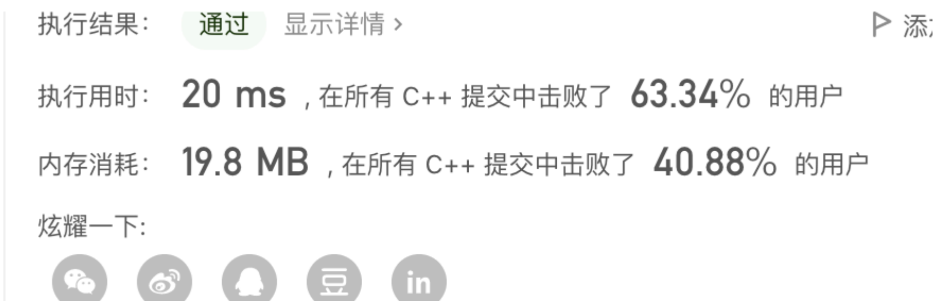
605 Can Place Flowers


执行结果： 通过 显示详情 › 添

执行用时： 20 ms ，在所有 C++ 提交中击败了 63.34% 的用户

内存消耗： 19.8 MB ，在所有 C++ 提交中击败了 40.88% 的用户

炫耀一下：

```cpp
class Solution {
public:
    bool canPlaceFlowers(vector<int>& flowerbed, int n) {
        int index = 0;
        while(index < flowerbed.size() && n != 0) {
            while(index < flowerbed.size() && flowerbed[index] == 1)
                index++;

            if(index == flowerbed.size())
                break;

            if((index == 0 || (flowerbed[index - 1] == 0))
            && (index == flowerbed.size() - 1 || flowerbed[index + 1] == 0)){
                n--;
                index += 2;
            }else
                index++;
        }

        return n == 0;
    }
};
```

611 Valid Triangle Number

```cpp
class Solution {
public:
    int triangleNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end());

        int res = 0;
        int size = nums.size();
        for(int i = 0; i < size - 2; i++){
            int k = i + 2;
            for(int j = i + 1; j < nums.size() - 1 && nums[i] != 0; j++){
                while(k < nums.size() && nums[i] + nums[j] > nums[k])
                    k++;
                res += k - j - 1;
            }
        }

        return res;
    }
};
```

617 Merge Two Binary Tree

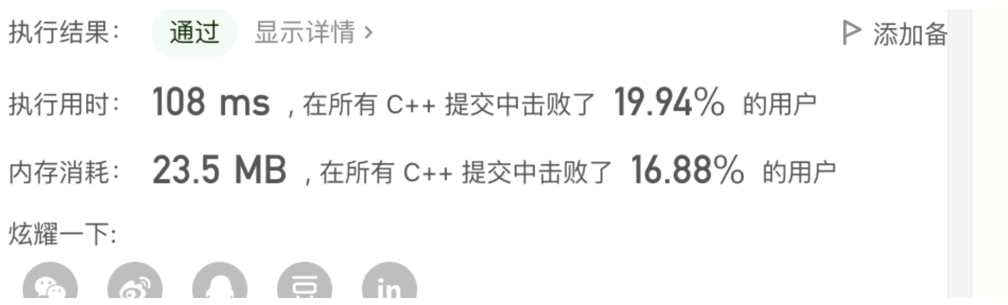
```
    public TreeNode mergeTrees(TreeNode root1, TreeNode root2) {
        if(root1 == null || root2 == null){
            return root1 == null ? root2 : root1;
        }

        root1.val += root2.val;
        root1.left  = mergeTrees(root1.left, root2.left);
        root1.right = mergeTrees(root1.right, root2.right);

        return root1;
    }s
```

621 Task Scheduler

```
//ref: http://zxi.mytechroad.com/blog/greedy/leetcode-621-task-scheduler/
```

```cpp
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        vector<int> count(26, 0);
        for (const char task : tasks)
            ++count[task - 'A'];
        const int max_count = *max_element(count.begin(), count.end());
        size_t ans = (max_count - 1) * (n + 1);
        ans += count_if(count.begin(), count.end(),
                        [max_count](int c){ return c == max_count; });
        return max(tasks.size(), ans);
    }
};
```

638 Shopping Offers


执行结果: 通过 显示详情 ›

添加备

执行用时: 108 ms , 在所有 C++ 提交中击败了 19.94% 的用户

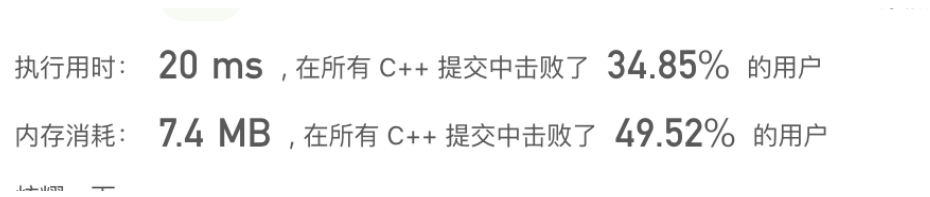
内存消耗: 23.5 MB , 在所有 C++ 提交中击败了 16.88% 的用户

