

力扣500题刷题笔记

207. 课程表

你这个学期必须选修 `numCourses` 门课程，记为 `0` 到 `numCourses - 1`。在选修某些课程之前需要一些先修课程。先修课程按数组 `prerequisites` 给出，其中 `prerequisites[i] = [ai, bi]`，表示如果要学习课程 `ai` 则必须先学习课程 `bi`。

- 例如，先修课程对 `[0, 1]` 表示：想要学习课程 `0`，你需要先完成课程 `1`。

请你判断是否可能完成所有课程的学习？如果可以，返回 `true`；否则，返回 `false`。

示例 1:

```
1 输入: numCourses = 2, prerequisites = [[1,0]]
2 输出: true
3 解释: 总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。这是可能的。
```

示例 2:

```
1 输入: numCourses = 2, prerequisites = [[1,0],[0,1]]
2 输出: false
3 解释: 总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0；并且学习课程 0 之前，你还应先完成课程 1。这是不可能的。
```

提示:

- `1 <= numCourses <= 10^5`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= ai, bi < numCourses`
- `prerequisites[i]` 中的所有课程对 **互不相同**

思路

(拓扑排序) $O(n + m)$

① 统计每个节点的入度

② $in[u] = 0$ 入队

```
while (true) {
    // ...
}
```



拓扑排序:

对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序,是将G中所有顶点排成一个线性序列,使得图中任意一对顶点u和v,若 $\langle u, v \rangle \in E(G)$,则u在线性序列中出现在v之前。

一个合法的选课序列就是一个拓扑序,拓扑序是指一个满足有向图上,不存在一条边出节点在入节点前的线性序列,如果有向图中有环,就不存在拓扑序。可以通过拓扑排序算法来得到拓扑序,以及判断是否存在环。

拓扑排序步骤:

- 1、建图并记录所有节点的入度。
- 2、将所有入度为0的节点加入队列。
- 3、取出队首的元素 `now`, 将其加入拓扑序列。
- 4、访问所有 `now` 的邻接点 `next`, 将 `next` 的入度减1, 当减到0后, 将 `next` 加入队列。
- 5、重复步骤3、4, 直到队列为空。
- 6、如果拓扑序列个数等于节点数, 代表该有向图无环, 且存在拓扑序。

时间复杂度分析: 假设 n 为点数, m 为边数, 拓扑排序仅遍历所有的点和边一次, 故总时间复杂度为 $O(n + m)$ 。

c++代码

```

1  class Solution {
2  public:
3      bool canFinish(int n, vector<vector<int>>& edges) {
4          vector<vector<int>> g(n);
5          vector<int> d(n); // 存储每个节点的入度
6          for(auto edge : edges){
7              g[edge[1]].push_back(edge[0]); // 建图
8              d[edge[0]]++; // 入度加1
9          }
10
11         queue<int> q;
12         for(int i = 0; i < n; i++){
13             if(d[i] == 0) q.push(i); // 将所有入度为0的节点加入队列。

```

```

14         }
15
16         int cnt = 0; //统计拓扑节点的个数
17         while(q.size()){
18             int t = q.front();
19             q.pop();
20             cnt++;
21             for(int i : g[t]){ //访问t的邻接节点(出边)
22                 d[i]--;
23                 if(d[i] == 0) q.push(i);
24             }
25         }
26
27         return cnt == n;
28     }
29 };

```

438. 找到字符串中所有字母异位词 *

题目

给定两个字符串 `s` 和 `p`，找到 `s` 中所有 `p` 的 **异位词** 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

异位词 指由相同字母重排列形成的字符串（包括相同的字符串）。

示例 1:

```

1  输入: s = "cbaebabacd", p = "abc"
2  输出: [0,6]
3  解释:
4  起始索引等于 0 的子串是 "cba", 它是 "abc" 的异位词。
5  起始索引等于 6 的子串是 "bac", 它是 "abc" 的异位词。

```

示例 2:

```

1  输入: s = "abab", p = "ab"
2  输出: [0,1,2]
3  解释:
4  起始索引等于 0 的子串是 "ab", 它是 "ab" 的异位词。
5  起始索引等于 1 的子串是 "ba", 它是 "ab" 的异位词。
6  起始索引等于 2 的子串是 "ab", 它是 "ab" 的异位词。

