

解题思路一：

递推法：

对于数字的变化，使用for循环遍历即可，nums[i]要第一位数字，nums[j]要最后一位数字，前者/10，后者mod 10即可。对于互质数字的判断：  
①如果两个数其中一个为1，则两个数互质。  
②如果两个数正好可以整除，则不互质  
③如果没有整除则进行递推判断

最后判断返回值是否为1即可

代码：

#include <iostream>

#include <vector>

using namespace std;

class Solution {

public:

int countBeautifulPairs(vector<int>& nums) {

int count = 0;

int len = nums.size();

//遍历

for (int i = 0; i < len - 1; i++) {

int firstDigit = getFirstDigit(nums[i]);

for (int j = i + 1; j < len; j++) {

int lastDigit = getLastDigit(nums[j]);

if (gcd(firstDigit, lastDigit) == 1) {

count++;

}

}

}

return count;

}

private:

//获取当前下标的第一位数字

int getFirstDigit(int num) {

while (num >= 10) {

num /= 10;

}

return num;

}

//获取下一位下标的最后一位数字

int getLastDigit(int num) {

return num % 10;

}

//判断是不是互质

int gcd(int a, int b) {

while (b != 0) {

int t = a % b;

a = b;

b = t;

}

return a;

}

};

int main() {

Solution solution;

vector<int> array = { 31, 25, 72, 79, 74 };

int count = solution.countBeautifulPairs(array);

cout << count << endl; //

return 0;

}

解题思路二：

递归法：

①b为0，a为最大公约数  
 ②b不为0，返回gcd(b,a%b)的最大公约数  
 如果互质，最后会返回1；不互质返回最大公约数

代码：

#include <iostream>

#include <vector>

using namespace std;

class Solution {

public:

int countBeautifulPairs(vector<int>& nums) {

int count = 0;

int len = nums.size();

for (int i = 0; i < len - 1; i++) {

int firstDigit = getFirstDigit(nums[i]);

for (int j = i + 1; j < len; j++) {

int lastDigit = getLastDigit(nums[j]);

if (gcd(firstDigit, lastDigit) == 1) {

count++;

}

}

}

return count;

}

private:

int getFirstDigit(int num) {

while (num >= 10) {

num /= 10;

}

return num;

}

int getLastDigit(int num) {

return num % 10;

}

//递归

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

};

int main() {

Solution solution;

vector<int> array = { 31, 25, 72, 79, 74 };

int count = solution.countBeautifulPairs(array);

cout << count << endl;

return 0;

}

解题思路三：

由于最高位在 [1,9]中，我们可以在遍历数组的同时，统计最高位的出现次数，这样就只需枚举 [1,9] 中的与x%10 互质的数，把对应的出现次数加到答案中。

代码：

class Solution {

public:

int countBeautifulPairs(vector<int>& nums) {

int ans = 0, cnt[10]{};

for (int x : nums) {

for (int y = 1; y < 10; y++) {

//对于y下标的值其实就是后面x/10后的值，即第一位数字的值,存储其出现次数

if (cnt[y] && gcd(y, x % 10) == 1) {

ans += cnt[y];

}

}

while (x >= 10) {

x /= 10;

}

cnt[x]++; // 统计最高位的出现次数

}

return ans;

}

private:

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

};



解题思路一：

哈希表：

使用哈希表存储遍历的单词的结果，对于每个单词，如果单词是字符从前缀，开始递归检查剩下的字母能不能让单词拼出来，如果可以拼出来，返回true,不能返回在循环结束后false。对于每个word的判断，我们使用substr即可，如果说符合word,substr(word.size()),使其变成从word.size()以后的字符串

#include <iostream>

#include <vector>

#include <string>

#include <unordered\_map>

using namespace std;

class Solution {

public:

// 主函数：判断是否可以用字典中的单词拼出字符串 s

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

unordered\_map<string, bool> memo; // 记忆化哈希表

return canFormStringHelper(s, wordDict, memo);

}

private:

// 辅助递归函数：带有记忆化功能

bool canFormStringHelper(const string& s, const vector<string>& wordDict, unordered\_map<string, bool>& memo) {

if (s.empty()) { // 如果字符串为空，返回 true

return true;

}

if (memo.find(s) != memo.end()) { // 如果已经在 memo 中，直接返回结果

return memo[s];

}

// 遍历字典中的每个单词

for (const string& word : wordDict) {

//从拷贝当前字符串第0个拷贝，检查是否和word相等

if (s.substr(0, word.size()) == word) { // 检查单词是否是字符串 s 的前缀

// 递归检查剩余部分是否可以用字典中的单词拼出

if (canFormStringHelper(s.substr(word.size()), wordDict, memo)) {

return memo[s] = true; // 可以拼出，存储结果并返回 true

}

}

}

return memo[s] = false; // 无法拼出，存储结果并返回 false

}

};

int main() {

Solution solution;

string s = "applepenapple"; // 输入字符串

vector<string> wordDict = { "apple", "pen" }; // 单词字典

if (solution.wordBreak(s, wordDict)) {

cout << "正确" << endl; // 输出“正确”表示可以拼出

}

else {

cout << "失败" << endl; // 输出“失败”表示无法拼出

}

return 0;

}

解题思路二：

动态规划，对于字符串中每个单词长度的字符，我们假设它能在字典中找到，那么dp[word.size()]应该是true,从下标0开始，我们设置下标0为ture;那么对于每个能匹配到的单词的下标之和应该就是ture,如第一个单词的长度是6且匹配到,那么dp[6]=ture,如果后续还能匹配到正确的单词，假定是3，那么dp[6+3]=ture。也就是说，我们只要确保最后一次输入的i的值是ture即可。由此观之，对于每个dp[i]有dp[i]=dp[j] && check(s[j..i−1])

代码：

class Solution {

public:

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

//字典入集合进行排序

auto wordDictSet = unordered\_set <string>();

for (auto word : wordDict) {

wordDictSet.insert(word);

}

auto dp = vector <bool>(s.size() + 1);

dp[0] = true;

//将字符串符合特定长度的字符进行拷贝然后进行验证

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

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

if (dp[j] && wordDictSet.find(s.substr(j, i - j)) != wordDictSet.end()) {

dp[i] = true;

break;

}

}

}

return dp[s.size()];

}

};