炫耀一下:

```cpp
//暴力回溯
class Solution {
public:
    int minPrice = 0;
    int shoppingOffers(vector<int>& price, vector<vector<int>>& special,
vector<int>& needs) {
        int sum = 0;
        for(int i = 0; i < price.size(); i++){
            sum += price[i] * needs[i];
        }

        minPrice = sum;

        backtrack(price, special, needs, 0);

        return minPrice;
    }

    int backtrack(vector<int>& price, vector<vector<int>>& special, vector<int>& needs, int expense){
        if(allSell(needs)){
            minPrice = min(minPrice, expense);
        }

        int res = minPrice;
        for(int i = 0; i < special.size(); i++){
            vector<int> sp = special[i];
            bool canUse = true;
            for(int j = 0; j < needs.size(); j++) {
                if (sp[j] > needs[j]) {
                    canUse = false;
                    break;
                }
            }

            if(canUse){
                for(int j = 0; j < price.size(); j++){
                    needs[j] -= sp[j];
                }

                res = min(backtrack(price, special, needs, expense + special[i][price.size()]), res);
                minPrice = (res + expense, minPrice);

                for(int j = 0; j < price.size(); j++){
                    needs[j] += sp[j];
                }
            }
        }

        int temp = 0;
        for(int i = 0; i < price.size(); i++){
            temp += price[i] * needs[i];
        }

        minPrice = min(temp + expense, minPrice);

        return res;
    }

    bool allSell(vector<int>& needs){
        for(int i = 0; i < needs.size(); i++){
            if(needs[i] != 0)
```

```cpp
                return false;
        }

        return true;
    }

    int min(int i, int j){
        return i >= j ? j : i;
    }
};
```

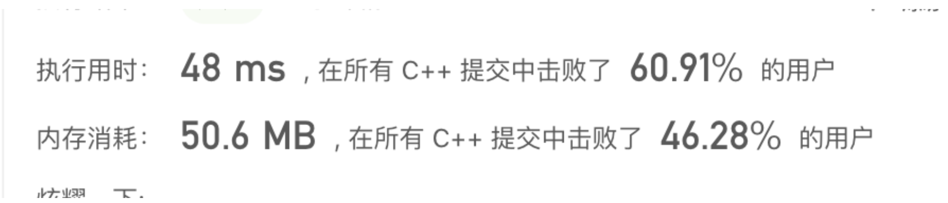
647 Palindromic Substrings


执行用时： 20 ms ，在所有 C++ 提交中击败了 34.85% 的用户

内存消耗： 7.4 MB ，在所有 C++ 提交中击败了 49.52% 的用户

```cpp
// 灵感来源第五题
class Solution {
public:
    int countSubstrings(string s) {
        int size = s.size();
        vector<vector<bool>> dp(size, vector<bool>(size, false));

        for(int i = 0; i < size; i++){
            dp[i][i] = true;
        }

        int count = 0;
        for(int i = size - 1; i >= 0; i--){
            for(int j = i + 1; j < size; j++){
                if(s[i] == s[j]){
                    if(i + 1 == j)
                        dp[i][j] = true;
                    else
                        dp[i][j] = dp[i + 1][j - 1];
                }

                if(dp[i][j])
                    count++;
            }
        }

        count += size;
        return count;
    }
};
```

