golden sachs

Sunday, February 21, 2021 11:20 AM

- 83. Remove Duplicates from Sorted List

```
public ListNode deleteDuplicates(ListNode head) {
    if (head == null) {
        return head;
    }

    ListNode cur = head;

    while (cur != null) {
        ListNode dup = cur.next;

        while (dup != null && dup.val == cur.val) {
            dup = dup.next;
        }
        cur.next = dup;
        cur = cur.next;
    }
    return head;
}
```

- 82. Remove Duplicates from Sorted List II

```
//法一
public ListNode deleteDuplicates(ListNode head) {
    if (head == null) {
        return null;
    }
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode cur = dummy;

    while (cur.next != null) {//用root.next是因为，因为所有dup不保存，所以root不能在dup element上，不然就回不去了，
        //决定root.next是不是dup，是就换掉root。next
        ListNode dup = cur.next;
        if (dup.next != null && dup.val == dup.next.val) {//当从一个dup list出来，可能会进入另一个dup，所以root还不能到next，只有root。next不是有dup的元素才可以跳
            while (dup.next != null && dup.val == dup.next.val) {
                dup = dup.next;
            }
            cur.next = dup.next;
        } else {
            cur = cur.next;
        }
    }
    return dummy.next;
}
```

//法二

```
public ListNode deleteDuplicates(ListNode head) {
    if (head == null) {
        return null;
    }
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode cur = head;
    ListNode pre = dummy;

    while (cur != null) {
        ListNode dup = cur.next;
        if (dup != null && dup.val == cur.val) {
            while (dup != null && dup.val == cur.val) {
                dup = dup.next;
            }
            pre.next = dup;
            cur = dup;
        } else {
            pre = cur;
            cur = cur.next;
        }
    }
    return dummy.next;
}
```

- 724. Find Pivot Index

```
public int pivotIndex(int[] nums) {
    int sum = getSum(nums);
    int total = 0;

    for (int i = 0; i < nums.length; i++) {
        if (total * 2 + nums[i] == sum) {
            return i;
        }
        total += nums[i];
    }
    return -1;
}

private int getSum(int[] nums) {
    int sum = 0;
    for (int v : nums) {
        sum += v;
    }
    return sum;
}
```

- 1358. Number of Substrings Containing All Three Characters

//法一

```
public int numberOfSubstrings(String s) {
    char[] charArr = s.toCharArray();
    int len = s.length();

    int count = 0;
    int[] last = {-1, -1, -1};

    for(int j = 0; j < len; j++) {
        last[charArr[j] - 'a'] = j;
        count += 1 + Math.min(last[0], Math.min(last[1], last[2]));
    }

    return count;
}
```

//法二

```
public int numberOfSubstrings(String s) {
    char[] arr = s.toCharArray();
    int[] letter = new int[3];

    int fast = 0;
    int slow = 0;

    int count = 0;

    while (fast < arr.length) {
        letter[arr[fast] - 'a']++;

        while (slow < fast && letter[0] > 0 && letter[1] > 0 && letter[2] > 0) {
            count += arr.length - fast;
            letter[arr[slow++] - 'a']--;
        }
        fast++;
    }
    return count;
}
```

- 740. Delete and Earn

```
public int deleteAndEarn(int[] nums) {
    int[] count = new int[10001];
    for (int v : nums) {
        count[v]++;
    }

    int take = 0;
    int dont = 0;
    int pre = -1;
```

```
    for (int i = 0; i < 10001; i++){
        if (count[i] > 0) {
            int curMax = Math.max(take, dont);
            if (pre == i - 1){
                take = i * count[i] + dont;
            } else {
                take = i * count[i] + curMax;
            }
            dont = curMax;
            pre = i;
        }
    }
    return Math.max(take, dont);
}
```

