【实例 001】 反转一个 3 位整数

1. 问题描述

反转一个三位数。

2. 问题示例

输入 number＝123,输出 321。

3. 代码实现

相关代码如下：

```
#include <iostream>
using namespace std;
class Solution {
public:
    //定义一个反转整数的函数,输入为一个三位数,输出为反转后的数
    int reverseInteger(int number) {
        int a, b, c, m;
        a = number/100;                    //获取百位数字
        b = (number - a * 100)/10;         //获取十位数字
        c = number - a * 100 - b * 10;     //获取个位数字
        m = a + b * 10 + c * 100;          //将百位、十位、个位数字组合成一个新的整数
        return m;
    }
};
int main() {
    int a = 123, b = 0;
    cout << "输入:" << a << endl;
    Solution solution;
    b = solution.reverseInteger(a);
    cout << "输出:" << b << endl;
    return 0;
}
```

4. 运行结果

输入：123

输出：321

【实例 002】 合并排序数组

1. 问题描述

合并两个升序的整数数组 A 和 B,形成一个新的数组,新数组也要有序。

2. 问题示例

输入 arr1＝[1 2 3],arr2＝[4 5 6],输出[1 2 3 4 5 6],返回合并所有元素后的数组。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    vector<int> mergeSortedArray(vector<int> &a, vector<int> &b) {
        //初始化两个数组的指针和结果数组的指针
        int i = 0, j = 0, k = 0, l, n;
        //获取两个数组的长度
        l = a.size();
        n = b.size();
        //创建一个新的数组,长度为两个输入数组之和
        vector<int> c(n + l);
        //合并两个有序数组
        while (i < l && j < n) {
            if (a[i] <= b[j]) {
                c[k++] = a[i++];
            } else {
                c[k] = b[j];
                k++;
                j++;
            }
        }
        //如果数组 A 还有剩余元素,将其添加到结果数组中
        while (i < l) {
            c[k] = a[i];
            k++;
            i++;
        }
        //如果数组 B 还有剩余元素,将其添加到结果数组中
        while (j < n) {
            c[k] = b[j];
            k++;
            j++;
        }
        return c;
    }
};
int main() {
    //初始化两个有序数组
    vector<int> arr1 = {1, 2, 3};            //修改为实际元素值
    vector<int> arr2 = {4, 5, 6};
    Solution solution;
    //调用合并函数,得到合并后的有序数组
    vector<int> arr3 = solution.mergeSortedArray(arr1, arr2);
    cout << "输入:[" ;
    for (int i = 0; i < arr1.size(); i++) {
        cout << arr1[i] << " ";
    }
    cout << "],[";
    for (int i = 0; i < arr2.size(); i++) {
```

```
        cout << arr2[i] << " ";
    }
    cout << "]" << endl;
    cout << "输出:[" ;
    for (int i = 0; i < arr3.size(); i++) {
        cout << arr3[i] << " ";
    }
    cout << "]" << endl;
    return 0;
}
```

4. 运行结果

输入：[1 2 3],[4 5 6]

输出：[1 2 3 4 5 6]

【实例 003】 旋转字符串

1. 问题描述

给定一个字符串(以字符数组的形式)和一个偏移量,根据偏移量从左向右原地旋转字符串。

2. 问题示例

输入 str="abcdefg",offset=3,输出"efgabcd"。输入 str="abcdefg",offset=0,输出"abcdefg"。输入 str="abcdefg",offset=1,输出"gabcdef",返回旋转后的字符串。输入 str="abcdefg",offset=2,输出"fgabcde",返回旋转后的字符串。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <string>
using namespace std;
class Solution {
public:
    void reverse(string &str, int start, int end) {
        while (start < end) {
            char temp = str[start];
            str[start] = str[end];
            str[end] = temp;
            start++;
            end-- ;
        }
    }
    string rotate_string(string str, int offset) {
        int len = str.size();
        if (offset == 0 || offset >= len) {
            return str;
        }
        reverse(str, 0, len - 1);                    //旋转整个字符串
```

```
        reverse(str, 0, offset - 1);                    //旋转前 offset 个字符
        reverse(str, offset, len - 1);                  //旋转剩余字符
        return str;
    }
};
int main() {
    string str = "abcdefg";
    Solution solution;
    int offset = 0;
    cout << "输入:";
    cout << "str = \"" << str << "\", offset = " << offset << endl;
    cout << "输出:" ;
    cout << "\"" << solution.rotate_string(str, offset) << "\"" << endl;
    offset = 1;
    cout << "输入:";
    cout << "str = \"" << str << "\", offset = " << offset << endl;
    cout << "输出:";
    cout << "\"" << solution.rotate_string(str, offset) << "\"" << endl;
    offset = 2;
    cout << "输入:";
    cout << "str = \"" << str << "\", offset = " << offset << endl;
    cout << "输出:";
    cout << "\"" << solution.rotate_string(str, offset) << "\"" << endl;
    offset = 3;
    cout << "输入:";
    cout << "str = \"" << str << "\", offset = " << offset << endl;
    cout << "输出:";
    cout << "\"" << solution.rotate_string(str, offset) << "\"" << endl;
    return 0;
}
```