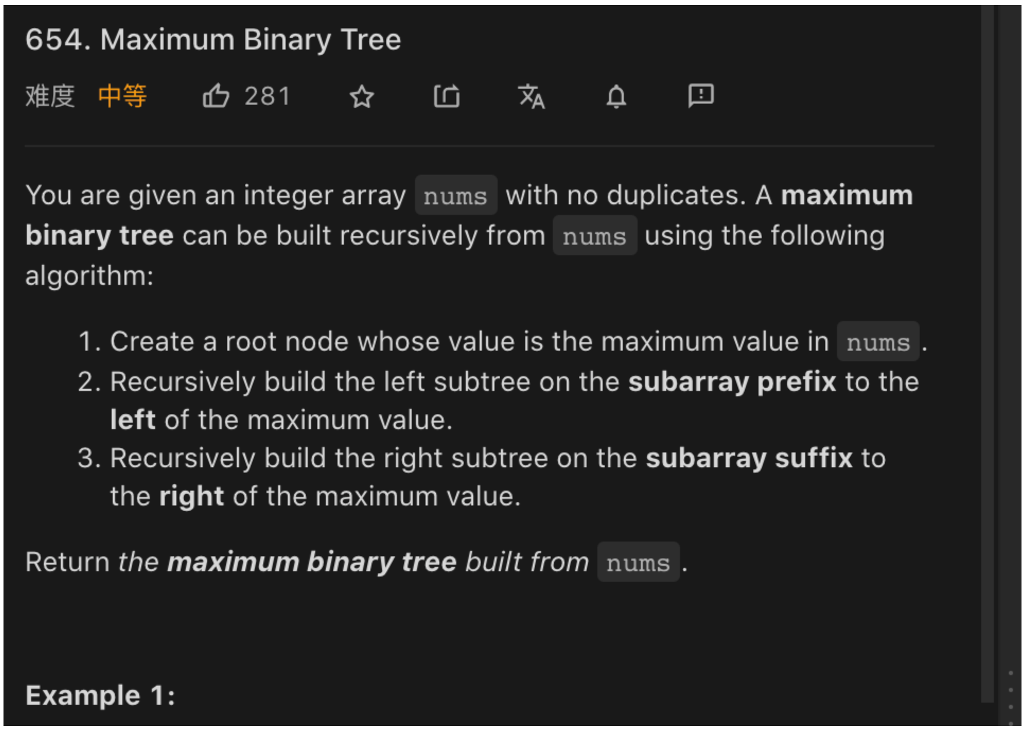
652 Find Duplicate Subtrees


执行用时： 48 ms ，在所有 C++ 提交中击败了 60.91% 的用户

内存消耗： 50.6 MB ，在所有 C++ 提交中击败了 46.28% 的用户

```
class Solution {
public:
    unordered_map<string, vector<TreeNode*>> map;
    vector<TreeNode*> findDuplicateSubtrees(TreeNode* root) {
        postorder(root);

        vector<TreeNode*> res;
        for(auto it = map.begin(); it != map.end(); it++){
            if(it->second.size() >= 2){
                res.push_back(it->second[0]);
            }
        }

        return res;
    }

    string postorder(TreeNode* root){
        if(root == nullptr)
            return "#";

        string left = postorder(root->left);
        string right = postorder(root->right);

        string cur = to_string(root->val) + ">"+ left +"<" + right;
        //cout << to_string(root->val) << " " << cur << endl;
        map[cur].push_back(root);
        return cur;
    }
};
```

654 Maximum Binary Tree



654. Maximum Binary Tree

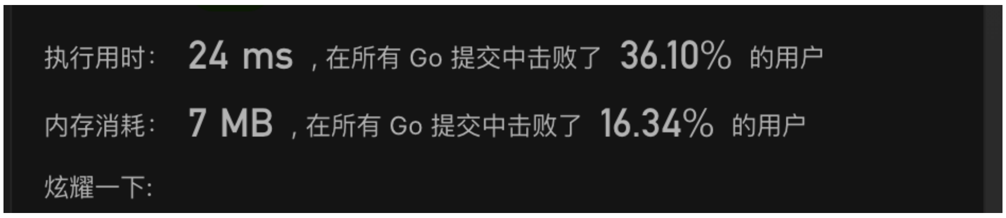
难度 中等 281

You are given an integer array `nums` with no duplicates. A **maximum binary tree** can be built recursively from `nums` using the following algorithm:

1. Create a root node whose value is the maximum value in `nums`.
2. Recursively build the left subtree on the **subarray prefix** to the **left** of the maximum value.
3. Recursively build the right subtree on the **subarray suffix** to the **right** of the maximum value.

Return *the* ***maximum binary tree*** *built from* `nums`.

Example 1:



执行用时： 24 ms ，在所有 Go 提交中击败了 36.10% 的用户

内存消耗： 7 MB ，在所有 Go 提交中击败了 16.34% 的用户

炫耀一下:

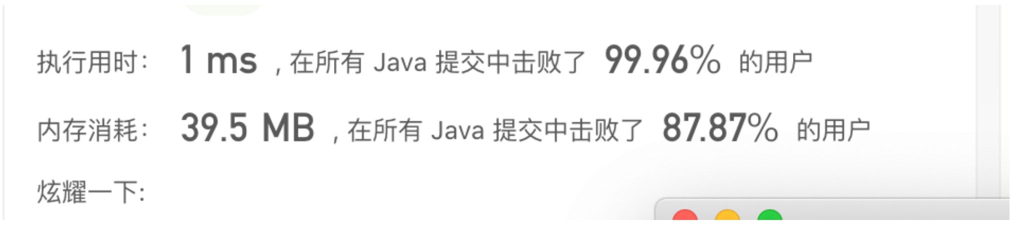
```go
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func constructMaximumBinaryTree(nums []int) *TreeNode {
  return buildTree(nums, 0, len(nums) - 1)

}


func buildTree(nums []int, start int, end int) *TreeNode{
  if start > end{
    return nil
  }else if start == end{
    root := new(TreeNode)
    root.Val = nums[start]
    return root
  }

  index := start
  max := nums[start]

  for i := start; i <= end; i++{
    if max < nums[i]{
      index = i
      max   = nums[i]
    }
  }

  root := new(TreeNode)
  root.Val = max
  root.Left  = buildTree(nums, start, index  - 1)
  root.Right = buildTree(nums, index + 1, end)

  return root
}

