写在最前面：感谢勇哥（@YongLiu）提供我的这份学习资料，把我从一无所知解救了出来。

1. 有些code我写了，但是还没来得及在这更新，需要的话请在微信里告知或者在这个doc写
2. comment提醒我，我会及时更新。
3. 欢迎在leetcode上test，发现问题也请告知
4. 欢迎在code上加自己的comment或者改动，前人的学习笔记对后人也是帮助

# **Session 1 Binary Search**

1. L278 first bad version

| ***/\* You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad. Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad. You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API. \*/* public int firstBadVersion(int n) {  int left = 1;*//这里的left不是index，所以应该从1开始。这样后面就不需要check*  int right = n;  int mid = 0;  while (left <= right) {  mid = left + (right - left) / 2;  if (isBadVersion(mid)) {  right = mid - 1;  } else{  left = mid + 1;  }   }  return left;  }** |
| --- |

1. L374 Guess Number

| ***/\* We are playing the Guess Game. The game is as follows:***  ***I pick a number from 1 to n. You have to guess which number I picked.***  ***Every time you guess wrong, I'll tell you whether the number is higher or lower.***  ***You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):***  ***-1 : My number is lower 猜大了***  ***1 : My number is higher 猜小了***  ***0 : Congrats! You got it!***  ***Example :***  ***Input: n = 10, pick = 6***  ***Output: 6***  ***\*/***  **public int guessNumber(int n) {**  **int l = 1;**  **int r = n;**  **while(l <= r){**  **int mid = l + (r - l) / 2;**  **if(guess(mid) == 0) return mid;**  **else if(guess(mid) == 1) l = mid + 1; //猜小了，**  **else**  **r = mid - 1;**  **}**  **return -1; //找不到**  **}** |
| --- |
|  |

1. L34 search for a range

| */\* Given an array of integers sorted in ascending order, find the starting and ending position of a given target value. Your algorithm runtime complexity must be in the order of O(log n). If the target is not found in the array, return [-1, -1]. For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4]. \*/* **class** **Solution** {  **public** **int**[] searchRange(**int**[] nums, **int** target) {  **if**(nums == **null** || nums.length == 0) **return** **new** **int**[]{-1,-1};  **int** left = leftPos(nums, target);  **int** right = rightPos(nums, target);  **return** **new** **int**[]{left, right};  }  **private** **int** **leftPos**(**int**[] nums, **int** target){  **int** left = 0;   **int** right = nums.length - 1;  **while**(left <= right){  **int** mid = left + (right - left)/2;  **if**(nums[mid] < target){  left = mid + 1;  }**else**{  right = mid - 1;  }  }  **return** (left == nums.length || nums[left] != target) ? -1 : left;  }  **private** **int** **rightPos**(**int**[] nums, **int** target){  **int** left = 0;   **int** right = nums.length - 1;  **while**(left <= right){  **int** mid = left + (right - left)/2;  **if**(nums[mid] <= target){  left = mid + 1;  }**else**{  right = mid - 1;  }  }  **return** (right < 0 || nums[right] != target) ? -1 : right;  } } |
| --- |

1. L658 K closest position to target

| */\* Given a sorted array, two integers k and x, find the k closest elements to x in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred. Example 1: Input: [1,2,3,4,5], k=4, x=3 Output: [1,2,3,4] \*/*  *//分两步，第一步先找好有效的位置，第二步就是从left+1开始，将符合要求的element加入到//result list中去。要注意的是，left可能会出界的问题* public List<Integer> findClosestElements(int[] arr, int k, int x) {  if(arr == null || arr.length ==0)return null;  List<Integer> res = new ArrayList<Integer>();  int left = 0;  int right = arr.length-1;  int mid = 0;  while(left + 1 < right){  mid = left + (right-left)/2;  if(arr[mid] < x){  left = mid;  }else{  right = mid;  }  }  for(int i = 0; i < k; i++){  if(left>=0 && right <= arr.length-1){  if(Math.abs(arr[left]-x)<=Math.abs(arr[right]-x)){  left--;  }else{  right++;  }  }else if(left < 0){  right++;  }else{  left--;  }  }  int start = left < 0 ? 0 : left+1;//left跳出之后是在起始位置的左边一个值  for(int i = 0; i < k; i++){  res.add(arr[start+i]);  }  return res;  } |
| --- |

1. Largest Smaller position of target (不存在return -1)

| public int largestSmaller(int[] array, int target){  if(array == null || array.length == 0) return -1;  int left = 0;  int right = array.length - 1;  int mid = 0;  while(left <= right){  mid = left + (right - left)/2;  if(array[mid] < target){  left = mid + 1;  }else{  right = mid - 1;  }  }  return left - 1; // 如果不存在smaller，相当于无解，return -1  } |
| --- |

1. L35 search insertion position.(Smallest larger position)

| */\* Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You may assume no duplicates in the array. Example 1: Input: [1,3,5,6], 5 Output: 2 \*/*  *//需要clarify在相等的时候，是insert 在那个位置，以及在有重复的时候，放在哪个位置* **public** **int** **searchInsert**(**int**[] nums, **int** target){  public int searchInsert(int[] nums, int target) {  if(nums == null || nums.length == 0) return -1;  int left = 0;  int right = nums.length - 1;  int mid = 0;  if(target > nums[right]){*//把这个当作corner case处理*  return right + 1;  }  if(target < nums[left]){  return left;  }  while(left <= right){  mid = left + (right - left)/2;  if(nums[mid] >= target){  right = mid - 1;  }else{  left = mid + 1;  }  }  return left;  }  } |
| --- |

1. L162 Find peak valley element

| */\* A peak element is an element that is greater than its neighbors. Given an input array where num[i] ≠ num[i+1], find a peak element and return its index. The array may contain multiple peaks, in that case return the index to any one of the peaks is fine. You may imagine that num[-1] = num[n] = -∞. For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2. \*/* public int findPeakElement(int[] nums) {  if(nums.length == 1) return 0;  int len = nums.length;  int left = 0;  int right = len - 1;  int mid = 0;  if(nums[len-1] > nums[len-2]) return len-1;  if(nums[0] > nums[1]) return 0;  while(left <= right){  mid = left + (right - left)/2;  if(nums[mid] < nums[mid+1]){  left = mid + 1;  }else{  right = mid - 1;  }  }  return left;  }  //或者  public int findPeakElement(int[] nums) {  if(nums == null || nums.length ==0) return -1;  int left = 0;  int right = nums.length-1;  int mid = 0;  while(left +1 < right){  mid = left + (right - left)/2;  if(nums[mid] > nums[mid-1] && nums[mid] > nums[mid+1]){  return mid;  }  if(nums[mid] > nums[mid-1]){  left = mid;  }  if(nums[mid] < nums[mid-1]){  right = mid;  }  }  return nums[left]>nums[right]?left:right;  } |
| --- |

1. L702 Streaming flow find target

| */\*\*  \* // This is ArrayReader's API interface.  \* // You should not implement it, or speculate about its implementation  \* interface ArrayReader {  \* public int get(int index) {}  \* }  \*   \* 思路是先通过binary search找到ending所落在的range  \* 再通过binary search在找到的range里面找到ending的点  \*  \*/*   **public** **int** **search**(ArrayReader reader, **int** target) {  **int** right = 0, val = reader.get(right);  **while** (val != Integer.MAX\_VALUE) {  **if** (val < target) {  right = right == 0 ? 1 : right \* 2;  val = reader.get(right);  } **else** {  **break**;  }  }    **int** left = right / 2, mid;  **while** (left <= right) {  mid = left + (right - left) / 2;  val = reader.get(mid);  **if** (val == target) {  **return** mid;  } **else** **if** (val < target) {  left = mid + 1;  } **else** {  right = mid - 1;  }  }  **return** -1;  } |
| --- |

1. L74 search int a 2D matrix

| */\*  [1, 3, 5, 7],  [10, 11, 16, 20],  [23, 30, 34, 50] \*/* **public** **boolean** **searchMatrix**(**int**[][] matrix, **int** target) {  **if**(matrix == **null** || matrix.length == 0 || matrix[0] == **null** || matrix[0].length == 0) **return** **false**;  **int** row = matrix.length;  **int** col = matrix[0].length;  **int** left = 0;   **int** right = row\*col - 1;  **int** mid = 0;  **while**(left <= right){  mid = left + (right - left)/2;  **int** r = mid / col;  **int** c = mid % col;  **if**(matrix[r][c] == target){  **return** **true**;  }**else** **if**(matrix[r][c] > target){  right = mid - 1;  }**else**{  left = mid + 1;  }  }  **return** **false**;  } |
| --- |

1. L240 search in a 2D matrix II

| ***/\*  [1, 4, 7, 11, 15],  [2, 5, 8, 12, 19],  [3, 6, 9, 16, 22],  [10, 13, 14, 17, 24],  [18, 21, 23, 26, 30] \*/* *//从左下角开始找，比target大就往上走，比target小就往右走*** public boolean searchMatrix(int[][] matrix, int target) {  if(matrix == null || matrix.length == 0 || matrix[0] == null || matrix[0].length == 0) return false;  int row = matrix.length;  int col = matrix[0].length;  int r = row - 1;  int c = 0;  while(r >= 0 && c < col){  if(matrix[r][c] == target){  return true;  }else if(matrix[r][c] > target){  r--;  }else{  c++;  }  }  return false;  } |
| --- |

1. L33 search in a rotated sorted array

| */\* 通过target和nums[0]， 或者nums[left]比较，来判断target是在哪一个区域，然后分别在两个区域分情况讨论。 \*/* **public** **int** **search**(**int**[] nums, **int** target) {  **if**(nums == **null** || nums.length == 0) **return** -1;  **int** left = 0;  **int** right = nums.length-1;  **int** mid = 0;  **while**(left + 1 < right){  mid = left + (right - left)/2;    **if**(target >= nums[0]){*//target是落在前半个区域*  **if**(nums[mid] > target){  right = mid;  }**else**{  **if**(nums[mid] > nums[0]){  left = mid;  }**else**{  right = mid;  }  }  }**else**{  **if**(nums[mid] > target) {*//target在落在后半个区域*  **if**(nums[mid] > nums[0]){  left = mid;  }**else**{  right = mid;  }  }**else**{  left = mid;  }  }  }  **if**(nums[left] == target) **return** left;  **if**(nums[right] == target) **return** right;  **return** -1;  } |
| --- |

1. L81 search in a rotated sorted array II

| ***/\* Follow up for "Search in Rotated Sorted Array": What if duplicates are allowed? Would this affect the run-time complexity? How and why? \*/*** public boolean search(int[] nums, int target) {  if(nums == null || nums.length == 0)return false;  int left = 0;  int right = nums.length-1;  int mid = 0;  while(left <= right){  mid = left + (right - left)/2;  if(nums[mid] == target) return true;  if(nums[mid] > nums[left]){  if(nums[mid] >= target && target >= nums[left]){  right = mid - 1;  }else{  left = mid + 1;  }  }else if(nums[mid] < nums[left]){  if(nums[mid] <= target && target <= nums[right]){  left = mid + 1;  }else{  right = mid - 1;  }  }else{  left++;  }  }  return false;  } |
| --- |

1. L153 find minimum in rotated sorted array

| ***/\* Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2). Find the minimum element. You may assume no duplicate exists in the array. \*/*** public int findMin(int[] nums) {  if(nums == null || nums.length == 0) return -1;  if(nums.length == 1) return nums[0];  int left = 0;  int right = nums.length - 1;  int mid = 0;  if(nums[left] < nums[right]) return nums[left];  while(left + 1 < right){  mid = left + (right - left)/2;  if(nums[mid] >= nums[left]){  if(nums[mid] > nums[right]){  left = mid;  }else{  right = mid;  }   }  if(nums[mid] <= nums[right]){  if(nums[mid] > nums[left]){  left = mid;  }else{  right = mid;  }  }  }  return Math.min(nums[left], nums[right]);  } |
| --- |

| **public** **int** **findMin**(**int**[] nums) {  **if** (nums == **null** || nums.length == 0) **return** -1;    **int** len = nums.length;  **if** (nums.length == 1 || nums[0] < nums[len - 1]) **return** nums[0];    **int** left = 0, right = len - 1, mid = 0;  **while** (left <= right) {  mid = left + (right - left) / 2;  **if** (nums[mid] >= nums[0]) {  left = mid + 1;  } **else** {  right = mid - 1;  }  }  **return** nums[left];  } |
| --- |

1. L154 find minimum in rotated sorted array II

| **public** **int** **findMin**(**int**[] nums) {  **if** (nums == **null** || nums.length == 0) {  **return** -1;  }   **int** len = nums.length;  **if** (nums.length == 1 || nums[0] < nums[len - 1]) {  **return** nums[0];  }   **if** (nums[0] == nums[len - 1]) {  **for** (**int** i = 1; i < len - 1; i++) {  **if** (nums[i] < nums[i - 1]) **return** nums[i];  }  **return** nums[0];  } **else** {  **int** left = 0, right = len - 1, mid = 0;  **while** (left <= right) {  mid = left + (right - left) / 2;  **if** (nums[mid] >= nums[0]) {  left = mid + 1;  } **else** {  right = mid - 1;  }  }  **return** nums[left];  }  } |
| --- |

1. LT 183 wood cut

| ***/\* Given n pieces of wood with length L[i] (integer array). Cut them into small pieces to guarantee you could have equal or more than k pieces with the same length. What is the longest length you can get from the n pieces of wood? Given L & k, return the maximum length of the small pieces.***  ***Example***  ***Example 1***  ***Input:***  ***L = [232, 124, 456]***  ***k = 7***  ***Output: 114***  ***Explanation: We can cut it into 7 pieces if any piece is 114cm long, however we can't cut it into 7 pieces if any piece is 115cm long.***  ***Example 2***  ***Input:***  ***L = [1, 2, 3]***  ***k = 7***  ***Output: 0***  ***Explanation: It is obvious we can't make it.\*/*** public int woodCut(int[] L, int k) {  // write your code here  if (L == null || L.length == 0 || k <= 0) {  return 0;  }    int l = 1;  int r = 1;  for (int wood : L) {  r = Math.max(wood, r);  }  while(l + 1 < r){  int mid = l + (r - l) / 2;  if(getPieces(mid, L) < k) r = mid;  else  l = mid;  }  if(getPieces(r, L) >= k) return r;  if(getPieces(l, L) >= k) return l;  return 0;  }    private int getPieces(int len, int[] L){  int cnt = 0;  for(int wood : L){  cnt += wood / len;  }  return cnt;  } |
| --- |

# 

# **Session 2 Sorting Algorithm**

1. Selection sort
2. mergeSort

| *//arraylist版本* **public** List<Integer> **mergeSort**(ArrayList<Integer> array){  **if**(array == **null** || array.size() == 0) **return** **null**;  ArrayList<integer> sortedArray = mergeSort(array, 0, array.size()-1);  **return** sortedArray; }  **private** ArrayList<Integer> **mergeSort**(ArrayList<> array, **int** left, **int** right){  List<Integer> res = **new** ArrayList<Integer>();  **if**(left == right){  res.add(array.get(left));  **return** res;  }  **int** mid = left + (right - left)/2;  ArrayList<Integer> leftRes = mergeSort(array, left, mid);  ArrayList<Integer> rightRes = mergeSort(array, mid+1, right);  **return** merge(leftRes, rightRes); }  **private** ArrayList<Integer> **merge**(ArrayList<Integer> left, ArrayList<Integer> right){  }  *//array 版本* **public** **int**[] mergeSort(**int**[] array){  *//cc*  **int**[] helper = **new** **int**[array.length];  mergeSort(array, 0, array.length-1, helper);  **return** array; }  **private** **void** **mergeSort**(**int**[] array, **int** left, **int** right, **int**[] helper){  **if**(left == right) **return**;  **int** mid = left + (right - left)/2;  mergeSort(array, left, mid, helper);  mergeSort(array, mid+1, right, helper);  merge(array, left, right, mid, helper); }  **private** **void** **merge**(**int**[] array, **int** left, **int** right, **int** mid, **int**[] helper){  **int** leftStart = left;  **int** rightStart = mid +1;   **for**(**int** i = left; i <= right; i++){  helper[i] = array[i];  }  **while**(leftStart <= mid && rightStart <= right){  **if**(helper[leftStart] <= helper[rightStart]){  array[left++] = helper[leftStart++];  }**else**{  array[left++] = helper[rightStart++];  }  }   **while**(leftStart <= mid){  array[left++] = helper[leftStart++];  } } |
| --- |

1. quick sort

| **public** **int**[] quickSort(**int**[] array){  *//cc*  **int** left = 0;  **int** right = array.length - 1;  quickSort(array, left, right);  **return** array; } **private** **void** **quickSort**(**int**[] array, **int** left, **int** right){  **if**(left >= right) **return**;   **int** pivotIdx = findPivotAndPartition(array, left, right);  quickSort(array, left, pivotIdx-1);  quickSort(array, pivotIdx+1, right); }  **private** **int** **findPivotAndPartition**(**int**[] array, **int** left, **int** right){  **int** leftIdx = left;  **int** rightIdx = right;  Random rand = **new** Random();  **int** pivot = left + rand.nextInt(right - left +1);  **int** pivotVal = array[pivot];  swap(array, pivot, right);  rightIdx--;  **while**(leftIdx <= rightIdx){  **if**(array[leftIdx] <= pivotVal){  leftIdx++;  }**else** **if**(array[rightIdx] > pivotVal){  rightIdx--;  }**else**{  swap(array, leftIdx++, rightIdx--);  }  }  swap(array, leftIdx, right);  **return** leftIdx; } |
| --- |

1. count sort/bucket sort

L75 sort color

| */\* Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue. Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively. Note: You are not suppose to use the library's sort function for this problem. \*/* **public** **void** **sortColors**(**int**[] nums){  if(nums == null || nums.length == 0) return;  List<Integer> arr0list0 = new ArrayList<>();  List<Integer> arr1list1 = new ArrayList<>();  List<Integer> arr2list2 = new ArrayList<>();  for(int i = 0; i < nums.length; i++){  if(nums[i] == 0){  arr0.add(nums[i]);  }else if(nums[i] == 1){  arr1.add(nums[i]);  }else{  arr2.add(nums[i]);  }  }  int idx = 0;  for(int i = 0; i < arr0.size(); i++){  nums[idx] = arr0.get(i);  idx++;  }  for(int i = 0; i < arr1.size(); i++){  nums[idx] = arr1.get(i);  idx++;  }  for(int i = 0; i < arr2.length; i++){  nums[idx] = arr2.get(i);  idx++;  }  }  // in place  class Solution {  public void sortColors(int[] nums) {  if(nums == null || nums.length == 0) return;  int left = 0;  int index = 0;  int right = nums.length - 1;  while(index <= right){  if(nums[index] == 0){  swap(nums, index, left);  left++;  index++;  } else if(nums[index] == 1){  index++;  } else if(nums[index] == 2){  swap(nums, right, index);  right--;  }  }  }  public void swap(int nums[], int i, int j){  int temp = nums[i];  nums[i] = nums[j];  nums[j] = temp;  }  } |
| --- |

# 

# **Session 3 LinkedList**

1. L206 reverse Linkedlist

| */\* reverse Linkedlist both in iteration and recursion  \*/* public ListNode reverseList(ListNode head) {  if(head == null || head.next == null) return head;  ListNode prev = null;  ListNode next = null;  ListNode cur = head;  while(cur != null){  next = cur.next;  cur.next = prev;  prev = cur;  cur = next;  }  return prev;  }  public ListNode reverseList(ListNode head) {  if(head == null || head.next == null) return head;  ListNode newHead = reverseList(head.next);  head.next.next = head;  head.next = null;  return newHead  } |
| --- |

1. L92 reverse linkedlist II by range

| */\* Reverse a linked list from position m to n. Do it in-place and in one-pass. For example: Given 1->2->3->4->5->NULL, m = 2 and n = 4, return 1->4->3->2->5->NULL. Note: Given m, n satisfy the following condition: 1 ≤ m ≤ n ≤ length of list. \*/*  **class** **Solution** {  **public** ListNode **reverseBetween**(ListNode head, **int** m, **int** n) {  **if**(head == **null** || head.next == **null**) **return** head;  ListNode cur = head;  ListNode prev = **null**;  // 找到第m个ListNode  **for**(**int** i = 0; i < m-1; i++){  prev = cur;  cur = cur.next;  }  // reverse第m个到第n个  **if**(prev != **null**){  prev.next = reverseByN(cur, n - m + 1);  **return** head;   }**else**{  return reverseByN(head, n - m + 1);  }  }  **private** ListNode **reverseByN**(ListNode head, **int** n){  **if**(head.next == **null**) **return** head;  ListNode cur = head;  ListNode next = **null**;  ListNode prev = **null**;  **for**(**int** i = 0; i < n; i++){  next = cur.next;  cur.next = prev;  prev = cur;  cur = next;  }   head.next = next;    **return** prev;  } } |
| --- |

1. L24 Reverse LinkedList by pair

| */\* Given a linked list, swap every two adjacent nodes and return its head. Example: Given 1->2->3->4, you should return the list as 2->1->4->3. Note: Your algorithm should use only constant extra space. You may not modify the values in the list's nodes, only nodes itself may be changed. \*/* **public** ListNode **swapPairs**(ListNode head) {  **if**(head == **null** || head.next == **null**) **return** head;  ListNode subHead = swapPairs(head.next.next);  ListNode newHead = head.next;  head.next.next = head;  head.next = subHead;  **return** newHead;  } |
| --- |

1. L25,Reverse linkedlist by k

| */\* Given a linked list, reverse the nodes of a linked list k at a time and return its modified list. k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is. Example: Given this linked list: 1->2->3->4->5 For k = 2, you should return: 2->1->4->3->5 For k = 3, you should return: 3->2->1->4->5*  *Given this linked list: 1->2->3->4->5->6->7->8->9->10*  *For k = 4, you should return: 3->2->1->4->5 Note: Only constant extra memory is allowed. You may not alter the values in the list's nodes, only nodes itself may be changed. \*/* class Solution {  public ListNode reverseKGroup(ListNode head, int k) {  if(head == null) return null;  // 找到k size的linkedlist，并且截取下来  ListNode cur = head;  //这个也可以当做base case  for(int i = 0; i < k-1; i++){  cur = cur.next;  if(cur == null){  return head;  }  }  ListNode next = cur.next;  cur.next = null;  // 将k size的 linkedlist reverse，并且将原来的tail定义成newHead  ListNode newHead = reverse(head);  // 做下一个recursion，并要求将下一层的newHead返回，接上当前的newTail  ListNode newTail = head;  newTail.next = reverseKGroup(next, k);  return newHead;  }  private ListNode reverse(ListNode head){  if(head == null || head.next == null) return head;  ListNode reversedHead = reverse(head.next);  head.next.next = head;  head.next = null;  return reversedHead;  }  } |
| --- |

1. L141 check the cycle of LinkedList

| **public** **boolean** **hasCycle**(ListNode head) {  **if**(head == **null** || head.next == **null**) **return** **false**;  ListNode slow = head;  ListNode fast = head;  **while**(fast.next != **null** && fast.next.next != **null**){  slow = slow.next;  fast = fast.next.next;  **if**(slow == fast){  **return** **true**;  }  }  **return** **false**;  } |
| --- |

1. L160 check intersection

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

//先让cur1和cur2分别从头开始，并用count计算出他们各自路径的node的个数。然后再讲cur1和cur2再次从头开始。如果count1>count2,