4. 运行结果

输入：str="abcdefg",offset=0

输出："abcdefg"

输入：str="abcdefg",offset=1

输出："gabcdef"

输入：str="abcdefg",offset=2

输出："fgabcde"

输入：str="abcdefg",offset=3

输出："efgabcd"

【实例 004】 相对排名

1. 问题描述

根据 N 名运动员的得分，找到相对等级和获得最高分前三名的人，分别获得金牌、银牌和铜牌。N 是正整数，并且不超过 10000，所有运动员的成绩都保证是独一无二的。

2. 问题示例

输入[5,4,3,2,1],输出["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"],前三名运动员得分较高,根据得分依次获得金牌、银牌和铜牌。对于后两名运动员,根据分数输出相对等级。

3. 代码实现

相关代码如下:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main() {
    //创建一个整数向量 arr,并初始化为{5, 4, 3, 2, 1}
    vector<int> arr = {5, 4, 3, 2, 1};
    int n = arr.size();
    cout << "输入:" << endl;
    cout << "[";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << "]" << endl;
    sort(arr.begin(), arr.end(), greater<int>());
    //输出排序后的数组及对应的奖牌
    cout << "输出:" << endl;
    for (int i = 0; i < n; i++) {
        string medal;
        if (i == 0) {
            medal = "金牌";
        } else if (i == 1) {
            medal = "银牌";
        } else if (i == 2) {
            medal = "铜牌";
        } else {
            medal = "第" + to_string(i + 1) + "名";
        }
        cout << medal << ",得分:" << arr[i] << endl;
    }
    return 0;
}
```

4. 运行结果

输入:[5 4 3 2 1]

输出:金牌,得分:5
银牌,得分:4
铜牌,得分:3
第 4 名,得分:2
第 5 名,得分:1

【实例 005】 二分查找

1. 问题描述

给定一个排序的整数数组(升序)和一个要查找的目标整数 target,请查找 target 第 1 次出现的下标(从 0 开始),如果 target 不存在于数组中,返回－1。

2. 问题示例

输入数组[1,4,4,5,7,7,8,9,9,10]和 target＝1,输出其所在的位置 0。

3. 代码实现

相关代码如下:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    //二分查找函数,返回目标值在数组中的第一个位置
    int binarySearch(vector<int> &nums, int target) {
        int low = 0, high = nums.size();
        while (low < high) {
            int mid = low + (high - low)/2;
            if (target == nums[mid]) {                    //如果中间位置的值等于目标值
                high = mid;                               //将 high 指针移动到 mid 位置
            } else if (target < nums[mid]) {              //如果目标值小于中间位置的值
                high = mid;
            } else {
                low = mid + 1;
            }
        }
        return (low < nums.size() && nums[low] == target) ? low : -1;
        //如果找到目标值,返回其位置,否则返回 -1
    }
};
int main() {
    Solution solution;                                    //创建 Solution 类的对象
    vector<int> nums = {1, 4, 4, 5, 7, 7, 8, 9, 9, 10};   //定义整数数组
    int target = 1;
    cout << "输入:[" ;
    for (int i = 0; i < nums.size(); i++) {
        cout << nums[i] << " ";
    }
    cout << "],target = 1" << endl;
    int result = solution.binarySearch(nums, target);     //调用函数,获取结果
    cout << "输出:" << result << endl;
    return 0;
}
```

