

19/11/2024

DSA PRACTICE DAY 7

1. Next Permutation:

```
public void nextPermutation(int[] nums) {
    int ind1=-1;
    int ind2=-1;
    for(int i=nums.length-2;i>=0;i--){
        if(nums[i]<nums[i+1]){
            ind1=i;
            break;
        }
    }
    if(ind1==-1){
        reverse(nums,0);
    }
    else{
        for(int i=nums.length-1;i>=0;i--){
            if(nums[i]>nums[ind1]){
                ind2=i;
                break;
            }
        }
        swap(nums,ind1,ind2);
        reverse(nums,ind1+1);
    }
}

void swap(int[] nums,int i,int j){
    int temp=nums[i];
    nums[i]=nums[j];
    nums[j]=temp;
}

void reverse(int[] nums,int start){
    int i=start;
    int j=nums.length-1;
    while(i<j){
        swap(nums,i,j);
        i++;
        j--;
    }
}
```

```
1 class Solution {
2     public void nextPermutation(int[] nums) {
3         int ind1=-1;
4         int ind2=-1;
5         for(int i=nums.length-2;i>=0;i--){
6             if(nums[i]<nums[i+1]){
7                 ind1=i;
8                 break;
9             }
10        }
11        if(ind1==-1){
12            reverse(nums,0);
13        }
14        else{
15            for(int i=nums.length-1;i>=0;i--){
16                if(nums[i]>nums[ind1]){
17                    ind2=i;
18                    break;
19                }
20            }
21            swap(nums,ind1,ind2);
22            reverse(nums,ind1+1);
23        }
24    }
25    void swap(int[] nums,int i,int j){
26        int temp=nums[i];
27        nums[i]=nums[j];
28        nums[j]=temp;
29    }
30    void reverse(int[] nums,int start){
31        int i=start;
32        int j=nums.length-1;
33        while(i<j){
34            swap(nums,i,j);
35            i++;
36            j--;
37        }
38    }
39 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =

[1,2,3]

Output

[1,3,2]

Expected

[1,3,2]

Contribute a testcase

2. Spiral Matrix

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int row = matrix.length;
        int col = matrix[0].length;
        int x = 0;
        int y = 0;
        int dx = 1;
        int dy = 0;
        List<Integer> res = new ArrayList<>();
        for (int i = 0; i < row * col; i++) {
            res.add(matrix[y][x]);
            matrix[y][x] = -101;
            if ((0 <= x + dx && x + dx < col && 0 <= y + dy && y + dy < row) && !matrix[y+dy][x+dx]) {
                int temp = dx;
                dx = -dy;
                dy = temp;
            }
            x += dx;
            y += dy;
        }
        return res;
    }
}
```

```
1 class Solution {
2     public List<Integer> spiralOrder(int[][] matrix) {
3         int row = matrix.length;
4         int col = matrix[0].length;
5         int x = 0;
6         int y = 0;
7         int dx = 1;
8         int dy = 0;
9         List<Integer> res = new ArrayList<>();
10        for (int i = 0; i < row * col; i++) {
11            res.add(matrix[y][x]);
12            matrix[y][x] = -101;
13            if ((0 <= x + dx && x + dx < col && 0 <= y + dy && y + dy < row) && !matrix[y+dy][x+dx]) {
14                int temp = dx;
15                dx = -dy;
16                dy = temp;
17            }
18            x += dx;
19            y += dy;
20        }
21        return res;
22    }
23 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

matrix =

[[1,2,3],[4,5,6],[7,8,9]]

Output

[1,2,3,6,9,8,7,4,5]

Expected

[1,2,3,6,9,8,7,4,5]

3. Longest substring without repeating characters

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int left = 0;
        int maxLength = 0;
        HashSet<Character> charSet = new HashSet<>();
        for (int right = 0; right < s.length(); right++) {
            while (charSet.contains(s.charAt(right))) {
                charSet.remove(s.charAt(left));
                left++;
            }
            charSet.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```