//就让cur1走，直到走到count1 == count2。此时，在一起同步走，相遇的时候就是intersection

| ListNode cur1 = headA;  ListNode cur2 = headB;  **int** count1 = 0;  **int** count2 = 0;  **while**(cur1 != **null**){  cur1 = cur1.next;  count1++;  }  **while**(cur2 != **null**){  cur2 = cur2.next;  count2++;  }  cur1 = headA;  cur2 = headB;  **while**(count1>count2){  cur1 = cur1.next;  count1--;  }  **while**(count2>count1){  cur2 = cur2.next;  count2--;  }  **while**(cur1!=**null** || cur2!=**null**){  **if**(cur1 == cur2 ){  **return** cur1;  }  cur1 = cur1.next;  cur2 = cur2.next;  }  **return** **null**;  } |
| --- |

1. L147 Insertion sort list O(n^2)

| */\* Sort a linked list using insertion sort. \*/* **public** ListNode **insertionSortList**(ListNode head) {  *//inserttion sort的思路是双指针站肩，先确定一个j，j从1开始。先将j的位置的值cache出来，然后设置一个指针i， i从j-1往前走。遇到比j的值大的值*  *//的时候，i的值就往后移动一位，直到不能移动位置。这时候，再把j的位置的值insert进去*  **if**(head == **null** || head.next == **null**) **return** head;  ListNode cur = head;*//用来遍历原来的linkedlist*  ListNode dummy = **new** ListNode(0);  ListNode prev = dummy;  ListNode insertNode = **null**;  **while**(cur != **null**){  prev = dummy;*//每次loop，都需要将sorted好的LinkedList遍历一遍，找到可以插入的node*  **while**(prev.next != **null** && cur.val > prev.next.val){*//如果新的orted list中的值小于cur.val的话，就继续往后走，直到比cur大的*  prev = prev.next;  }    insertNode = cur;  cur = cur.next;  insertNode.next = prev.next;  prev.next = insertNode;    }  **return** dummy.next;  } |
| --- |

1. L148 sort list with O(nlogN)

| **class** **Solution** {  **public** ListNode **sortList**(ListNode head) {  **if**(head == **null** || head.next == **null**) **return** head;  ListNode prev = findMid(head);  ListNode next = prev.next;  prev.next = **null**;  ListNode head1 = sortList(head);  ListNode head2 = sortList(next);  ListNode newHead = merge(head1,head2);  **return** newHead;  }    **private** ListNode **merge**(ListNode head1, ListNode head2){  **if**(head1 == **null**) **return** head2;  **if**(head2 == **null**) **return** head1;    **if**(head1.val < head2.val){  head1.next = merge(head1.next, head2);  **return** head1;  }**else**{  head2.next = merge(head1, head2.next);  **return** head2;  }  }  **private** ListNode **findMid**(ListNode head){  ListNode prev = **null**;  ListNode slow = head;  ListNode fast = head;  **while**(fast != **null** && fast.next != **null**){  prev = slow;  slow = slow.next;  fast = fast.next.next;  }  **return** prev;  } } |
| --- |

1. L19 remove nth node from end of list

| */\* Given a linked list, remove the n-th node from the end of list and return its head. Example: Given linked list: 1->2->3->4->5, and n = 2.  After removing the second node from the end, the linked list becomes 1->2->3->5. Note: Given n will always be valid. Follow up: Could you do this in one pass? \*/*  **public** ListNode **removeNthFromEnd**(ListNode head, **int** n) {  ListNode slow = head;  ListNode fast = head;  **int** count = 0;  **while**(count < n && fast != **null**){  fast = fast.next;  count++;  }  *//这条是判断当n的值等于LinkedList的节点个数的时候的情况，很关键*  **if**(fast == **null**) **return** head.next;  **while**(fast.next != **null**){  slow = slow.next;  fast = fast.next;  }    slow.next = slow.next.next;  **return** head;  } |
| --- |

1. Q5 L203, 207, L83

# **Session 4: Stack & Heap**

1. L232 Design queue by stack
2. L 225 design stack by queue
3. follow-up: LinkedList left shift by K
4. L155 design Min Stack

3.1 follow-up: O(1)时间get medium

1. 汉诺塔， selection sort using 3 stacks

4.1 follow-up: using 2 stacks

# **Session 5 Binary Search Tree and Divide Conquer**

1. L144, L145, L94 preorder, postorder, inorder traversal
2. L104 max depth

| **public** **int** **maxDepth**(TreeNode root) {  **if**(root == **null**) **return** 0;  **int** left = maxDepth(root.left);  **int** right = maxDepth(root.right);  **return** Math.max(left, right) + 1;  } |
| --- |

1. L111, minimum depth

| **public** **int** **minDepth**(TreeNode root) {  **if**(root == **null**) **return** 0;  **int** left = minDepth(root.left);  **int** right = minDepth(root.right);  **if**(left != 0 && right != 0) **return** Math.min(left, right) + 1;  **else** **return** left + right + 1;    } |
| --- |

1. L110 Balanced Binary Tree

| **class** **Solution** {  **public** **boolean** **isBalanced**(TreeNode root) {  **if** (root == **null**) **return** **true**;  **return** getHeight(root) != -1;  }    **private** **int** **getHeight**(TreeNode root) {  **if** (root == **null**) **return** 0;    **int** left = getHeight(root.left);  **if** (left == -1) {  **return** -1;  }  **int** right = getHeight(root.right);  **if** (right == -1 || Math.abs(left - right) > 1) {  **return** -1;  }  **return** Math.max(left, right) + 1;  } } |
| --- |

1. L101 Symmetric Tree

| **class** **Solution** {  **public** **boolean** **isSymmetric**(TreeNode root) {  **if**(root == **null**) **return** **true**;  **return** isSymmetric(root.left, root.right);  }   **public** **boolean** **isSymmetric**(TreeNode left, TreeNode right){  **if**(left == **null** && right == **null**) **return** **true**;  **if**(left == **null** || right == **null**) **return** **false**;  **if**(left.val != right.val) **return** **false**;    **return** isSymmetric(left.left, right.right) && isSymmetric(left.right, right.left);  } } |
| --- |

1. L100 Same Tree

| **public** **boolean** **isSameTree**(TreeNode p, TreeNode q) {  **if**(p == **null** && q == **null**) **return** **true**;  **if**(p == **null** || q == **null**) **return** **false**;  **if**(p.val != q.val) **return** **false**;  **return** isSameTree(p.left, q.left) && isSameTree(p.right, q.right); } |
| --- |

1. L572 subtree of another tree

| **class** **Solution** {  **public** **boolean** **isSubtree**(TreeNode s, TreeNode t) {  **if**(s == **null**) **return** **false**;  **if**(isSameTree(s, t)) **return** **true**;*//也可以当做一个base case来看待*   **return** isSubtree(s.left, t) || isSubtree(s.right, t);   }  **private** **boolean** **isSameTree**(TreeNode left, TreeNode right){  **if**(left == **null** && right == **null**) **return** **true**;  **if**(left == **null** || right == **null**) **return** **false**;  **if**(left.val != right.val) **return** **false**;   **return** isSameTree(left.left, right.left) && isSameTree(left.right, right.right);  } } |
| --- |

1. LT 470 Tweaked identical binary tree

| *//niu a niu* public boolean niuAniu(TreeNode left, TreeNode right){  if(left == null && right == null) return true;  if(left == null || right == null) return false;  if(left.val != right.val) return false;  return niuAniu(left.left,right.left) && niuAniu(left.right, right.right) || niuAniu(left.left, right.right) && niuAniu(left.right, right.left);  } |
| --- |

1. L98 Validate Binary Search Tree

| *//算法哥写的* **class** **Solution** {  TreeNode prev = **null**;  **public** **boolean** **isValidBST**(TreeNode root) {  **if**(root == **null**) **return** **true**;   **if**(!isValidBST(root.left)) **return** **false**;//如果左边invalid，不用往下做   **if**(prev != **null** && prev.val >= root.val){  **return** **false**;  }  prev = root;   **return** isValidBST(root.right);   } } |
| --- |

| *//自己写的* **class** **Solution** {  TreeNode prev;  **public** **boolean** **isValidBST**(TreeNode root) {  **if**(root == **null**) **return** **true**;    **if**(!isValidBST(root.left)) **return** **false**;    **if**(prev == **null**){  prev = root;  }**else** {  **if**(prev.val >= root.val){  **return** **false**;  }**else**{  prev = root;  }   }   **return** isValidBST(root.right);   } } |
| --- |

1. LT 11 Search range in BST

| **public** **class** **Solution** {  */\*\*  \* @param root: param root: The root of the binary search tree  \* @param k1: An integer  \* @param k2: An integer  \* @return: return: Return all keys that k1<=key<=k2 in ascending order  \*/*    **public** List<Integer> **searchRange**(TreeNode root, **int** k1, **int** k2) {  *// write your code here*  List<Integer> res = **new** ArrayList<Integer>();  **if**(root == **null**) **return** res;  dfs(root, k1, k2, res);  **return** res;  }  **private** **void** **dfs**(TreeNode root, **int** k1, **int** k2, List<Integer> res){  **if**(root == **null**) **return**;    **if**(root.val > k1){//  dfs(root.left, k1, k2, res);  }    **if**(root.val >= k1 && root.val <= k2){  res.add(root.val);  }    **if**(root.val < k2){  dfs(root.right, k1, k2, res);  }    } } |
| --- |

1. L270 closest target in binary search tree

| **public** **int** **closestValue**(TreeNode root, **double** target) {   TreeNode closest = root;  TreeNode cur = root;  **while**(cur != **null**){   **if**(Math.abs(cur.val - target) < Math.abs(closest.val - target)){  closest = cur;  }  **if**(cur.val < target){  cur = cur.right;  }**else**{  cur = cur.left;  }  }  **return** closest.val;  } |
| --- |

# **Session 6 Heap and HashMap**

1. L347 Top K frequency element

| ***/\* Given a non-empty array of integers, return the k most frequent elements. For example, Given [1,1,1,2,2,3] and k = 2, return [1,2]. Note:  You may assume k is always valid, 1 ≤ k ≤ number of unique elements. Your algorithm's time complexity must be better than O(n log n), where n is the array's size.  \*/* *//用 n size 的 maxHeap，将所有的 非重复的element都加进去，然后poll k个出来* class Solution {   class Element{  int val;  int frequency;  public Element(int val, int frequency){  this.val = val;  this.frequency = frequency;  }  }    public List<Integer> topKFrequent(int[] nums, int k) {  List<Integer> res = new ArrayList<Integer>();  if(nums == null) return res;  HashMap<Integer, Integer> map = new HashMap<>();*//key is value , value is frequency*  for(int i = 0; i < nums.length; i++){  if(map.containsKey(nums[i])){  int count = map.get(nums[i]);  count++;  map.put(nums[i], count);  }else{  map.put(nums[i], 1);  }  }    PriorityQueue<Element> maxHeap = new PriorityQueue<>(nums.length, new Comparator<Element>(){  @Override  public int compare(Element e1, Element e2){  return e2.frequency - e1.frequency;  }  });    for(Integer key : map.keySet()){  maxHeap.offer(new Element(key, map.get(key)));  }  while(k-- > 0){  res.add(maxHeap.poll().val);  }  return res;  } }** |
| --- |

| *//用k size 的min heap* **class** **Solution** {   **class** **Element**{  **int** val;  **int** frequency;  **public** **Element**(**int** val, **int** frequency){  **this**.val = val;  **this**.frequency = frequency;  }  }    **public** List<Integer> **topKFrequent**(**int**[] nums, **int** k) {  List<Integer> res = **new** ArrayList<Integer>();  **if**(nums == **null**) **return** res;  HashMap<Integer, Integer> map = **new** HashMap<>();*//key is value , value is frequency*  **for**(**int** i = 0; i < nums.length; i++){  **if**(map.containsKey(nums[i])){  **int** count = map.get(nums[i]);  count++;  map.put(nums[i], count);  }**else**{  map.put(nums[i], 1);  }  }    PriorityQueue<Element> minHeap = **new** PriorityQueue<>(k, **new** Comparator<Element>(){  **@Override**  **public** **int** **compare**(Element e1, Element e2){  **return** e1.frequency - e2.frequency;  }  });    **for**(Integer key : map.keySet()){    **if**(minHeap.size() < k){  minHeap.offer(**new** Element(key, map.get(key)));  } else **if**(map.get(key) > minHeap.peek().frequency){  minHeap.poll();  minHeap.offer(**new** Element(key, map.get(key)));  }  }  **while**(minHeap.size() != 0){  res.add(0, minHeap.poll().val);  }  **return** res;  } } |
| --- |

# **Session 7 数据结构之间的相互实现**

1. design stack using LinkedList

| **public** **class** **myStack{**  **class ListNode{**  **int value;**  **int next;**  **public ListNode(int value){**  **this.value = value;**  **}**  **}**  **}** |
| --- |

1. Implement HashMap

//hashmap的内层实现是用一个linkedList

# **Session 8 BFS DFS and Dijkstra**

//微软面经

//amazon:

四轮，每一轮 一道BQ + 一道题

第一轮： level order tracersal 带 next 指针

第二题：missing number，n个出现奇数次，m个出现偶数次，内存放不下怎么办

第三题：BFS解题，题意是模拟一个游戏，在一个2Dmatrix上，有一些点是炮台，每个炮台有一定的射击范围，如果范围是1的话，就表示可以射击的范围是周围一圈，如果是2的话就表示是周围两圈。现在需要空降一些伞兵下去，问伞兵不被射击的概率有多大。

第四题：meeting room，两种 解法。size特别大怎么办，涉及到外排序。

what is the day to day basis

what is the most challenging part of this position

1. L314 Binary Vertical order traversal

| ***/\* Given a binary tree, return the vertical order traversal of its nodes' values. (ie, from top to bottom, column by column). If two nodes are in the same row and column, the order should be from left to right. \*/* */\*用dfs没法做到上下有序，即再同一个vertical level的情况下，需要从上到下有序排排列 \*／*** |
| --- |

| *//用两个queue，一个存level上的node，一个存vertical上的column的信息。用一个hashmap存column的信息和arraylist，arraylist存的是node的value*  **public** List<List<Integer>> verticalOrder(TreeNode root) {  List<List<Integer>> res = **new** ArrayList<>();  **if**(root == **null**) **return** res;  *//keep 一个min和max，用于后面能够有序将hashMap中的list按照从左到右的顺序添加到res里面去*  **int** min = Integer.MAX\_VALUE;  **int** max = Integer.MIN\_VALUE;    Queue<TreeNode> levelQue = **new** LinkedList<>();  Queue<Integer> colQue = **new** LinkedList<>();  HashMap<Integer, ArrayList<Integer>> map = **new** HashMap<>();    levelQue.offer(root);  colQue.offer(0);    **while**(!levelQue.isEmpty()){  TreeNode pollNode = levelQue.poll();  **int** col = colQue.poll();    min = Math.min(col, min);  max = Math.max(col, max);  *//更新hashmap里面存的值*  **if**(map.containsKey(col)){  map.get(col).add(pollNode.val);  }**else**{  ArrayList<Integer> list = **new** ArrayList<>();  list.add(pollNode.val);  map.put(col, list);  }  *//更新queue里面存的值*  **if**(pollNode.left != **null**){  levelQue.offer(pollNode.left);  colQue.offer(col - 1);  }    **if**(pollNode.right != **null**){  levelQue.offer(pollNode.right);  colQue.offer(col + 1);  }  }  **for**(**int** i = min; i <= max; i++){  res.add(map.get(i));  }    **return** res;  } |
| --- |

1. Geeksforgeeks.Validate a complete binary tree

| **public** boolean isCompleteBinaryTree(TreeNode root){  if(root == null) return true;  Queue<TreeNode> queue = new LinkedList<TreeNode>();  boolean flag = false;  queue.offer(root);  while(!queue.isEmpty()){  TreeNode pollNode = queue.poll();  if(pollNode != null){  if(flag) return false;  queue.offer(pollNode.left);  queue.offer(pollNode.right);  }else{  flag = true;  }  }  return true;  } |
| --- |

# **Session 9 Bit Operation and**

1. L191 number of one

//Hamming weight

//n右移

| **public** **int** **hammingWeight**(**int** n){  **if**(n == 0) **return** 0;  **int** count = 0;  **for**(**int** i = 0; i < 32; i++){  **if**(n == 0) **break**;  **if**((n >>i) & 1 != 0) count++;  }  **return** count; } |
| --- |

1. L136 Single number

//Given an array of integers, every element appears *twice* except for one. Find single one.

//0和任何数偶异或都还是那个数

| **public** **int** **singleNumber**(**int**[] nums){  **if**(nums == **null** || nums.length == 0) **throw** **new** IllegalArgumentException();  **int** res = 0;  **for**(**int** i = 0; i < nums.length; i++){  res^=nums[i];  }  **return** res; } |
| --- |

1. L231 Power of two

//Given an integer, write a function to determine if it is a power of two.

//比特操作解法

| **public** **boolean** **isPowofTwo**(**int** n){  **if**(n<=0) **throw** IllegalArgumentException();  **int** count = 0;  **while**(n != 0){  **if**((n & 1) != 0) count++;  n >>= 1;  }  **return** count == 1; } |
| --- |
|  |

//s2

| **public** **boolean** **isPowOfTwo**(**int** n){  *//cc*  **return** n && (n-1) == 0; } |
| --- |

1. L217

//Given an array of integers, find if the array contains any duplicates.

//hashSet解法

1. L220

/\*

Given an array of integers, find out whether there are two distinct indices *i* and *j* in the array such that the **absolute** difference between **nums[i]** and **nums[j]** is at most *t* and the **absolute** difference between *i* and *j* is at most *k*.

用TreeSet, 算法哥课之后做

\*/

| **public** **boolean** **containsNearbyAlmostDuplicate**(**int**[] nums, **int** k, **int** t){  **if**(nums == **null** || nums.length == 0) **return** **false**;  } |
| --- |

1. check duplicate in a word

| **public** **boolean** **dupInWord**(**char**[] word){  **if**(word == **null** || word.length == 0) **return** **true**;  **int**[] map = **new** **int**[8];  **int** row = 0;  **int** col = 0;  **for**(**char** c : word){  row = c / 32;  col = c % 32;  **if**(map[row] & (1<<col) != 0) **return** **false**;  **else**{  map[row] |= 1 << col;  }  }  **return** **true**; } |
| --- |

1. L190 reverse Bit

//

| **public** **int** **reverseBits**(**int** n){  **if**(n == 0 || n == -1) **return** n;  **for**(**int** i = 0; i < 16; i++){  **if**((n >> i) & 1 != (n>>(31-i)) & 1){  n^=1<< i;  n ^ =1(31-i);  }  }  **return** n; } |
| --- |

1. Reverse Integer including negative

| **public** **int** **reverseInteger**(**int** n){  **if**(Math.abs(n) < 10) **return** n;  **int** sign = n > 0 ? 1 : -1;  **int** res = 0;  **while**(n > 0){  **if**(res > Integer.MAX\_VALUE/10) **return** 0;  res = res \* 10 + n%10;  n = n/10;  **if**(res > Integer.MAX\_VALUE || res < Integer.MIN\_VALUE) **return** 0  }  **return** sign\*res; } |
| --- |

1. 十进制到十六进制

| **public** String **decToHex**(**int** n){  **if**(n < 10) **return** **new** String("0X") + n;  StringBuilder sb = **new** StringBuilder;  } |
| --- |

1. a

# 

# **Session 10 Array And String**

1. Remove chars from string

//Example: “yougetoffer” remove f and o, then return “yugeter”

| **public** String **removeCharFromString**(String str){  **if**(str == **null** || str.length() == 0) **return** **null**;  **int** slow = 0;  **char**[] array = str.toCharArray();  **for**(**int** fast = 0; fast < array.length; fast++){  **if**(array[fast] != 'o' || array[fast] != 'f'){  array[slow++] = array[fast];  }  }  **return** **new** String(array, 0, slow); } |
| --- |

1. Remove leading/trailing/ and duplicate space with one remaining

| *//input "\_ \_ \_ you\_ \_ \_get\_ \_ \_ offer\_ \_ \_" , output "you\_get\_offer"* *//2 solutions, keep the space before the word, or keep the space after the word* *//code of solution 1: keep the space after the word* |
| --- |

| **public** String **removeSpaceLeftOne**(String str){  **if**(str == **null** || str.length() == 0) **return** **null**;  **char**[] array = str.toCharArray();  **int** slow = 0;  **for**(**int** fast = 0; fast < array.length; fast++){  **if**(array[fast] != ' ' || fast != 0 && array[fast - 1] != ' '){  array[slow++] = array[fast];  }  }  **if**(slow == 0) **return** “”;*//slow doesn't need to move*  **return** array[slow - 1] == ' ' ? **new** String(array, 0, slow-1) : **new** String(array, 0, slow); } |
| --- |

//code for solution2: keep the space before the word

| **public** String **removeSpaceLeftOne**(String str){  *//cc*  **char**[] array = str.toCharArray();  **int** slow = 0;  **for**(**int** fast = 0; fast < array.length(); fast++){  **if**(array[fast] != ' ' || fast < array.length - 1 && array[fast + 1] != ' '){  array[slow++] = array[fast];  }  }  **if**(slow == 0) **return** “”;  **return** array[0] == ' ' ? **new** String(array, 1, slow - 1) : **new** String(array, 0, slow); *// new string(array, 1, xxx) 这里的xxx表示的是sub array 的length，所以应该是slow - 1， 1 + slow - 1 = slow (Joe Zhou)* } |
| --- |

1. Remove duplicate from sorted array

//code for with 0 left

| *//S1 hashmap* **public** **int**[] dedupWithZeroLeft(**int**[] array){ *//cc*  HashMap<Integer, Integer> map = **new** HashMap<Integer, Integer>();  **int** count = 0;  **ArrayList<integer> res = new ArrayList<Integer>()**;  **for**(**int** i = 0; i < array.length; i++){  **if**(map.containsKey(array[i])){  count = map.get(array[i])+1;  map.put(array[i], count)；  }**else**{  map.put(array[i], 1);  }  }  for(int i = 0; i < array.length; i++){  if(map.get(array[i]) == 1){  res.add(array[i]);  }  }  **return** res.toArray(); } |
| --- |

// stack + flag

//code for with 1 left

| **public** **char**[] removeDupLeftOne(**char**[] array){  //cc  **int** slow = 1;  **for** (**int** fast = 1; fast < array.length; fast++){  **if**(array[fast] != array[slow - 1]){  array[slow++] = array[fast];  }  }  **return** array;// } |
| --- |

//code for with k left, [0,slow)

| **public** String **removeWithKLeft**(String str){  **if**(str == **null** || str.length() < k) **return** str;  **char**[] array = str.toCharArray();  **int** slow = k;  **for** (**int** fast = k; fast < str.length; fast++) {  **if**(array[fast] != array[slow - k]){  array[slow++] = array[fast];  }  }  **return** **new** String(array, 0, slow); } |
| --- |