4. 运行结果

输入:[1 4 4 5 7 7 8 9 9 10],target＝1

输出:0

【实例 006】 下一个更大的数

1. 问题描述

给定两个不重复的数组 nums1 和 nums2,其中 nums1 是 nums2 的子集。在 nums2 的相应位置找到 nums1 所有元素的下一个更大数字。

nums1 中数字的下一个更大数字是 nums2 中右边第 1 个更大的数字。如果它不存在,则为此数字输出－1。nums1 和 nums2 中的所有数字都是唯一的,nums1 和 nums2 的长度均不超过 1000。

2. 问题示例

输入 nums1＝[4,1,2],nums2＝[1,3,4,2],输出[－1,3,－1],对于第 1 个数组中的数字 4,在第 2 个数组中找不到下一个更大的数字,因此输出－1;对于第 1 个数组中的数字 1,第 2 个数组中的下一个更大数字是 3,则输出 3;对于第一个数组中的数字 2,第 2 个数组中没有下一个更大的数字,因此输出－1。

3. 代码实现

相关代码如下:

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;
class Solution {
public:
 vector<int> nextGreaterElement(vector<int> &nums1, vector<int> &nums2) {
        unordered_map<int, int> mp;          //创建哈希表,用于存储 nums2 中每个元素的索引
        vector<int> ans;                     //创建一个向量,用于存储结果
        for (int i = 0; i < nums2.size(); i++) {
            mp[nums2[i]] = i;                //将 nums2 中的元素及其索引存入哈希表
        }
        for (int i = 0; i < nums1.size(); i++) {
            int j = mp[nums1[i]];            //获取 nums1 中元素在 nums2 中的索引
            auto it = find_if(nums2.begin() + j, nums2.end(), [&](int x) {
                return x > nums1[i];         //查找 nums2 中下一个大于 nums1 中当前元素的值
            });
            if (it != nums2.end()) {
                ans.emplace_back(*it);       //如果找到元素值,将其添加到结果向量中
            } else {
                ans.emplace_back(-1);        //如果没找到元素值,添加 -1 到结果向量中
            }
        }
        return ans;
    }
};
int main() {
    Solution solution;                       //创建一个 Solution 对象
    vector<int> nums1 = {4, 1, 2};           //定义 nums1 向量
```

```
    vector<int> nums2 = {1, 3, 4, 2};      //定义 nums2 向量
    cout << "输入:[";
    for (int num : nums1) {
        cout << num << " ";                //输出 nums1 向量的元素
    }
    cout << "],[" ;
    for (int num : nums2) {
        cout << num << " ";                //输出 nums2 向量的元素
    }
    cout << "]" << endl;
    vector<int> result = solution.nextGreaterElement(nums1, nums2);
    cout << "输出:[";
    for (int num : result) {
        cout << num << " ";
    }
    cout << "]" << endl;
    return 0;
}
```

4. 运行结果

输入：[4 1 2],[1 3 4 2]

输出：[-1 3 -1]

【实例 007】 字符串中的单词数

1. 问题描述

计算字符串中的单词数，其中一个单词定义为不含空格的连续字符串。

2. 问题示例

输入"Hello, my name is John"，输出 5。

3. 代码实现

相关代码如下：

```
#include <iostream>
using namespace std;
class Solution {
public:
    int countSegments(string &s) {
        int res = 0;
        for (int i = 0; i < s.length(); i++) {
        //如果当前字符不是空格，前一个字符是空格或者是字符串的第一个字符，则计数器加 1
            if (s[i] != ' '&& (i == 0 || s[i - 1] == ' ')) {
                res++;
            }
        }
        return res;
    }
};
int main() {
    Solution solution;
```

```
    string input = "Hello, my name is John";
    cout << "输入:" << input << endl;
    int segments = solution.countSegments(input);
    cout << "输出: " << segments << endl;
    return 0;
}
```

4. 运行结果

输入：Hello，my name is John

输出：5

【实例 008】 勒索信

1. 问题描述

给定一个表示勒索信内容的字符串和一个表示杂志内容的字符串，写一个方法判断能否通过剪下杂志中的内容构造出这封勒索信，若可以，返回 True，否则返回 False。注：杂志字符串中的每个字符仅能在勒索信中使用一次。

2. 问题示例

输入 ransomNote="aa"，magazine="aab"，输出 True。勒索信的内容可以从杂志内容中构造出来。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <unordered_map>
using namespace std;
class Solution {
 public:
    bool canConstruct(string &ransomNote, string &magazine) {
        int n = ransomNote.size(), m = magazine.size();
        if (n == 0)
            return true;
        if (n > m)
            return false;
        unordered_map<char, int> charCount;
        for (char ch : magazine)
            charCount[ch]++;
        for (char ch : ransomNote) {
            if (charCount[ch] == 0)
                return false;
            else
                charCount[ch]--;
        }
        return true;
    }
};
int main() {
 Solution solution;
```

```
    string ransomNote = "aa";
    string magazine = "aab";
    cout << "输入:" << ransomNote << "," << magazine << endl;
    bool result = solution.canConstruct(ransomNote, magazine);
    cout << "输出:" << (result ? "True" : "False") << endl;
    return 0;
}
```

