140. Word Break II <Hard>

***DFS(Depth First Search) + Memoization***

class Solution {

public:

vector<string> wordBreak(string s, vector<string>& wordDict) {

unordered\_set<string> wordSet{begin(wordDict), end(wordDict)};

unordered\_map<string, vector<string>> memo;

return wordBreak(s, wordSet, memo);

}

private:

vector<string> wordBreak(const string& s,

const unordered\_set<string>& wordSet,

unordered\_map<string, vector<string>>& memo) {

if (memo.count(s))

return memo[s];

vector<string> ans;

// 1 <= prefix.length() < s.length()

for (int i = 1; i < s.length(); ++i) {

const string& prefix = s.substr(0, i);

const string& suffix = s.substr(i);

if (wordSet.count(prefix))

for (const string& word : wordBreak(suffix, wordSet, memo))

ans.push\_back(prefix + " " + word);

}

// contains whole string, so don't add any space

if (wordSet.count(s))

ans.push\_back(s);

return memo[s] = ans;

}

};

class Solution {

public:

vector<string> wordBreak(string s, vector<string>& wordDict) {

unordered\_map<int, vector<string>> ump; //To store the result

unordered\_set<string> words(wordDict.begin(), wordDict.end());

return helper(s,0,words,ump);

}

vector<string> helper(string s, int index, unordered\_set<string> &words, unordered\_map<int, vector<string>> &ump){

vector<string> res, sub;

int n = s.size();

if(n == index)

{

res.push\_back(" ");

return res;

}

if(ump.count(index))

return ump[index];

string temp;

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

{

temp += s[i];

if(words.count(temp))

{

sub = helper(s, i+1, words, ump);

for(auto s : sub){

res.push\_back(s == " " ? temp : temp + " " + s);

}

}

}

return ump[index] = res;

}

};