Online Interview Questions - Problems Wise

deepwaterooo

February 7, 2015

Contents

Chapter 1

4 Sum, wrong answers

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, $a \leqslant b \leqslant c \leqslant d$)

The solution set must not contain duplicate quadruplets.

Example

For example, given array S = {1 0 -1 0 -2 2}, and target = 0. A solution set is:

```
(-1, 0, 0, 1)
(-2, -1, 1, 2)
(-2, 0, 0, 2)
```

```
/**
 * @param numbers : Give an array numbersbers of n integer
 * @param target : you need to find four elements that's sum of target
 * @return : Find all unique quadruplets in the array which gives the sum of
 *           zero.
 */
public ArrayList<ArrayList<Integer>> fourSum(int[] numbers, int target) {
    int n = numbers.length;
    HashSet<ArrayList<Integer>> set = new HashSet<ArrayList<Integer>>();
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (n < 4) return res;
    Arrays.sort(numbers);
    int x = 0, y = 0, tmp = 0;
    for (int i = 0; i < n - 3; i++) {
        if (i > 0 && numbers[i] == numbers[i - 1]) continue;
        for (int j = i + 1; j < n - 2; j++) {
            x = j + 1;
            y = n - 1;
            while (x < y) {
                tmp = target - numbers[i] - numbers[j];
                if (numbers[x] + numbers[y] == tmp) {
                    set.add(new ArrayList<Integer>(Arrays.asList(numbers[i], numbers[j]
                    x++;
                    y--;
                }
                else if (numbers[x] + numbers[y] < tmp) x++;
                else y--;
            }
        }
    }
    res.addAll(set);
    return res;
}
```

Chapter 2

2 Sum: O(1) Space, O(nlogn) Time undone

25% Accepted

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

Note

You may assume that each input would have exactly one solution

Example

numbers=[2, 7, 11, 15], target=9

return [1, 2]

Challenge

2.1 O(1) Space, O(nlogn) Time

2.2 O(n) Space, O(n) Time

```java
public int[] twoSum(int[] numbers, int target) { // O(nlogn), O(1)
    Arrays.sort(numbers);
    int [] res = new int[2];
    int i = 0, j = numbers.length - 1;
    while (i < j) {
        if (numbers[i] + numbers[j] == target) {
            res[0] = i + 1;
            res[1] = j + 1;
            return res;
        }
        if (numbers[i] + numbers[j] < target) i++;
        else j--;
    }
    return res;
}
```

Chapter 3

3 Sum

19% Accepted

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.

Note

Elements in a triplet (a,b,c) must be in non-descending order. (ie, $a \leqslant b \leqslant c$)

The solution set must not contain duplicate triplets.

Example

For example, given array S = {-1 0 1 2 -1 -4}, A solution set is:

```
(-1, 0, 1)
(-1, -1, 2)
```

```
public ArrayList<ArrayList<Integer>> threeSum(int[] numbers) {
    int n = numbers.length;
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    HashSet<List<Integer>> set = new HashSet<List<Integer>>();
    Integer [] one = new Integer[3];
    int k = 0;
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            k = j + 1;
            while (k < n - 1 && numbers[i] + numbers[j] + numbers[k] != 0) k++;
            if (k <= n - 1 && numbers[i] + numbers[j] + numbers[k] == 0) {
                one[0] = numbers[i];
                one[1] = numbers[j];
                one[2] = numbers[k];
                Arrays.sort(one);
                set.add(new ArrayList<Integer>(Arrays.asList(one)));
                k++;
            }
        }
    }
    for(List<Integer> i : set)
        res.add(new ArrayList(i));
    return res;
}
```

Chapter 4

3 Sum Closest

30% Accepted

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers.

Note

You may assume that each input would have exactly one solution.

Example

For example, given array S = {-1 2 1 -4}, and target = 1. The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

```
public int threeSumClosest(int[] numbers ,int target) {
    int n = numbers.length;
    int res = Integer.MAX_VALUE;
    int k = 0;
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            k = j + 1;
            while (k <= n - 1) {
                if (Math.abs(numbers[i] + numbers[j] + numbers[k] - target) < Math.abs(
                    res = numbers[i] + numbers[j] + numbers[k];
                k++;
            }
        }
    }
    return res;
}
```

Chapter 5

A + B Problem, no idea;;;

60% Accepted
For given numbers a and b in function aplusb, return the sum of them.
Note
You don't need to parse the input and output. Just calculate and return.
Example
If a = 1 and b = 2 return 3
Challenge
Can you do it without + operation?
Clarification
Are a and b both 32-bit integers?