4. 运行结果

输入：ransomNote="aa",magazine="aab"

输出：True

【实例 009】 不重复的两个数

1. 问题描述

给定一个数组,其中除了 2 个数,其他数均出现 2 次,请找到不重复的 2 个数并返回。

2. 问题示例

给出 a=[1,2,5,5,6,6],返回 [1,2],除 1 和 2 外其他数都出现了 2 次,因此返回[1,2]。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
std::vector<int> findUniqueNumbers(const std::vector<int> &a) {
    std::unordered_map<int, int> countMap;      //创建哈希表,存储每个数字出现的次数
    for (int num : a) {                         //遍历数组中的每个数字
        countMap[num]++;                        //将数字添加到哈希表中,并更新其出现次数
    }
    std::vector<int> result;
    for (const auto &pair : countMap) {         //遍历哈希表中的每个键值对
        if (pair.second == 1) {                 //如果某个数字出现次数为1,说明它是唯一的
            result.push_back(pair.first);       //将该数字添加到结果向量中
        }
    }
    return result;
}
int main() {
    std::vector<int> a = {1, 2, 5, 5, 6, 6};    //定义一个数组
    cout << "输入:[";
    for (int num : a) {                         //遍历数组中的每个数字
        cout << num << " ";
    }
    cout << "]";
    std::vector<int> uniqueNumbers = findUniqueNumbers(a);
    cout << "输出: [";
    for (int num : uniqueNumbers) {             //遍历结果向量中的每个数字
        std::cout << num << " ";
```

```
    }
    cout << "]" << endl;
    return 0;
}
```

4. 运行结果

输入：[1 2 5 5 6 6]

输出：[1 2]

【实例 010】 双胞胎字符串

1. 问题描述

给定两个字符串 s 和 t，每次可以任意交换 s 的奇数位或偶数位上的字符，即奇数位上的字符能与其他奇数位的字符互换，偶数位上的字符也能与其他偶数位的字符互换，问能否经过若干次交换，使 s 变成 t?

2. 问题示例

输入 s="abcd"，t="cdab"，输出"Yes"，第一次 a 与 c 交换，第二次 b 与 d 交换。输入 s="abcd"，t="bcda"，输出"No"，无论如何交换，都无法得到 bcda。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <string>
using namespace std;
bool canSwapToEqual(string s, string t) {
    int n = s.size();
    if (n != t.size())
        return false;
    int odd_cnt[26] = {0}, even_cnt[26] = {0};
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0) {
            even_cnt[s[i] - 'a']++;
            even_cnt[t[i] - 'a']--;
        } else {
            odd_cnt[s[i] - 'a']++;
            odd_cnt[t[i] - 'a']--;
        }
    }
    for (int i = 0; i < 26; i++) {
        if (odd_cnt[i] != 0 || even_cnt[i] != 0)
            return false;
    }
    return true;
}
int main() {
    string s = "abcd", t = "cdab";
    cout << "输入:" << s << "," << t << endl;
    //cin >> s >> t;
```

```
    if (canSwapToEqual(s, t)) {
        cout << "输出:Yes" << endl;
    } else {
        cout << "输出:No" << endl;
    }
    return 0;
}
```