```

650 2 Keys Keyboard 不错的DP 题目

```
class Solution {
public:
    int minSteps(int n) {
        if(n == 1)
            return 0;
        vector<int> dp(n + 1, INT_MAX);
        dp[1] = 0;
        dp[2] = 2;

        for(int i = 3; i <= n; i++){
            dp[i] = i;

            for(int j = 2; j * j <= i; j++){
                if(i % j == 0)
```

```
                    dp[i] = dp[j] + dp[i / j];
                }
            }

            return dp[n];
        }
};
```

665 Non Decreasing Array

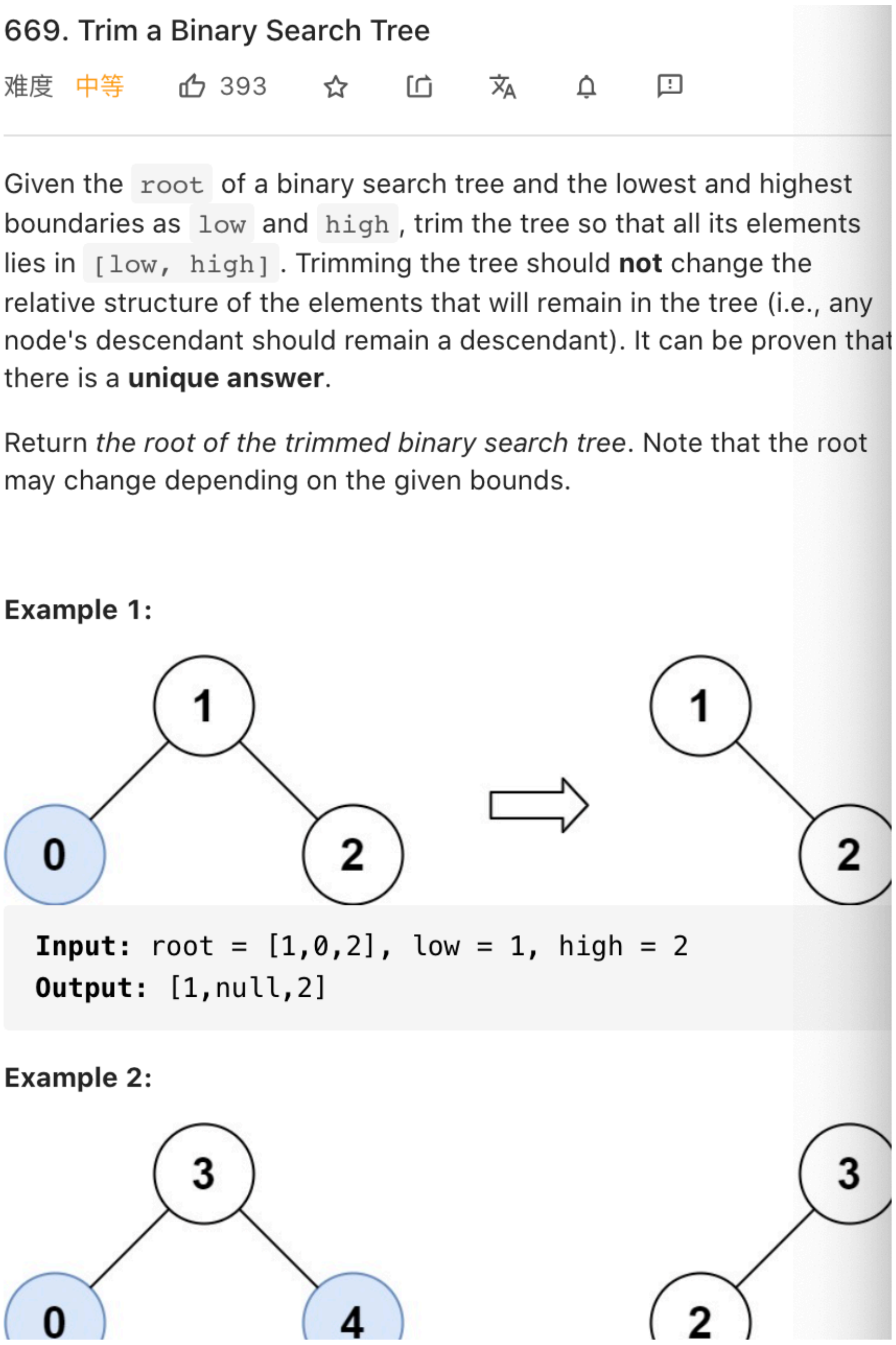

执行用时： 1 ms ，在所有 Java 提交中击败了 99.96% 的用户

内存消耗： 39.5 MB ，在所有 Java 提交中击败了 87.87% 的用户

炫耀一下:

```
    public boolean checkPossibility(int[] nums) {
        int len = nums.length;
        int left = 0, right = len - 1;

        while(left < len - 1 && nums[left] <= nums[left + 1])
            left++;

        if(left == len - 1)
            return true;

        while(right >= 1 && nums[right] >= nums[right - 1])
            right--;

        if(left + 1 != right)
            return false;

        if(left == 0 || right == len - 1)
            return true;

        return nums[left - 1] <= nums[right] || nums[right + 1] >= nums[left];
    }
```