1. L344 Reverse String

//3 solutions, two pointers, tail recursion, and original recursion

| *//S1 two pointers and swap*  **public** String **reverseString**(String str){  **if**(str == **null**) **return** **null**;  **char**[] chars = str.toCharArray();  **int** sow = 0;  **int** right = chars.length;  **while** (left < right){  swap(array, left++, right--);  }  **return** **new** String(array, 0, array.length); } **private** **void** **swap**(**char**[] chars, **int** left, **int** right){  **char** temp = chars[left];  chars[left] = chars[right];  chars[right] = temp; } |
| --- |

| //S2 recursion  **public** String **reverseString**(String str){  *//cc*  **char**[] chars = str.toCharArray();  **int** left = 0;  **int** right = chars.length-1;  reverseString(**char**, left, right);  **return** **new** String(chars, 0, chars.length); } **private** **void** **reverseString**(**char**[] chars, **int** left, **int** right){  **if**(left >= right) **return**;  swap(chars,left, right);  **return** reverseString(chars, left + 1, right - 1); } |
| --- |

1. L151 Reverse Words in a String

//L151,

| **/\***  there are spaces in the trail/body/tail, and the method need to remove the trailing and tail space and left only one space between words  **\*/**  **class** **Solution**{  **public** String **reverseWordsInString**(String str){  **if**(str == **null** || str.length() == 0)**return** **null**;  String trimmedStr = removeSpace(str);  **char**[] chars = trimmedStr.toCharArray();  reverseString(chars, 0, chars.length-1);  **int** start = -1;  **for**(**int** end = 0; end < chars.length; end++){  **if**(chars[end] == ' '){  reverseString(chars, start + 1, end - 1);  start = end;  }  }  reverse(chars, start + 1, end - 1);*//post processing since when end go beyond the range, the last word will not be reversed*  **return** chars; } **private** String **removeSpace**(String str){  **char**[] array = str.toCharArray();  **int** slow = 0;  **for**(**int** fast = 0; fast < array.length; fast++){  **if**(array[fast] != ' ' || fast != 0 && array[fast - 1] != ' '){  array[slow++] = array[fast];  }  }  **if**(slow == 0) **return** "";  **return** array[slow - 1] == ' ' ? **new** String(array, 0, slow - 1) : **new** String(array, 0, slow); }  **private** **void** **reverseString**(**char**[] chars, **int** left, **int** right){  **while**(left < right){  swap(chars, left++, right--)  }  **return** chars; } **private** **void** **swap**(**char**[] chars, **int** left, **int** right){  **char** temp = chars[left];  chars[left] = chars[right];  chars[right] = temp;  } } |
| --- |

1. follow-up of Q5. let’s go to ‘new york‘

| /\*  what if the input is let’s go to ‘new york’, and the output should be ‘new york’ to go let’s  assume the string is trimmed  **\*/**  **public** String **reverseString**(String str){  *//cc*  **char**[] chars = str.toCharArray();  reverseString(chars, 0, chars.length-1);  **int** slow = -1;  **int** fast = 0  **while**(fast < chars.length){  **if**(chars[fast] == ' '&& fast !== 0 && chars[fast - 1] == ' \' '){  reverseString(chars, slow + 1, fast-1);  **break**;  }  fast++;  }  **while**(fast < chars.length){  **if**(chars[fast] == ' '){  reverseString(chars, slow+1, fast - 1);  }  fast++;  }  reverseString(chars, slow+1, fast-1);  **return** **new** String(chars, 0, fast); } |
| --- |

//L186 is almost the same because the input is char[]

| class Solution {  public void reverseWords(char[] str) {  if(str == null || str.length == 0) return;    reverse(str, 0, str.length - 1);    int start = 0;  for(int end = 0; end <= str.length; end++) {  if(end == str.length || str[end] == ' ') {  reverse(str, start, end-1);  start = end + 1;  }  }  }    private void reverse(char[] arr, int start, int end) {  while(start < end) {  char temp = arr[start];  arr[start++] = arr[end];  arr[end--] = temp;  }    }  } |
| --- |

1. 6. String Rotate

//LT 8, rotate string. Given a char[] and an offset, rotate the char[] from the place of offset

//example: abcdeft, and offset = 3, the output is efgabcd

| **public** String **rotateString**(String str, **int** offset){  **if**(str == **null** || str.length() == 0) **return** **null**  **char**[] chars = str.toCharArray();  offset = offset % chars.length;   reverseString(chars, 0, chars.length-1);  reverse(chars, 0, offset-1);  reverse(chars, offset, chars.length-1);  **return** chars; } |
| --- |

1. L44, L10, space replacement

//L44 wildcard match: 算法加强再做

//L10 regular expression match: 算法加强再做

| /\*  Write a method to replace all spaces in a string with %20. The string is given in a characters array, you can assume it has enough space for replacement and you are given the true length of the string.  \*/  **public** **int** **replaceBlank**(**char**[] string, **int** length) {  // write your code here  **if**(string == **null**) **return** 0;  **int** count = 0;  **for** (**int** i = 0; i < string.length; i++){  **if**(string[i] == ' ')count++;  }  **int** slow = 2\*count + length-1;  **for**(**int** fast = length-1; fast >= 0; fast--){  **if**(string[fast] == ' '){  string[slow--] = '0';  string[slow--] = '2';  string[slow--] = '%';  }**else**{  string[slow--] = string[fast];  }  }  **return** string.length;  } |
| --- |

1. Encoding and decoding

//Q8 Encoding: aaabbbbccdeee to a3b4c2d1e3, Leetcode 443

| **public** String **encoding**(String str){  **if**(str == **null** || str.length == 0) **return** **null**;  **char**[] chars = str.toCharArray();  Arrays.sort(chars);  HashMap<Character, Integer> hm = **new** HashMap<>();  **int** count = 0;  **for**(**int** i = 0; i < chars.length; i++){  **while**(i < chars.length && chars[i] == chars[i+1]){  count++;  }  hm.put(chars[i], count);  count = 0;  } } |
| --- |

L443 String Compression

| **class** **Solution** {  **public** **int** **compress**(**char**[] chars) {  **if**(chars == **null** || chars.length == 0) **return** 0;    **int** count = 0;  **int** slow = 0;  **int** len = chars.length;  **for**(**int** fast = 0; fast <= len; fast++) {  **if**(fast == len || (fast != 0 && chars[fast] != chars[fast - 1])) {  chars[slow++] = chars[fast - 1];  **if**(count > 1) {  **char**[] countChar = String.valueOf(count).toCharArray();  **for** (**int** i = 0; i < countChar.length; i++) {  chars[slow++] = countChar[i];  }  }  count = 1;  }**else** {  count++;  }  }    **return** slow;  } } |
| --- |

1. //Q8.1 Decode String: L394, 算法加强的时候做

//s = "3[a]2[bc]", return "aaabcbc".

| **public** String **decodeString**(String s){  **if**(s == **null** || s.length() == 0) **return** **null**; } |
| --- |

1. **//**Q8.2 Matching: i18h-->internationalization， **算法加强的时候做**
2. **//**Q8.3 L91 decode ways
3. **//**Q8.4 count and say
4. **//**Q8.5 L8 String to Integer

| **class** **Solution** {  **public** **int** **myAtoi**(String str) {  */\*  需要考虑四个问题：  1. space的问题  2. sign的问题  3. string是否valid的问题  4. overflow的问题  \*/*  **if**(str == **null** || str.length() == 0) **return** 0;  **int** index = 0;  **int** sign = 1;  **int** len = str.length();  **long** res = 0;  *//处理空格*  **while**(str.charAt(index) == ' ') {  index++;  **if**(index == len) **return** 0;  }    *//处理符号.走到这里的时候，表示空格都已经走完了，接下来第一个如果是符号的话，就更新符号*  **if**(str.charAt(index) == '-' || str.charAt(index) == '+') {  sign = str.charAt(index) == '+' ? 1 : -1;  index++;  }    *//处理invalid的情况*  **while**(index < len) {  **int** num = str.charAt(index) - '0';  **if**(num < 0 || num > 9) **break**;    **if**(res \* 10 > Integer.MAX\_VALUE || res \* 10 + num > Integer.MAX\_VALUE) {  **return** sign == 1 ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;  }    res = res\*10 + num;  index++;  }    **return** (**int**)res\*sign;  } } |
| --- |

1. 9. L28, Implement strStr

**//S1 O(n\*m)**

| **public** **int** **strStr**(String haystack, String needle) {  **if**((needle == **null** || needle.length() == 0))**return** 0;  **if**(haystack == **null** || haystack.length() == 0 ) **return** -1;  **if**(haystack.length() < needle.length()) **return** -1;    **for**(**int** i = 0; i <= haystack.length()- needle.length(); i++){  **int** j = 0;  **for**(j = 0; j < needle.length(); j++){  **if**(haystack.charAt(i+j) != needle.charAt(j)){  **break**;  }  }  **if**(j == needle.length())**return** i;  }  **return** -1;  } |
| --- |

| *// Donator @Joannasijie Jiang* **class** **Solution** {  **public** **int** **strStr**(String s1, String s2){  **if**(s1 == **null** || s1.length() == 0 || s2 == **null** || s2.length() == 0) {  **return** -1;  }  **if**(s2.length() >= s1.length()) {  **return** strStr(s2, s1);  }  **for**(**int** i = 0; i <= s1.length() - s2.length(); i++){  **int** j = 0;  **for**(; j < s2.length(); j++){  **if**(s1.charAt(i + j)!= s2.charAt(j)) {  **break**;  }  }  **if**(j == s2.length()) {  **return** i;  }  }  **return** -1;  } } |
| --- |

| ***//S2 robin carb O(n)*** |
| --- |

1. String permutation without duplicate

| **public** List<List<Integer>> permutation(**int**[] array){  List<List<Integer>> res = **new** ArrayList<List<Integer>>();  **if**(array == **null** || array.length == 0) **return** res;  permutation(res, array, 0);  **return** res;  } **private** **void** **permutation**(List<List<Integer>> res, **int**[] array, **int** level){  **if**(level == array.length-1){  List<Integer> path = **new** ArrayList<Integer>();  **for**(**int** i = 0; i < array.length; i++){  path.add(array[i]);  }  res.add(path);  **return**;  }  **for**(**int** i = level; i < array.length; i++){  swap(array, i , level);  permutation(res, array, level+1);  swap(array, i , level);  } } |
| --- |

1. //**10.1 permutation with duplicate**

| **public** List<List<Integer>> permutationWithDuplicate(**int**[] nums){  List<List<Integer>> res = **new** ArrayList<List<Integer>>();  **if**(nums == **null** || nums.length == 0) **return** res;  helper(res, nums, 0);  **return** res; } **private** **void** **helper**(List<List<Integer>> res, **int**[] nums, **int** level){  **if**(level == nums.length - 1){  res.add(**new** String(nums, 0, nums.length));//integer cannot be convert to string  **return**;  }  Set<Integer> set = **new** HashSet<Integer>();  **for**(**int** i = level; i < nums.length; i++){  **if**(set.add(nums[i])){  swap(nums, i, level);  helper(res, nums, level+1);  swap(nums, i, level);  }  } } |
| --- |

| **class** **Solution** {  **public** List<List<Integer>> permuteUnique(**int**[] nums) {  List<List<Integer>> res = **new** ArrayList<>();  **if**(nums == **null** || nums.length == 0) **return** res;  Arrays.sort(nums);  Set<Integer> set = **new** HashSet<>();  dfs(nums, res, set, **new** ArrayList<>());  **return** res;  }    **private** **void** **dfs**(**int**[] nums, List<List<Integer>> res, Set<Integer> set, List<Integer> path) {   **if**(path.size() == nums.length) {    res.add(**new** ArrayList(path));  **return**;  }    **for**(**int** i = 0; i < nums.length; i++) {   **if**(i > 0 && nums[i] == nums[i-1] && !set.contains(i-1)) {  **continue**;  } **else**{  **if**(set.add(i)) {  path.add(nums[i]);  dfs(nums, res, set, path);  path.remove(path.size() - 1);  set.remove(i);  }   }   }   } } |
| --- |

1. 11. L242, valid anagram

//Example: s = "anagram", t = "nagaram", return true.

//用一个hashmap做

| **public** **boolean** **isanagram**(String s1, String s2){  **if**(s1 == **null**) **return** s2 ==**null**;  **if**(s1.length() != s2.length()) **return** **false**;//长度必须相等  HashMap<Character, Integer> hm = **new** HashMap<Character, Integer>();  **char**[] char1 = s1.toCharArray();  **char**[] char2 = s2.toCharArray();  **int** count = 0;  **for**(**char** c1 : char1){  **if**(hm.containsKey(c1)){  count = hm.get(c1) + 1;  hm.put(c1, count);  }**else**{  hm.put(c1, 1);  }  }   **for**(**char** c2 : char2){  **if**(hm.containsKey(c2)){  **if**(hm.get(c2) == 1) {  hm.remove(c2);  }**else** {  count = hm.get(c2) - 1;  hm.put(c2, count);  }  }**else**{  **return** **false**;  }  }    **return** hm.size() == 0; } |
| --- |

//用int[26]做，相当于26进制

| **public** **boolean** **isAnagram**(String s, String t){  **if**(s == **null**) **return** t ==**null**;  **if**(t == **null**) **return** s == **null**;  **if**(s.length() != t.length()) **return** **false**;  **int**[] count = **new** **int**[26];  **char** temp;  **for**(**int** i = 0; i < s.length(); i++){  temp = s.charAt(i);  count[temp - 'a']++;  }  **for**(**int** j = 0; j < t.length(); j++){  temp = t.charAt(j);  **if**(count[temp - 'a'] > 0){  count[temp - 'a']--;  }**else**{  **return** **false**;  }  }  **for**(**int** k = 0; k < count.length; k++){  **if**(count[k] != 0) **return** **false**;  }  **return** **true**; } |
| --- |

1. //Group Anagram

| **class** **Solution** {  **public** List<List<String>> groupAnagrams(String[] strs) {  List<List<String>> res = **new** ArrayList<>();  **if**(strs == **null** || strs.length == 0) **return** res;    Map<String, List<String>> map = **new** HashMap<>();    **for**(String str : strs) {  String sortedStr = sortString(str);  List<String> list = map.get(sortedStr);  **if**(list != **null**) {  list.add(str);  }**else** {  List<String> newList = **new** ArrayList<>();  newList.add(str);  map.put(sortedStr, newList);  }  }    **for**(String entry : map.keySet()) {  res.add(map.get(entry));  }  **return** res;  }    **private** String **sortString**(String s) {    **char**[] str = s.toCharArray();  Arrays.sort(str);    **return** **new** String(str);  } } |
| --- |

1. 11.1 L125 valid palindrome

//从两边往中间走

| **public** **boolean** **isPalindrome**(String s){  **if**(s == **null** || s.length() == 0) **return** **true**;  **int** left = 0;  **int** right = s.length() - 1;  **while**(left < right){  **if**(s.charAt(left) != s.charAt(right)) **return** **false**;  left++;  right--;  }  **return** **true**; } |
| --- |

11.2 L9 palindrome number

//

public

1. 11.3 L205 Isomorphic String

//题目假设两个string的长度是一样的，其实如果不一样的话，直接在corner case里面//check就可以了

| **public** **boolean** **isIsomorphic**(String s, String t) {  **if**(s == **null**) **return** t == **null**;  **if**(t == **null**) **return** s == **null**;  HashMap<Character, Integer> mapS = **new** HashMap<Character, Integer>();  HashMap<Character, Integer> mapT = **new** HashMap<Character, Integer>();  **int** len = s.length();  **char** c1;  **char** c2;  **for**(**int** i = 0; i < len; i++){  c1 = s.charAt(i);  c2 = t.charAt(i);  **if**(!mapS.containsKey(c1)){  mapS.put(c1, i);  }  **if**(!mapT.containsKey(c2)){  mapT.put(c2,i);  }    **if**(mapS.get(c1) != mapT.get(c2)){  **return** **false**;  }  }  **return** **true**;  } |
| --- |

1. 12. L5 Longest palindrome Substring

//算法陪练的解法

| **public** String **longestPalindrome**(String s){  **if**(s == **null** || s.length() == 0)  **int** left = 0;  **int** right = 0;//left 和right是用来表示最长的palindrom的头尾  **int** l, r;  **for**(**int** i = 0; i < s.length(); i++){  l = i;  r = i;  **while**(r+1 < s.length() && s.charAt(i) == s.charAt(i+1)){  r++;  i++;  }  **while**(l > 0 && r < s.length() - 1 && s.charAt(l-1) == s.charAt(r+1)){  l--;  r++;  }  **if**(r - l > right - left){  right = r;  left = l;  }  }  **return** s.substring(left, right+1);  } |
| --- |

//算法加强的解法

| **public** String **longestPalindrome**(String s){  **if**(s == **null** || s.length() == 0) **return** **null**;  String res = "";  **for** (**int** i = 0; i < s.length(); i++){  String str = search(s, i, i);  **if**(str.length() > res.length()){  res = str;  }  str = search(s, i, i+1);  **if**(str.length() > res.length()){  res = str;  }  }  **return** res; } **private** String **search**(String s, **int** left, **int** right){  **while**(left >=0 && right < s.length() && s.charAt(left) == s.charAt(right)){  left--;  right++;   }  **return** s.substring(left + 1, right); } |
| --- |

1. 13. Leetcode 5: Longest substring without repeating character

| **class Solution {  public int lengthOfLongestSubstring(String s) {  if(s == null || s.length() == 0) return 0;    Map<Character, Integer> map = new HashMap<>();  int start = 0;  int max = 0;  for(int end = 0;end < s.length(); end++) {  char c = s.charAt(end);  Integer idx = map.get(c);  if(idx != null) {  start = Math.max(start, idx + 1);  }  max = Math.max(end - start + 1, max);  map.put(c, end);  }  return max;  } }** |
| --- |

1. 13.1 longest substring length with at most k repeating character

# Session 11 Dynamic Programming and Greedy

1. 1.Fibonacci and climbing stairs

//code of recursion

| **public** **long** **fibRec**(**int** n){*//注意可能出界的问题，用long比较保险，stand out*  **if**(n<=1) **return** n;  **return** fib(n-1) + fib(n-2) } *//code for dp with O(n) space* **public** **long** **fibDP1**(**int** n){  **if**(n<=1) **return** n;  **int**[] dp = **new** **int**[n];  dp[0] = 0;  dp[1] = 1;  **for**(**int** i = 2; i <=n; i++){  dp[i] = dp[i-1] + dp[i-2];  }  **return** dp[n]; } |
| --- |

//code for dp with O(1) space

| **public** **int** **fibDP2**(**int** n){  **if**(n <= 1) **return** n;  **int** pp = 0;  **int** p = 1;  **int** res = 0;  **for** (**int** i = 2; i <= n; i++){  res = pp+p;  pp = res;  p = pp;  }  **return** res; } |
| --- |

1. 32. L118, L119, L120 Pascal’s Triangle minimum path sum

//2.1 L118 Generate Pascal’s Triangle

//code

//2.2 L119

//2.3 L120 Min Path Sum

//leetcode 的input是List<List<Integer>>, 要求O(n)的空间复杂度

//S1 DP 空间复杂度为O(n^2)

| **public** **int** **minimumTotal**(List<List<Integer>> triangle) {  *//if(triangle == null) throw new IllegalArgumentException();*  **int** len = triangle.size();  **int**[][] dp = **new** **int**[len][len];  **for**(**int** i = 0; i < len; i++){  dp[len-1][i] = triangle.get(len-1).get(i);  }  **for**(**int** i = len-2; i >=0; i--){  **for**(**int** j = 0; j <= i; j++){  dp[i][j] = Math.min(dp[i+1][j], dp[i+1][j+1]) + triangle.get(i).get(j);  }  }  **return** dp[0][0];  } |
| --- |

//S2 DP 空间复杂度为 O(n)

| **public** **int** **minimumTotal**(List<List<Integer>> triangle) {  *//if(triangle == null) throw new IllegalArgumentExcption();*  **int** len = triangle.size();  **int**[] dp = **new** **int**[len];  **for**(**int** i = 0; i < len; i++){  dp[i] = triangle.get(len-1).get(i);  }  **for**(**int** i = len-2; i >=0; i--){  **for**(**int** j = 0; j <= i; j++){  dp[j] = Math.min(dp[j], dp[j+1]) + triangle.get(i).get(j);  }  }  **return** dp[0];  } |
| --- |

//**S3 top down recursion**

| **Int min = Integer.MAX\_VALUE;**  **public** **int** **minimumTotal**(**int**[][] matrix){  *//cc*  if (matrix == null || matrix.length == 0 || matrix[0] == null || matrix[0].length == 0) return 0;    minimumTotal(0, 0, 0, matrix);  return min;  } **private** **void**  **minimumTotal**(**int** i, **int** j, **int** sum, **int**[][] matrix){  **if**(i == matrix.length) {  if(sum < min) {  sum = min;  }  return;  }  **minimumTotal**(i + 1, j, sum + matrix[i][j], matrix);  **minimumTotal**(i + 1, j + 1, sum + matrix[i][j], matrix); } |
| --- |

//**S4 bottotm up recursion**

| **public** **int** **minimumTotal**(**int**[][] matrix){  *//cc*  **return** minimumTotal(0,0,matrix); } **private** **int** **minimumTotal**(**int** i, **int** j, **int**[][] matrix){  **if**(i == n) **return** 0;  **int** left = minimumTotal(i+1, j, matrix);  **int** right = minimumTotal(i+1, j+1, matrix);  **return** Math.min(left, right) + matrix[i][j]; } |
| --- |

//**S5 带剪枝的recursion,用记忆化存储的方法**

| **public** **int** **minimumTotal**(**int**[][] matrix){  *//cc*  Integer[][] dp = **new** Integer[matrix.length()][];  **return** minimumTotal(0,0,dp, matrix); } **private** **int** **minimumTotal**(**int** i, **int** j, Integer[][] dp, **int**[][] matrix){  **if**(i == n) **return** 0;  If (dp[i] == null){  dp[i] = new Integer[matrix[i].length];  }  **if**(dp[i][j] != **null**) **return** dp[i][j];   **int** left = minimumTotal(i+1, j, dp, matrix);  **int** right = minimumTotal(i+1, j+1, dp, matrix);  dp[i][j] = Math.min(left, right) + matrix[i][j];  **return** dp[i][j]; }  int[][] dp; // suggest all the memorization global  **public int minimumTotal**(List<List<Integer>> triangle) {  // recursion + dp：bottom up  dp = new int[triangle.size()][triangle.size()];  for (int[] i : dp) Arrays.fill(i, Integer.MAX\_VALUE);  return minTotalHelper(0, 0, triangle);  }  **private int minTotalHelper**(int i, int j, List<List<Integer>> triangle) {  int level = triangle.size();  // base case  if (i == level) {  return 0;  }  // check if the node has called  if (dp[i][j] != Integer.MAX\_VALUE) {  return dp[i][j];  }  int left = minTotalHelper(i + 1, j, triangle);  int right = minTotalHelper(i + 1, j + 1, triangle);  // restore the minimum value of sum to dp  dp[i][j] = triangle.get(i).get(j) + Math.min(left, right);  return dp[i][j];  } |
| --- |

1. 33. L53 The length of longest ascending subarray/substring/consecutive 1

// S1 DP with O(n) space

| **public** **int** **legnthOfLISA**(**int**[] nums){  **if**(num == **null** || nums.length == 0) **return** 0;  **int**[] dp = **new** **int**[nums.length];  dp[0] = 1;  **int** max = dp[0];*//max的赋值最好不要hard coding写一个具体的数进去*  **for** (**int** i = 1; i < nums.length; i++){  **if**(nums[i] > nums[i - 1]){  dp[i] = dp[i - 1] + 1;   max = Math.max(dp[i], max);  }**else**{  dp[i] = 1;  }  }  **return** max; } |
| --- |

//S2 DP with O(1)space

| **public** **int** **lengthOfLISA**(**int**[] nums){  *//cc*  **int** cur = 1;  **int** max = 1;  **for** (**int** i = 1; i < nums.length; i++){  **if**(nums[i] > nums[i-1]){  cur++;  max = Math.max(max, cur);  }**else**{  cur = 1;  }  }  **return** max; } |
| --- |

3.1 L300 length of longest increasing subsequence

//

1. 4**.L152** maximum product subarray

//dp

| **public** |
| --- |

1. //L300 max sum of subArray

| **public** |
| --- |

1. 5**. //**L55 Jump Game

//give [2,3,1,1,4], return true

//S1 recursion

//S2 DP 从后往前看, Leetcode超时

| **public** **boolean** **canJump**(**int**[] nums){  **if**(nums == **null** || nums.length == 0) **return** **true**  **boolean**[] dp = **new** **boolean**[nums.length];  dp[nums.length-1] = **true**;  **for** (**int** i = nums.length - 2; i >= 0; i--){  **for**(**int** j = 1; j <= nums[i]; j++){  **if**(dp[i+j] == **true**){  dp[i] = **true**;  **break**;  }**else**{  **return** **false**;  }   }  }  **return** dp[0]; } |
| --- |

//S3 greedy

| **public** **boolean** **canJump**(**int**[] array){  **if**(array == **null** || array.length == 0) **return** **true**;  **int** maxRange = array[0];  **for**(**int** i = 0; i < nums.length; i++){  **if**(maxRange < i) **return** **false**;  maxRange = Math.max(maxRange, array[i] + i);  **if**(maxRange >= nums.length-1) **return** **true**;  }  **return** **false**; } |
| --- |

1. 6**. Jump game II, L45**

// return the minimum steps

//S1 DP (Leetcode跑不过)

| **public** **int** **canJump**(**int** array){  **if**(array == **null** || array.length < 2) **return** 0;  **int** len = array.length;  **int**[] dp = **new** **int**[len];  dp[len-1] = 0;  **for**(**int** i = len-2; i >= 0; i--){  **int** minStep = Integer.MAX\_VALUE;  **for**(**int** j = 1; j <= array[i]; j++){  **if**(i + j < len){  minStep = Math.min(dp[i+j], minStep);  }  }  **if** (minStep == Integer.MAX\_VALUE) dp[i] = Integer.MAX\_VALUE;  **else** dp[i] = minStep+1;  }   **return** dp[0];  } |
| --- |

//Greedy

| **public** **int** **canJump**(**int**[] array){  **if**(array == **null** || array.length < 2) **return** 0;  **int** minStep = 0;  **int** prevRange = 0;  **int** curRange = 0;  **for**(**int** i = 0; i < array.length; i++){  **if**(curRange >= array.length - 1) **return** minStep + 1;  **if**(i > prevRange){  prevRange = curRange;  minStep++;  }  curRange = Math.max(curRange, i + array[i]);  }  **return** minStep; } |
| --- |

1. 7. Cut rope//cut stone GeeksforGeeks

//一根绳子，可以在任何位置下刀切，求切完之后绳子长度最大的乘积

| pubic |
| --- |

1. 8. Word break L 139

//s = "leetcode", dict = ["leet", "code"].