```

思路

c++代码

```

1  class Solution {
2  public:
3      vector<int> findAnagrams(string s, string p) {
4          unordered_map<char, int> hs, hp;
5          for(int c : p) hp[c]++;
6          vector<int> res;
7          for(int i = 0, j = 0; i < s.size(); i++){
8              hs[s[i]]++;
9              while(hs[s[i]] > hp[s[i]]) hs[s[j++]]--;

```

```

10         if(i - j + 1 == p.size()){
11             res.push_back(j);
12         }
13     }
14     return res;
15 }
16 };

```

621. 任务调度器

给你一个用字符数组 `tasks` 表示的 CPU 需要执行的任务列表。其中每个字母表示一种不同种类的任务。任务可以以任意顺序执行，并且每个任务都可以在 1 个单位时间内执行完。在任何一个单位时间，CPU 可以完成一个任务，或者处于待命状态。

然而，两个 **相同种类** 的任务之间必须有长度为整数 `n` 的冷却时间，因此至少有连续 `n` 个单位时间内 CPU 在执行不同的任务，或者在待命状态。

你需要计算完成所有任务所需要的 **最短时间**。

示例 1:

```

1  输入: tasks = ["A","A","A","B","B","B"], n = 2
2  输出: 8
3  解释: A -> B -> (待命) -> A -> B -> (待命) -> A -> B
4      在本示例中，两个相同类型任务之间必须间隔长度为 n = 2 的冷却时间，而执行一个任务只需要一个单位时间，所以中间出现了（待命）状态。

```

示例 2:

```

1  输入: tasks = ["A","A","A","B","B","B"], n = 0
2  输出: 6
3  解释: 在这种情况下，任何大小为 6 的排列都可以满足要求，因为 n = 0
4      ["A","A","A","B","B","B"]
5      ["A","B","A","B","A","B"]
6      ["B","B","B","A","A","A"]
7      ...
8  诸如此类

```

示例 3:

```

1  输入: tasks = ["A","A","A","A","A","A","B","C","D","E","F","G"], n = 2
2  输出: 16
3  解释: 一种可能的解决方案是:
4      A -> B -> C -> A -> D -> E -> A -> F -> G -> A -> (待命) -> (待命) -> A ->
      > (待命) -> (待命) -> A

```

提示:

- `1 <= task.length <= 104`
- `tasks[i]` 是大写英文字母
- `n` 的取值范围为 `[0, 100]`

思路

c++代码

```

1  class Solution {
2  public:
3      int leastInterval(vector<char>& tasks, int n) {
4          unordered_map<char, int> hash;
5          for (auto c: tasks) hash[c] ++ ;
6          int maxc = 0, cnt = 0;
7          for (auto [k, v]: hash) maxc = max(maxc, v);
8          for (auto [k, v]: hash)
9              if (maxc == v)
10                 cnt ++ ;
11          return max((int)tasks.size(), (maxc - 1) * (n + 1) + cnt);
12      }
13  };
14

```

581. 最短无序连续子数组 *

题目

给你一个整数数组 `nums`，你需要找出一个 **连续子数组**，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

请你找出符合题意的 **最短** 子数组，并输出它的长度。

示例 1:

```

1  输入: nums = [2,6,4,8,10,9,15]
2  输出: 5
3  解释: 你只需要对 [6, 4, 8, 10, 9] 进行升序排序，那么整个表都会变为升序排序。

```

示例 2:

```

1  输入: nums = [1,2,3,4]
2  输出: 0

```

示例 3:

```

1  输入: nums = [1]
2  输出: 0

```

提示:

- `1 <= nums.length <= 10^4`
- `-10^5 <= nums[i] <= 10^5`

思路1

(排序) $O(n \log n)$

- 1、将原数组拷贝一份，然后对拷贝后的数组排序。
- 2、对比原数组和排序后的数组，除去前半部分和后半部分相同的数字后，剩余数字的长度就是答案。

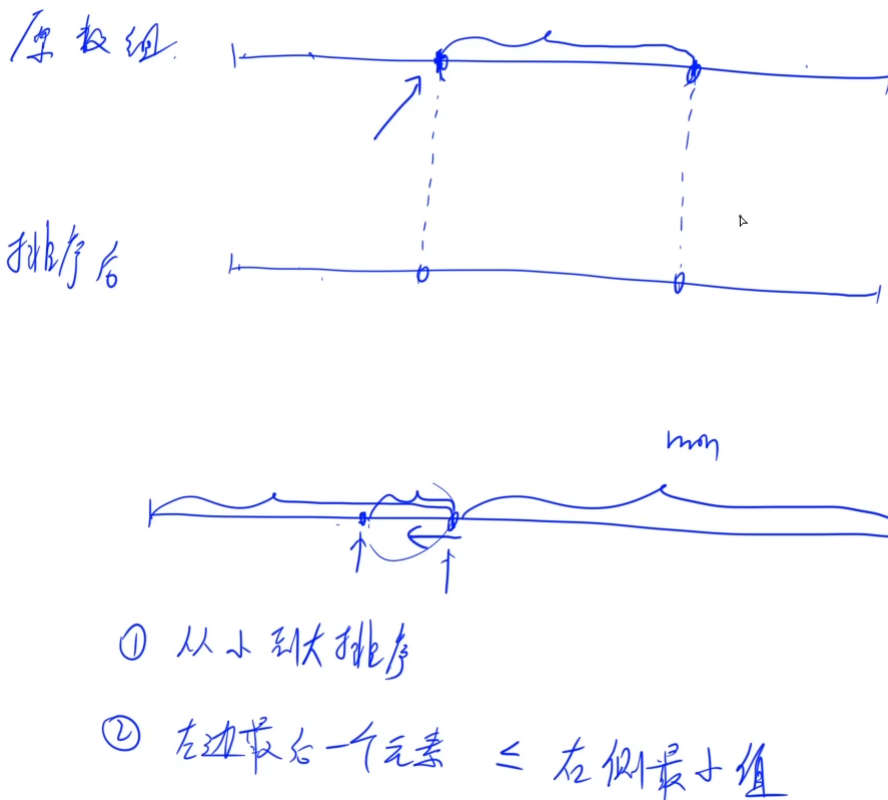
c++代码1

```

1  class Solution {
2  public:
3      int findUnsortedSubarray(vector<int>& nums) {
4          int n = nums.size();
5          vector<int> a(nums);
6          sort(a.begin(), a.end());
7          int i = 0, j = n - 1;
8          while(i < n && nums[i] == a[i]) i++;
9          while(j >= 0 && nums[j] == a[j]) j--;
10         return max(0, j - i + 1);
11     }
12 };

```

思路2



有序 + 无序 + 有序

递增有序的最左边界满足：

- 1、从小到大排序
- 2、左边最后一个元素 \leq 右侧最小值

具体过程：

- 1、遍历数组找到左边保持升序的最后一个点的位置 l ，和从右向左看保持降序的最后一个点的位置 r 。
- 2、从 $l+1$ 的位置向右扫描，如果遇到有比 $nums[l]$ 小的元素，说明最起码 l 不在正确位置上，则 $l--$ 。
- 3、从 $r-1$ 的位置向左扫描，如果遇到有比 $nums[r]$ 大的元素，说明最起码 $nums[r]$ 不在正确的位置上， $r++$ 。
- 4、最后返回 $r - l - 1$ 。

c++代码2

```

1  class Solution {
2  public:
3      int findUnsortedSubarray(vector<int>& nums) {
4          int n = nums.size();
5          int l = 0, r = n - 1;
6          while(l + 1 < n && nums[l + 1] >= nums[l]) l++;
7          if(l == r) return 0;
8          while(r - 1 >= 0 && nums[r - 1] <= nums[r]) r--;
9          for(int i = l + 1; i < n; i++){
10             while(l >= 0 && nums[l] > nums[i]) l--;
11         }
12         for(int i = r - 1; i >= 0; i--){
13             while(r < n && nums[r] < nums[i]) r++;
14         }
15         return r - l - 1;
16     }
17 };

```

538. 把二叉搜索树转换为累加树

思路

本题中要求我们将每个节点的值修改为原来的节点值加上所有大于它的节点值之和。这样我们只需要反序中序遍历该二叉搜索树，记录过程中的节点值之和，并不断更新当前遍历到的节点的节点值，即可得到题目要求的累加树。

c++代码

```

1  class Solution {
2  public:
3      int sum = 0;
4      TreeNode* convertBST(TreeNode* root) {
5          if(root != nullptr){
6              convertBST(root->right);
7              sum += root->val;
8              root->val = sum;
9              convertBST(root->left);
10         }
11         return root;
12     }
13 };