- 1163. Last Substring in Lexicographical Order

```
public String lastSubstring(String s) {
  char[] arr = s.toCharArray();
  char max = arr[0];
  int index = 0;
  for (int i = 0; i < arr.length; i++) {
      if (arr[i] > max) {
          max = arr[i];
          index = i;
      }
  }

  for (int i = index + 1; i < arr.length; i++) {
        if (arr[i] == max){
            int tmp = index;
            int cur = i;

            while (cur < arr.length && arr[cur] == arr[tmp]) {
                tmp++;
                cur++;
            }

            if (cur < arr.length && arr[cur] > arr[tmp]) {
                index = i;
            }
            if (cur == arr.length) {
                return s.substring(index);
            }
        }
  }
  return s.substring(index);
}
```

- 780. Reaching Points

```
public boolean reachingPoints(int sx, int sy, int tx, int ty) {
    while (tx >= sx && ty >= sy) {
        if (tx == ty) {
            break;
        }
        if (tx > ty) {
            if (ty > sy) {
                tx %= ty;
            } else {
                return (tx - sx) % ty == 0;
            }
        } else {
            if (tx > sx) {
                ty %= tx;
            } else {
                return (ty - sy) % tx == 0;
            }
        }
    }
    return sx == tx && sy == ty;
}
```

- 322. Coin Change

```
public int coinChange(int[] coins, int amount) {
    if (amount == 0) {
        return 0;
    }

    int[] minimal = new int[amount + 1];
    Arrays.fill(minimal, amount + 1);
    minimal[0] = 0;

    for (int i = 1; i <= amount; i++) {
        for (int k = 0; k < coins.length; k++) {
            if (coins[k] <= i) {
                minimal[i] = Math.min(minimal[i], minimal[i - coins[k]] + 1);
            }
        }
    }
    return minimal[amount] == amount + 1 ? -1 : minimal[amount];
}
```

- 532. K-diff Pairs in an Array

```
public int findPairs(int[] nums, int k) {
    int pairs = 0;
    Map<Integer, Integer> map = getMap(nums);

    for (Map.Entry<Integer, Integer> e : map.entrySet()) {
        int key = e.getKey();
```

```
            int value = e.getValue();

            if (k > 0 && map.containsKey(key + k)) {
                pairs++;
            }
            if (k == 0 && value > 1) {
                pairs++;
            }
        }
        return pairs;
    }

    private Map<Integer, Integer> getMap(int[] nums) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int v : nums) {
            map.put(v, map.getOrDefault(v, 0) + 1);
        }
        return map;
    }



    public int findPairs(int[] nums, int k) {

        int pairs = 0;
        Map<Integer, Integer> map = getMap(nums, k);

        for (Map.Entry<Integer, Integer> e : map.entrySet()) {
            int key = e.getKey();
            int value = e.getValue();

            if (k > 0 && map.containsKey(key - k)) {
                pairs++;
            }
            if (k == 0 && value > 1) {
                pairs++;
            }
        }
        return pairs;
    }

    private Map<Integer, Integer> getMap(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int v : nums) {
            map.put(v + k, map.getOrDefault(v + k, 0) + 1);
        }
        return map;
    }
```

- 547. Number of Provinces

```
    public int findCircleNum(int[][] isConnected) {
```

```
        boolean[] visited = new boolean[isConnected.length];
        int count = 0;
        for(int i = 0; i < isConnected.length; i++) {
            if (!visited[i]) {
                count++;
                dfs(isConnected, i, visited);
            }
        }
        return count;
    }

    private void dfs(int[][] isConnected, int index, boolean[] visited) {
        for (int j = 0; j < isConnected.length; j++) {
            if (isConnected[index][j] == 1 && !visited[j]) {
                visited[j] = true;
                dfs(isConnected, j, visited);
            }
        }
    }
```