```
1 class Solution {
2     public int lengthOfLongestSubstring(String s) {
3         int left = 0;
4         int maxLength = 0;
5         HashSet<Character> charSet = new HashSet<>();
6         for (int right = 0; right < s.length(); right++) {
7             while (charSet.contains(s.charAt(right))) {
8                 charSet.remove(s.charAt(left));
9                 left++;
10            }
11            charSet.add(s.charAt(right));
12            maxLength = Math.max(maxLength, right - left + 1);
13        }
14        return maxLength;
15    }
16 }
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

s =

"abcabcbb"

Output

3

Expected

3

4. Remove linked list elements

The screenshot shows a coding challenge interface. On the left, the code editor displays a Java solution for removing elements from a linked list. The code uses a dummy node and a while loop to traverse the list and remove nodes with values equal to 'val'. On the right, the 'Test Result' panel shows the test case 'Case 1' as 'Accepted' with a runtime of 0 ms. The input is a linked list with values [1, 2, 6, 3, 4, 5, 6] and val = 6. The output is [1, 2, 3, 4, 5], which matches the expected result.

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode ans = new ListNode(0, head);
        ListNode dummy = ans;

        while (dummy != null) {
            while (dummy.next != null && dummy.next.val == val) {
                dummy.next = dummy.next.next;
            }
            dummy = dummy.next;
        }
        return ans.next;
    }
}
```

More challenges

- 237. Delete Node in a Linked List
- 2095. Delete the Middle Node of a Linked List
- 3263. Convert Doubly Linked List to Array I

5. Palindrome linked list

The screenshot shows a coding challenge interface. On the left, the code editor displays a Java solution for checking if a linked list is a palindrome. The solution converts the linked list into an ArrayList and then uses two pointers (left and right) to compare elements from both ends. On the right, the 'Test Result' panel shows the test case 'Case 1' as 'Accepted' with a runtime of 0 ms. The input is a linked list with values [1, 2, 2, 1], and the output is true, which matches the expected result.

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        List<Integer> list = new ArrayList();
        while (head != null) {
            list.add(head.val);
            head = head.next;
        }

        int left = 0;
        int right = list.size() - 1;
        while (left < right && list.get(left) == list.get(right)) {
            left++;
            right--;
        }
        return left >= right;
    }
}
```

6. Minimum path sum

The screenshot shows a coding challenge interface. On the left, the code editor displays a Java solution for finding the minimum path sum in a grid. The solution uses dynamic programming, calculating the minimum path sum for each cell based on the values from the top and left cells. On the right, the 'Test Result' panel shows the test case 'Case 1' as 'Accepted' with a runtime of 0 ms. The input is a grid with values [[1, 3, 1], [1, 5, 1], [4, 2, 1]], and the output is 7, which matches the expected result.

```
class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;

        int[] previousRow = grid[0];

        for (int j = 1; j < n; j++) previousRow[j] += previousRow[j - 1];

        for (int i = 1; i < m; i++) {
            int[] row = grid[i];
            previousRow[0] += row[0];
            for (int j = 1; j < n; j++) previousRow[j] = row[j] + Math.min(previousRow[j - 1], previousRow[j]);
        }
        return previousRow[n - 1];
    }
}
```

7. Validate binary search tree

The screenshot shows a coding challenge interface. On the left, the code editor displays a Java solution for validating a binary search tree. The solution uses a recursive approach, checking if the root is null and then recursively validating the left and right subtrees. On the right, the 'Test Result' panel shows the test case 'Case 1' as 'Accepted' with a runtime of 0 ms. The input is a binary tree with root = [2, 1, 3], and the output is true, which matches the expected result.