```

238. 除自身以外数组的乘积

思路

申请两个数组，一个用来记录每个位置左边的乘积，和它右边的乘积，再把两个数组乘起来即可。

c++代码

```

1  class Solution {
2  public:
3      vector<int> productExceptSelf(vector<int>& nums) {
4          int n = nums.size();
5          vector<int> p(n, 1);
6          for(int i = 1; i < n; i++) p[i] = p[i - 1] * nums[i - 1]; //前缀乘
7          积
8          for(int i = n - 1, s = 1; i >= 0; i--){
9              p[i] *= s;
10             s *= nums[i]; // 后缀乘积
11         }
12         return p;
13     };

```

437. 路径总和 III *

思路1

(dfs) $O(n^2)$

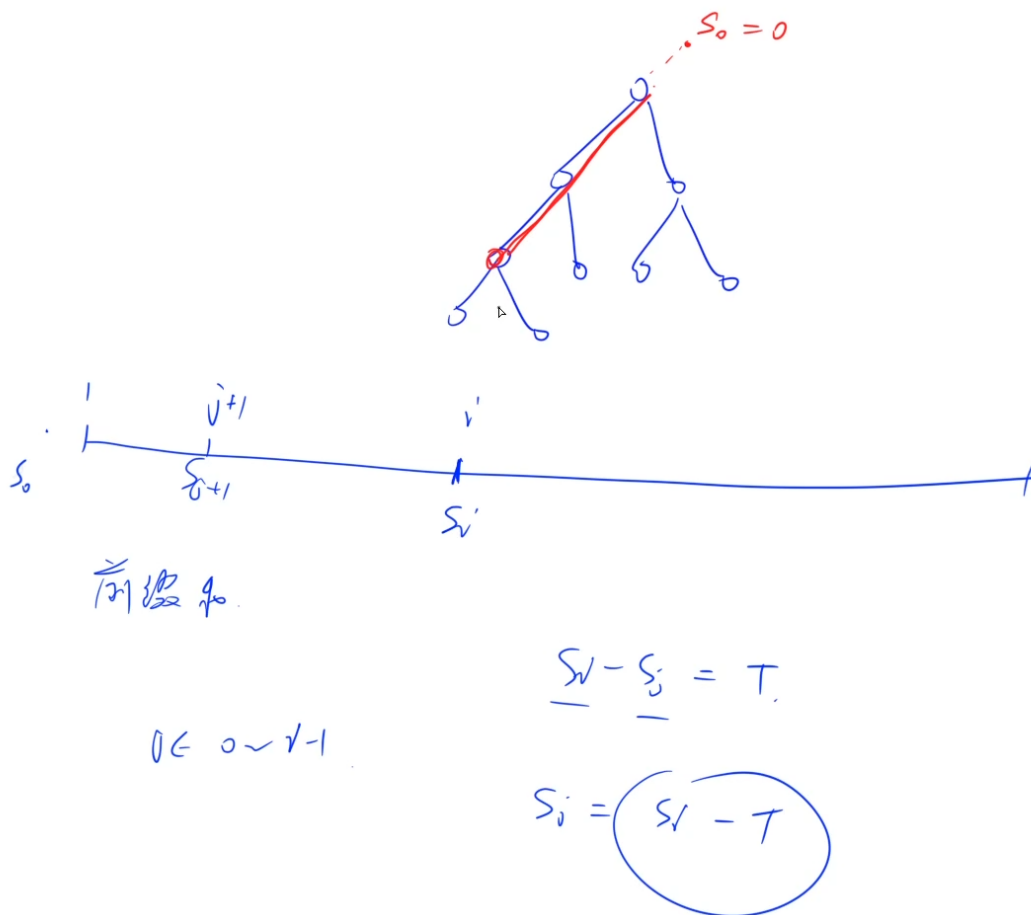
我们遍历每一个节点 `node`，搜索以当前节点 `node` 为起始节点往下延伸的所有路径，并对路径总和为 `targetSum` 的路径进行累加统计。

c++代码

```

1  class Solution {
2  public:
3      int res = 0;
4      int pathSum(TreeNode* root, int targetSum) {
5          if(!root) return 0;
6          dfs(root, targetSum);
7          pathSum(root->left, targetSum);
8          pathSum(root->right, targetSum);
9          return res;
10     }
11     void dfs(TreeNode* root, int sum){
12         if(!root) return ;
13         sum -= root->val;
14         if(sum == 0) res++;
15         if(root->left) dfs(root->left, sum);
16         if(root->right) dfs(root->right, sum);
17     }
18 };