- 416. Partition Equal Subset Sum

```
    public boolean canPartition(int[] nums) {
        int sum = getSum(nums);
        if (sum % 2 != 0) {
            return false;
        }

        boolean[] partition = new boolean[sum / 2 + 1];
        partition[0] = true;

        for (int element : nums) {
            for (int k = sum / 2; k >= element; k--) {
                partition[k] |= partition[k - element];
            }
        }
        return partition[sum / 2];

    }

    private int getSum(int[] a) {
        int sum = 0;
        for (int v : a) {
            sum += v;
        }
        return sum;
    }
    }
```

- 131. Palindrome Partitioning

```
public List<List<String>> partition(String s) {
    if (s == null || s.length() == 0) {
        return new ArrayList<>();
    }

    List<List<String>> sol = new ArrayList<>();
    List<String> ans = new ArrayList<>();
    dfs(sol, ans, 0, s);
    return sol;
}

private void dfs(List<List<String>> sol, List<String> ans, int index, String s) {
    if (index == s.length()) {
        sol.add(new ArrayList<>(ans));
        return;
    }

    for (int i = index + 1; i <= s.length(); i++) {
        String sub = s.substring(index, i);
        if (palindrome(sub)) {
            ans.add(sub);
            dfs(sol, ans, i, s);
            ans.remove(ans.size() - 1);
        }
    }
}

private boolean palindrome(String s) {
    int left = 0;
    int right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

- 1219. Path with Maximum Gold

```
public int getMaximumGold(int[][] grid) {
    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return 0;
    }
    int[][] dir = new int[][]{{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
    int[] max = new int[1];
    for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[0].length; j++) {
```

```
            if (grid[i][j] > 0) {
                dfs(grid, dir, max, 0, i, j);
            }
        }
    }
    return max[0];
}

private void dfs(int[][] grid, int[][] dir, int[] max, int ans, int y, int x) {
    ans += grid[y][x];
    max[0] = Math.max(max[0], ans);

    int tmp = grid[y][x];
    grid[y][x] = 0;

    for (int i = 0; i < dir.length; i++) {
        int newY = y + dir[i][0];
        int newX = x + dir[i][1];
        if (valid(newY, newX, grid)) {
            dfs(grid, dir, max, ans, newY, newX);
        }
    }

    grid[y][x] = tmp;
}

private boolean valid (int y, int x, int[][] grid) {
    return y >= 0 && y < grid.length && x >= 0 && x < grid[0].length && grid[y][x] != 0;
}
```

- 647. Palindromic Substrings

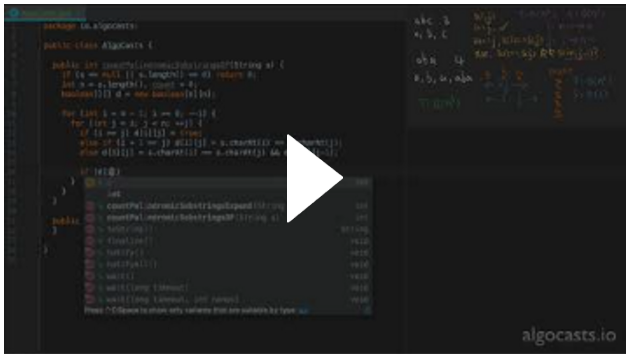
```
public int countSubstrings(String s) {
    if (s == null || s.length() == 0) {
        return 0;
    }
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        count += expand(s, i, i);//两种开花方法，从一个字母开始开
        count += expand(s, i, i + 1);//或者从字母中间
    }
    return count;

}

private int expand(String s, int left, int right) {
    int count = 0;
    while (left >= 0 && right < s.length()) {
        if (s.charAt(left) != s.charAt(right)) {
            break;
        }
        count++;
        left--;
```

```
            right++;
        }
        return count;
    }
```

[LeetCode 647. Palindromic Substrings - AlgoCasts 讲解]



- 775. Global and Local Inversions

```
public boolean isIdealPermutation(int[] A) {
    for (int i = 0; i < A.length; i++) {
        if (Math.abs(A[i] - i) > 1) {
            return false;
        }
    }
    return true;
}
```

