

}

return BBFS(dictionary, length+1, workingSet, endSet, beginSet);

}

int ladderLength(string beginWord, string endWord, vector<string>& wordList)

{

unordered\_set<string> dictionary(wordList.begin(), wordList.end());

if(dictionary.count(endWord) == 0)

{

return 0;

}

unordered\_set<string> beginSet;

beginSet.insert(beginWord);

unordered\_set<string> endSet;

endSet.insert(endWord);

unordered\_set<string> workingSet;

return BBFS(dictionary, 1, beginSet, endSet, workingSet);

}