```

(前缀和 + 哈希) $O(n)$

求出二叉树的前缀和，统计以每个节点 **node** 节点为路径结尾的合法路径的数量，记录一个哈希表 **cnt**，维护每个前缀和出现的次数。

对于当前节点 **root**，前缀和为 **cur**，累加 **cnt[cur - sum]** 的值，看看有几个路径起点满足。

c++代码

```

1  class Solution {
2  public:
3      unordered_map<int, int> cnt;
4      int res = 0;
5      int pathSum(TreeNode* root, int targetSum) {
6          cnt[0] = 1;
7          dfs(root, targetSum, 0);
8          return res;
9      }
10     void dfs(TreeNode* root, int sum, int cur){
11         if(!root) return ;
12         cur += root->val;
13         res += cnt[cur - sum];
14         cnt[cur]++;
15         dfs(root->left, sum, cur), dfs(root->right, sum, cur);
16         cnt[cur]--; //回溯
17     }
18 };

```

309. 最佳买卖股票时机含冷冻期

思路

(动态规划) $O(n)$

状态表示: $f[i]$ 表示第 i 天结束后不持有股票的最大收益, $g[i]$ 表示第 i 天结束后持有股票的最大收益。

状态计算:

- $f[i] = \max(f[i - 1], g[i - 1] + \text{prices}[i])$, 表示第 i 天什么都不做, 或者卖掉持有的股票。
- $g[i] = \max(g[i - 1], f[i - 2] - \text{prices}[i])$, 表示第 i 天什么都不做, 或者买当天的股票, 但需要从上两天的结果转移。

初始化:

c++代码

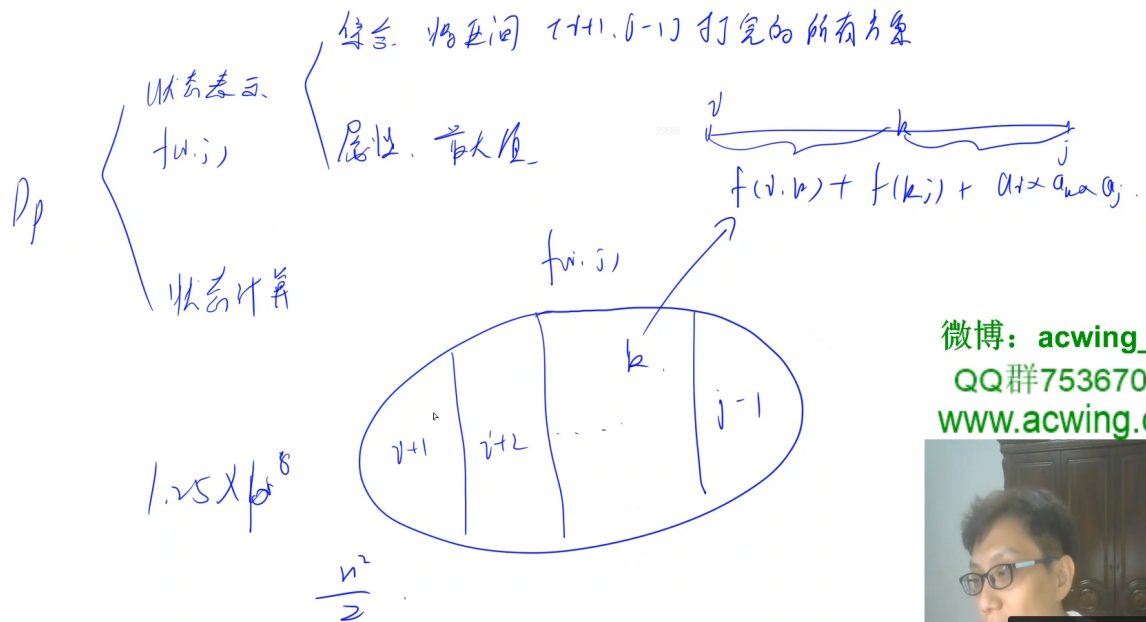
```
1  class Solution {
2  public:
3      int maxProfit(vector<int>& prices) {
4          int n = prices.size();
5          vector<int> f(n + 1), g(n + 1);
6          f[0] = 0, g[0] = -prices[0];
7          for(int i = 1; i < n; i++){
8              f[i] = max(f[i - 1], g[i - 1] + prices[i]);
9              if(i >= 2) g[i] = max(g[i - 1], f[i - 2] - prices[i]);
10             else g[i] = max(g[i - 1], -prices[i]);
11         }
12         return f[n - 1];
13     }
14 };
```