669 Trim a Binary Search Tree



669. Trim a Binary Search Tree

难度 中等 393

Given the `root` of a binary search tree and the lowest and highest boundaries as `low` and `high`, trim the tree so that all its elements lies in `[low, high]`. Trimming the tree should **not** change the relative structure of the elements that will remain in the tree (i.e., any node's descendant should remain a descendant). It can be proven that there is a **unique answer**.

Return *the root of the trimmed binary search tree*. Note that the root may change depending on the given bounds.

Example 1:

```
Input: root = [1,0,2], low = 1, high = 2
Output: [1,null,2]
```

Example 2:

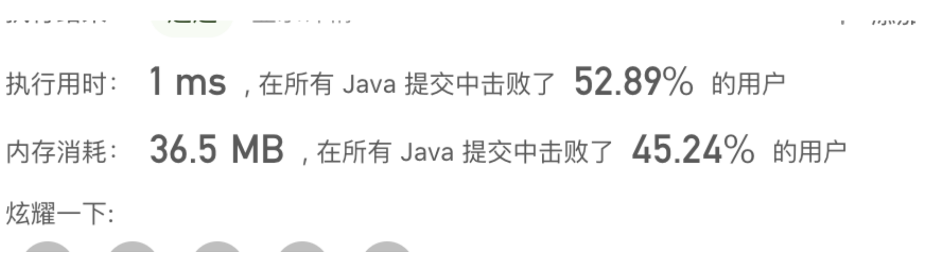
```java
    public TreeNode trimBST(TreeNode root, int low, int high) {
        root = helper(root, low, high);
        return root;
    }

    private TreeNode helper(TreeNode root, int low, int high){
        if(root == null)
```

```
            return null;

        root.left  = helper(root.left , low, high);
        root.right = helper(root.right, low, high);

        if(low <= root.val && root.val <= high){
            return root;
        }else{
            if(root.right == null)
                return root.left;
            else{
                TreeNode cur = root.right;
                while(cur.left != null)
                    cur = cur.left;

                cur.left = root.left;
                root.left = null;

                return root.right;
            }

        }
    }
```

674 Longest Continuous Increasing Subsequence

```
func findLengthOfLCIS(nums []int) int {
  left, right := 0, 1
  res := 1

  for ;right < len(nums);{
    for ;right < len(nums) && nums[right] > nums[right - 1];{
      res = max(res, right - left + 1)
      right++
    }

    if right == len(nums){
      break
    }

    left = right
    right = right + 1
  }


  return res
}

func max(a int, b int) int{
  if a > b{
    return a
  }

  return b
}
```

678 Valid Parenthesis String


执行用时： 1 ms ，在所有 Java 提交中击败了 52.89% 的用户

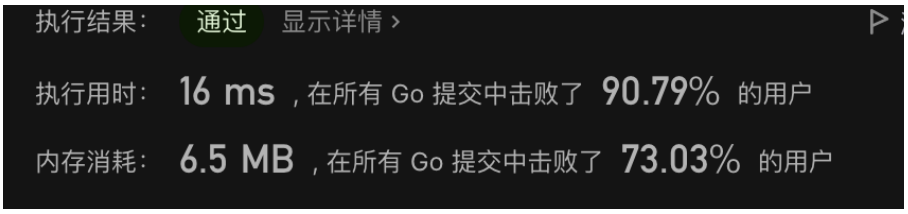
内存消耗： 36.5 MB ，在所有 Java 提交中击败了 45.24% 的用户

炫耀一下:

```

    public boolean checkValidString(String s) {
        List<Character> stack = new ArrayList<>();
        int star = 0;
        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) == '(' || s.charAt(i) == '*'){
                stack.add(s.charAt(i));
                if(s.charAt(i) == '*')
                    star++;
            }else{
                if(stack.isEmpty() || stack.get(stack.size() - 1) == ')')
```

```
                    return false;

                boolean seen = false;
                for(int j = stack.size() - 1; j >= 0; j--){
                    if(stack.get(j) == '(') {
                        stack.remove(j);
                        seen = true;
                        break;
                    }
                }

                if(!seen){
                    stack.remove(stack.size() - 1);
                    star--;
                }
            }
        }

        star = 0;
        for(int i = stack.size() - 1; i >= 0; i--){
            if(stack.get(i) == '*')
                star++;
            else{
                if(star > 0){
                    star--;
                }else{
                    return false;
                }
            }
        }

        return true;
    }
```

