**1048. Longest String Chain: -**

Medium Accepted: 337.8K Submissions: 563.2K Acceptance Rate: 60.0%

You are given an array of words where each word consists of lowercase English letters.

wordA is a **predecessor** of wordB if and only if we can insert **exactly one** letter anywhere in wordA **without changing the order of the other characters** to make it equal to wordB.

* For example, "abc" is a **predecessor** of "abac", while "cba" is not a **predecessor** of "bcad".

A **word chain**is a sequence of words [word1, word2, ..., wordk] with k >= 1, where word1 is a **predecessor** of word2, word2 is a **predecessor** of word3, and so on. A single word is trivially a **word chain** with k == 1.

Return *the****length****of the****longest possible word chain****with words chosen from the given list of*words.

**Example 1:**

**Input:** words = ["a","b","ba","bca","bda","bdca"]

**Output:** 4

**Explanation**: One of the longest word chains is ["a","ba","bda","bdca"].

**Example 2:**

**Input:** words = ["xbc","pcxbcf","xb","cxbc","pcxbc"]

**Output:** 5

**Explanation:** All the words can be put in a word chain ["xb", "xbc", "cxbc", "pcxbc", "pcxbcf"].

**Example 3:**

**Input:** words = ["abcd","dbqca"]

**Output:** 1

**Explanation:** The trivial word chain ["abcd"] is one of the longest word chains.

["abcd","dbqca"] is not a valid word chain because the ordering of the letters is changed.

**Constraints:**

* 1 <= words.length <= 1000
* 1 <= words[i].length <= 16
* words[i] only consists of lowercase English letters.

**Code: -**

class Solution {

public:

    int helper(string str, unordered\_map<string,bool> &mp, unordered\_map<string,int> &dp){

      // base case

      // dp found case

      if(dp[str] != -1)

        return dp[str];

      // recursive case

      int maxi = 0;

      string  temp = "";

      for(int i=0; i<str.size(); ++i){

        temp = "";

        temp += str.substr(0, i);

        temp += str.substr(i+1, str.size()-i+1);

        if(mp[temp] == true)

          maxi = max(maxi, helper(temp, mp, dp));

      }

      // return from current state

      return dp[str] = 1 + maxi;

    }

    int longestStrChain(vector<string>& word) {

      int n = word.size();

      if(n == 1)

        return 1;

      sort(word.begin(), word.end());

      unordered\_map<string,bool> mp;

      unordered\_map<string,int> dp;

      for(auto str:word){

        mp[str] = true;

        dp[str] = -1;

      }

      int ans = 1;

      for(int i=n-1; i>=0; --i){

        ans = max(ans, helper(word[i], mp, dp));

      }

      return ans;

    }

};