312. 戳气球 *

题目

思路

(区间DP) $O(n^3)$



状态表示: $f[i][j]$ 表示戳破区间 (i, j) 所有气球所能获得硬币的最大数量。

状态计算:

假设最后一次戳破编号为 k 的气球

$$f[i][j] = \max(f[i][j], f[i][k] + f[k][j] + a[i] * a[k] * a[j])$$

c++代码

```

1  class Solution {
2  public:
3      int maxCoins(vector<int>& nums) {
4          int n = nums.size();
5          vector<int> a(n + 2, 1); //全部初始化为1
6          for(int i = 1; i <= n; i++) a[i] = nums[i - 1]; //下标从1开始
7          vector<vector<int>> f(n + 2, vector<int>(n + 2));
8          for(int len = 3; len <= n + 2; len++) //枚举长度
9              for(int i = 0; i + len - 1 <= n + 1; i++){ // [0, n + 1] //左边界
10                 int j = i + len - 1; // (i, j) 右边界
11                 for(int k = i + 1; k < j; k++)
12                     f[i][j] = max(f[i][j], f[i][k] + f[k][j] + a[i] * a[j] *
a[k]);
13             }
14         return f[0][n + 1];
15     }
16 };
```

459. 重复的子字符串

思路1

暴力解法

c++代码1

```

1  class Solution {
2  public:
3      bool repeatedSubstringPattern(string s) {
4          int n = s.size();
```

```

5         for(int len = 1; len <= n / 2; len++){
6             bool flag = true;
7             string str = s.substr(0, len);
8             for(int i = 0; i < n; i += len){
9                 for(int j = 0; j < len; j++){
10                    {
11                        if(s[j] != s[j + i]){
12                            flag = false;
13                            break;
14                        }
15                    }
16                }
17                if(flag) return true;
18            }
19            return false;
20        }
21    };

```

思路2

KMP

$S = \text{"abababc"}$ $i=7$
 $P = \text{"abababab"}$ $j=6$ $p[j+1]$ $S[i]$

$next[1]=0$ $next[6]=4$
 $next[2]=0$ $next[7]=5$
 $next[3]=1$ $next[8]=6$
 $next[4]=2$
 $next[5]=3$

next数组:

next 数组, $next[i]$ 表示子串 $p[1,2,\dots,i-1,i]$ 的最长相等前后缀的前缀最后一位下标, 或者说是子串的最长相等前后缀的长度, 因为我们是从下标1开始的, 这也体现出了从1开始的好处

定理1:

假设 S 的长度为 len , 则 S 存在最小循环节, 循环节的长度 L 为 $len - next[len]$, 子串为 $S[0 \dots len - next[len] - 1]$ 。

- 1、如果 len 可以被 $len - next[len]$ 整除, 则表明字符串 S 可以完全由循环节循环组成, 循环周期 $T = len / L$ 。
- 2、如果不能, 说明还需要再添加几个字母才能补全。需要补的个数是循环个数 $L - len \% L = L - (len - L) \% L = L - next[len] \% L$, $L = len - next[len]$ 。

c++代码2

```

1  class Solution {
2  public:
3      bool repeatedSubstringPattern(string s) {
4          int n = s.size();
5          s = ' ' + s; //下标从1开始
6          vector<int> next(n + 1);
7          for(int i = 2, j = 0; i <= n; i++){
8              while(j && s[i] != s[j + 1]) j = next[j];
9              if(s[i] == s[j + 1]) j++;
10             next[i] = j;
11         }
12         int t = n - next[n];
13         return t < n && n % t == 0;
14     }
15 };