//Return true because "leetcode" can be segmented as "leet code".

//S1 没有带记忆化存储的话，leetcode会超时

| **public** **boolean** **wordBreak**(String s, List<Integer> wordList){  **if**(s == **null** || s.length() == 0) **return** **true**;  **if**(wordList == **null**) **throw** **new** IllegalArgumentException();  Set<String> set = **new** HashSet<>();  Boolean[] goodIndex = **new** Boolean[s.length+1];  **for**(String str : wordList){  set.add(str);  }  **return** wordBreak(s, set, 0,goodIndex); } **private** **boolean** **wordBreak**(String s, HashSet<String> set, **int** index, **boolean**[] goodIndex){  *//base case*  **if**(index == s.length()) **return** **true**;*//整个string都切完了，表示成功了。失败的case不 //在base case里面，在recursion body里面check*  **if**(goodIndex[index] != **null**) **return** goodIndex[index];   **for**(**int** i = index; i < s.length(); i++){  String word = s.substring(index, i+1);  **if**(set.contains(word)){*//分叉是分在当前的单词存在的情况下，如果后面的单词也return回来一个true的话，则当前层可return true了*  **if**(wordBreak(s, set, i+1, goodIndex){  goodIndex[index] = **true**;  **return** **true**;  }  }  }  goodIndex[index] = **false**;  **return** goodIndex[index]; } |
| --- |

//S2 DP, 不用recursion

| **public** **boolean** **wordBreak**(String s, List<Integer> wordList){ *//cc* Set<String> set = **new** HashSet<>(); **boolean**[] dp =  **for**(String str : wordList){ set.add(str); } **for**(**int** i = 0; i < s.length(); i++){  } } |
| --- |

1. 9. Word Break II L140

//

1. 10. Edit distance L72

//DP

| **public** **int** **minDistance**(String word1, String word2){  **if**(word1 == **null**) **return** word2.length();  **if**(word2 == **null**) **return** word1.length();   **int** len1 = word1.length();  **int** len2 = word2.length();  **int**[][] dp = **new** **int**[len1+1][len2+1];  **for**(**int** i = 0; i <= len1; i++){  dp[0][i] = i;  }  **for**(**int** i = 0; i <= len2; i++){  dp[i][0] = i;  }   **for**(**int** i = 1; i <= len1; i++){  **for**(**int** j = 1; j <= len2; j++){  **if**(word1.charAt(i-1) == word2.charAt(j-1)) dp[i][j] = dp[i-1][j-1];  **else** dp[i][j] = Math.min((Math.min(dp[i-1][j], dp[i-1][j-1]), dp[i][j-1])) + 1;  }  }  **return** dp[len1][len2]; } |
| --- |

1. 11. One Edit distance Facebook on site 题

//问s1到s2的edit distance是否exactly是1. for loop从左往右，遇到两个string中哪个值不 //一样的时候，check三种情况，只要满足其中任何一种，就可以return true

//代码未验证

| **public** **boolean** **isOneStep**(String word1, String word2){  *//cc*  **int** len1 = word1.length();  **int** len2 = word2.length();  **if**(Math.abs(len1 - len2) > 1) **return** **false**;  **if**(len1 < len2) **return** isOneStep(word2, word1);  **for**(**int** i = 0; i < len1; i++){  **if**(word1.charAt(i) != word2.cahrAt(i)){  **if**(! ( word1.substring(i+1)..equals(word2.substring(i+1)) || word1.substring(i).equals(word2.substring(i+1)) || word1.substring(i+1).equals(word2.substring(i)) )) **return** **false**;  }  }  **return** **true**; } |
| --- |

1. 13. L115 算法加强会讲
2. 14. L97 算法加强会讲
3. Q3, **The longest ascending subarray, return the length of this subarray**
4. 674. Longest Continuous Increasing Subsequence (that is 103)

| //DP with space O(1)  **public** **int** **longestAsceningSubArray**(**int**[] array){  **if**(array == **null** || array.length == 0) **return** 0;  **int** cur = 1;  **int** max = cur;  **for**(**int** i = 1; i < array.length; i++){  **if**(array[i] > array[i-1]){  cur++;  max = Math.max(cur, max);  }**else**{  cur = 1;  }  }  **return** cur; } |
| --- |

1. **Q4:**  L152, maximum product subarray

| public int maxProduct(int[] nums) {  */\*  由于负数的存在，使得在loop的过程中，需要保持两个数，一个是当前遇到的最大值，一个是当前遇到的最小值。因为如果下一个是负数的话，那么通过和  当前的最小值相乘，仍有机会成为最大值。而当前值为最大的话，和负数相乘，就会变成最小值。  1. 计算max[i]的时候有三个途径，一是如果max[i-1]为负数，当前nums[i]也为负数；二是两者都为正数；再者就是nums[i]自己本身是最大值  2. min[i]同样的思路  3. 在每次更新的过程中，还要更新result  \*/*  if(nums == null || nums.length == 0) return 0;  int[] max = new int[nums.length];  int[] min = new int[nums.length];  max[0] = nums[0];  min[0] = nums[0];  int res = max[0];  for(int i = 1; i < nums.length; i++) {  max[i] = Math.max(Math.max(max[i-1] \* nums[i], min[i-1]\*nums[i]), nums[i]);  min[i] = Math.min(Math.min(max[i-1] \* nums[i], min[i-1]\* nums[i]), nums[i]);  res = Math.max(res, max[i]);  }  return res;  } |
| --- |

1. Q5 L53 Maximum sum of subarray

| **public** **int** **maxSumOfSubArray**(**int**[] array){  **if**(array == **null** || array.length == 0) **throw** **new** IllegalArgumentException();  **int**[] dp = **new** **int**[array.length];  dp[0] = array[0];  **int** max = dp[0];  **for**(**int** i = 1; i < array.length; i++){  **if**(dp[i-1] > 0){  dp[i] = dp[i-1] + array[i];  }**else**{  dp[i] = array[i];  }  max = Math.max(max, dp[i]);  */\*这里还可以这样写  dp[i] = Math.max(array[i], dp[i-1] + array[i]);  max = Math.max(max, dp[i]);  \*/*  }  **return** max; } |
| --- |

//O(1) space

| **public** **int** **maxSumOfSubArray**(**int**[] array){  **if**(array == **null** || array.length == 0) **throw** **new** IllegalArgumentException();  **int** cur = array[0];  **int** max = cur;  **for**(**int** i = 1; i < array.length; i++){  **if**(cur > 0) {  cur += array[i];  }**else**{  cur = array[i];  }  max = Math.max(cur,max);  */\*  cur = Math.max(cur+array[i], array[i]);  max = Math.max(cur, max);  \*/*  }  **return** max; } |
| --- |

1. Q6 L55 Jump Game

| *//S1 recursion* **public** **boolean** **canJump**(**int**[] array){  **if**(array == **null** || array.length < 2) **return** **true**;  **return** canJump(array, 0); } **private** **boolean** **canJump**(**int**[] array, **int** step){  **if**(step > array.length - 1) **return** **true**;  **for**(**int** i = 1; i < array[index]; i++){  **if**(canJump(array, index+i)) **return** **true**;  }  **return** **false**; } |
| --- |

| *//DP* **public** **boolean** **canJump**(**int**[] array){  *//cc*  **int** len = array.length;  **int**[] dp = **int**[len];  dp[len-1] = **true**;  **for**(**int** i = len-2; i >= 0; i--){  **for**(**int** j = 1; j <= array[i]; j++){  **if**(i+j >= len-1 || dp[i+j]) {  dp[i] = **true**;  **break**;  }  }  }  **return** dp[0]; } |
| --- |

| *//Greedy* **public** **boolean** **canJump**(**int**[] array){  *//cc*  **int** len = array.length;  **int** max = 0;  **for**(**int** i = 0; i < len-1; i++){  **if**(max < i) **return** **false**;  max = Math.max(i + array[i], max);  **if**(max >= len-1) **return** **true**;  }  **return** **false**; } |
| --- |

1. Q7 L45 Jump Game II

| *//recursion,时间复杂度K^n, k为每个点能跳的最远步数，n为array中点的个数*  **public** **int** **minJumpStep**(**int**[] array){  **if**(array == **null** || array.length < 2) **return** 0;  **int**[] minStep = **new** **int**[1];  minJumpStep(array, 0, 0, minStep);   **return** minStep[0]; } **private** **void** **minJumStep**(**int**[] array, **int** index, **int** step, **int**[] minStep){   **if**(index >= array.length-1){  **if**(step < minStep[0]){  minStep[0] = step;  }  **return**;  }  **for**(**int** i = 1; i <= array[index]; i++){  minJumpStep(array, index+i, step+1, minStep);  } } |
| --- |

| *// DP,*  **public** **int** **minJumpStep**(**int**[] array){  *//cc*  **int** len = array.length;  **int**[] dp = **new** **int**[len];  dp[len-1] = 1;  **for**(**int** i = len-1; i >= 0; i--){  **int** min = Integer.MAX\_VALUE;  **for**(**int** j = 1; j < array[i]; j++){  **if**(i+j < len){  min = Math.min(min, dp[i+j]);  }  }  **if**(min = Integer.MAX\_VALUE) dp[i] = Integer.MAX\_VALUE;  **else** dp[i] = min + 1;  }  **return** dp[0]; } |
| --- |

| *///Greedy* **public** **int** **minJumpStep**(**int**[] nums){  *//cc*  if (nums.length < 2) return 0;  **int** len = nums.length;  **int** res = 0;  **int** preMax = 0;  **int** curMax = 0;  **for**(**int** i = 0; i < len; i++){  **if**(i > preMax){  preMax = curMax;  res++;  }  curMax = Math.max(curMax, i + nums[i]);  **if**(curMax >= len - 1) **return** res + 1;  }  **return** res; } |
| --- |

1. Q8 Cut Rope/cut stone

| */\* Geeksforgeeks Set 36 Given a rope of length n meters, cut the rope in different parts of integer lengths in a way that maximizes product of lengths of all parts. You must make at least one cut. Assume that the length of rope is more than 2 meters. \*/* |
| --- |

| *//Recursion* */\* 因为至少要切一刀，i\*(n-i)表示在i的位置切一刀的时候，所能产生的乘积。i\*maxProd(n-i)表示有n米长的绳子在i处切一刀，右边的部分切下去任意刀所能产生的最大的值。recursion是一个从大到小的过程。所以会有重复的计算。 \*/* **public** **int** **maxProd**(**int** n){  **if**(n ==0 || n ==1) **return** 0;  **int** max = Integer.MIN\_VALUE;  **for**(**int** i = 1; i <= n; i++){  **int** curMax = Math.max(i\*(n-i), i\*maxProd(n-i));  max = Math.max(curMax, max);  }  **return** max; } |
| --- |

| *//DP* */\* dp[i]的定义是长度为i米的绳子最多能产生的最大的product。dp是一个从小到大的过程，可以重复使用历史上计算过的值 \*/* **public** **int** **maxProd**(**int** n){  **if**(n == 0 || n == 1) **return** 0;  **int**[] dp = **new** **int**[n+1];  dp[0] = 0;  dp[1] = 0;  **for**(**int** i = 1; i <= n; i++){  **int** curMax = 0;  **for**(**int** j = 1; j < i/2; j++){  curMax = Math.max(curMax, Math.max(j\*(i-j), j\*dp[i-j]));  }  dp[i] = curMax;  }  **return** max; } |
| --- |

| *//Q8.1 LT 700 cutting rod* /\*  \*/  **public** **int** **cutting**(**int**[] prices, **int** n) {  */\* Write your code here*    \*/  **if**(n == 0) **return** 0;  **if**(n == 1) **return** prices[0];  **int**[] dp = **new** **int**[n+1];  dp[0] = 0;  **for**(**int** i = 1; i <= n; i++) {  *//int max = Integer.MIN\_VALUE;*  **for** (**int** j = 0; j < i; j++) {  dp[i] = Math.max(dp[i], dp[j] + prices[i - j - 1]);  }  *//dp[i] = max;*  }  **return** dp[n];  } |
| --- |

1. Q9 LintCode, Merger Stone

| */\* LT 476. 沙子归并问题 There is a stone game. At the beginning of the game the player picks n piles of stones in a line. The goal is to merge the stones in one pile observing the following rules: 1. At each step of the game,the player can merge two adjacent piles to a new pile. 2. The score is the number of stones in the new pile. You are to determine the minimum of the total score. [4,1,1,4] score = 18 \*/* |
| --- |

public class Score{

public int mergeStone(int[] array){

if(array == null || array.length == 0) return 0;

int len = array.length;

int[] dp = new int[len];

dp[0] = array[0];

for(int i = 1; i < len; i++){

int minSum = Integer.MAX\_VALUE;

for()

}

}

}

1. Q10 L140 Word Break. Top 6 高频题, 微软Onsite原题

| */\* recursion without memorization, 时间复杂度O(2^n)，最差的情况是每个单词总是能命中，但是在最后的时候，总是失败。这样的话，每个字母都需要分叉，然后往下继续做。这样的话，每个字母的地方都会分两个叉，类似一个二叉树，树的高度是string的长度n，所以worst case的时候，时间复杂度是O(2^n)  \*/* **public** **boolean** **wordBreak**(String s, List<String> wordDict){  **if**(s == **null** || s.length() == 0) **return** **true**;  **if**(wordList == **null**) **throw** **new** IllegalArgumentException;  Set<String> dict = **new** HashSet<>();  **for**(String str : wordList){  dict.add(str);  }  **return** helper(s, 0, dict); } **private** **boolean** **helper**(String s, **int** index, Set<String> dict){  **if**(index == s.length()){  **return** **true**;  }  **for**(**int** i = index; i < s.length(); i++){  String temp = s.subString(index, i+1);  **if**(dict.contains(temp)){  **if**(helper(s, i+1, dict)) **return** **true**;  }  }  **return** **false**; } |
| --- |

| *//recursion with memorization* **public** **boolean** **wordBreak**(String s, List<String> wordDict){  **if**(s == **null** || s.length() == 0) **return** **true**;  **if**(wordList == **null**) **throw** **new** IllegalArgumentException;  Set<String> dict = **new** HashSet<>();  **for**(String str : wordList){  dict.add(str);  }  Boolean[] visited = **new** Boolean[s.length+1];  **return** helper(s, 0, dict, visited); } **private** **boolean** **helper**(String s, **int** index, HashSet<String> dict, Boolean[] visited){  **if**(index == s.length) **return** **true**;  **if**(visited[index] != **null**) **return** **true**;  **for**(**int** i = index, i <= s.length; i++){  String word = s.subString(index, i+1);  **if**(dict.contains(word)){  **if**(helper(s,i+1, dict, visited)){  visited[index] = **true**;  **return** **true**;  }  }  }  visiteded[index] = **false**;  **return** visited[index]; } |
| --- |

| ***// DP*** **public** **boolean** **wordBreak**(String s, List<String> wordDict){  **if**(s == **null** || s.length() == 0) **return** **true**;  **if**(wordList == **null**) **throw** **new** IllegalArgumentException;  Set<String> dict = **new** HashSet<>();  **for**(String str : wordList){  dict.add(str);  }  **int**[] dp = **new** **int**[s.length+1];  dp[0] = **true**;  **for**(**int** i = 1; i <= s.length; i++){  **for**(**int** j = 0; j <= i; j++){  **if**(dp[j] && dict.contains(s.subString(j, i))){  dp[i] = **true**;  **break**;  }  }  }  **return** dp[s.length] } |
| --- |

1. Q11 L72 Edit distance. Two Sequence

| *//space O(m\*n)* **public** **int** **editDistance**(String s1, String s2){  **if**(s1==**null** || s2 == **null**) **throw** **new** IllegalArgumentException();  **if**(s1.length == 0) **return** s2.length();  **if**(s2.length == 0) **return** s2.length();  **int** len1 = s1.length();  **int** len2 = s2.length();  **int**[][] dp = **new** **int**[len1+1][len2+1];  dp[0][0] = 0;  **for**(**int** i = 1; i <= len1; i++){  dp[i][0] = i;  }  **for**(**int** i = 1; i < len2; i++){  dp[0][i] = i;  }   **for**(**int** i = 1; i <= len1; i++){  **for**(**int** j = 1; j< len2; j++){  **if**(s1.charAt(i-1) == s2.charAt(j-1)) dp[i][j] = dp[i-1][j-1];  **else**{  dp[i][j] = Math.min(dp[i-1][j-1], Math.min(dp[i-1][j],dp[i][j-1]))+1;  }  }  }  **return** dp[len1][len2]; } |
| --- |

| *//Space O(m+n)* |
| --- |

1. L214, Q12, 一个单词，变成palindrome的最小的步数

| **public** |
| --- |

1. L161 One distance (facebook题)

| **public** |
| --- |

1. Q13，记录operation sequence (walmart 电面题)

| **public** |
| --- |

1. L115, Distinct Subsequences
2. L97

# **Session 12 Recursion and Divide Conquer**

1. 45. L51, L52 N queen



| *//The n-queens puzzle is the problem of placing n queens on an n×n chessboard such //that //no two queens attack each other.* **class** **Solution**{   **public** List<List<String>> solveNQuenes(**int** n){  **if**(n <= 0) **throw** **new** IllegalArgumentException();  List<List<String>> res = **new** ArrayList<>();  List<String> path = **new** ArrayList<String>();  solveNQueens(res, path, n, 0);  **return** res;  }    **private** **void** **solveNQueens**(List<List<String>> res, List<String> path, **int** n, **int** level){  **if**(path.size() == n) {  List<String> newPath = **new** ArrayList<>();  **for**(String str : path) {  newPath.add(str);  }  res.add(newPath);  **return**;  }   **for**(**int** i = 0; i < n; i++){  **char**[] chars = **new** **char**[n];  **for** (**int** j = 0; j < n; j++){  chars[j] = '.';  }  chars[i] = 'Q';  path.add(**new** String(chars));    **if**(isValid(path, level, i)){  solveNQueens(res, path, n, level+1);  }  path.remove(path.size() - 1);  }  }  *//isValid可以用DP优化*  **public** **boolean** **isValid**(List<String> path, **int** level, **int** col){  **if**(path == **null** || path.size() == 1) **return** **true**;  **for**(**int** i = 0; i < level; i++){  **if**(path.get(i).charAt(col) == 'Q') **return** **false**;  }  **for**(**int** i = 0; i < level; i++){  **int** prevQ =path.get(i).indexOf('Q');  **if**(Math.abs(prevQ - col) == level - i ) **return** **false**;  }  **return** **true**;  } } |
| --- |

1. 46. L36, L37 Sudoku Solver

//

1. 47. L54, L59 spiral print

// follow up:

// 逆时针打印

//顺逆交替

//m\*n的长方形

//input 个数为n，generate a matrix {{1,2,3},{8,9,4},{7,6,5}}

// input 为n, 螺旋打印123698745

1. L54 **Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.**

| *//m\*n* **public** List<Integer> **spiralOrder2**(**int**[][] matrix){  List<Integer> res = **new** ArrayList<>(); **if**(matrix == **null** || matrix.length == 0 || matrix[0] == **null** || matrix[0].length ==0) **return** res;   spiralOrder2(res, matrix, 0, matrix.length, matrix[0].length);  **return** res; } **private** **void** **spiralOrder2**(List<Integer> res, **int**[][] matrix, **int** offset, **int** rowSize, **int** colSize){  **if**(rowSize == 1){  **for**(**int** i = 0; i < colSize; i++ ){  res.add(matrix[offset][offset + i]);  }  **return**;  }  **if**(colSize == 1){  **for**(**int** i = 0; i < rowSize ; i++){  res.add(matrix[offset + i][offset]);  }  **return**;  }  **if**(rowSize == 0 || colSize == 0){  **return**;  }  **for**(**int** i = 0; i < colSize - 1; i++){  res.add(matrix[offset][offset + i]);  }  **for**(**int** i = 0; i < rowSize - 1; i++){  res.add(matrix[offset + i][offset + colSize -1]);  }  **for**(**int** i = 0; i < colSize - 1; i++){  res.add(matrix[offset + rowSize - 1][offset + colSize - 1 - i]);  }  **for**(**int** i = 0; i < rowSize - 1; i++){  res.add(matrix[offset + rowSize - 1 - i][offset]);  }  spiralOrder2(res, matrix, offset + 1, rowSize - 2, colSize - 2); } |
| --- |