- Yes.

Chapter 6

Anagrams My Submissions

28% Accepted
Given an array of strings, return all groups of strings that are anagrams.
Note
All inputs will be in lower-case
Example

```
Given a string list: ["lint","intl","inlt","code"]
return ["lint","inlt","intl"]

public String mySort(String s) {
    char [] tmp = s.toCharArray();
    Arrays.sort(tmp);
    return new String(tmp);   //tmp.toString(); doesn't work
}

public List<String> anagrams(String[] strs) {
    Map<String, List<Integer>> m = new HashMap<String, List<Integer>>();
    ArrayList<String> res = new ArrayList<String>();
    for (int i = 0; i < strs.length; i++) {
        String tmp = mySort(strs[i]);
        if (!m.containsKey(tmp))
            m.put(tmp, new ArrayList<Integer>(Arrays.asList(i)));
        else m.get(tmp).add(i);
    }
    for (String key : m.keySet())
        if (m.get(key).size() > 1)
            for (int i = 0; i < m.get(key).size(); i++)
                res.add(strs[m.get(key).get(i)]);
    return res;
}
```

Chapter 7

Backpack: still feeling difficult for me now ...

17% Accepted

Given n items with size A[i], an integer m denotes the size of a backpack. How full you can fill this backpack?

Note

You can not divide any item into small pieces.

Example

If we have 4 items with size [2, 3, 5, 7], the backpack size is 11, we can select 2, 3 and 5, so that the max size we can fill this backpack is 10. If the backpack size is 12. we can select [2, 3, 7] so that we can fulfill the backpack.

You function should return the max size we can fill in the given backpack.

Chapter 8

Balanced Binary Tree

46% Accepted

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example

Given binary tree A={3,9,20,#,#,15,7}, B={3,#,20,15,7}

```
A)  3            B)    3
   / \                  \
  9  20                 20
      /  \               / \
     15   7             15  7
```

The binary tree A is a height-balanced binary tree, but B is not.

Chapter 9

Best Time to Buy and Sell Stock Show Result

45% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example

Given an example [3,2,3,1,2], return 1

```
public int maxProfit(int[] prices) {
    if (prices.length == 0) return 0;
    int n = prices.length;
    int [] buy = new int[n];
    buy[0] = prices[0];
    for (int i = 1; i < n; i++)
        buy[i] = Math.min(buy[i - 1], prices[i]);
    int [] sell = new int[n];
    sell[n - 1] = prices[n - 1];
    int res = Integer.MIN_VALUE;
    for (int i = n - 2; i >= 0; i--) {
        sell[i] = Math.max(sell[i + 1], prices[i]);
        res = Math.max(res, sell[i] - buy[i]);
    }
    return res;
}
```

Chapter 10

Best Time to Buy and Sell Stock II

62% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example

Given an example [2,1,2,0,1], return 2

Chapter 11

Best Time to Buy and Sell Stock III

25% Accepted

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

Note

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example

Given an example [4,4,6,1,1,4,2,5], return 6

Chapter 12

Binary Representation

6% Accepted

Given a (decimal - e g 3.72) number that is passed in as a string,return the binary representation that is passed in as a string.If the number can not be represented accurately in binary, print “ERROR”

Example

n = 3.72, return ERROR

n = 3.5, return 11.1

Chapter 13

Binary Search My Submissions: arr.length > Integer.MAX$_{VALUE}$? don't understand ...

27% Accepted

Binary search is a famous question in algorithm.

For a given sorted array (ascending order) and a target number, find the first index of this number in O(log n) time complexity.

If the target number does not exist in the array, return -1.

Example

If the array is [1, 2, 3, 3, 4, 5, 10], for given target 3, return 2.