```
// 这个方法比较general, 不受array的内容限制
public boolean isIdealPermutation(int[] A) {
    int local = 0;
    for (int i = 1; i < A.length; i++) {
        if (A[i - 1] > A[i]) {
            local++;
        }
    }

    int[] global = new int[1];
    int[] helper = new int[A.length];
    mergeSort(A, helper, 0, A.length - 1, global);
    System.out.println(local + " " + global[0]);
    return global[0] == local;
}

private void mergeSort(int[] arr, int[] h, int left, int right, int[] count) {
    if (left >= right) {
        return;
    }
    int mid = left + (right - left) / 2;
    mergeSort(arr, h, left, mid, count);
```

```
        mergeSort(arr, h, mid + 1, right, count);
        merge(arr, h, left, mid, right, count);
    }

    private void merge(int[] a, int[] h, int l, int m, int r, int[] count) {
        for (int i = l; i <= r; i++) {
            h[i] = a[i];
        }

        int s1 = l;
        int s2 = m + 1;
        int index = l;

        while (s1 <= m) {
            if (s2 > r || h[s1] <= h[s2]) {// 如果是等于，肯定不是inversion，如果h[s1] < h[s2]肯定也不是inversion，inversion必须是：h[s1] 》 h[s2]，长点脑子吧！！跟等于没啥关系！！
                //这里的理解是这样的：在s1和（s2 -1)比较，肯定是 h[s1] > h[s2 - 1](如果是相等我们选是s1)，所以m + 1 到s2-1都比s1小，count加的就是这段
                count[0] += s2 - (m + 1);//1
                a[index++] = h[s1++];
            } else {
                // count[0] += m - s1 + 1;//2   1,2 选一个都行 。一个特别需要注意的小细节！！！！！；我们必须以小的那个点，为中心的，算另一个和*它！！有几个inversion
                //而不是！！不是！！以大的那边算小的那边有几个和它对应的inversion，因为小的比完这一轮就移动了！！就走了！！，我们要以移动的那个点为基准!
                a[index++] = h[s2++];
            }
        }
    }
```

- 54. Spiral Matrix

```
    public List<Integer> spiralOrder(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        List<Integer> so = new ArrayList<>(0);

        int left = 0;
        int right = n - 1;
        int top = 0;
        int bot = m - 1;

        while (left < right && top < bot) {
            for (int i = left; i < right; i++) {
                so.add(matrix[top][i]);
            }

            for (int i = top; i < bot; i++) {
                so.add(matrix[i][right]);
            }

            for (int i = right; i > left; i--) {
                so.add(matrix[bot][i]);
```

```
            }

            for (int i = bot; i > top; i--) {
                so.add(matrix[i][left]);
            }

            left++;
            right--;
            top++;
            bot--;
        }

        if (left == right) {
            for (int i = top; i <= bot; i++) {
                so.add(matrix[i][right]);
            }
        } else if (top == bot) {
            for (int i = left; i <= right; i++) {
                so.add(matrix[top][i]);
            }
        }
        return so;
    }
```

- 70. Climbing Stairs

```
    public int climbStairs(int n) {
        int[] step = new int[n + 1];
        step[0] = 1;//几种方式走0部?  1种 站着不动
        step[1] = 1;//几种方式走1步

        for (int i = 2; i <= n; i++) {
            step[i] = step[i - 1] + step[i - 2];
        }

        return step[n];

    }
```

- 78. Subsets

```
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> sol = new ArrayList<>();
        List<Integer> ans = new ArrayList<>();
        dfs(nums, sol, ans, 0);
        return sol;

    }

    private void dfs(int[] nums, List<List<Integer>> sol, List<Integer> ans, int index) {
        if (index == nums.length) {
            sol.add(new ArrayList<>(ans));
            return;
```

```
        }

        dfs(nums, sol, ans, index + 1);
        ans.add(nums[index]);
        dfs(nums, sol, ans, index + 1);
        ans.remove(ans.size() - 1);
    }
```

- 76. Minimum Window Substring