680 Valid Palindrome II



执行结果： 通过 显示详情 ›

执行用时： 16 ms ，在所有 Go 提交中击败了 90.79% 的用户

内存消耗： 6.5 MB ，在所有 Go 提交中击败了 73.03% 的用户

```go
/*
  思路就是 递归 + 双指针， 因为可能有两种情况

  同时注意 go 的全局变量可能会导致 OJ 有问题，因此需要调整， 变成指针
*/
func validPalindrome(s string) bool {
  var firstCounter bool = false
  return myValidPalindDrome(s, &firstCounter)

}

func myValidPalindDrome(s string, firstCounter *bool) bool{
  if isPalindrome(s){
    return true
  }

  for left, right := 0, len(s) - 1; left < right;{

    if s[left] == s[right]{
      left++
      right--
    }else{
      if left + 1 == right{
        return true
      }else{
        if *firstCounter{
          return false
        }

        *firstCounter = true

        if s[left + 1] == s[right] || s[left] == s[right - 1]{
          if s[left + 1] == s[right] && s[left] == s[right - 1] {
            return myValidPalindDrome(s[left+2:right], firstCounter) ||
myValidPalindDrome(s[left+1:right-1], firstCounter)
          }else if s[left + 1] == s[right]{
            return myValidPalindDrome(s[left+2:right], firstCounter)
          }else if  s[left] == s[right - 1]{
            return  myValidPalindDrome(s[left+1:right-1], firstCounter)
          }
        }
      }
    }
  }

  return true
}

func isPalindrome(s string) bool{
  for i, j := 0, len(s) - 1; i < j; {
    if s[i] != s[j]{
      println(s[0:i])
      return false
    }

    i++
    j--
  }

  return true
}
```

```go
//超出时间限制
func validPalindrome(s string) bool {
  for i := 0; i < len(s); i++{
    newStr := s[0:i] + s[i + 1:]

    if isPalindrome(newStr){
      return true
    }

  }

  return false
}

func isPalindrome(s string) bool{
  for i, j := 0, len(s) - 1; i < j; {
    if s[i] != s[j]{
      return false
    }

    i++
    j--
  }

  return true
}
```

681 Next Closet Time

执行结果： 通过 显示详情 >

执行用时： **0 ms** ，在所有 C++ 提交中击败了 **100.00%** 的用户

内存消耗： **6.3 MB** ，在所有 C++ 提交中击败了 **15.09%** 的用户

```cpp
class Solution {
public:
    string END_TIME   = "23:59";
    string START_TIME = "0:0";
    unordered_set<string> possibleStr;
    string nextClosestTime(string time) {
        unordered_set<char> set;
        for(char ch : time){
            if(ch == ':')
                continue;
            set.insert(ch);
        }

        generate(time, 0, set);

        string res = "";
        int minDiff = INT_MAX;
        for(const string& str1 : possibleStr){
            if(str1 == time)
                continue;
            int diff = difference(time, str1);
            if(diff < minDiff){
                minDiff = diff;
                res = str1;
            }


           // cout << "time ->" << str1 << " diff -> " << diff << endl;
        }

        return res == "" ? time : res;
    }

    void generate(string& time, int curPos, unordered_set<char>& set){
        if(curPos == time.size())
            return;

        for(char ch : set){
            if(curPos == 2){
                generate(time, curPos + 1, set);
            }else{
                string nextTime(time);
```

```
                    nextTime[curPos] = ch;

                    if(curPos == 4){
                        int hour = getHour(nextTime);
                        int min  = getMin(nextTime);

                        if(hour >= 24 || min >= 60)
                            continue;

                        possibleStr.insert(nextTime);
                    }else{
                        generate(nextTime, curPos + 1, set);
                    }

                }


            }
        }


        //the time difference between s1 and s2, in mins
        int difference(const string& s1, const string& s2){
            int res = 0;

            int hour1 = getHour(s1);
            int hour2 = getHour(s2);

            int min1  = getMin(s1);
            int min2  = getMin(s2);

            if(hour1 > hour2 || (hour1 == hour2 && min1 > min2)){
                int diff1 = difference(s1, END_TIME);
                int diff2 = difference(START_TIME, s2);

                return diff1 + diff2 + 1;
            }else{
                if(min1 <= min2) {
                    int mins = min2 - min1;
                    int hour = hour2 - hour1;

                    return hour * 60 + mins;
                }else{
                    int mins = 60 - min1 + min2;
                    int hour = hour2 - hour1 - 1;

                    return mins + hour * 60;
                }
            }
```

```cpp
    }


    int getHour(const string& s){
        int res = 0;
        int index = 0;
        while(s[index] != ':'){
            res *= 10;
            res += s[index] - '0';
            index++;
        }

        return res;
    }

    int getMin(const string& s){
        int res = 0;
        int index = 0;
        while(s[index] != ':')
            index++;

        index++;
        while(index < s.size()){
            res *= 10;
            res += s[index] - '0';
            index++;
        }

        return res;
    }
};