Challenge

If the count of numbers is bigger than MAXINT, can your code work properly?

```
/**
 * @param nums: The integer array.
 * @param target: Target to find.
 * @return: The first position of target. Position starts from 0.
 */
public int binarySearch(int[] nums, int target) {
    if (target < nums[0] || target > nums[nums.length - 1]) return -1;
    int bgn = 0, end = nums.length - 1;
    if (bgn == end - 1) {
        if (target == nums[bgn]) return bgn;
        else if (target == nums[end]) return end;
        else return -1;
    }
    while (bgn < end - 1) {
        int mid1 = bgn + (end - bgn) / 2;
        int mid2 = mid1 + 1;
        if (target < nums[mid1]) end = mid1 - 1;
        else if (target > nums[mid2]) bgn = mid2 + 1;
        else if (target == nums[mid1]) end = mid1;
        else if (target == nums[mid2] && nums[mid1] < nums[mid2]) bgn = mid2;
    }
    if (bgn == end - 1) {
        if (target == nums[bgn]) return bgn;
        else if (target == nums[end]) return end;
        else return -1;
    } else if (bgn == end)
        return nums[bgn] == target ? bgn : -1;
    else return -1;
}
```

Chapter 14

Binary Tree Inorder Traversal: Iterative undone···

37% Accepted

Given a binary tree, return the inorder traversal of its nodes' values.

Example

Given binary tree {1,#,2,3},

```
1
 \
  2
 /
3
```

return [1,3,2].

Challenge

Can you do it without recursion?

```java
public void inorderTraversal(TreeNode root, ArrayList<Integer> res) {
    if (root == null) return;
    inorderTraversal(root.left, res);
    res.add(root.val);
    inorderTraversal(root.right, res);
}

public ArrayList<Integer> inorderTraversal(TreeNode root) {
    ArrayList<Integer> res = new ArrayList<Integer>();
    inorderTraversal(root, res);
    return res;
}
```

Chapter 15

Binary Tree Level Order Traversal Show Result My Submissions

33% Accepted

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

Example

Given binary tree {3,9,20,#,#,15,7},

```
  3
 / \
9  20
  /  \
 15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

Challenge

Using only 1 queue to implement it.

```
public ArrayList<ArrayList<Integer>> levelOrder(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
                q.add(curr);
                continue;
            } else {
                res.remove(res.size() - 1);
                return res;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            res.get(n).add(curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 16

Binary Tree Level Order Traversal II

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

Example

Given binary tree {3,9,20,#,#,15,7},

```
  3
 / \
9  20
  /  \
 15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

```
public ArrayList<ArrayList<Integer>> levelOrderButtom(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
                q.add(curr);
                continue;
            } else {
                res.remove(res.size() - 1);
                ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>
                for (int i = res.size() - 1; i >= 0; i--)
                    result.add(new ArrayList<Integer>(res.get(i)));
                return result;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            res.get(n).add(curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 17

Binary Tree Maximum Path Sum: some part missing….

23% Accepted

Given a binary tree, find the maximum path sum.

The path may start and end at any node in the tree.

Example

Given the below binary tree,

```
  1
 / \
2   3
```

Return 6.

Chapter 18

Binary Tree Zigzag Level Order Traversal

26% Accepted

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

Example

Given binary tree {3,9,20,#,#,15,7},

```
  3
 / \
9  20
  /  \
 15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

```java
public ArrayList<ArrayList<Integer>> zigzagLevelOrder(TreeNode root) {
    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
    if (root == null) return  res;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(null);
    q.add(root);
    TreeNode curr = root;
    TreeNode prev = null;
    int n = 0;
    int cnt = 0;
    while (!q.isEmpty()) {
        prev = curr;
        curr = q.poll();
        if (curr == null) {
            if (prev != curr) {
                res.add(new ArrayList<Integer>());
                q.add(curr);
                ++cnt;
                continue;
            } else {
                res.remove(res.size() - 1);
                return res;
            }
        }
        if (curr != null) {
            n = res.size() - 1;
            if (cnt % 2 == 1)
                res.get(n).add(curr.val);
            else
                res.get(n).add(0, curr.val);
            if (curr.left != null) q.add(curr.left);
            if (curr.right != null) q.add(curr.right);
        }
    }
    return res;
}
```

Chapter 19

Climbing Stairs

40% Accepted
You are climbing a stair case. It takes n steps to reach to the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?
Example
Given an example n=3 , 1+1+1=2+1=1+2=3
return 3

```
public int climbStairs(int n) {
    int [] res = new int[n];
    res[0] = 1;
    if (n < 2) return 1;
    res[1] = 2;
    for (int i = 2; i < n; i++)
        res[i] = res[i - 1] + res[i - 2];
    return res[n - 1];
}
```

Chapter 20

Combination Sum

26% Accepted

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

For example, given candidate set 2,3,6,7 and target 7, A solution set is:

```
[7]
[2, 2, 3]
```

Note

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).

The solution set must not contain duplicate combinations.

Example

given candidate set 2,3,6,7 and target 7,

A solution set is:

```
[7]
[2, 2, 3]
```

```
public void combinationSum(int [] candidates, int gap, int idx,
                           List<List<Integer>> res, List<Integer> path) {
    if (gap == 0)
        res.add(new ArrayList<Integer>(path));
    for (int i = idx; i < candidates.length; i++) {
        if (candidates[i] <= gap) {
            path.add(candidates[i]);
            combinationSum(candidates, gap - candidates[i], i, res, path);
            path.remove(path.size() - 1);
        }
    }
}

public List<List<Integer>> combinationSum(int[] candidates, int target) {
    int n = candidates.length;
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
    Arrays.sort(candidates);
    combinationSum(candidates, target, 0, res, path);
    return res;
}
```

Chapter 21

Combination Sum II

24% Accepted

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).

The solution set must not contain duplicate combinations.

Example

For example, given candidate set 10,1,6,7,2,1,5 and target 8,

A solution set is:

```
[1,7]
[1,2,5]
[2,6]
[1,1,6]
```

```
public void combinationSum2(int [] candidates, int gap, int idx,
                            List<List<Integer>> res, List<Integer> path,
                            boolean [] used) {
    if (gap == 0)
        res.add(new ArrayList<Integer>(path));
    for (int i = idx; i < candidates.length; i++) {
        if (i > 0 && candidates[i] == candidates[i - 1] && !used[i - 1]) continue;
        if (candidates[i] <= gap) {
            used[i] = true;
            path.add(candidates[i]);
            combinationSum2(candidates, gap - candidates[i], i + 1, res, path, used);
            path.remove(path.size() - 1);
            used[i] = false;
        }
    }
}

public List<List<Integer>> combinationSum2(int[] candidates, int target) {
    int n = candidates.length;
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
    Arrays.sort(candidates);
    boolean [] used = new boolean[n];
    combinationSum2(candidates, target, 0, res, path, used);
    return res;
}
```

Chapter 22

Combinations

31% Accepted

Given two integers n and k, return all possible combinations of k numbers out of 1 ⋯ n.

Example

For example,

If n = 4 and k = 2, a solution is:

```
[[2,4],[3,4],[2,3],[1,2],[1,3],[1,4]]
```

```
public void combine(int n, int k, int idx, List<Integer> src, List<Integer> path,
                    List<List<Integer>> res) {
    if (path.size() == k) {
        List<Integer> one = new ArrayList<Integer>(path);
        Collections.sort(one);
        res.add(new ArrayList(one));
        return;
    }
    for (int i = idx; i < n; i++) {
        path.add(src.get(i));
        combine(n, k, i + 1, src, path, res);
        path.remove(path.size() - 1);
    }
}

public List<List<Integer>> combine(int n, int k) {
    List<Integer> src = new ArrayList<Integer>();
    for (int i = 0; i < n; i++)
        src.add(i + 1);
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    List<Integer> path = new ArrayList<Integer>();
    combine(n, k, 0, src, path, res);
    return res;
}
```

Chapter 23

Compare Strings

32% Accepted
Compare two strings A and B, determine whether A contains all of the characters in B.
The characters in string A and B are all Upper Case letters.
Example
For A = "ABCD", B = "ABC", return true.
For A = "ABCD" B = "AABC", return false.

```java
public boolean compareStrings(String a, String b) {
    if (b == null) return a == null;
    if (a.length() < b.length()) return false;
    Map<Character, Integer> bm = new HashMap<Character, Integer>();
    Map<Character, Integer> am = new HashMap<Character, Integer>();
    for (int i = 0; i < b.length(); i++) {
        if (!bm.containsKey(b.charAt(i)))
            bm.put(b.charAt(i), 1);
        else bm.put(b.charAt(i), bm.get(b.charAt(i)) + 1);
    }
    for (int i = 0; i < a.length(); i++) {
        if (!am.containsKey(a.charAt(i)))
            am.put(a.charAt(i), 1);
        else am.put(a.charAt(i), am.get(a.charAt(i)) + 1);
    }
    for (Character key : bm.keySet())
        if (!am.containsKey(key) || am.get(key) < bm.get(key)) return false;
    return true;
}
```

Chapter 24