```
    public String minWindow(String s, String t) {
        Map<Character, Integer> count = countLetter(t);

        int min = Integer.MAX_VALUE;
        int start = 0;

        int match = 0;
        int slow = 0;
        int fast = 0;

        while (fast < s.length()) {
            Integer last = count.get(s.charAt(fast));
            if (last != null) {
                if (last == 1) {
                    match++;
                }
                count.put(s.charAt(fast), --last);

            }
            fast++;

            while (match == count.size() && slow <= fast) {

                if (fast - slow < min) {
                    min = fast - slow;
                    start = slow;
                }

                Integer first = count.get(s.charAt(slow));
                if (first != null) {
                    if (first == 0) {
                        match--;
                    }
                    count.put(s.charAt(slow), ++first);
                }
                slow++;
            }
        }
        return min == Integer.MAX_VALUE ? "" : s.substring(start, start + min);

    }

    private Map<Character, Integer> countLetter(String s) {
        Map<Character, Integer> map = new HashMap<>();
```

```
        for (Character l : s.toCharArray()) {
            map.put(l, map.getOrDefault(l, 0) + 1);
        }

        return map;
    }
```

- 289. Game of Life

```
    public void gameOfLife(int[][] board) {
        for (int i = 0; i < board.length; i++) {
            for (int k = 0; k < board[0].length; k++) {
                int lives  = count(board, i, k);

                if (lives == 3 || lives - board[i][k] == 3) {
                    board[i][k] |= 0b10;
                }
            }
        }

        for (int i = 0; i < board.length; i++) {
            for (int k = 0; k < board[0].length; k++) {
                board[i][k] >>= 1;
            }
        }
    }

    private int count(int[][] b, int y, int x) {
        int count = 0;
        for (int i = -1; i <= 1; i++) {
            for (int k = -1; k <= 1; k++) {
                if(valid(b, y + i, x + k)) {
                    count++;
                }
            }
        }
        return count;
    }

    private boolean valid(int[][] b, int y, int x) {
        return y >= 0 && y < b.length && x >= 0 && x < b[0].length && (b[y][x] & 1) == 1;//记得这里要and，因为可以已经update过了变成11（bit），但是我们只看没update的值就是最后一个bit
    }
```

- 387. First Unique Character in a String

```
    public int firstUniqChar(String s) {
        Map<Character, Integer> map = new HashMap<>();

        for (Character c : s.toCharArray()) {
            map.put(c, map.getOrDefault(c, 0) + 1);
        }

        for (int i = 0; i < s.length(); i++) {
            if (map.get(s.charAt(i)) == 1) {
```

```
                return i;
            }
        }
        return -1;
    }
```

- 1041. Robot Bounded In Circle

```
    public boolean isRobotBounded(String instructions) {
        int[][] dir = new int[][]{{1, 0},  {0, 1}, {-1, 0}, {0, -1},};
        int y = 0;
        int x = 0;
        int index = 0;
        int[] direction = dir[index];

        for (char c : instructions.toCharArray()) {
            if (c == 'G') {
                y += direction[0];
                x += direction[1];
            }

            if (c == 'L') {
                index--;
                index = mod(index, 4);
                direction = dir[index];
            }

            if (c == 'R') {
                index++;
                index = mod(index, 4);
                direction = dir[index];
            }
        }

        return (x == 0 && y == 0) || index != 0;

    }

    private int mod(int f, int s) {
        while (f < 0) {
            f += s;
        }
        return Math.abs(f) % s;
    }
```

- 443. String Compression

```
    public int compress(char[] chars) {
        int slow = 0;//slow左边的
        int fast = 0;

        while (fast < chars.length) {
            int count = 0;
            while (fast + count < chars.length && chars[fast] == chars[fast + count]) {
                count++;
            }
```