```

682 Baseball Game

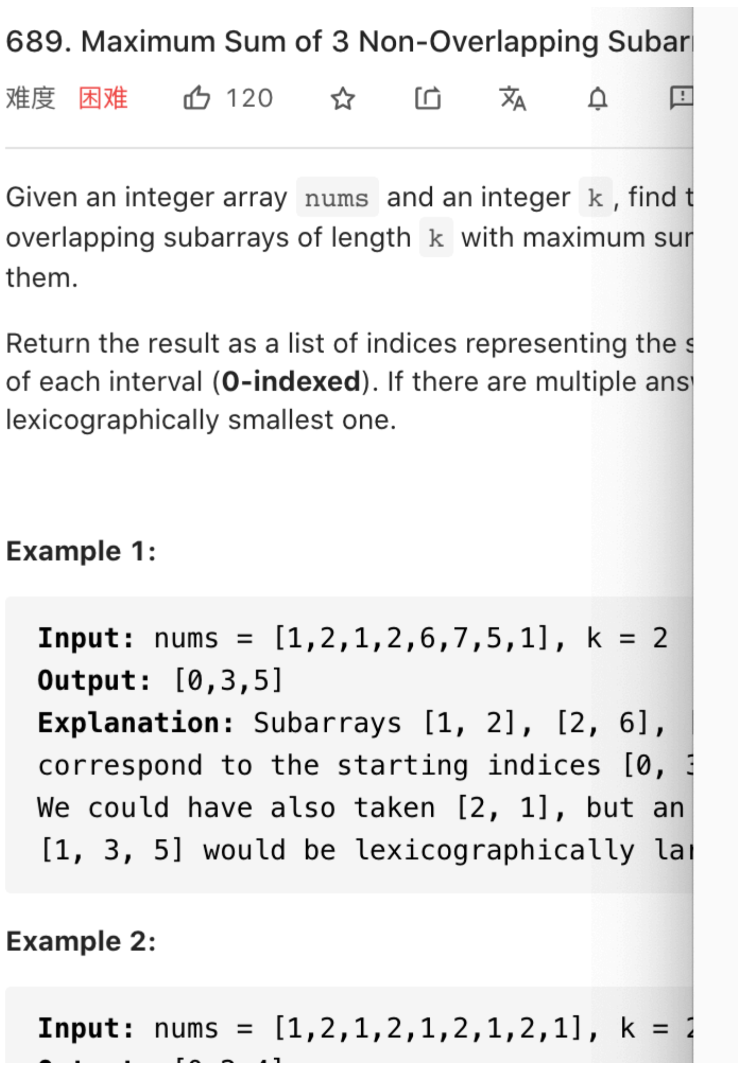
```go
func calPoints(ops []string) int {
  res := make([]int, 0)

  for _, op := range ops{
    if num, err := strconv.Atoi(op); err == nil{
      res = append(res, num)
    }else{
      if op == "C"{
        res = res[:len(res) - 1]
      }else if op == "D"{
        temp := res[len(res) - 1]
        res = append(res, temp * 2)
      }else {
        temp := res[len(res) - 1] + res[len(res) - 2]
        res = append(res, temp)
      }
    }
  }

  sum := 0
  for _, num := range res{
    sum += num
  }

  return sum
}
```

688 Knight Probability In Chessboad

```
实际上 dp 里面存储的是 方法种数

  比如到达 dp[i][j] 有几种方法， 类似 63 unique path
  同时 采用 swap 技术， 降维 dp

  这样每次 dp1 是新的， dp0 是旧的
  最后进行交换
class Solution {
public:
    double knightProbability(int n, int k, int row, int column) {
        vector<vector<double>> dp0(n, vector<double>(n, 0));
        dp0[row][column] = 1.0;

        int dir[8][2] = {{-1, -2}, {-2, -1}, {1, -2}, {2, -1}, {-2, 1}, {-1, 2}, {1, 2}, {2, 1}};

        for(int round = 0; round < k; round++){
            vector<vector<double>> dp1(n, vector<double>(n, 0.0));
            for(int i = 0; i < n; i++){
                for(int j = 0; j < n; j++){
                    for(int m = 0; m < 8; m++){
                        int newX = i + dir[m][0];
                        int newY = j + dir[m][1];
```

```

                          if(newX >= 0 && newY >= 0 && newX < n && newY < n){
                              dp1[newX][newY] += dp0[i][j];
                          }
                      }
                  }
              }

              std::swap(dp1, dp0);
          }

          double total = 0;
          for(int i = 0; i < n; i++){
              for(int j = 0; j < n; j++){
                  total += dp0[i][j];
              }
          }

          return total / pow(8, k);

      }
};
```

689 Maximum Sum of 3 Non-Overlapping subarray



689. Maximum Sum of 3 Non-Overlapping Subar

难度 困难 120

Given an integer array `nums` and an integer `k`, find t
overlapping subarrays of length `k` with maximum sur
them.