1. L59. generate a **matrix with input as n without extra space. n is the row and col length of this matrix**

| **//需要设定一个start的量**  **public** **int**[][] generateMatrix(**int** n) {  **int**[][] matrix = **new** **int**[n][n];  fillMatrix(matrix, 0, n, n, 1);  **return** matrix;  }  **private** **void** **fillMatrix**(**int**[][] matrix, **int** offset, **int** size, **int** n, **int** start) {  **if** (size == 0) **return**;  **if** (size == 1) {  matrix[offset][offset] = n \* n;  **return**;  }  **for** (**int** i = 0; i < size - 1; i++) {  matrix[offset][offset + i] = start + i;  }  **for** (**int** i = 0; i < size - 1; i++) {  matrix[offset + i][offset + size - 1] = start + size - 1 + i;  }  **for** (**int** i = 0; i < size - 1; i++) {  matrix[offset + size - 1][offset + size - 1 - i] = start + 2 \* (size - 1) + i;  }  **for** (**int** i = 0; i < size - 1; i++) {  matrix[offset + size - 1 - i][offset] = start + 3 \* (size - 1) + i;  }  start = start + 4 \* (size - 1);  fillMatrix(matrix, offset + 1, size - 2, n, start);  } |
| --- |

1. **count the numbers of nodes in left subtree, and count complete tree nodes**

| *//get the subTree nodes numbers in O(1)* *//add a field the class of TreeNode* **public** **class** **Solution**{  **class** **TreeNode**{  **int** val;   **int** leftSubtreeNodes;  TreeNode left;  TreeNode right;  **public** **TreeNode**(**int** val){  **this**.val = val;  leftSubtreeNodes = 0;  left = **null**;  right = **null**;  }  }  **public** **int** **getLeftSubtreeNodes**(TreeNode root){  **if**(root == **null**) **return** 0;   **int** left = getLeftSubtreeNodes(root.left);  **int** right = getLeftSubtreeNodes(root.right);  root.leftSubtreeNodes = left;  **return** left + right + 1;  } } |
| --- |

1. **find a node with max difference between left subtree and right subtree**

| **public** TreeNode **maxDiffInSubtree**(TreeNode root){  **if**(root == **null**) **return** **null**;  **int**[] max = Integer.MIN\_VALUE;  TreeNode[] maxDiff = **new** TreeNode[1];  maxDiffInSubtree(root, max, maxDiff);  **return** maxDiff[0];  } **private** **int** **maxDiffInSubtree**(TreeNode root, **int**[] max,TreeNode[] maxDiff){  **if**(root == **null**) **return** 0;  **int** left = maxDiffInSubtree(root.left);  **int** right = maxDiffInSubtree(root.right);  **if**(Math.abs(left - right) > max[0]){  max[0] = Math.abs(left - right);  maxDiff[0] = root;  }  **return** left + right + 1; } |
| --- |

L236 Q14 LCA的问题

//14.1 在BST中找

//14.2 LCA with parent, no left and right

//14.3 二叉树，求k个node的LCA

//14.4 k叉树，找2个node的LCA

//14.5 at most k叉树，找m个node的LCA

1. **// Q14 LCA in BT**

| **public** TreeNode **lowestCommonAncestor**(TreeNode root, TreeNode p, TreeNode q) {  **if**(root == **null**) **return** **null**;  **if**(root == p || root == q) **return** root;   TreeNode left = lowestCommonAncestor(root.left, p, q);   TreeNode right = lowestCommonAncestor(root.right,p,q);  **if**(left != **null** && right != **null**) **return** root;   **return** left == **null** ? right : left;   } |
| --- |

1. **//** Q14.1 **L235, LCA in BST**(通过比较值的大小，可以提前剪枝)

| **public** TreeNode **lowestCommonAncestor**(TreeNode root, TreeNode p, TreeNode q) {  **if**(root == **null**) **return** **null**;  **if**(root == p || root == q) **return** root;  TreeNode left = **null**;  TreeNode right = **null**;  **if**(root.val > p.val || root.val > q.val){  left = lowestCommonAncestor(root.left, p, q);  }  **if**(root.val < p.val || root.val <q.val ){  right = lowestCommonAncestor(root.right, p, q);  }    **if**(left != **null** && right != **null**) **return** root;  **return** left == **null** ? right : left;  } |
| --- |

1. **LCA 点可能不在树上**, recursion必须要call到底才能return，因为即便找到一个点的话，还需要找到另一个点是否也在树上

| **public** **class** **Solution**{  **class** **Result**{  TreeNode node;  **int** count;  **public** **Result**(TreeNode node, **int** count){  **this**.node = node;  **this**.count = count;  }  }  **public** TreeNode **LCA**(TreeNode root, TreeNode p, TreeNode q){  **if**(root == **null**) **return** **null**;  Result res = LCARecursion(root, p, q);  **return** res.count == 2 ? res.node : **null**;  }  **private** TreeNode **LCARecursion**(TreeNode root, TreeNode q, TreeNode q){  **if**(root == **null**) **return** **new** Result(**null**, 0);   Result left = LCARecursion(root.left, p, q);  Result right = LCARecursion(root.right, p,q);  **if**(root == p || root == q){  **return** **new** Result(root, left.count + right.count + 1);  }**else** {  **if**(left.count > 0 && right.count>0){  **return** **new** Result(root, left.count + right.count);  }**else** **if**(left.count > 0){  **return** **new** Result(root, left.count);  }**else** **if**(right.count > 0){  **return** **new** Result(right.node, right.count);  }**else**{  **return** **new** Result(**null**, 0);  }  }  } } |
| --- |

1. **//** LCA 可能有target点不在tree上，而且还有可能有duplicate

| **public** |
| --- |

1. **2叉树，求k个node的LCA**

| **public** TreeNode **LCAofKNodes**(TreeNode root, Set<TreeNode> nodes){  **if**(root == **null**) **return** **null**;  **if**(nodes.contains(root)) **return** root;   TreeNode left = LCAofKNodes(root.left, nodes);  TreeNode right = LCAofKNodes(root.right, nodes);  **if**(left != **null** && right != **null**) **return** root;  **return** left == **null** ? right : left; } |
| --- |

1. **2**叉树，求k个node的LCA，有些node可能不在tree上的情况，如果有一个node不在tree上的话，就return null

| **public** **class** **Solution**{  **class** **Result**{  TreeNode node;  **int** count；  **public** **Result**(**int** count, TreeNode node){  **this**.node = node;  **this**.count = count;  }  }   **public** TreeNode **LCAOfK**(TreeNode root, Set<TreeNode> nodes){  **if**(root == **null**) **return** **null**;  Result res = LCAOfKRecursion(root, nodes);  **return** res.count == nodes.size() ? res.node : **null**;  }  **private** Result **LCAOfKRecursion**(TreeNode root, Set<TreeNode> nodes){  **if**(root == **null**){  **return** **new** Result(**null**, 0);  }  Result left = LCAOfKRecursion(root.left, nodes);  Result right = LCAOfKRecursion(root.right, nodes);   **if**(nodes.contains(root)){  **return** **new** Result(root, left.count+right.count+1);  }  **else**{  **if**(left.count > 0 && right.count > 0){  **return** **new** Result(root, left.count+right.count);  }**else** **if**(left.count > 0){  **return** **new** Result(left.node, left.count);  }**else** **if**(right.count > 0){  **return** **new** Result(right.node, right.count);  }**else**{  **return** **new** Result(**null**, 0);  }  }  } } |
| --- |

1. **k叉树，求2个node的LCA**

| public class Solution{  class TreeNode{  int val;  ArrayList<TreeNode> children;  public TreeNode(int val){  this.val = val;  children = new ArrayList<TreeNode>();  }  }  public TreeNode LCA(TreeNode root, TreeNode p, TreeNode q){  if(root == null) return null;  if(root == p || root == q) return root;  int count = (root == p || root == q) ? 1 : 0;  TreeNode lastFoundLCA = null;  for(TreeNode node : root.children){  TreeNode target = LCA(node, p, q);  if(target != null){  count ++;  lastFoundLCA = target;  }  if(count >= 2){  return root;  }  }  return lastFoundLCA;  } |
| --- |

1. **m**叉树，n个node的LCA

| **public static void main(){  int a = 0; }** |
| --- |

**L124 Binary Tree maximum Path Sum(any node to any node)**

//Q15.1 any leaf node to any leaf node

//Q15.2 any leaf root to root node

//Q15.3 any node to root node

//Q15.4 any node to any leaf

1. **Q15**.1 any leafnode to any leafnode

| **public** **int** **pathSum**(TreeNode root){  **if** (root == **null**) **return** Integer.MIN\_VALUE;  **int**[] max = **new** **int**[1];  max[0] = Integer.MIN\_VALUE;  pathSum(root, max);  **return** max[0]; } **private** **int** **pathSum**(TreeNode root, **int**[] max){  **if**(root == **null**) **return** Integer.MIN\_VALUE;  **int** left = pathSum(root.left, max);  **int** right = pathSum(root.right, max);  max[0] = Math.max(max[0], left+right+root.value);  **return** Math.max(left, right) + root.value; } |
| --- |

1. Q15.2 any leaf node to root, 找最大的那一条

| public int pathSum(TreeNode root){  if(root == null) return Integer.MIN\_VALUE;  int left = pathSum(root.left);  int right = pathSum(root.right);  return Math.max(left, right) + root.val; } |
| --- |

1. **Q15.3 any node to root node**

| public int pathSum(TreeNode root){  if(root == null) return Integer.MIN\_VALUE;  int left = pathSum(root.left);  int right = pathSum(root.right);  *//if(left< 0 && right <0) return root.val;*  return Math.max(0, Math.max(left, right)) + root.val; } |
| --- |

1. **//**终极boss：**Any node to any node**

//正确的解法

| **class** **Solution** {  **public** **int** **maxPathSum**(TreeNode root) {  **if** (root == **null**) **throw** **new** IllegalArgumentException();  **int**[] max = **new** **int**[1];  max[0] = Integer.MIN\_VALUE;  pathSum(root, max);  **return** max[0];   }    **private** **int** **pathSum**(TreeNode root, **int**[] max){  **if**(root == **null**) **return** 0;  **int** left = pathSum(root.left, max)   **int** right = pathSum(root.right, max);  max[0] = Math.max(max[0], Math.max(0, left) + Math.max(0, right) + root.val);  **return** Math.max(0, Math.max(0, Math.max(left, right)) + root.val);*//此处要注意的问题都是当左右两边有一边传上来的值是负数的时候*  } } |
| --- |

| *//自己写的错误的解法* **public** **int** **pathSum**(TreeNode root){  **if** (root == **null**) **return** Integer.MIN\_VALUE;  **int**[] max = **new** **int**[1];  max[0] = Integer.MIN\_VALUE;  pathSum(root, max);  **return** max[0]; } **private** **void** **pathSum**(TreeNode root, **int**[] max){  **if**(root == **null**) **return** Integer.MIN\_VALUE;  **int** left = pathSum(root.left, max);  **int** right = pathSum(root.right, max);  max[0] = Math.max(max[0], left+right+root.value);  **return** Math.max(0, Math.max(left, right)) + root.value; } |
| --- |

Q16 **Path Sum II, return path**

1. Q16.1 given a binary tree, find the maximum path sum from **root to leaf**, **print all path**

**// Donated @Letian Xu**

| **public** List<List<Integer>> pathSum(TreeNode root, **int** sum) {  List<List<Integer>> res = **new** ArrayList<>();  **if**(root == **null**) **return** res;  pathSum(root, sum, **new** ArrayList<>(), res);  **return** res; }  **private void pathSum(TreeNode root, int sum, List<Integer> cur, List<List<Integer>> res) {  if (root == null) {  return;  }   if (root.left == null && root.right == null) {  if (sum == root.val) {  cur.add(root.val);  res.add(new ArrayList<>(cur));  cur.remove(cur.size() - 1);  return;  }   }**  **cur.add(root.val);   pathSum(root.left, sum - root.val, cur, res);  pathSum(root.right, sum - root.val, cur, res);  cur.remove(cur.size() - 1); }** |
| --- |

// Lotte Xu comment：这是path sum to target, 不是maximum path，答案和题意不一样

| **class** **Solution** {  **public** List<List<Integer>> pathSum(TreeNode root, **int** sum) {  List<List<Integer>> res = **new** ArrayList<>();  **if**(root == **null**) **return** res;  helper(root, sum, res, **new** ArrayList<>());  **return** res;  }    **private** **void** **helper**(TreeNode root, **int** sum, List<List<Integer>> res, List<Integer> path) {    **if**(root.left == **null** && root.right == **null**) {  **if**(sum - root.val == 0) {  path.add(root.val);  res.add(**new** ArrayList<>(path));  path.remove(path.size() - 1);  }  **return**;  }    path.add(root.val);  **if**(root.left != **null**) {  helper(root.left, sum - root.val, res, path);  }  path.remove(path.size() - 1);    path.add(root.val);  **if**(root.right != **null**) {  helper(root.right, sum - root.val, res, path);  }  path.remove(path.size() - 1);  } } |
| --- |

1. **//**Q16.2 given a binary tree, find the maximum path sum from **one leaf to root**, **print path**

先recursion再加

| public |
| --- |

1. **Q17 L113 Path sum II ：Path Sum to target: return path add up to target**

**Q17.1 find all root to leaf paths where each paths sum equals the given target**

| class Solution {  **public** List<List<Integer>> pathSum(TreeNode root, **int** sum) {  List<List<Integer>> res = **new** ArrayList<>();  **if** (root == **null**) **return** res;  List<Integer> path = **new** ArrayList<>();  path.add(root.val);  pathSum(root,sum,root.val,res,path);  **return** res; } **private** **void** **pathSum** (TreeNode root, **int** sum, **int** curSum, List<List<Integer>> res, List<Integer> path){ **if** (root.left == **null** && root.right == **null**){  **if** (curSum == sum){  res.add(**new** ArrayList<Integer> (path));  }  **return**;  }  *//left*  **if** (root.left != **null**){  path.add(root.left.val);  pathSum(root.left,sum,curSum + root.left.val, res, path);  path.remove(path.size() - 1);  }    *//right*  **if** (root.right != **null**){  path.add(root.right.val);  pathSum(root.right,sum,curSum + root.right.val, res, path);  path.remove(path.size() - 1);  }   } } |
| --- |

1. Q17.2 check if there is a root to leaf path where each path’s sum equals the given sum
2. Q17.3 find all **any to any** paths where each path’s sum equals to target(非常复杂)

| **public** List<List<Integer>> pathSum3(TreeNode root, **int** target){  List<List<Integer>> res = **new** ArrayList<>();  **if**(root == **null**) **return** res;  List<Integer> path = **new** ArrayList<>();  helper(res, path, root, target);  **return** res; } **private** **void** **helper**(List<List<Integer>> res, List<Integer> path, TreeNode root, **int** target){  **if**(root == **null**) **return** } |
| --- |

1. Q17.4 check if there is any to any path where each path’s sum equals to target

| **public** |
| --- |

1. Q18 Given a binary tree, find the maximum path sum from any to any, return the maximum sum path.

1. L297 Binary tree + binary search tree (**一定要掌握**)

| */\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/* **public** **class** **Codec** {  **public** String **serialize**(TreeNode root) {  StringBuilder sb = **new** StringBuilder();  preOrderDFS(root, sb);  sb.setLength(sb.length() - 1);  **return** sb.toString();  }  **private** **void** **preOrderDFS**(TreeNode root, StringBuilder sb){  **if**(root == **null**){  sb.append("null,");  **return**;  }  sb.append(root.val);  sb.append(',');  preOrderDFS(root.left, sb);  preOrderDFS(root.right, sb);  }  *// Decodes your encoded data to tree.*  **public** TreeNode **deserialize**(String data) {  TreeNode root = **null**;  String[] data\_array = data.split(",");  **return** buildTree(root, data\_array, **new** **int**[1]);  }  **private** TreeNode **buildTree**(TreeNode root, String[] data\_array, **int**[] index){  **if**(data\_array[index[0]].equals("null")){  index[0]++;  root = **null**;  **return** root;  }  root = **new** TreeNode(Integer.valueOf(data\_array[index[0]]));  index[0]++;  root.left = buildTree(**null**, data\_array, index);  root.right = buildTree(**null**, data\_array, index);  **return** root;  } } |
| --- |

1. LT 378 Binary Tree to Doubly Linked List

| *//Convert a binary search tree to doubly linked list within-order traversal.要求输出doublyListNode* **public** **class** **Solution** {  */\*  \* @param root: The root of tree  \* @return: the head of doubly list node  \*/*    DoublyListNode dummy = **new** DoublyListNode(0);  DoublyListNode cur = dummy;  DoublyListNode tail = dummy;    **public** DoublyListNode **bstToDoublyList**(TreeNode root) {  *// write your code here*  **if**(root == **null**) **return** **null**;  dfs(root);  **return** dummy.next;  }    **private** **void** **dfs**(TreeNode root) {  **if**(root == **null**) **return**;    dfs(root.left);    tail = **new** DoublyListNode(root.val);  cur.next = tail;  tail.prev = cur;  cur = tail;    dfs(root.right);   } } |
| --- |

1. **given a binary tree, sequence it to a doubly LinkedList with in/post/pre order sequence**

| *//amazon and facebook onsite* *//S1 in order*  TreeNode head;  TreeNode tail; **public** TreeNode **flatten**(TreeNode root){  **if**(root == **null**) **return** **null**;  TreeNode[] prev = **new** TreeNode[1];  helper(root, prev);  head.left = tail;*//需要circular的时候*  tail.right= head;*//需要circular的时候*  **return** head; }  **private** void **helper**(TreeNode root, TreeNode[] prev){  **if**(root == **null**) **return**;  helper(root.left, prev);   **if**(prev[0] == **null**) {  head = root;   }else{  root.left = prev[0];  prev[0].right = root;  }  *//tail = root;//如果需要circular的话*  prev[0] = root;   helper(root.right, prev); } |
| --- |

//S2 preOrder

public class solution{

//代码存档的解法，不推荐

public TreeNode BinaryTreeToDDL(TreeNode root){

if(root == null) return null;

TreeNode head = BinaryTreeToDDL(root, null);

return head;

}

private TreeNode BinaryTreeToDDL(TreeNode root, TreeNode tail){

if(root == null) return tail;

TreeNode right = BinaryTreeToDDL(root.left, BinaryTreeToDDL(root.right, tail));

root.right = right;

if(right != null) right.left = root;

return root;

}

//自己的解法

private TreeNode prev;

private TreeNode head;

public TreeNode BinaryTreeToDDL(TreeNode root){

if(root == null) return null;

head = BinaryTreeToDDL(root);

return head;

}

private void BinaryTreeToDDL(TreeNode root){

if(root == null) return;

TreeNode right = root.right;

if(prev != null){

prev.right = root;

}else{

head = root;

}

root.left = prev;

prev = root;

BinaryTreeToDDL(root.left);

BinaryTreeToDDL(right);

}

}

1. GeeksforGeeks convert binary to circular LinkedList
2. Q20 L114 Flatten Binary Tree to LinkedList

//Given a binary tree, flatten it to a linked list in-place

| **//先走到最右边**  **public** **void** **flatten**(TreeNode root) {  **if**(root == **null**) **return**;  TreeNode[] pre = **new** TreeNode[1];  helper(root, pre);  }    **private** **void** **helper**(TreeNode root, TreeNode[] pre){  **if**(root == **null**) **return**;  helper(root.right, pre);  helper(root.left, pre);  root.right = pre[0];  root.left = **null**;  pre[0] = root;  } |
| --- |

| **class** **Solution** {  **public** **void** **flatten**(TreeNode root) {  **if**(root == **null**) **return**;  TreeNode[] prev = **new** TreeNode[1];  flatten(root, prev);  **return**;  }  **private** **void** **flatten**(TreeNode root, TreeNode[] prev){  **if**(root == **null**) **return**;  TreeNode right = root.right;*//因为后面prev[0] = root, 随后prev[0].right通过deference的方式修改了root.right，所以要将right存下来*  **if**(prev[0] != **null**){  prev[0].right = root;  prev[0].left = **null**;  }   prev[0] = root;  flatten(root.left, prev);  flatten(right, prev);    } } |
| --- |

Q21 **Construct a tree from inorder and xxx-order sequence**

LC105 preorder + inorder

| **class** **Solution** {  **public** TreeNode **buildTree**(**int**[] preorder, **int**[] inorder) {  **if**(preorder == **null** || inorder == **null**) **return** **null**;  HashMap<Integer, Integer> map = **new** HashMap<>();  **int** len = preorder.length;  **for**(**int** i = 0; i < inorder.length; i++){  map.put(inorder[i], i);  }  TreeNode root = buildTree(preorder, 0, len - 1, inorder, 0, len-1, map);  **return** root;  }  **private** TreeNode **buildTree**(**int**[] preorder, **int** preStart, **int** preEnd, **int**[] inorder, **int** inStart, **int** inEnd, HashMap<Integer, Integer> map){  **if**(preStart > preEnd) **return** **null**;*//此处只能是大于，如果是>=的话，在只有一个node的时候，就接不上去，直接return null了*  **int** rootVal = preorder[preStart];  **int** rootIdx = map.get(rootVal);  TreeNode root = **new** TreeNode(rootVal);  root.left = buildTree(preorder, preStart+1,rootIdx - inStart + preStart, inorder, inStart, rootIdx-1, map);  root.right = buildTree(preorder, rootIdx + preStart + 1 - inStart, preEnd, inorder, rootIdx + 1, inEnd, map);  **return** root;  } } |
| --- |

1. L106 postorder + inorder

| **class** **Solution** {  **public** TreeNode **buildTree**(**int**[] inorder, **int**[] postorder) {  **if**(inorder == **null** || postorder == **null** || inorder.length != postorder.length) **return** **null**;  HashMap<Integer, Integer> map = **new** HashMap<>();  **for**(**int** i = 0; i < inorder.length; i++){  map.put(inorder[i], i);  }  TreeNode root = helper(inorder, 0, inorder.length-1, postorder, 0, postorder.length-1, map);  **return** root;  }  **private** TreeNode **helper**(**int**[] inorder, **int** inStart, **int** inEnd, **int**[] postorder, **int** postStart, **int** postEnd, HashMap<Integer, Integer> map){  **if**(inStart > inEnd) **return** **null**;*//要严格大于才行，不然只有一个node 的时候，就return null了*  **int** rootVal = postorder[postEnd];  TreeNode root = **new** TreeNode(rootVal);  **int** rootIdx = map.get(rootVal);  root.left = helper(inorder, inStart, rootIdx - 1, postorder, postStart, rootIdx - inStart + postStart - 1, map);  root.right = helper(inorder, rootIdx + 1, inEnd, postorder, rootIdx - inEnd + postEnd, postEnd - 1, map);  **return** root;  } } |
| --- |

1. Level order + inorder

| **public** |
| --- |

Q22 L108 L109 Convert sorted array/List to binary search tree

//结果不唯一。

1. L108 Given an array where elements are sorted in ascending order, convert it to a height balanced BST.如果要高度最小的话，每次都去最中间的。 时间复杂度1+2+4+8+...+2^logn = n

| **class** **Solution** {  **public** TreeNode **sortedArrayToBST**(**int**[] nums) {  **if**(nums == **null** || nums.length == 0) **return** **null**;  **int** len = nums.length;  TreeNode root = helper(nums, 0, len-1);  **return** root;  }  private TreeNode helper(int[] array, int start, int end){  if(start > end) return null;  int mid = start + (end - start)/2;  int rootVal = array[mid];  TreeNode root = new TreeNode(rootVal);  root.left = helper(array, start, mid-1);  root.right = helper(array, mid+1, end);  return root;  } } |
| --- |