4. 运行结果

输入：s=abcd,t=cdab

输出：Yes

【实例 011】 最接近 target 的值

1. 问题描述

给出一个数组,在数组中找到两个数,使得它们的和最接近但不超过目标值,返回它们的和。

2. 问题示例

输入 array=[1,3,5,11,7],target=15,输出 14。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;
int closestSum(vector<int> &nums, int target) {
    sort(nums.begin(), nums.end());                          //对 nums 进行排序
    int left = 0, right = nums.size() - 1;                   //初始化左右指针
    int closestSum = INT_MIN;                                //初始化最接近的和为最小整数
    while (left < right) {                                   //当左指针小于右指针时,继续循环
        int currentSum = nums[left] + nums[right];           //左右指针指向两个数的和
        if (currentSum <= target) {                          //如果当前和小于或等于 target
            closestSum = max(closestSum, currentSum);        //更新最接近的和
            left++;                                          //左指针向右移动
        } else {
            right--;
        }
    }
    return closestSum;
}
int main() {
    vector<int> nums = {1, 3, 5, 11, 7};
    int target = 15;
    cout << "输入:[1, 3, 5, 11, 7], target = 15" << endl;
    int result = closestSum(nums, target);                   //调用 closestSum 函数,得到结果
    cout << "输出: " << result << endl;
    return 0;
}
```

4. 运行结果

输入：[1,3,5,11,7],target=15

输出：14

【实例 012】 点积

1. 问题描述

给出两个数组,求它们的点积。

2. 问题示例

输入 array1=[1,1,1],array2=[2,2,2],输出 6,即 1×2+1×2+1×2=6。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
int dotProduct(const vector<int> &array1, const vector<int> &array2) {
 if (array1.size() != array2.size()) {
     cerr << "Arrays must have the same length." << endl;
     return 0;              //返回 0 或者其他错误标识
 }
 int result = 0;
 for (size_t i = 0; i < array1.size(); ++i) {
     result += array1[i] * array2[i];
 }
 return result;
}
int main() {
 vector<int> array1 = {1, 1, 1};
 vector<int> array2 = {2, 2, 2};
 cout << "输入:[";
 for (int num : array1) {
     cout << num << " ";
 }
 cout << "], [";
 for (int num : array2) {
     cout << num << " ";
 }
 cout << "]" << endl;
 int result = dotProduct(array1, array2);
 cout << "输出: " << result << endl;
 return 0;
}
```

4. 运行结果

输入：[1,1,1],[2,2,2]

输出：6

【实例 013】 函数运行时间

1. 问题描述

给定一系列描述函数进入和退出的时间，求每个函数的运行时间。

2. 问题示例

输入 s=["F1 Enter 10","F2 Enter 18","F2 Exit 19","F1 Exit 20"]，输出["F1|10","F2|1"]，即 F1 在 10 时刻进入，20 时刻退出，运行时长为 10，F2 在 18 时刻进入，19 时刻退出，运行时长为 1。

输入 s=["F1 Enter 10","F1 Exit 18","F1 Enter 19","F1 Exit 20"]，输出["F1|9"]，即 F1 在 10 时刻进入，18 时刻退出；又在 19 时刻进入，20 时刻退出，总运行时长为 9。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <sstream>
using namespace std;
vector<string> calculateRuntime(const vector<string> &logs) {
    unordered_map<string, int> entryTimes;       //存储任务的进入时间
    unordered_map<string, int> totalRuntime;
    for (const string &log : logs) {
        istringstream iss(log);
        string task, action;
        int time;
        iss >> task >> action >> time;
        if (action == "Enter") {
            entryTimes[task] = time;
        } else {
            //如果动作是"Exit",则计算任务的运行时间并累加到总运行时间中
            int entryTime = entryTimes[task];
            totalRuntime[task] += (time - entryTime);
        }
    }
    vector<string> result;
    for (const auto &entry : totalRuntime) {
        const string &task = entry.first;
        int runtime = entry.second;
        result.push_back(task + "|" + to_string(runtime));
    }
    return result;
}
int main() {
    vector<string> logs1 = {"F1 Enter 10", "F2 Enter 18", "F2 Exit 19", "F1 Exit 20"};
    cout << "输入: [";
    for (size_t i = 0; i < logs1.size(); ++i) {
        cout << "\"" << logs1[i] << "\"";
```