```

26. 删除有序数组中的重复项

思路1

取第一个数或者只要当前数和前一个数不相等，我们就取当前数（重复元素取最后一个）。

c++代码1

```

1  class Solution {
2  public:
3      int removeDuplicates(vector<int>& nums) {
4          int k = 0;
5          for(int i = 0; i < nums.size(); i++){
6              if(!i || nums[i] != nums[i - 1])
7                  nums[k++] = nums[i];
8          }
9          return k;
10     }
11 };

```

思路2

双指针，重复的一段元素我们只取第一个。

c++代码2

```

1  class Solution {
2  public:
3      int removeDuplicates(vector<int>& nums) {
4          int n = nums.size(), k = 0;
5          for(int i = 0, j = 0; i < n; i++){
6              j = i;
7              while(j < n && nums[i] == nums[j]) j++;
8              nums[k++] = nums[i];
9              i = j - 1;
10         }
11         return k;
12     }
13 };

```

268. 丢失的数字 *

思路

(基础数学) $O(n)$

直接对数组中的数字求和，然后用 $n(n+1)/2 - \text{sum}$ 即可得到答案。

时间复杂度分析：只有一次求和，故时间复杂度为 $O(n)$

c++代码

```
1 class Solution {
2 public:
3     int missingNumber(vector<int>& nums) {
4         int n = nums.size(), sum = 0;
5         for(int x : nums) sum += x;
6         return (1 + n) * n / 2 - sum;
7     }
8 };
```

706. 设计哈希映射 *

思路

(拉链法) $O(1)$

(拉链法) $O(1)$

哈希表的基本思想都是先开一个大数组，然后用某种哈希函数将key映射到数组的下标空间。不同算法的区别在于如何处理下标冲突，即当两个不同的key被映射到同一下标时，该怎么办。

一般有两种方式处理冲突：拉链法和开放寻址法。

首先我们来介绍拉链法。它的思想很简单，在哈希表中的每个位置上，用一个链表来存储所有映射到该位置的元素。

- 对于 `put(key, value)` 操作，我们先求出key的哈希值，然后遍历该位置上的链表，如果链表中包含key，则更新其对应的value；如果链表中不包含key，则直接将 (key, value) 插入该链表中。
- 对于 `get(key)` 操作，求出key对应的哈希值后，遍历该位置上的链表，如果key在链表中，则返回其对应的value，否则返回-1。
- 对于 `remove(key)`，求出key的哈希值后，遍历该位置上的链表，如果key在链表中，则将其删除。

时间复杂度分析：最坏情况下，所有key的哈希值都相同，且key互不相同，则所有操作的时间复杂度都是 $O(n)$ 。但最坏情况很难达到，每个操作的期望时间复杂度是 $O(1)$ 。

空间复杂度分析：一般情况下，初始的大数组开到总数据量的两到三倍大小即可，且所有链表的总长度是 $O(n)$ 级别的，所以总空间复杂度是 $O(n)$ 。

c++代码

```
1 const int N = 19997;
2 class MyHashSet {
3 public:
4     vector<int> h[N]; //数组模拟单调表，h[t]存贮key的哈希值，t为存贮地址
5     MyHashSet() {
6
7     }
8
9     int find(vector<int>&h, int key){ // 哈希表中是否包含key的哈希值
```

```

10     for(int i = 0; i < h.size(); i++){
11         if(h[i] == key) return i;
12     }
13     return -1;
14 }
15 void add(int key) {
16     int t = key % N;    //t是坑位
17     int k = find(h[t], key);
18     if(k == -1) h[t].push_back(key);
19 }
20
21 void remove(int key) {
22     int t = key % N;
23     int k = find(h[t], key);
24     if (k != -1) h[t].erase(h[t].begin() + k);
25 }
26
27 bool contains(int key) {
28     int t = key % N;
29     int k = find(h[t], key);
30     return k != -1;
31 }
32 };
33


```

168. Excel表列名称

思路

1	A	—	—	—	—
2	B	26	26 ²	26 ³	26 ⁴
3	C				
⋮					
26	Z	0 0 0	A 0	k	m
27	AA	0 0 1	B 1		
28	AB	0 1 2	⋮	n	k
29	AC	⋯ 25	2 25	m-1	m
		0 1 0			
		0 1 1			

微博: acv
 QQ群73
 www.acv



$$n = 54$$

$$n > 26$$

$$54 - 26 = 28$$

--

27.

1. 0.

B A.

c++代码

1 |