1. L109 Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

| *//primitive的做法,关键是找到中间的点* **class** **Solution** {  **public** TreeNode **sortedListToBST**(ListNode head) {  **if**(head == **null**) **return** **null**;  TreeNode root = helper(head, **null**);  **return** root;  }  **private** TreeNode **helper**(ListNode head, ListNode tail){  **if**(head == tail) **return** **null**;  ListNode mid = findMid(head, tail);  TreeNode root = **new** TreeNode(mid.val);  root.left = helper(head, mid);*//此处mid作为tail传进去，但是在base case中，即便最后留只剩下mid一个node，也不会接到tree上去，而是return一个null出来*  root.right = helper(mid.next, tail);  **return** root;  }  **private** ListNode **findMid**(ListNode head, ListNode tail){  ListNode slow = head;  ListNode fast = head;  **while**(fast != tail && fast.next != tail){  slow = slow.next;  fast = fast.next.next;  }  **return** slow;  }  } |
| --- |

| *//HashMap的做法* **public** TreeNode **sortedListToBST**(ListNode head){  **if**(head == **null**) **return** **null**;  **if**(head.next == **null**) **return** **new** TreeNode(head.val);  HashMap<Integer, ListNode> map = **new** HashMap<Integer, ListNode>();  **int** count = 0;  ListNode cur = head;  **while**(cur != **null**){  map.put(count, cur);  count++;  cur = cur.next;  }  TreeNode root = helper(0, count-1, map);  **return** root; } **private** TreeNode **helper**(**int** start, **int** end, HashMap<Integer, ListNode> map){  **if**(start > end) **return** **null**;  **int** mid = start + (end - start)/2;  TreeNode root = **new** TreeNode(map.get(mid).val);  root.left = helper(start, mid-1, map);  root.right = helper(mid+1, end, map);  **return** root; } |
| --- |

# **Session 13: Dynamic Programming II**

1. **Q1 L64 Minimum Path Sum**

| *／\**  *Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.* *Note: You can only move either down or right at any point in time.*  *单值问题，暗示可以考虑dp*  *\*／*  **public** **class** **MinPathSum**{  **public** **int** **minPathSum**(**int**[][] grid){  *//cc*  int rows = grid.length;  **int** cols = grid[0].length;  **int**[][] dp = **new** **int**[rows][cols];  dp[0][0] = grid[0][0];  **for**(**int** i = 1; i < cols; i++){  dp[0][i] = dp[0][i-1]+grid[0][i];  }  **for**(**int** i = 1; i < rows; i++){  dp[i][0] = dp[i-1][0]+grid[i][0];  }  **for**(**int** i = 1; i < rows; i++){  **for**(**int** j = 1; j < cols; j++){  dp[i][j] = Math.min(dp[i-1][j], dp[i][j-1]) + grid[i][j];  }  }  **return** dp[rows-1][cols-1];  } } |
| --- |

1. **Q2 L62 Unique Path**

| **class** **Solution** {  **public** **int** **uniquePaths**(**int** m, **int** n) {  *//cc*  **int**[][] dp = **new** **int**[m][n];  **for**(**int** i = 0; i < m; i++){  dp[i][0] = 1;  }  **for**(**int** i = 0; i< n; i++){  dp[0][i] = 1;  }  **for**(**int** i = 1; i < m; i++){  **for**(**int** j = 1; j < n; j++){  dp[i][j] = dp[i-1][j] + dp[i][j-1];  }  }  **return** dp[m-1][n-1];  } } |
| --- |

1. **Q2 L221 Given a matrix with only 0 and 1, find the largest square of 1’s, return area or** size

| **public** **int** **maximalSquare**(**int**[][] matrix) { *//cc*  **if** (matrix == **null** || matrix.length == 0 || matrix[0] == **null** || matrix[0].length == 0) **return** 0;  **int** rows = matrix.length;  **int** cols = matrix[0].length;  **int** max = matrix[0][0];  **int**[][] dp = **new** **int**[rows][cols];  dp[0][0] = matrix[0][0];  **for** (**int** i = 1; i < rows; i++) {  dp[i][0] = matrix[i][0];  max = Math.max(max, dp[i][0]);  }  **for** (**int** i = 1; i < cols; i++) {  dp[0][i] = matrix[0][i];  max = Math.max(max, dp[0][i]);  }   **for** (**int** i = 1; i < rows; i++) {  **for** (**int** j = 1; j < cols; j++) {  **if** (matrix[i][j] == 0) {  dp[i][j] = 0;  } **else** {  dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1])) + 1;  }  max = Math.max(max, dp[i][j]);  }  }  **return** max; } |
| --- |

1. 空心正方形

//需要preprocessing一下，准备好四个方向的dp，记录这个方向上最长的1

| **public** **int** **maxSquareSurrounding**(**int**[][] matrix){ *//cc* **int** row } |
| --- |

1. Q4 Given binary matrix, the largest + with all 1／largest X with all 1

//Q4.1 largest +

preprocessing 4个方向的连续1的数量，然后再填dp的表格的时候，取四个表中的最小的那个值。再过程中不断更新max的值。

//Q4.2 largest X

1. Q5 Given a matrix, largest sum of submatrix. 面试高频题

| **public** |
| --- |

1. 其他DP的题目，在算法加强中会cover

Q6 L95, L96, L91, L198, L213, L337, L132, L115, L97

# Session 15: String and Array II

1. Q1. L75 SortColor

/\*

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue. Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

\*/

| **class** **Solution** {  **public** **void** **sortColors**(**int**[] nums) {  **if**(nums == **null** || nums.length == 0) **return**;  **int** i = 0;   **int** j = 0;  **int** k = nums.length-1;  **while**(j <= k){  **if**(nums[j] == 0){  swap(nums, i, j);  i++;  j++;  }**else** **if**(nums[j] == 1){  j++;  }**else**{  swap(nums, j, k);  k--;  }  }  **return**;  }    **private** **void** **swap**(**int**[] nums, **int** x, **int** y){  **int** temp = nums[x];  nums[x] = nums[y];  nums[y] = temp;  } } |
| --- |

1. Q1 Given sorted/adjacent array(String), deduplicate with one remaining

// 如果是sorted，可以用stack去做

| **public** String **deduplicateWithOneLeft**(String s){  **if**(s == **null** || s.length() == 0) **return** **null**;  **char**[] array = s.toCharArray();  Stack<Character> stack = **new** Stack<Character>();  StringBuilder sb = **new** StringBuilder();  **for**(**int** i = 0; i < array.length; i++){  **if**(!stack.isEmpty()){  **if**(array[i] == stack.peek()) **continue**;  **else**{  stack.push(array[i]);  }  }**else**{  stack.push(array[i]);  }  }  **while**(!stack.isEmpty()){  sb.insert(0, stack.pop());  }  **return** sb.toString(); } |
| --- |

| *//如果是unsorted，而且不允许sort的话，可以采用hashmap+slow/fast指针* **public** String **dedupWithOneLeft**(String s){  **char**[] array = s.toCharArray();  HashMap<Character, Integer> map = **new** HashMap<Character, Integer>();  **int** slow = 0;  **for**(**int** fast = 0; fast < array.length; fast++){  **if**(!map.containsKey(array[fast])){  map.put(array[fast], 1);  array[slow++] = array[fast];  }  }  **return** **new** String(array, 0, slow); } |
| --- |

1. Q2 given sorted/adjacent array, deduplicate with 0 remained

| *//S1 hashmap, 但是不能保持原来的顺序* **public** **int**[] dedupWithZeroLeft(**int**[] array){  *//cc*  HashMap<Integer, Integer> map = **new** HashMap<Integer, Integer>();  **int** count = 0;  **List<integer> res = new ArrayList<Integer>()**;  **for**(**int** i = 0; i < array.length; i++){  **if**(map.containsKey(array[i])){  count = map.get(array[i])+1;  map.put(array[i], count)；  }**else**{  map.put(array[i], 1);  }  }  for(int i = 0; i < array.length; i++){  if(map.get(array[i]) == 1){  res.add(array[i]);  }  }  **return** res.toArray(); } |
| --- |

| *//S2 slow fast + FLAG, 返回最后array的长度(一定要写)* **public** String **removeDuplicate**(String s) {  **if** (s == **null** || s.length() <= 1) {  **return** s;  }   **int** len = s.length();  **char**[] sc = s.toCharArray();   *// slow is length*  **int** slow = 0;  **int** fast = 1;  **boolean** flag = **false**;  **while**(fast < len) {  **if**(sc[fast] != sc[fast - 1]) {  **if**(flag == true) {  flag = **false**;  } **else** {  sc[slow] = sc[fast - 1];  slow++;  }  } **else** {  flag = **true**;  }  fast++;  }  **if** (sc[len - 1] != sc[len - 2]) {  sc[slow] = sc[len - 1];  slow++;  }   **return** **new** String(sc).substring(0, slow);  } |
| --- |

| *//S3: Stack + flag, 最后一定要做一个post processing* **public** **int**[] removeDupWithZeroLeft(**int**[] array){  *//cc*  **boolean** flag = **false**;  Stack<Integer> stack = **new** Stack<Integer>();  stack.push(array[0]);  **for**(**int** i = 1; i < array.length; i++){  **if**(array[i] == stack.peek()){  flag = **true**;  //**continue**;  }**else**{  **if**(flag == **true**){  stack.pop();  stack.push(array[i]);  flag = **false**;  }**else**{  stack.push(array[i]);  }  }  }  if(flag == true) stack.pop();  **int**[] res = **new** **int**[stack.size()];  **int** i = stack.size()-1;  **while**(!stack.isEmpty()){  res[i--] = stack.pop();  }  **return** res; } |
| --- |

1. 4. G**iven unsorted string, deduplicate adjacent letters repeatedl**y

| **public** **class** **removeAllDuplicate** {  **public** **static** **void** **main**(String[] args) {  String s = "acaaabbbacdddd";  System.out.println(stringRemove(s, 0));   }  **public** **static** String **stringRemove**(String s, **int** n) {  **if** (s.length() == n) **return** s;   s = removeDuplicate(s);  **if** (s.length() != n) {  stringRemove(s,s.length());  }  **return** s;  }   **private** **static** String **removeDuplicate**(String s) {  **if** (s == **null** || s.length() <= 1) {  **return** s;  }   **int** len = s.length();  **char**[] sc = s.toCharArray();   *// slow is length*  **int** slow = 0;  **int** fast = 1;  **boolean** flag = **false**;  **while**(fast < len) {  **if**(sc[fast] != sc[fast - 1]) {  **if**(flag) {  flag = **false**;  } **else** {  sc[slow] = sc[fast - 1];  slow++;  }  } **else** {  flag = **true**;  }  fast++;  }  **if** (sc[len - 1] != sc[len - 2]) {  sc[slow] = sc[len - 1];  slow++;  }   **return** **new** String(sc).substring(0, slow);  } |
| --- |

1. L2**83 move zero with same order O(n) and O(1) space**

| *//s1 最后的位置补0*  **public** **int**[] moveZeroKeepOrder(**int**[] array){  **if**(array == **null** || array.length == 0) **return** **null**;  **int** slow = 0;  **for**(**int** fast = 0; fast < array.length; fast++){  **if**(array[fast] != 0){  array[slow++] = array[fast];  }  }  **while**(slow < array.length){  array[slow++] = 0;  }  **return** array;  } |
| --- |

| *//S2, 动的是slow指针。还是以fast指针去探索，当fast出界之后，slow之后补0* **public** **int**[] moveZero(int[] array){  //cc  int slow = 0;  int fast = 0;  while(slow < array.length){  if(fast < array.length){  if(array[fast] != 0){  array[slow++] = array[fast++];  }else{  fast++;  }  }else{  array[slow++] = 0;  }  }  return array;  } |
| --- |

| *//S3* **public** **void** **moveZeroes**(**int**[] nums) {  **if**(nums == **null** || nums.length <= 1) **return**;  **int** slow = 0;  **int** fast = slow+1;  **while**(fast < nums.length){  **if**(nums[slow]!=0){  slow++;  fast++;  }**else** **if**(nums[fast]==0){  fast++;  }**else**{  swap(nums, slow++, fast++);  }  }  }  **private** **void** **swap**(**int**[] nums, **int** i, **int** j){  **int** temp = nums[i];  nums[i] = nums[j];  nums[j] = temp;  } |
| --- |

//move all negative number to the right end while keep the order

| **//S1 时间O(n), 空间O(n)**  **static** **void** **segregateElements**(**int** arr[], **int** n)  {    *// Create an empty array to store result*  **int** temp[] = **new** **int**[n];    *// Traversal array and store +ve element in*  *// temp array*  **int** j = 0; *// index of temp*    **for** (**int** i = 0; i < n; i++)  **if** (arr[i] >= 0)  temp[j++] = arr[i];    *// If array contains all positive or all*  *// negative.*  **if** (j == n || j == 0)  **return**;    *// Store -ve element in temp array*  **for** (**int** i = 0; i < n; i++)  **if** (arr[i] < 0)  temp[j++] = arr[i];    *// Copy contents of temp[] to arr[]*  **for** (**int** i = 0; i < n; i++)  arr[i] = temp[i];  } |
| --- |

| *//S2 时间O(n), 空间O(1)，但是不能保持原来的顺序一样* **class** **GFG** {   **static** **void** **rearrange**(**int** arr[], **int** n){  **int** j = 0, temp;  **for** (**int** i = 0; i < n; i++){  **if** (arr[i] < 0){  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  j++;  }  }  }  *// A utility function to print an array*  **static** **void** **printArray**(**int** arr[], **int** n)  {  **for** (**int** i = 0; i < n; i++)  System.out.print(arr[i] + " ");  }    *// Driver code*  **public** **static** **void** **main**(String args[])  {  **int** arr[] = {-1, 2, -3, 4, 5, 6, -7, 8, 9};  **int** n = arr.length;    rearrange(arr, n);  printArray(arr, n);  } } |
| --- |

1. 7. given unsorted array, find largest and smallest number with least comparison

| /\*  *要求使得比较的次数最少，类似优化某一个api. 两两比较，大的放一组，小的放一组。然后从大的组里找出最大的，小的组里找出最小的。但是要注意奇偶的情况。*  *这样的话，比较的次数是n/2+n/2+n/2 = 3n/2. 第一次比较是n/2，然后在大的group里面for loop一次找最大的，时间是n/2，同样在小的group里面也是n/2。*  *这里也可以每一层都采用两两比较的方法，用recursion去做，但是这样的做法并没有优化。比较的次数是n/2 + n/4 + n/4 + n/8 + n/8 + ….+ 2+ 1 = 3n/2.用recursion的话，第一步是需要两边都要取，但是第一步之后，只需要取其中的一边。*  \*/  **public** **int**[] findLasgestAndSmallestNum(**int**[] array){  **if**(array == **null** || array.length == 0) **return** **null**;  **int**[] res = **new** **int**[2];  ArrayList<Integer> bigGroup = **new** ArrayList<>();  ArrayList<Integer>smallGroup = **new** ArrayList<>();  **int** left = 0;   **int** right = array.length-1;  **int** max = Integer.MIN\_VALUE;  **int** min = Integer.MAX\_VALUE;  **while**(left < right){  **if**(array[left] < array[right]){  bigGroup.add(array[right]);  smallGroup.add(array[left]);  }**else**{  bigGroup.add(array[left]);  smallGroup.add(array[right]);  }  left++;  right--;  }  **for**(**int** i = 0; i < bigGroup.size(); i++){  max = Math.max(max, bigGroup.get(i));  }  **for**(**int** j =0; j < smallGroup.size(); j++){  min = Math.min(min, smallGroup.get(j));  }  **if**(array[left] > max)res[0] = array[left];  **else** res[0] = max;  **if**(array[left] < min)res[1] = array[left];  **else** res[1] = min;  **return** res;  } |
| --- |

| **struct pair {  int min;  int max; };    struct pair getMinMax(int arr[], int low, int high) {  struct pair minmax, mml, mmr;   int mid;    */\* If there is only one element \*/*  if (low == high)  {  minmax.max = arr[low];  minmax.min = arr[low];   return minmax;  }     */\* If there are two elements \*/*  if (high == low + 1)  {   if (arr[low] > arr[high])   {  minmax.max = arr[low];  minmax.min = arr[high];  }   else  {  minmax.max = arr[high];  minmax.min = arr[low];  }   return minmax;  }    */\* If there are more than 2 elements \*/*  mid = (low + high)/2;   mml = getMinMax(arr, low, mid);  mmr = getMinMax(arr, mid+1, high);     */\* compare minimums of two parts\*/*  if (mml.min < mmr.min)  minmax.min = mml.min;  else  minmax.min = mmr.min;     */\* compare maximums of two parts\*/*  if (mml.max > mmr.max)  minmax.max = mml.max;  else  minmax.max = mmr.max;     return minmax; }** |
| --- |

1. 田忌赛马问题：given unsorted array, find the largest and second largest number with least comparison（类似leetcode 4）

| *//Geeksforgeeks Find the smallest and second smallest elements in an array* **class** **SecondSmallest** {  */\* Function to print first smallest and second smallest  elements \*/*  **static** **void** **print2Smallest**(**int** arr[])  {  **int** first, second, arr\_size = arr.length;    */\* There should be at least two elements \*/*  **if** (arr\_size < 2)  {  System.out.println(" Invalid Input ");  **return**;  }    first = second = Integer.MAX\_VALUE;  **for** (**int** i = 0; i < arr\_size ; i ++)  {  */\* If current element is smaller than first  then update both first and second \*/*  **if** (arr[i] < first)  {  second = first;  first = arr[i];  }    */\* If arr[i] is in between first and second  then update second \*/*  **else** **if** (arr[i] < second && arr[i] != first)  second = arr[i];  }  **if** (second == Integer.MAX\_VALUE)  System.out.println("There is no second" +  "smallest element");  **else**  System.out.println("The smallest element is " +  first + " and second Smallest" +  " element is " + second);  }    */\* Driver program to test above functions \*/*  **public** **static** **void** **main** (String[] args)  {  **int** arr[] = {12, 13, 1, 10, 34, 1};  print2Smallest(arr);  } } |
| --- |

1. L48, 2D matrix rotate by 90, 180,270度 （微软onsite题，amazon，facebook题)

| *//要采用拉的方式，不能采用推的方式。180度和270度都是要注意坐标的对应关系即可。*  public **void** **rotate**(**int**[][] matrix){  *//cc* helper(matrix, 0, matrix.length);  }  **private** **void** **helper**(**int**[][] matrix, **int** offset, **int** size){  **if**(size <= 1) **return**;  **for**(**int** i = 0; i < size-1; i++){  **int** temp = matrix[offset][offset+i];  matrix[offset][offset+i] = matrix[offset + size -1 - i][offset];  matrix[offset + size -1 - i][offset] = matrix[offset+size-1][offset+size-1-i];  matrix[offset+size-1][offset+size-1-i] = matrix[offset+i][offset+size-1];  matrix[offset+i][offset+size-1] = temp;  }  helper(matrix, offset+1, size-2);  } |
| --- |

考虑是长方形的时候，如何rotate

1. Q10 sort an array by the order defined by another array(AutoDesk 电面题)

| */\**  *如果在这个pattern里的话，就按照这个pattern排序；如果不在这个pattern的话，就按照native的大小关系排序。*  *例如: 对于一个array[]{2,1,3,5,7,2,9,7,2,6}, 需要按照另一个array{9,3,7}的pattern排序，那排哈之后的结果是{9,3,7,7,1,2,2,2,5,6}*  *通过改写Arrays.sort里面的comparator.需要注意的是，new comparator的时候，传入的必须是Integer Array*  *\*/*  **public** **int**[] sortByOtherPattern(**int**[] array, **int**[] pattern){  **if**(array == **null** || array.length == 0) **return** **null**;  HashMap<Integer, Integer> map = **new** HashMap<>();  **for**(**int** i = 0; i < pattern.length; i++){  map.put(pattern[i],i);  }  Integer[] wrap = **new** Integer[array.length];  **for**(**int** i = 0; i < array.length; i++) {  wrap[i] = array[i];  }  Arrays.sort(wrap, **new** Comparator<Integer>(){  **@Override**  **public** **int** **compare**(Integer a, Integer b) {  *// TODO Auto-generated method stub*  **if**(map.containsKey(a) && map.containsKey(b)){  **return** map.get(a) - map.get(b);  }**else** **if**(map.containsKey(a)){  **return** -1;  }**else** **if**(map.containsKey(b)){  **return** 1;  } **else**{  **return** a-b;  }  }  });  **for**(**int** i = 0; i< array.length; i++) {  array[i] = wrap[i];  }  **return** array;  } |
| --- |

1. Q11 Sort an array by the order defined by the first/last occurrence index

| *//同样是改写comparator*  **public** **int**[] sortByOccurence(**int**[] array){  *//cc*  HashMap<Integer,Integer> map = **new** HashMap<>();  Integer[] wrap = **new** Integer[array.length];  **for**(**int** i = 0; i < array.length; i++){  **if**(!map.containsKey(array[i])){  map.put(array[i],i);  }  }  **for**(**int** i = 0; i < array.length; i++) {  wrap[i] = array[i];  }  Arrays.sort(wrap, **new** Comparator<Integer>(){  **@Override**  **public** **int** **compare**(Integer a, Integer b){  **return** map.get(a) - map.get(b);  }  });  **for**(**int** i = 0; i < array.length; i++) {  array[i] = wrap[i];  }  **return** array;  }  S2：  class Solution {  public void moveZeroes(int[] nums) {  int i = 0;  int slow = nums.length - 1;  int fast = nums.length - 1;  while (i < nums.length - 1){  while (i < fast){  while(nums[fast] == nums[i] && fast > i ){//需要重复check fast有没有越过i，不然会继续改写  fast--;  }  nums[slow] = nums[fast];  slow--;  fast--;    }    int temp = nums[i];  for (int j = i + 1; j <= slow;j++){    nums[j] = temp;  }  i = slow + 1;  slow = nums.length - 1;  fast = nums.length - 1;  }    }  }  ps:这道题没有在LC上找到，所以testcase都是自己写的，如果有遗漏请订正！非常感谢 |
| --- |

1. L179 Q12 Largest Number

| */\* Given a list of non negative integers, arrange them such that they form the largest number. For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330. Note: The result may be very large, so you need to return a string instead of an integer 注意此处不能用变成Integer比大小的方式，因为可能会很长很长。 \*/* public String largestNumber(int[] nums) {  if(nums == null || nums.length == 0) return null;  StringBuilder sb = new StringBuilder();  Integer[] array = new Integer[nums.length];   for(int i = 0; i< nums.length; i++) {  array[i] = nums[i];   }    Arrays.sort(array, new Comparator<Integer>(){  @Override  public int compare(Integer a, Integer b){  String s1 = String.valueOf(a) + String.valueOf(b);  String s2 = String.valueOf(b) + String.valueOf(a);  return s2.compareTo(s1);  }  });  if(array[0] == 0) return "0";  for(int i = 0; i < array.length; i++){   sb.append(String.valueOf(array[i]));  }  return sb.toString();  } |
| --- |