```
            if (i < logs1.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
        vector<string> result1 = calculateRuntime(logs1);
        cout << "输出: [";
        for (size_t i = 0; i < result1.size(); ++i) {
            cout << "\"" << result1[i] << "\"";
            if (i < result1.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
        vector<string> logs2 = {"F1 Enter 10", "F1 Exit 18", "F1 Enter 19", "F1 Exit 20"};
        cout << "输入: [";
        for (size_t i = 0; i < logs2.size(); ++i) {
            cout << "\"" << logs2[i] << "\"";
            if (i < logs2.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
        vector<string> result2 = calculateRuntime(logs2);
        cout << "输出: [";
        for (size_t i = 0; i < result2.size(); ++i) {
            cout << "\"" << result2[i] << "\"";
            if (i < result2.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
        return 0;
    }
```

4. 运行结果

输入：["F1 Enter 10", "F2 Enter 18", "F2 Exit 19", "F1 Exit 20"]

输出：["F1|10", "F2|1"]

输入：["F1 Enter 10", "F1 Exit 18", "F1 Enter 19", "F1 Exit 20"]

输出：["F1|9"]

【实例 014】 查询区间

1. 问题描述

给定一个包含若干区间的 List 数组，长度是 1000，如[500,1500]、[2100,3100]。给定一个 number，判断 number 是否在这些区间内，如是则返回 True，否则返回 False。

2. 问题示例

输入 List=[[100,1100],[1000,2000],[5500,6500]]和 number=6000，输出 True，因

为 6000 在区间[5500,6500]。输入 List=[[100,1100],[2000,3000]]和 number=3500,输出 False,3500 不在 list 的任何一个区间中。

3. 代码实现

相关代码如下:

```
#include <iostream>
#include <vector>
using namespace std;
bool isNumberInIntervals(const vector<vector<int>> &intervals, int number) {
    for (const auto &interval : intervals) {
        if (number >= interval[0] && number <= interval[1]) {
            return true;
        }
    }
    return false;
}
int main() {
    vector<vector<int>> intervals1 = {{100, 1100}, {1000, 2000}, {5500, 6500}};
    int number1 = 6000;
    cout << "输入:[";
    for (size_t i = 0; i < intervals1.size(); ++i) {
        cout << "[";
        for (size_t j = 0; j < intervals1[i].size(); ++j) {
            cout << intervals1[i][j];
            if (j < intervals1[i].size() - 1) {
                cout << ", ";
            }
        }
        cout << "]";
        if (i < intervals1.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]," << number1 << endl;
    bool result1 = isNumberInIntervals(intervals1, number1);
    cout << "输出: " << (result1 ? "True" : "False") << endl;
    vector<vector<int>> intervals2 = {{100, 1100}, {2000, 3000}};
    int number2 = 3500;
    cout << "输入:[";
    for (size_t i = 0; i < intervals2.size(); ++i) {
        cout << "[";
        for (size_t j = 0; j < intervals2[i].size(); ++j) {
            cout << intervals2[i][j];
            if (j < intervals2[i].size() - 1) {
                cout << ", ";
            }
        }
        cout << "]";
        if (i < intervals2.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]," << number2 << endl;
```

```
    bool result2 = isNumberInIntervals(intervals2, number2);
    cout << "输出: " << (result2 ? "True" : "False") << endl;
    return 0;
}
```

4. 运行结果

输入：[[100,1100], [1000,2000], [5500,6500]],6000

输出：True

输入：[[100,1100], [2000,3000]],3500

输出：False

【实例 015】 两数之和

1. 问题描述

给定一个整数数组 nums 和一个整数目标值，请在该数组中找出和为目标值的两个整数，并返回它们的数组下标。可以假设每种输入只对应一个答案，但是数组中同一个元素在答案中不能重复出现，可以按任意顺序返回答案。

2. 问题示例

输入 nums=[2,7,11,15], target=9，输出[0,1]。

注：因为 nums[0]+nums[1]=9，所以返回[0, 1]。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    //定义一个名为 twoSum 的成员函数,接收一个整数向量 nums 和一个整数 target 作为参数
    vector<int> twoSum(vector<int> &nums, int target) {
        int n = nums.size();                    //获取 nums 的大小
        //使用两层循环遍历 nums 中的元素
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                //如果找到了和为 target 的两个整数,返回这两个数的下标
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }
        return {};
    }
};
int main() {
    Solution solution;                          //创建一个 Solution 类的实例
    vector<int> nums = {2, 7, 11, 15};          //定义一个整数向量 nums
    int target = 9;                             //定义一个整数 target
```

```
    vector<int> result = solution.twoSum(nums, target);
 cout << "输入:[2, 7, 11, 15], target = 9\n输出:";
 cout << "[" << result[0] << ", " << result[1] << "]" << endl;
 return 0;
}
```

4. 运行结果

输入：[2,7,11,15]，target=9

输出：[0,1]

