

题目描述通过笔记题解提交记录

2748. 美丽下标对的数目已解答

简单相关标签相关企业提示Ax

给你一个下标从 0 开始的整数数组 `nums` 。如果下标对 `i`、`j` 满足  $0 \leq i < j < \text{nums.length}$ ，如果 `nums[i]` 的第一个数字和 `nums[j]` 的最后一个数字互质，则认为 `nums[i]` 和 `nums[j]` 是一组 **美丽下标对**。

返回 `nums` 中 **美丽下标对** 的总数目。

对于两个整数 `x` 和 `y`，如果不存在大于 1 的整数可以整除它们，则认为 `x` 和 `y` **互质**。换言之，如果 `gcd(x, y) == 1`，则认为 `x` 和 `y` 互质，其中 `gcd(x, y)` 是 `x` 和 `y` 的 **最大公因数**。

**示例 1:**

**输入:** `nums = [2,5,1,4]`

**输出:** 5

**解释:** `nums` 中共有 5 组美丽下标对:

- `i = 0` 和 `j = 1`: `nums[0]` 的第一个数字是 2，`nums[1]` 的最后一个数字是 5。2 和 5 互质，因此 `gcd(2,5) == 1`。
- `i = 0` 和 `j = 2`: `nums[0]` 的第一个数字是 2，`nums[2]` 的最后一个数字是 1。2 和 1 互质，因此 `gcd(2,1) == 1`。
- `i = 1` 和 `j = 2`: `nums[1]` 的第一个数字是 5，`nums[2]` 的最后一个数字是 1。5 和 1 互质，因此 `gcd(5,1) == 1`。
- `i = 1` 和 `j = 3`: `nums[1]` 的第一个数字是 5，`nums[3]` 的最后一个数字是 4。5 和 4 互质，因此 `gcd(5,4) == 1`。
- `i = 2` 和 `j = 3`: `nums[2]` 的第一个数字是 1，`nums[3]` 的最后一个数字是 4。1 和 4 互质，因此 `gcd(1,4) == 1`。

因此，返回 5。

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]{};
    }
};

```


次数

```
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
        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);
}


};
```

## 139. 单词拆分

已解答 

中等

 相关标签

 相关企业

Aa

给你一个字符串 `s` 和一个字符串列表 `wordDict` 作为字典。如果可以利用字典中出现的一个或多个单词拼接出 `s` 则返回 `true`。

**注意：**不要求字典中出现的单词全部都使用，并且字典中的单词可以重复使用。

### 示例 1:

**输入：**`s = "leetcode", wordDict = ["leet", "code"]`

**输出：**`true`

**解释：**返回 `true` 因为 "leetcode" 可以由 "leet" 和 "code" 拼接成。

### 示例 2:

**输入：**`s = "applepenapple", wordDict = ["apple", "pen"]`

**输出：**`true`

**解释：**返回 `true` 因为 "applepenapple" 可以由 "apple" "pen" "apple" 拼接成。

注意，你可以重复使用字典中的单词。

### 示例 3:

**输入：**`s = "catsanddog", wordDict = ["cats", "dog", "sand", "and", "cat"]`

**输出：**`false`

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 `true`；那么对于每个能匹配到的单词的下标之和应该就是 `true`，如第一个单词的长度是 6 且匹配到，那么 `dp[6]=true`，如果后续还能匹配到正确的单词，假定是 3，那么 `dp[6+3]=true`。也就是说，我们只要确保最后一次输入的 `i` 的值是 `true` 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()];  
    }  
};
```