1. Q13 Merge K sorted array

| //S1：进行多个 Merge 2。代码没有online judge  **public** **class** **Solution**{  **int**[] arr;  **public** **int**[] sortArrays(List<**int**[]> list){  **if**(list == **null** || list.size() == 0) **return** **null**;  **return** helper(list, 0, list.size()-1);  }    **private** **int**[] helper(List<**int**[]> list, **int** start, **int** end){  **if**(list.size() == 1) {  arr = list.get(0).clone();  **return** arr;  }    List<**int**[]> path = **new** ArrayList<**int**[]>();  **while**(start <= end){  **int**[] res = merge(list, start, end);  path.add(res);  start++;  end--;  }  helper(path, 0, path.size()-1);  **return** arr;  }    **private** **int**[] merge(List<**int**[]> list, **int** start, **int** end){  **if**(start == end) **return** list.get(start);  **int**[] arr1 = list.get(start);  **int**[] arr2 = list.get(end);  **int**[] temp = **new** **int**[arr1.length+arr2.length];  **int** cur1 = 0;  **int** cur2 = 0;  **int** cur = 0;  **while**(cur1 < arr1.length && cur2 < arr2.length){  **if**(arr1[cur1] < arr2[cur2]){  temp[cur] = arr1[cur1++];  }**else**{  temp[cur] = arr2[cur2++];  }  cur++;  }  **while**(cur1<arr1.length){  temp[cur++] = arr1[cur1++];  }  **while**(cur2 < arr2.length){  temp[cur++] = arr2[cur2++];  }  **return** temp;  } } |
| --- |

//S2 用heap 做，每次都比较k个，把最小的那个poll出来，然后再offer一个进去

| **public** **class** **Solution**{ **class** **Element**{  **int** val;  **int** listIdx;  **int** arrayIdx;  **public** **Element**(**int** val, **int** listIdx, **int** arrayIdx){  **this**.val = val;  **this**.listIdx = listIdx;  **this**.arrayIdx = arrayIdx;  }  }  **public** List<Integer> **mergeKSortedArray**(List<**int**[]> list){  **if**(list == **null** || list.size() == 0) **return** **null**;  List<Integer> res = **new** ArrayList<Integer>();  PriorityQueue<Element> minHeap = **new** PriorityQueue<Element>(list.size(), **new** Comparator<Element>(){  **@Override**  **public** **int** **compare**(Element a, Element b){  **return** a.val - b.val;  }  });  **for**(**int** i = 0; i < list.size(); i++){  minHeap.offer(**new** Element( list.get(i)[0], i,0) );  }  **while**(!minHeap.isEmpty()){  Element temp = minHeap.poll();  res.add(temp.val);  **if**(temp.arrayIdx+1 < list.get(temp.listIdx).length){  minHeap.offer(**new** Element (list.get(temp.listIdx)[temp.arrayIdx+1], temp.listIdx, temp.arrayIdx+1));  }  }    **return** res;  } } |
| --- |

1. L23 Merge K Sorted Linked List

//用heap

| **public** ListNode **mergeKLists**(ListNode[] lists) {  **if**(lists == **null** || lists.length ==0) **return** **null**;  ListNode head = **null**;  ListNode cur = **null**;  PriorityQueue<ListNode> minHeap = **new** PriorityQueue<ListNode>(lists.length, **new** Comparator<ListNode>(){  **@Override**  **public** **int** **compare**(ListNode a, ListNode b){  **return** a.val - b.val;  }  });  **for**(**int** i = 0; i < lists.length; i++){  **if**(lists[i] != **null**){  minHeap.offer(lists[i]);  }  }  **while**(!minHeap.isEmpty()){  ListNode temp = minHeap.poll();  **if**(head == **null**){  head = temp;  cur = head;  }**else**{  cur.next = temp;  cur = cur.next;  }    **if**(temp.next != **null**){  minHeap.offer(temp.next);  }  }  **return** head;  } |
| --- |

1. L1 2sum, combination sum

| */\* Given an array of integers, return indices of the two numbers such that they add up to a specific target. \*/*  **public** **int**[] twoSum(**int**[] array, **int** target){  **if**(array == **null** || array.length == 0) **throw** **new** IllegalArgumentException();  Map<Integer, Integer> map = **new** HashMap<>();  **int**[] res = **new** **int**[2];  **for**(**int** i = 0; i < array.length; i++){  **if**(map.containsKey(target - array[i])){  res[0] = map.get(target - array[i]);  res[1] = i;  **return** res;  }**else**{  map.put(array[i], i);  }  }  **return** **null**; } |
| --- |

//2 sum, unsorted array, has duplicate

1. L15, 3Sum

| ***/\* Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero. \*/*** class Solution {  public List<List<Integer>> threeSum(int[] nums) {  List<List<Integer>> res = new ArrayList<>();  Arrays.sort(nums);  *// int start = 0;*  *// int end = nums.length - 1;*  for(int i = 0; i < nums.length; i++) {  if(i > 0 && nums[i] == nums[i-1]) {  continue;  }    int target = 0 - nums[i];  int start = i + 1;  int end = nums.length - 1;    while(start < end) {    if(nums[start] + nums[end] == target) {  List<Integer> path = new ArrayList<>();  path.add(nums[i]);  path.add(nums[start]);  path.add(nums[end]);  res.add(path);  while(start < end && nums[start] == nums[start + 1]) start++;  while(start < end && nums[end] == nums[end - 1]) end--;   start++;  end--;  } else if(nums[start] + nums[end] < target) {  start++;  } else {  end--;  }   }   }  *//-4, -1, -1, 0, 1, 2*  return res;  } } |
| --- |

1. 4 Sum
2. K sum
3. Find common elements in 2 arrays

| **public** List<Integer> **findCommon**(**int**[] arr1, **int**[] arr2){  List<Integer> res = **new** ArrayList<>();  **if**(arr1 == **null** || arr2 == **null**) **return** **null**;  Set<Integer> set = **new** HashSet<>();  **if**(arr1.length > arr2.length) **return** findCommon(arr2, arr1);  **for**(**int** i = 0; i < arr1.length; i++){  set.add(arr1[i]);  }  **for**(**int** j = 0; j < arr2.length; j++){  **if**(set.contains(arr2[j])){  res.add(arr[j]);  }  }  **return** res;  } |
| --- |

1. Find common elements in 3 arrays

| **public** List<Integer> findCommonInThreeArrays(){  } |
| --- |

1. Find common elements in 4 arrays
2. Find common elements in k sorted arrays. 是要求k个array中都含有的一个element

| *//用heap做,谁小移谁。每次都check一下当前的最大值和heap poll出来的值是否相等* **public** **class** **FindCommon**{  class Element{  int val;  int matrixIdx;  int arrayIdx;  public Element(int val, int matrixIdx, int arrayIdx){  this.val = val;  this.matrixIdx = matrixIdx;  this.arrayIdx = arrayIdx;  }  }  **public** List<Integer> **findCommonInKArrays**(**int**[][] arrays){  *//cc*  **int** maxSoFar = arrays[0][0];  List<Integer> res = **new** ArrayList<Integer>();  PriorityQueue<Cell> minHeap = **new** PriorityQueue<Cell>(arrays.length, **new** Comparator<Cell>(){  **@Override**  **public** **int** **compare**(Cell a, Cell b){  **return** a.val - b.val;  }  });  **for**(**int** i = 0; i < arrays.length; i++){  maxSoFar = Math.max(maxSoFar, arrays[i][0]);  minHeap.offer(**new** Cell(arrays[i][0], i, 0));  }    **while**(!minHeap.isEmpty()){  **if**(maxSoFar == minHeap.peek().val){  res.add(minHeap.peek().val);  }   Cell pollEle = minHeap.poll();  **if**(pollEle.arrayIdx + 1 < arrays[pollEle.matrixIdx].length){  minHeap.offer(**new** Cell(arrays[pollEle.matrixIdx][pollEle.arrayIdx+1], pollEle.matrixIdx, pollEle.arrayIdx + 1));  maxSoFar = Math.max(maxSoFar, arrays[pollEle.matrixIdx][pollEle.arrayIdx+1]);  }**else**{  **break**;  }   }  **return** res;  }  } |
| --- |

1. Q 22. Given two sorted array, find kth smallest element from them(一定要写)

| ***//S1 merge two sorted array, then find the kth one, 时间复杂度O(m+n)* public int kthSmallestFronTwoArray(int[] arr1, int[] arr2, int k){  *//cc*  int cur1 = 0;  int cur2 = 0;  int cur = 0;  int[] res = new int[arr1.length+arr2.length];  while(cur1 < arr1.length && cur2 < arr2.length){  if(arr1[cur1] < arr2[cur2]){  res[cur] = arr1[cur1++];  }else{  res[cur] = arr2[cur2++];  }  cur++;  }  while(cur1<arr1.length){  res[cur++] = arr1[cur1++];  }  while(cur2<arr2.length){  res[cur++] = arr2[cur2++];  }  return res[k];**  **}** |
| --- |

| *//S2 merg*e*前k个，time complexity O(k)* **public** **int** **kthSmallestFromTwoArray2**(**int**[] arr1, **int**[] arr2, **int** k){  **if**((arr1 == **null** && arr2 == **null**) || arr1.length + arr2.length < k) **throw** **new** IllegalArgumentException();  **int** cur1 = 0;  **int** cur2 = 0;  **int** cur = 0;  **int**[] res = **new** **int**[k+1];  **while**(cur1 + cur2 <= k){  **if**(arr1[cur1] < arr2[cur2]){  res[cur] = arr1[cur1++];  }**else**{  res[cur] = arr2[cur2++];  }  cur++;  }  **return** res[k]; } |
| --- |

| *//S3, binary reduction, time complexity is logK using recursion*  public intkthSmallestFromTwoArrays3(int[] arr1, int[] arr2, int k){  //cc  int[] res = new int[1];  helper(res, arr1, arr2, 0, 0, k);  return res[0];  }  private void helper(int[] res, int[] arr1, int[] arr2, int start1, int start2, int k){  if(start1 + start2 == k) return;  if(arr1.length - start1 > k/2 && arr2.length - start2 > k/2){  if(arr1[k/2] < arr2[k/2]){  }  }  } |
| --- |

1. L4 Given two sorted array, find the median of them in log(m+n)（微软面试题）

| class Solution {  public double findMedianSortedArrays(int[] nums1, int[] nums2) {    int len1 = nums1.length;  int len2 = nums2.length;  int len = len1 + len2;    if(len % 2 == 0) {  return (helper(nums1, nums2, 0, 0, len/2) + helper(nums1, nums2, 0, 0, len/2 + 1) )/2.0;  } else {  return helper(nums1, nums2, 0, 0, len/2 + 1);  }  }    private int helper(int[] nums1, int[] nums2, int start1, int start2, int k) {    if(start1 >= nums1.length) {  return nums2[start2 + k - 1];  }  if(start2 >= nums2.length) {  return nums1[start1 + k - 1];  }    if(k == 1) {  return Math.min(nums1[start1], nums2[start2]);  }    int val1 = start1 + k/2 - 1 < nums1.length ? nums1[start1+k/2-1] : Integer.MAX\_VALUE;  int val2 = start2 + k/2 - 1 < nums2.length ? nums2[start2+k/2-1] : Integer.MAX\_VALUE;    if(val1 < val2) {  return helper(nums1, nums2, start1 + k/2 - 1 + 1, start2, k - k/2);  } else {  return helper(nums1, nums2, start1, start2 + k/2 - 1 + 1, k - k/2);  }    }  } |
| --- |

1. 一个array的streaming flow, keep the median and access it with O(1), return median
2. given an unsorted array, find the max product of 3 element(k product)(intweet 的面试题)
3. L239 sliding window maximum

| */\* Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. For example, Given nums = [1,3,-1,-3,5,3,6,7], and k = 3. Therefore, return the max sliding window as [3,3,5,5,6,7]. \*/*  */\* s1 using queue to store the sliding window and get the max for each move. Accepted but time efficiency is porr. O(k\*(n-k+1)) \*/*  **public** **int**[] maxSlidingWindow(**int**[] nums, **int** k) {  **if**(nums == **null** ) **return** **null**;  **if**(nums.length == 0) **return** **new** **int**[0];  **int**[] res = **new** **int**[nums.length - k + 1];  Deque<Integer> queue = **new** LinkedList<Integer>();  **for**(**int** i = 0; i < k; i++){  queue.offerLast(nums[i]);  }  res[0] = findMax(queue,k);  **for**(**int** i = k; i < nums.length; i++){  queue.pollFirst();  queue.offerLast(nums[i]);  res[i - k + 1] = findMax(queue, k);  }  **return** res;  }  **private** **int** **findMax**(Deque<Integer> queue, **int** k){  **int** max = queue.peek();  **int** count = 0;  **while**(count < k){  **int** pollVal = queue.pollFirst();  max = Math.max(max, pollVal);  queue.offerLast(pollVal);  count++;  }  **return** max; } |
| --- |

| ***/\* S2 采用单调序列, QUEUE里面存的内容可以是nums的值，也可以是nums的index \*/* class Solution {** public int[] maxSlidingWindow(int[] nums, int k) {  if(nums == null) return null;  if(nums.length == 0)return new int[0];  int[] res = new int[nums.length - k +1];  Deque<Integer> queue = new LinkedList<Integer>();  for(int i = 0; i < k; i++){  update(queue, nums[i]);  }  res[0] = queue.peekFirst();  for(int i = k; i < nums.length; i++){   if(queue.peekFirst() == nums[i-k]){  queue.pollFirst();  }  update(queue, nums[i]);  res[i-k+1] = queue.peek();  }  return res;  } private void update(Deque<Integer> queue, int val){  while(!queue.isEmpty() && val > queue.peekLast()){  queue.pollLast();  }  queue.offerLast(val);  } **}** |
| --- |

# Session 16 综合演练 I

1. L138, copy list with random pointer / Skip pointer(非常重要),

| ***// s1: 1 HashMap, 2 pass. 第一个pass用来存原来的reference，第二个后的reference***  ***// Time complexity O(n), Space complexity O(n), n equals the size of list*** class RandomListNode {  int label;  RandomListNode next, random;  RandomListNode(int x) { this.label = x; } }  public RandomListNode copyRandomList(RandomListNode head) {  if(head == null ) return head;  HashMap<RandomListNode, RandomListNode> map = new HashMap<>();  RandomListNode cur = head;  while(cur != null){  map.put(cur, new RandomListNode(cur.label));  cur = cur.next;  }  cur = head;  while(cur != null){  map.get(cur).next = map.get(cur.next);  map.get(cur).random = map.get(cur.random);  cur = cur.next;  }  return map.get(head);  } |
| --- |

| ***//S2: 1 hashMap, 1 pass. 边copy 边存* class Node {  int val;  Node next, random;  Node(int x) { this.val = x; } } public RandomListNode copyRandomList(RandomListNode head) {** if(head == null ) return head;  HashMap<**Node**, **Node**> map = new HashMap<>();  **Node** cur = head;  ***// contributor @Amy amycoconut115@gmail.com***  while(cur != null){  if(!map.containsKey(cur)){  map.put(cur, new Node(cur.val));  }    if(cur.next != null) {  if (!map.containsKey(cur.next)){  map.put(cur.next, new Node(cur.next.val));  }  map.get(cur).next = map.get(cur.next);  }    if(cur.random != null) {  if (!map.containsKey(cur.random)){  map.put(cur.random, new Node(cur.random.val));  }  map.get(cur).random = map.get(cur.random);  }  cur = cur.next;  }   return map.get(head);  **}** |
| --- |
|  |

| ***//S2: concise version 1 hashMap, 1 pass. 边copy 边存* class Node {  int val;  Node next, random;  Node(int x) { this.val = x; } } public RandomListNode copyRandomList(RandomListNode head) {** if(head == null) return head;  HashMap<Node, Node> map = new HashMap<>();  Node cur = head;  map.put(cur, new Node(cur.val)); while(cur != null){  if(cur.next != null) {  map.putIfAbsent(cur.next, new Node(cur.next.val));  map.get(cur).next = map.get(cur.next);  }  if(cur.random != null) {  map.putIfAbsent(cur.random, new Node(cur.random.val));  map.get(cur).random = map.get(cur.random);  }  cur = cur.next;  }   return map.get(head);  **}** |
| --- |

1. deep copy a tree

| **//用recursion**  ***// Time complexity O(n), Space complexity O(n), n equals the size of tree***  **public** TreeNode **deepCopyTree**(TreeNode root){  if(root == null) return null;  TreeNode newRoot = new TreeNode(root.val);  newRoot.left = deepCopyTree(root.left);  newRoot.right = deepCopyTree(root.right);  return newRoot;  } |
| --- |

1. Deep copy a Graph

Recursive

| ***// Time complexity O(n), Space complexity O(n), n equals the size of graph***  **public** Node **cloneGraph**(Node node) {  **if** (node == **null**) {  **return** **null**;  }  **return** cloneGraphHelper(node,**new** HashMap<>());  }    **private** Node **cloneGraphHelper**(Node node, Map<Node,Node> nodeMap) {  **if** (nodeMap.containsKey(node)) **return** nodeMap.get(node);  Node newNode = **new** Node(node.val, **new** ArrayList<>());  nodeMap.put(node,newNode);  **for** (Node n : node.neighbors) {  newNode.neighbors.add(cloneGraphHelper(n,nodeMap));  }  **return** newNode;  } |
| --- |

Iterative

| **public** Node **cloneGraph**(Node node) {  **if** (node == **null**) {  **return** **null**;  }    Queue<Node> que = **new** LinkedList<>();  Map<Node,Node> nodeMap = **new** HashMap<>();  que.add(node);  nodeMap.put(node,**new** Node(node.val, **new** ArrayList<>()));    **while** (!que.isEmpty()) {  Node cur = que.poll();  Node curClone = nodeMap.get(cur);    **for** (Node n : cur.neighbors) {  Node nextClone = nodeMap.get(n);  **if** (nextClone == **null**) {  nextClone = **new** Node(n.val, **new** ArrayList<>());  nodeMap.put(n, nextClone);   que.add(n);  }  curClone.neighbors.add(nextClone);  }  }  **return** nodeMap.get(node);  } |
| --- |

1. L270 Closest Binary Search Tree value

| *//double比较的时候，比较特殊* **public** **int** **closestValue**(TreeNode root, **double** target) {  *//cc*  TreeNode closest = root;  TreeNode cur = root;  **while**(cur != **null**){  **if**(cur.val == target){  **return** cur.val;  }**else** **if**(cur.val < target){  **if**(Math.abs(cur.val - target) < Math.abs(closest.val - target)){  closest = cur;  }  cur = cur.right;  }**else**{  **if**(Math.abs(cur.val - target) < Math.abs(closest.val - target)){  closest = cur;  }  cur = cur.left;  }  }  **return** closest.val;  } |
| --- |

1. L272, K closest Binary search tree value

| */\* S1 算法哥的解法，用两个stack做.leftStack存储的是比target小的node，rightStack存储的是比target大的node。 先用一个cur去找node，如果cur.val>target的话，就将cur压进left，然后再往右边走(因为根据bst的特性，如果cur比target小的话，那么其左子树都会比target小)；如果cur.val<target,将cur压进right，然后再往左子树走。leftStack是按照从大到小的顺序依次排列，rightStack是按照从小到大的顺序依次排列。 \*/* **class** **Solution** {  **public** List<Integer> **closestKValues**(TreeNode root, **double** target, **int** k) {  List<Integer> res = **new** ArrayList<Integer>();  **if**(root == **null**) **return** res;  Stack<TreeNode> leftStack = **new** Stack<>();  Stack<TreeNode> rightStack = **new** Stack<>();  TreeNode cur = root;  **while**(cur != **null**){  **if**(cur.val < target){  leftStack.push(cur);  cur = cur.right;  }**else**{  rightStack.push(cur);  cur = cur.left;  }  }  **while**(k-->0){  **if**(!leftStack.isEmpty() && !rightStack.isEmpty()){  **if**(Math.abs(leftStack.peek().val - target) < Math.abs(rightStack.peek().val - target)){  res.add(getPrev(leftStack));  }**else**{  res.add(getNext(rightStack));  }  }**else** **if**(!leftStack.isEmpty()){  res.add(getPrev(leftStack));  }**else**{  res.add(getNext(rightStack));  }  }  **return** res;  } */\* getPrev的作用是将leftStack打开，往右走，将各个right node的左子树压进去，然后保持leftStack从大到小排列 \*／* private int getPrev(Stack<TreeNode> stack){  TreeNode top = stack.pop();  TreeNode cur = top.left;  while(cur!=null){  stack.push(cur);  cur = cur.right;  }  return top.val;  }    private int getNext(Stack<TreeNode> stack){  TreeNode top = stack.pop();  TreeNode cur = top.right;  while(cur!=null){  stack.push(cur);  cur = cur.left;  }  return top.val;  } } |
| --- |

| *// S2 inorder transverse*   **public** List<Integer> **closestKValues**(TreeNode root, **double** target, **int** k) {  List<Integer> res = **new** LinkedList<>();  inOrder(root, target, k, res);  **return** res;  }    **private** **boolean** **inOrder**(TreeNode root, **double** target, **int** k, List<Integer> res) {  **if** (root == **null**) **return** **true**;  **if** (!inOrder(root.left, target, k, res)) **return** **false**;  **if** (k > res.size()) {  res.add(root.val);  } **else** {  **if** (Math.abs(res.get(0) - target) <= Math.abs(root.val - target)) {  **return** **false**;  }  res.add(root.val);  res.remove(0);  }  **return** inOrder(root.right, target, k, res);  } |
| --- |

| *// S3 priority queue solution* **public** List<Integer> **closestKValues**(TreeNode root, **double** target, **int** k) {  List<Integer> res = **new** ArrayList<Integer>();  PriorityQueue<TreeNode> maxHeap = **new** PriorityQueue<TreeNode>(k, (a, b) -> {  **double** val1 = Math.abs(a.val - target), val2 = Math.abs(b.val - target);  **return** val1 == val2 ? 0 : val1 > val2 ? -1 : 1;  });  dfsHelper(root, target, k, maxHeap);  **for**(**int** i = 0; i < k; i++){  res.add(maxHeap.poll().val);  }  **return** res;  }    **private** **void** **dfsHelper**(TreeNode root, **double** target, **int** k, PriorityQueue<TreeNode> maxHeap) {  **if** (root == **null**) {  **return**;  }    maxHeap.offer(root);  **if** (maxHeap.size() > k) {  maxHeap.poll();  }  dfsHelper(root.left, target, k, maxHeap);  dfsHelper(root.right, target, k, maxHeap);  } |
| --- |

1. largest smaller binary search tree value(比target小的最大的) (Salesforce onsite 题目)

| */\**  *这道题需要注意的点是edge case的处理，由于是找比target小的最大值，属于单边逼近的问题，所以会出现可能不存在的情况。例如对于一个bst = [15, 5, 20, 3, 14, 18, 23 ],如果给的target是0，比0小的最大的是没有的，所以一开始如果将closest = root的话，会导致最后输出的结果是root，但是其实是应该是没有答案。*  *S1 返回的是TreeNode的方法*  *\*／*  **public TreeNode largestSmaller(TreeNode root, int target){**  if(root == null) return null;  TreeNode closest = null;  TreeNode cur = root;  while(cur != null){  if(cur.val < target){  if(cur.val > closest.val){  closest = cur;  }  cur = cur.right;  }else{  cur = cur.left;  }  }  return closest;  **}** |
| --- |

| *//S2 return的是int* **public** int largestSmaller(TreeNode root, int target){  if(root == null) throw new IllegalArgumentException();  int closest = root.val;  TreeNode cur = root;  while(cur != null){  if(cur.val < target){  if(cur.val > closest){  closest = cur.val;  }  cur = cur.left;  }else{  cur = cur.right;  }  }  if(closest < target) return closest;  else throw IllegalArgumentException();  *//if(closest == root.val) throw new IllegalArgumentException();*  *//return closest;*  } |
| --- |