```
            chars[slow++] = chars[fast];
            fast += count;

            if (count == 1) {
                continue;
            }

            slow = appendDigits(chars, slow, count);
        }
        return slow;
    }

    private int appendDigits(char[] chars, int slow, int count) {
        int len = 0;
        for (int i = count; i > 0; i /= 10) {
            len++;
            slow++;
        }

        while (count != 0) {
            chars[--slow] = (char)(count % 10 + '0');
            count /= 10;
        }

        return slow + len;
    }
```

- 870. Advantage Shuffle

```
    public int[] advantageCount(int[] A, int[] B) {
        Map<Integer, List<Integer>> map = new HashMap<>();
        List<Integer> remaining = new ArrayList<>();//凑数的

        int[] sortA = Arrays.copyOf(A, A.length);
        int[] sortB = Arrays.copyOf(B, B.length);

        Arrays.sort(sortA);
        Arrays.sort(sortB);


        for (int v : B) {
            map.put(v, new ArrayList<>());
        }

        int aIndex = 0;
        int bIndex = 0;

        while (aIndex < A.length) {
            if (sortA[aIndex] > sortB[bIndex]) {
                map.get(sortB[bIndex++]).add(sortA[aIndex++]);
            } else {
```

```
                remaining.add(sortA[aIndex++]);
            }
        }

        for (int i = 0; i < B.length; i++) {
            List<Integer> larger = map.get(B[i]);

            if (larger.size() > 0) {
                A[i] = larger.remove(larger.size() - 1);;
            } else {
                A[i] = remaining.remove(remaining.size() - 1);
            }
        }

        return A;
    }
```

- 624. Maximum Distance in Arrays

```
    public int maxDistance(List<List<Integer>> arrays) {
        int min = arrays.get(0).get(0);
        int max = arrays.get(0).get(arrays.get(0).size() - 1);
        arrays.remove(0);
        int dist = 0;

        for (List<Integer> list : arrays) {
            int curMin = list.get(0);
            int curMax = list.get(list.size() - 1);
            dist = Math.max(dist, Math.max(Math.abs(max - curMin), Math.abs(curMax - min)));
            max = Math.max(max, curMax);
            min = Math.min(min, curMin);
        }
        return dist;
    }
```

- 209. Minimum Size Subarray Sum

```
    public int minSubArrayLen(int target, int[] nums) {
        int slow = 0;
        int fast = 0;
        int len = Integer.MAX_VALUE;
        int sum = 0;

        while (fast < nums.length) {
            sum += nums[fast++];

            while (slow < fast && sum >= target) {
                len = Math.min(len, fast - slow);

                sum -= nums[slow++];
            }
        }
        return len == Integer.MAX_VALUE ? 0 : len;
    }
```

- 1010. Pairs of Songs With Total Durations Divisible by 60

```java
public int numPairsDivisibleBy60(int[] time) {
    int count = 0;
    int[] rem = new int[60];
    for (int t : time) {
        if (t % 60 == 0) {
            count += rem[0];
        } else {
            count += rem[60 - t % 60];
        }
        rem[t % 60]++;
    }
    return count;
}
```

- 52. N-Queens II

```java
public int totalNQueens(int n) {
    List<Integer> sol = new ArrayList<>();
    int[] count = new int[1];
    dfs(sol, n, count);
    return count[0];
}

private void dfs(List<Integer> sol, int n, int[] count) {
    if (sol.size() == n) {
        count[0]++;
        return;
    }

    for (int i = 0; i < n; i++) {
        if (valid(sol, i, sol.size())) {
            sol.add(i);
            dfs(sol, n, count);
            sol.remove(sol.size() - 1);
        }
    }
}

private boolean valid(List<Integer> list, int col, int row) {
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) == col || row - i == Math.abs(col - list.get(i))) {
            return false;
        }
    }
    return true;
}
```

- Count Inversions of size three in a given array