【实例 016】 二进制求和

1. 问题描述

给定两个二进制字符串 a 和 b,以二进制字符串的形式返回它们的和。

2. 问题示例

输入 a=11,b=1,输出 100。

3. 代码实现

相关代码如下：

```
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
class Solution {
public:
    string addBinary(string a, string b) {
        string ans;
        reverse(a.begin(), a.end());          //反转字符串 a,方便从低位开始相加
        reverse(b.begin(), b.end());          //反转字符串 b,方便从低位开始相加
        int n = max(a.size(), b.size()), carry = 0;
        for (size_t i = 0; i<n; ++i) {        //遍历每一位然后相加
            carry += i<a.size() ? (a.at(i) == '1') : 0;
            //如果当前位在字符串 a 中,则加上对应的值(0 或 1),否则不加
            carry += i<b.size() ? (b.at(i) == '1') : 0;
            //如果当前位在字符串 b 中,则加上对应的值(0 或 1),否则不加
            ans.push_back((carry % 2) ? '1' : '0');
            //将当前位的结果添加到结果字符串中
            carry /= 2;
        }
        if (carry) {                          //如果最后还有进位,则将其添加到结果字符串中
            ans.push_back('1');
        }
        reverse(ans.begin(), ans.end());      //反转结果字符串,使其恢复原来的顺序
        return ans;                           //返回结果字符串
    }
};
int main() {
    Solution solution;
    string a = "11";                          //定义字符串 a
```

```
    string b = "1";                          //定义字符串 b
    string result = solution.addBinary(a, b);
    cout << "输入:a = " << a << ", b = " << b << endl;
    cout << "输出:" << result << endl;
    return 0;
}
```

4. 运行结果

输入：a=11,b=1

输出：100

【实例 017】 数组剔除元素后的乘积

1. 问题描述

给定一个整数数组 A。定义 B[i]=A[0]×...＊×A[i−1]＊×A[i+1]×...×A[n−1]，即 B[i]为剔除 A[i]元素之后所有数组元素之积，计算数组 B 时不要使用除法，输出数组 B。

2. 问题示例

输入 A=[1,2,3]，输出[6,3,2]，即 B[0]=A[1]×A[2]=6；B[1]=A[0]×A[2]=3；B[2]=A[0]×A[1]=2。

3. 代码实现

相关代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    vector<long long> productExcludeItself(vector<int> &nums) {
        int iSize = nums.size();                    //获取数组长度
        vector<long long> multiRet;                 //存储结果的向量
        long long multi = 1;                        //用于计算乘积的变量
        int i = 0, j = 0, k = 0;
        for (i = 0; i < iSize; i++) {
            multi = 1;
            for (j = 0; j < i; j++) {
                multi *= nums[j];
            }
            for (k = i + 1; k < iSize; k++) {
                multi *= nums[k];
            }
            //将当前元素的乘积添加到结果向量中
            multiRet.push_back(multi);
        }
        return multiRet;                            //返回结果向量
    }
};
int main() {
    Solution solution;
```

```
    vector<int> nums = {1, 2, 3};
    vector<long long> result = solution.productExcludeItself(nums);
    cout << "输入:[";
    for (int num : nums) {
        cout << num << " ";
    }
 cout << "]\n 输出:[";
 for (int num : result) {
     cout << num << " ";
 }
 cout << "]" << endl;
 return 0;
}
```

4. 运行结果

输入:[1 2 3]

输出:[6 3 2]

【实例 018】 键盘的一行

1. 问题描述

给定一个单词列表,返回可以在键盘上使用字母键输入的单词。可以多次使用键盘中的一个字符,输入字符串仅包含字母表中的字母。

2. 问题示例

输入["Hello","Alaska","Dad","Peace"],输出["Alaska","Dad"],即这 2 个单词可以在键盘的第 3 行输出。

3. 代码实现

相关代码如下:

```
#include<iostream>
#include<vector>
using namespace std;
class Solution{
public:
    vector<string> findWords(vector<string> &words) {
        //定义 3 个字符串,分别表示键盘上的 3 行字符
        vector<string> s = {"qwertyuiopQWERTYUIOP",
                            "asdfghjklASDFGHJKL",
                            "ZXCVBNMzxcvbnm"
                            };
        vector<string> res;                                    //存储向量
        for (int i = 0; i < words.size(); i++) {               //遍历输入的单词列表
            for (int j = 0; j < s.size(); j++) {               //遍历键盘上的 3 行字符
                int pos = s[j].find(words[i][0]);
                if (pos == -1)
                    continue;                                  //如果找不到,跳过当前行
                int k = 1;
                for (k = 1; k < words[i].size(); k++) {        //遍历当前单词的剩余字符
```