1. Smallest larger to target in a binary search tree

| *//* **public** TreeNode **smallestLarger**(TreeNode root){  if(root == null) return null;  TreeNode closest = null;  TreeNode cur = root;  while(cur != null){  if(cur.val > target){  closest = cur;  cur = cur.left;  }else{  cur = cur.right;  }  }  return closest;*//如果return的是int的话，这里需要判断一下最后target是否比最大的那个还要大*  } |
| --- |

1. L450 Remove target from BST（重要）

| ***//先找到和target值相等所在的node。如果是大于／小于target的话，采用recursion的方式，往左子树／右子树方向找。如果是等于target的话，表示已经locate到了需要delete的那个node。此时还有三种情况：a）左子树和右子树都不为空，b）左子树为空，c）右子树为空* class Solution {** public TreeNode deleteNode(TreeNode root, int key) {  if(root == null) return root;    if(root.val < key){  root.right = deleteNode(root.right, key);*//需要删出的node在右边，所以只要recursion call下去，右边传上来的就是删除好的版本*  }else if(root.val > key){  root.left = deleteNode(root.left, key);*//和右边的情况一样*  }else{  if(root.left != null && root.right != null){//两边子树都不为空的情况  root.val = findSmallestInRight(root.right).val;*//也可以从左边找最大的*  root.right = deleteNode(root.right, root.val);  }else{  root = root.left == null ? root.right : root.left;//  }  }  return root;  }  private TreeNode findSmallestInRight(TreeNode root){  while(root.left != null){  root = root.left;  }  return root;  } **}** |
| --- |

1. LT85 Insert Target into BST

| ***//Given a binary search tree and a new tree node, insert the node into the tree. You should keep //the tree still be a valid binary search tree.* *//S1 Recursion*** public TreeNode insertNode(TreeNode root, TreeNode node) {  *// write your code here*  if(root == null) return node;  if(root.val > node.val) {  root.left = insertNode(root.left,node);  //if(root.left == null) root.left = node;  }else {  root.right = insertNode(root.right, node);  //if(root.right == null) root.right = node;  }  return root;  } |
| --- |

| **//S2 iteration**  **public TreeNode insertNode(TreeNode root, TreeNode node) {** *// write your code here*  if(root == null && node == null) return node;  if(node == null) return root;  if(root == null) return node;  TreeNode cur = root;  TreeNode prev = root;  while(cur != null){  prev = cur;  if(cur.val < node.val){  cur = cur.right;  }else{  cur = cur.left;  }  }  if(prev.val > node.val){  prev.left = node;  }else if(prev.val < node.val){  prev.right = node;  }  return root;  **}** |
| --- |

1. L156 Binary search tree upside down --> only upside down

| *／\** *Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root. 用root.left作base case，最后通过指针的转换来实现反转 \*／* **public** TreeNode **upsideDownBinaryTree**(TreeNode root) {  *//if(root == null) return null;*  **if**(root == **null** || root.left == **null**) **return** root;*//root==null是corner case*  TreeNode newRoot = upsideDownBinaryTree(root.left);  root.left.left = root.right;  root.left.right = root;  root.left = **null**;  root.right = **null**;  **return** newRoot; } |
| --- |

1. Cut Stone

| *//S2 recursion* ***public******int*** |
| --- |

| *//S2 recursion + pruning* **public** **int** |
| --- |

| *//S3 DP* **public** **int** |
| --- |

1. Stones merge 著名的沙子归并问题
2. L20 valid parentheses

| */\* Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not. \*/* *// 用Stack做* **public** **boolean** **isValid**(String s) {  **if**(s == **null** || s.length() == 0) **return** **true**;  Stack<Character> stack = **new** Stack<>();  **for**(**int** i = 0; i < s.length(); i++){  **char** c = s.charAt(i);  **if**(stack.isEmpty()){  stack.push(c);  }**else**{  **if**(c == '(' || c == '{' || c == '['){  stack.push(c);  }**else**{  **if**(c == ')'){  **if**(stack.isEmpty() || stack.pop() != '(') **return** **false**;  }  **if**(stack.isEmpty() || c == ']'){  **if**(stack.pop() != '[') **return** **false**;  }  **if**(stack.isEmpty() || c == '}'){  **if**(stack.pop() != '{') **return** **false**;  }  }  }  }  **return** stack.isEmpty();  }   *//比较精炼的写法* **public** **boolean** **isValid**(String s) {  **if**(s == **null** || s.length()==0) **return** **true**;  Stack<Character> stack = **new** Stack<Character>();  **for**(**char** c : s.toCharArray()){  **if**(c == '('){  stack.push(')');  }**else** **if**(c == '['){  stack.push(']');  }**else** **if**(c == '{'){  stack.push('}');  }**else** {  **if**(stack.isEmpty() || stack.pop()!=c){  **return** **false**;  }  }  }  **return** stack.isEmpty();  } |
| --- |

1. valid xml html

这里是k种

1. n pairs () 的permutations
2. validate permutation of n pairs (), m pairs [], p pairs {}.
3. Best Time to buy and sell stock I II III IV V cooldown
4. L318 find two strings from dictionary without common character, max product of length

| */\* Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower-case letters. If no such two words exist, return 0. Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"] Return 16 The two words can be "abcw", "xtfn". \*/* *//brute force. 遍历String[]中的每一个元素，取出来做比较，如果含有相同的字符，或长度小于max就进下一个* **class** **Solution** {  **public** **int** **maxProduct**(String[] words) {  **if**(words == **null** || words.length == 0) **return** 0;  **int** max = 0;//是针对任何一个w1和w2的pair  **for**(**int** i = 0; i < words.length-1; i++){  **for**(**int** j = i+1; j < words.length; j++){  String word1 = words[i];  String word2 = words[j];  **if**(word1.length() \* word2.length() < max) **continue**;  **else** **if**(isCommon(word1, word2) == **false**){  max = Math.max(max, word1.length()\*word2.length());  }  }  }  **return** max;  }    **private** **boolean** **isCommon**(String s1, String s2){  **if**(s1.length() > s2.length()) **return** isCommon(s2,s1);  HashSet<Character> set = **new** HashSet<>();  **for**(**int** i = 0; i < s1.length(); i++){  set.add(s1.charAt(i));  }  **for**(**int** j= 0; j < s2.length(); j++){  **if**(set.contains(s2.charAt(j))){  **return** **true**;  }  }  **return** **false**;  } |
| --- |

| *//S2 比特操作优化，或者hashSet优化* **public int** **maxProduct**(String[] words){  if(words == null || words.length == 0) return 0;  int[] val = new int[words.length];//每一个单词用一个int来表示  int max = 0;  for(int i = 0; i< words.length; i++){  for(int j = 0; j < words[i].length(); j++){  //如果某个字母出现的话，把相应的bit位设置为1  val[i] = val[i] | (1 << words[i].charAt(j) - ‘a’);  }  }  for(int i = 0; i < words.length - 1; i++){  for(int j = i+1; j < words.length; j++){  if((val[i] & val[j]) == 0){  max = Math.max(max, words[i].length()\*words[j].length());  }  }  }  return max;  } |
| --- |

| *//S3 BFS优化* |
| --- |

1. kth smallest number in f(x,y,z) = a^x \* b^y \* c^z (x,y,z>0, a,b,c > 1) (facebook onsite)

| *// x=y=z=1的时候，得到的结果最小，所以用heap先将111 offer进去。* **static** **class** **Element**{  **int** x;  **int** y;  **int** z;  **public** **Element**(**int** x, **int** y, **int** z){  **this**.x = x;  **this**.y = y;  **this**.z = z;  }   **@Override**  **public** **int** **hashCode**() {  **return** **this**.x \* 31^2 + **this**.y \* 31 + **this**.z;  }   **@Override**  **public** **boolean** **equals**(Object obj) {  **if**(obj **instanceof** Element){  Element c = (Element) obj;  **return** **this**.x == c.x && **this**.y == c.y && **this**.z == c.z;  }**else**{  **return** **false**;  }  } }  **public** **static** **int** **kthSmallestNumber**(**int** x, **int** y, **int** z, **final** **int** a, **final** **int** b, **final** **int** c, **int** k){  PriorityQueue<Element> minHeap = **new** PriorityQueue<Element>(k, **new** Comparator<Element>(){  **@Override**  **public** **int** **compare**(Element e1, Element e2){  **return** (**int**) (Math.pow(a, (e1.x)) +Math.pow(b, (e1.y)) + Math.pow(c, (e1.z)) -  (Math.pow(a, (e2.x)) + Math.pow(b, (e2.y)) + Math.pow(c, (e2.z))));  }  });   HashSet<Element> set = **new** HashSet<Element>();  Element ele1 = **new** Element(1,1,1);  minHeap.offer(ele1);  set.add(ele1);  **int** res = 0;  **while**(k-- > 0 && !minHeap.isEmpty()) {  Element ele = minHeap.poll();  res = (**int**) ((**int**)Math.pow(a, ele.x) + Math.pow(a, ele.y) + Math.pow(c, ele.z));  *//System.out.print(res + " ");*  **int** i = ele.x;  **int** j = ele.y;  **int** l = ele.z;   **if**(i + 1 < k){  Element eleI = **new** Element(i+1,j,l);  **if**(set.add(eleI)){  minHeap.offer(eleI);  }  }   **if**(j + 1 < k){  Element eleJ = **new** Element(i,j+1,l);  **if**(set.add(eleJ)){  minHeap.offer(eleJ);  }  }   **if**(l + 1 < k){  Element eleL = **new** Element(i,j,l+1);  **if**(set.add(eleL)){  minHeap.offer(eleL);  }  }   }   **return** res; } |
| --- |

| */\* L668 Memory Limit exceed 空间复杂度太高O(K),估计题目要求O（1）space \*/* **class** **Cell**{  **int** x;  **int** y;  **public** **Cell**(**int** x, **int** y){  **this**.x = x;  **this**.y = y;  } } **public** **int** **findKthNumber**(**int** m, **int** n, **int** k) {  **if**(m == 0 || n == 0 || m\*n < k) **throw** **new** IllegalArgumentException();  PriorityQueue<Cell> maxHeap = **new** PriorityQueue<Cell>(k, **new** Comparator<Cell>(){  **@Override**  **public** **int** **compare**(Cell c1, Cell c2){  **return** c2.x\*c2.y - c1.x\*c1.y;  }  });  HashSet<Integer> set = **new** HashSet<Integer>();  **int** count = 1;  **for**(**int** i = 1; i <= m; i++){  **for**(**int** j = 1; j <= n; j++){  Cell cell = **new** Cell(i, j);  **if**(count <= k){  **if**(set.add(i\*m+j)){  *//System.out.print(set.contains(cell) + " ");*  maxHeap.offer(cell);  }  count++;  }**else**{  **if**(set.add(i\*m+j) && cell.x\*cell.y < maxHeap.peek().x \* maxHeap.peek().y){  maxHeap.poll();  maxHeap.offer(cell);  }  }  }  }  **return** maxHeap.peek().x \* maxHeap.peek().y; } |
| --- |

1. kth closest position in 2D 3D space from k sorted array(all positive number)
2. L84 直方图面积Largest Rectangle in Histogram(Uber onsite)

| ***/\* Brute force，从左往右扫一遍，每到一个位置的时候，就向左右走，分别找到第一个比自己小的。从宽度差乘以本身的高度，能得出面积。然后更新最大的面积。***  ***时间复杂度O(n^2), 空间复杂度O(1), leetcode上超时。 \*／*** *class Solution {  public int largestRectangleArea(int[] heights) {  if(heights == null || heights.length == 0) return 0;  int maxArea = 0;  for(int i = 0; i < heights.length; i++){  int width = findWidth(i, heights);  maxArea = Math.max(maxArea, width\*heights[i]);  }  return maxArea;  }    private int findWidth(int idx, int[] heights){  int start = idx;  int end = idx;  while(start >= 0) {  if (start == 0 || heights[idx] > heights[start - 1]) {  break;  }  start--;  }  while(end < heights.length){  if(end == heights.length - 1 || heights[idx] > heights[end+1]){  break;  }  end++;  }  end = end+1;  return end - start;  } }* |
| --- |

| *//s2， 用两个array和一个stack，one pass去做。leftArray存左边界，rightArray存右边界，stack用来存储还没有计算出来面积的立柱的index，当哪个立柱的面积可以计算了，就可以将其踢出stack。* public int largestRectangleArea(int[] heights){  if(heights == null || heights.length == 0)return 0;  int[] leftBound = new int[heights.length];  int[] rightBound = new int[heights.length];  Stack<Integer> stack = new Stack<>();  for(int i = 0; i < heights.length; i++){  if(stack.isEmpty()){  stack.push(i);  left[i] = i;  }else{  if(heights[i])  }  }  } |
| --- |



| *//S3 只用一个stack，因为在stack中已经可以得到左右边界的信息了。*  **public** **int** **largestRectangleArea**(**int**[] heights) {  **if**(heights == **null** || heights.length == 0) **return** 0;  Stack<Integer> stack= **new** Stack<Integer>();  **int** max = 0;  **for**(**int** i = 0; i <= heights.length; i++){//边界条件为<=  **int** h = (i < heights.length ? heights[i]:0);*//最后位置的立柱高度为0，可以在最后全部清理掉stack中存的数据。不然最后还要post processing一次*  **if**(stack.isEmpty() || h >= heights[stack.peek()]){*//等于的时候也要加进去*  stack.push(i);  }**else**{  **while**(!stack.isEmpty() && h < heights[stack.peek()]){  **int** popIdx = stack.pop();  **int** area = 0;  **if**(stack.isEmpty()){  area = heights[popIdx]\*i;*//能够留在最左边的位置，说明是高度最低的那个立柱，所以其宽度应该是整个系列的宽度*  }**else**{  area = heights[popIdx] \* (i - stack.peek() - 1);  }  max = Math.max(max, area);  }  stack.push(i);  }  }  **return** max;  } |
| --- |

1. L42 Trapping rain water

| */\* Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining. For example,  Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6. \*/* */\* S1: Brute force： 对每一个点，for loop往左看，找到左边的最大值；再往右看，找到右边的最大值.取两个中较小的那个值，作为当前index的能够盛的水的高度。 时间复杂度为O(n^2) \*/*  */\* S2: 3 pass。第一个pass是从左往右扫，用一个array leftMax记录往左看的最大值；第二个pass是从右往左扫，用一个array rightMax记录的是往右看的最大值。然后用一个minMax array记录两个array 中对应位置的较小值。*  *时间复杂度为O(n), 空间复杂度为O(2n) \*/* **public** **int** **trap**(**int**[] height){  **if**(height == **null** || height.length ==0) **return** 0;  **int** len = height.length;  **int**[] leftMax = **new** **int**[len];  leftMax[0] = height[0];  **int**[] rightMax = **new** **int**[len];  rightMax[len-1] = height[len-1];  **int**[] minMax = **new** **int**[len];  **int** leftCurMax = leftMax[0];  **int** rightCurMax = rightMax[len-1];  **int** trapWater = 0;  **for**(**int** i = 1; i < len; i++){  **if**(height[i] > leftCurMax){  leftCurMax = height[i];  }  leftMax[i] = leftCurMax;  }    **for**(**int** j = len - 2; j>=0; j--){  **if**(height[j] > rightCurMax){  rightCurMax = height[j];  }  rightMax[j] = rightCurMax;  }    **for**(**int** i = 0; i < len; i++){  **int** h = Math.min(leftMax[i], rightMax[i]);  trapWater += (h-height[i]);  }  **return** trapWater;  } |
| --- |

| ***/\* s3 用两个指针left和right放在左右两端，同时借用leftMax，rightMax两个变量，在扫一遍的过程中更新trapWater的值. \*/* public int trap(int[] height){  if(height == null || height.length == 0) return 0;  int len = height.length;  int left = 0;  int right = len-1;  int leftMax = height[0];  int rightMax = height[len-1];  int trapWater = 0;  while(left < right){  if(height[left] >= leftMax){  leftMax = height[left];  }  if(height[right] >= rightMax){  rightMax = height[right];  }  trapWater += (leftMax-height[left]) + (rightMax - height[right]);  if(leftMax < rightMax){  left++;  }else {  right--;  }  }  return trapWater; }** |
| --- |

| *//S3 更优的写法* **public** **int** **trap**(**int**[] array){  **if**(array == **null** || array.length == 0) **return** 0;  **int** left = 0;   **int** right = array.length;  **int** leftMax = 0;  **int** rightMax = 0;  **int** water = 0;  **while**(left <= right){//最好写成<=  **if**(leftMax < rightMax){  leftMax = Math.max(array[left], leftMax);  water += (leftMax - array[left]);*//移动left之前先把水量加上去*  left++;  }**else**{*//把==的情况吸收进去，虽然只移动一边，但是再次进入while loop的时候会更新*  rightMax = Math.max(array[right], rightMax);  water += (rightMax - array[right]);  right--;  }  }  **return** water; } |
| --- |

1. 2D water Trapping(google面试题)

# Session 17 综合演练 II

1. L90，Q13.2 subset II with all subsequene of sorted/adjacent string with dupicate
2. L146, LRU cache

| */\* Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and put. get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1. put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item. Follow up: Could you do both operations in O(1) time complexity? \*/*  **class** **LRUCache** {   **private** DLLN head; // DLLN = Doubly LinkedList Node  **private** DLLN tail;  **private** Map<Integer, DLLN> map;  **private** **final** **int** capacity;  **private** **int** size;    **public** **LRUCache**(**int** capacity) {  **this**.capacity = capacity;  **this**.map = **new** HashMap<>();  **this**.head = **new** DLLN(0,0);  **this**.tail = **new** DLLN(0,0);  head.next = tail;  tail.prev = head;  size = 0;  }    **public** **int** **get**(**int** key) {  **if** (!map.containsKey(key)) **return** -1;  DLLN ret = map.get(key);  deleteNodeFromChain(ret);  moveToHead(ret);  **return** ret.val;   }    **public** **void** **put**(**int** key, **int** value) {  DLLN node = **new** DLLN(key,value);  **if** (!map.containsKey(key)) {  **if** (size >= capacity) {  map.remove(tail.prev.label);  deleteNodeFromChain(tail.prev);  size--;  }  size++;  } **else** {  deleteNodeFromChain(map.get(key));  }  map.put(key,node);  moveToHead(node);  }    **private** **void** **deleteNodeFromChain**(DLLN node) {  node.prev.next = node.next;  node.next.prev = node.prev;  }    **private** **void** **moveToHead**(DLLN node) {  node.next = head.next;  node.prev = head;  head.next = node;  node.next.prev = node;  }    **class** **DLLN** {  **int** label;  **int** val;  DLLN prev;  DLLN next;  DLLN (**int** label, **int** val) {  **this**.label = label;  **this**.val = val;  **this**.prev = **null**;  **this**.next = **null**;  }  } } |
| --- |

1. Given a String, find first non repeating character/first repeating/second non repeating/ second repeating
2. L387 Given Streamflow, find first non repeating character with O(1) time

| **public** **int** **firstUniqChar**(String s) {  **int**[] chars = **new** **int**[26];  Arrays.fill(chars, -1);  **for** (**int** i = 0; i < s.length(); i++) {  **int** idx = s.charAt(i) - 'a';  chars[idx] = chars[idx] == -1 ? i : -2;  }  **int** idx = Integer.MAX\_VALUE;  **for** (**int** i = 0; i < 26; i++) {  **if** (chars[i] >= 0) {  idx = Math.min(idx, chars[i]);  }  }  **return** idx == Integer.MAX\_VALUE ? -1 : idx;  } |
| --- |

1. https://www.geeksforgeeks.org/find-first-non-repeating-character-stream-characters/
2. L41 first missing positive

| **public** **int** **firstMissingPositive**(**int**[] nums) {  **int** len = nums.length, temp = 0;  **for** (**int** i = 0; i < len; i++) {  **while** (nums[i] != i + 1 && nums[i] > 0 && nums[i] <= len && nums[i] != nums[nums[i] - 1]) {  temp = nums[i];  nums[i] = nums[temp - 1];  nums[temp - 1] = temp;  }  }    **for** (**int** i = 0; i < len; i++) {  **if** (nums[i] != i + 1) **return** i + 1;  }  **return** len + 1;  } |
| --- |

1. Longest common substring /subarray/subsequence length between two string

| */\* 可以用dp做 \*/* **public** |
| --- |

1. L300. Longest ascending/descending/consecutive subarray/subsequence

| */\* Given an unsorted array of integers, find the length of longest increasing subsequence. For example, Given [10, 9, 2, 5, 3, 7, 101, 18], The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length. Your algorithm should run in O(n2) complexity. Follow up: Could you improve it to O(n log n) time complexity? \*/* **class** **Solution** {  **public** **int** **lengthOfLIS**(**int**[] nums) {  **if**(nums == **null** || nums.length == 0) **return** 0;  **int** max = 0;  List<Integer> list = **new** ArrayList<Integer>();  **for**(**int** i = 0; i < nums.length; i++){  **if**(list.size() == 0){  list.add(nums[i]);  }**else**{  **if**(i - 1 >= 0 && nums[i] > list.get(list.size() - 1)){  list.add(nums[i]);  }**else**{  **int** idx = findLargestSmaller(list, nums[i]);  list.set(idx+1, nums[i]);  }  }  }  **return** list.size();  }  **private** **int** **findLargestSmaller**(List<Integer> list, **int** target){  **int** left = 0;  **int** right = list.size();  **int** mid = 0;  **while**(left <= right){  mid = left + (right - left)/2;  **if**(list.get(mid) < target){  left = mid + 1;  }**else**{  right = mid - 1;  }  }  **return** left - 1;  } } |
| --- |

1. L149 Max point on a line(面试高频题)

| */\* Given n points on a 2D plane, find the maximum number of points that lie on the same straight line. \*/*  **public** **int** **maxPoints**(**int**[][] points) {  **if** (points == **null**) **return** 0;  **if** (points.length <= 2) **return** points.length;   Map<Integer, Map<Integer, Integer>> map = **new** HashMap<>();  **int** result = 0;  **for** (**int** i = 0; i < points.length; i++) {  map.clear();  **int** overlap = 1, max = 0;  **for** (**int** j = i + 1; j < points.length; j++) {  **int** x = points[j][0] - points[i][0];  **int** y = points[j][1] - points[i][1];  **if** (x == 0 && y == 0) {  overlap++;  **continue**;  }   **int** gcd = generateGCD(x, y);  x /= gcd;  y /= gcd;   Map<Integer, Integer> ymap = map.computeIfAbsent(x, f -> **new** HashMap<>());  ymap.put(y, ymap.getOrDefault(y, 0) + 1);  max = Math.max(max, map.get(x).get(y));  }  result = Math.max(result, max + overlap);  }  **return** result;  }   **private** **int** **generateGCD**(**int** a, **int** b) {  **return** b == 0 ? a : generateGCD(b, a % b);  } |
| --- |

1. L169 Majority number ½, ⅓, 1/k.

# Session 18 Probability and Randomanization

1. Q1 given a biased coin, face up with p, face down with 1-p
2. 如何通过random 5实现random 7
3. Q3 能否用random2去生成所有的random(n)
4. implement shuffle() in out card desig
5. Google phone interview

public boolean isStringPattern(String s, int p){

if(s == null)

}