```
class Fenwick {
    int[] sum;
    public Fenwick(int n) {
        sum = new int[n];
    }

    public int query(int index) {
        int total = 0;
        while (index > 0) {
            total += sum[index];
            index -= lowBit(index);
        }
        return total;
    }

    public void update(int index, int diff) {
        while (index < sum.length) {
            sum[index] += diff;
            index += lowBit(index);
        }
    }

    public void clear() {
        Arrays.fill(sum, 0);
    }

    private int lowBit(int x) {
        return x & (-x);
    }
}

public int countInversions(int[] arr) {
    int[] sorted = Arrays.copyOf(arr, arr.length);
    Arrays.sort(sorted);

    Map<Integer, Integer> rank = getRank(sorted);

    Fenwick tree = new Fenwick(rank.size() + 1);

    int n = arr.length;
    int[] greater = new int[n];
    int[] smaller = new int[n];

    for (int i = n - 1; i >= 0; i--) {
        smaller[i] = tree.query(rank.get(arr[i]) - 1);
        tree.update(rank.get(arr[i]), 1);
    }

    tree.clear();

    for (int i = 0; i < n; i++) {
```

```
            greater[i] = i - tree.query(rank.get(arr[i]));
            tree.update(rank.get(arr[i]), 1);
        }

        int count = 0;
        for (int i = 0; i < n; i++) {
            count += smaller[i] * greater[i];
        }
        return count;
    }

    private Map<Integer, Integer> getRank(int[] sorted) {
        int rank = 1;
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < sorted.length; i++) {
            if (i == 0 || sorted[i] != sorted[i -1]) {
                map.put(sorted[i], rank++);
            }
        }
        return map;
    }
```

- 796. Rotate String

```
    public boolean rotateString(String A, String B) {
        return A.length() == B.length() && (A + A).contains(B);
    }
```

- 1051. Height Checker

```
    public int heightChecker(int[] heights) {
        int len = heights.length;
        int count = 0;
        if(len == 1) {
            return count;
        }

        int[] a = Arrays.copyOf(heights,len);
        Arrays.sort(a);

        for(int i =0; i <len; i++){
            if(heights[i]!=a[i]) {
                count++;
            }
        }
        return count;
    }
```

- 1086. High Five

```
public int[][] highFive(int[][] items) {
    Map<Integer, PriorityQueue<Integer>> map = new TreeMap<>();

    for (int[] item : items) {
        int id = item[0];
        int score = item[1];

        PriorityQueue<Integer> list = map.get(id);
        if (list == null) {
            map.put(id, new PriorityQueue<>(Collections.reverseOrder()));
        }
        map.get(id).add(score);
    }

    int[][] sol = new int[map.size()][2];

    int index = 0;

    for (int id : map.keySet()) {
        int sum = 0;
        PriorityQueue<Integer> scores = map.get(id);
        for (int i = 0; i < 5; i++) {
            sum += scores.poll();
        }

        sol[index][0] = id;
        sol[index][1] = sum / 5;
        index++;
    }
    return sol;
}
```

- 643. Maximum Average Subarray I

```
public double findMaxAverage(int[] nums, int k) {
    int fast = 0;

    double sum = 0;
    double max = Integer.MIN_VALUE;

    while (fast < k) {
        sum += nums[fast++];
    }

    max = sum;

    while (fast < nums.length) {
        sum += nums[fast];
        sum -= nums[fast - k];
        max = Math.max(max, sum);
        fast++;
    }
```

```
        return max / k;

    }
```

- 746. Min Cost Climbing Stairs

```
    public int minCostClimbingStairs(int[] cost) {
        for(int i = 2; i < cost.length; i++) {
            cost[i] += Math.min(cost[i - 1], cost[i - 2]); //当前step i已经付过钱，所以下一步的时候所以可以走下一步
        }

        return Math.min(cost[cost.length - 1], cost[cost.length - 2]);//所以这里倒数第一和倒数第二比较，cost[cost.length - 1] 和cost[cost.length - 2]都是那个index 的stair都已经
        //付过钱了，都可以reach end，就看哪个cost最小
    }
```