Return the result as a list of indices representing the s
of each interval (**0-indexed**). If there are multiple ans
lexicographically smallest one.

Example 1:

```
Input: nums = [1,2,1,2,6,7,5,1], k = 2
Output: [0,3,5]
Explanation: Subarrays [1, 2], [2, 6],
correspond to the starting indices [0, 3
We could have also taken [2, 1], but an
[1, 3, 5] would be lexicographically lar
```

Example 2:

```
Input: nums = [1,2,1,2,1,2,1,2,1], k = 2
```

```
class Solution {
public:
    vector<int> maxSumOfThreeSubarrays(vector<int>& nums, int k) {
        int size = nums.size();
        int len = size - k + 1;

        vector<int> subArrarySum(len, 0);


        int sum = 0;
        for(int i = 0; i < size; i++){
            sum += nums[i];

            if(i - k >= 0){
                sum -= nums[i - k];
            }

            if(i >= k - 1)
                subArrarySum[i - k + 1] = sum;
        }

        int maxIndex = 0;
        vector<int> leftSum(len, 0);
        vector<int> rightSum(len, 0);

        for(int i = 0; i < len; i++){
            if(subArrarySum[maxIndex] < subArrarySum[i]){
                maxIndex = i;
            }

            leftSum[i] = maxIndex;
        }

        maxIndex = len - 1;
        for(int i = len - 1; i >= 0; i--){
            if(subArrarySum[i] >= subArrarySum[maxIndex]){
                maxIndex = i;
            }

            rightSum[i] = maxIndex;
        }

        vector<int> res(3, -1);
        for(int i = k; i < len - k; i++){
            if(res[0] == -1 ||
            subArrarySum[res[0]] + subArrarySum[res[1]] + subArrarySum[res[2]] < subArrarySum[i] + subArrarySum[leftSum[i - k]] + subArrarySum[rightSum[i + k]]){
                res[0] = leftSum[i - k];
                res[1] = i;
                res[2] = rightSum[i + k];
            }

        }
        return res;
    };
};
```

694 Number of Distinct Islands

694. Number of Distinct Islands

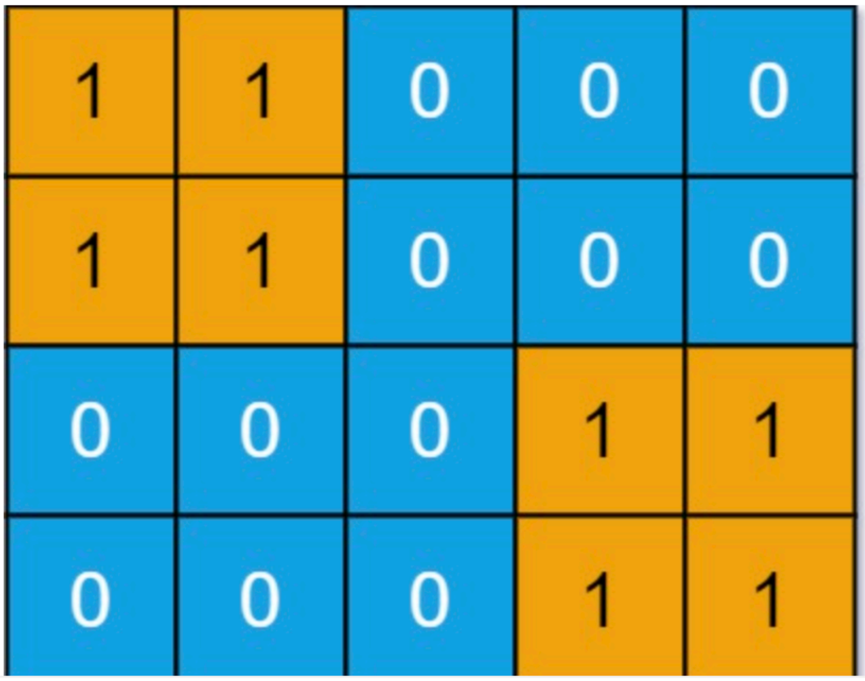
难度 中等 76

You are given an `m x n` binary matrix `grid`. An island is a group of `1`'s (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

Return *the number of **distinct** islands*.

Example 1:



1	1	0	0	0
1	1	0	0	0
0	0	0	1	1
0	0	0	1	1