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        if (root == null) return true;
        Deque<TreeNode> stack = new LinkedList<>();
        TreeNode prev = null;
        boolean onRightSideOfPrev = false;
        while (root != null || !stack.isEmpty()) {
            while (root != null) {
                stack.push(root);
                root = root.left;
            }
            root = stack.pop();
            if (prev != null && ((!onRightSideOfPrev && prev.val > root.val) || (onRightSideOfPrev && prev.val < root.val))) return false;
            prev = root;
            root = root.right;
            onRightSideOfPrev = root == null ? false : true;
        }
        return true;
    }
}
```

8. Word ladder

Program:

```
class Solution {  
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {  
        Set<String> wordSet = new HashSet<>(wordList);  
        if (!wordSet.contains(endWord)) return 0;  
        Queue<String> queue = new LinkedList<>();  
        queue.offer(beginWord);  
        Set<String> visited = new HashSet<>();  
        visited.add(beginWord);  
        int length = 1;  
        while (!queue.isEmpty()) {  
            int levelSize = queue.size();  
            for (int i = 0; i < levelSize; i++) {  
                String currentWord = queue.poll();  
                if (currentWord.equals(endWord)) return length;  
                for (String neighbor : getNeighbors(currentWord, wordSet)) {  
                    if (!visited.contains(neighbor)) {  
                        visited.add(neighbor);  
                        queue.offer(neighbor);  
                    }  
                }  
            }  
            length++;  
        }  
        return 0;  
    }  
    private List<String> getNeighbors(String word, Set<String> wordSet) {  
        List<String> neighbors = new ArrayList<>();  
        char[] wordChars = word.toCharArray();  
        for (int i = 0; i < wordChars.length; i++) {  
            char originalChar = wordChars[i];  
            for (char c = 'a'; c <= 'z'; c++) {  
                if (c == originalChar) continue;  
                wordChars[i] = c;  
                String neighbor = new String(wordChars);  
                if (wordSet.contains(neighbor) && !visited.contains(neighbor)) {  
                    neighbors.add(neighbor);  
                }  
            }  
            wordChars[i] = originalChar;  
        }  
        return neighbors;  
    }  
}
```

```

        wordChars[i] = c;

        String transformedWord = new String(wordChars);

        if (wordSet.contains(transformedWord)) {
            neighbors.add(transformedWord);
        }
    }

    wordChars[i] = originalChar;
}

return neighbors;
}
}

```

Output:

```

beginWord =
    "hit"

endWord =
    "cog"

wordList =
    ["hot", "dot", "dog", "lot", "log", "cog"]

```

9. Word ladder

Program:

```
class Solution {  
    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {  
        Map<String,Integer> hm = new HashMap<>();  
        List<List<String>> res = new ArrayList<>();  
  
        Queue<String> q = new LinkedList<>();  
        q.add(beginWord);  
        hm.put(beginWord,1);  
  
        HashSet<String> hs = new HashSet<>();  
        for(String w : wordList) hs.add(w);  
        hs.remove(beginWord);  
        while(!q.isEmpty()){  
            String word = q.poll();  
            if(word.equals(endWord)){  
                break;  
            }  
  
            for(int i=0;i<word.length();i++){  
                int level = hm.get(word);  
                for(char ch='a';ch<='z';ch++){  
                    char[] replaceChars = word.toCharArray();  
                    replaceChars[i] = ch;  
                    String replaceString = new String(replaceChars);  
  
                    if(hs.contains(replaceString)){  
                        q.add(replaceString);  
                        hm.put(replaceString,level+1);  
                        hs.remove(replaceString);  
                    }  
                }  
            }  
        }  
    }  
}
```

```

    }

    if(hm.containsKey(endWord) == true){
        List<String> seq = new ArrayList<>();
        seq.add(endWord);
        dfs(endWord,seq,res,beginWord,hm);
    }

    return res;
}

public void dfs(String word,List<String> seq,List<List<String>> res,String
beginWord,Map<String,Integer> hm){
    if(word.equals(beginWord)){
        List<String> ref = new ArrayList<>(seq);
        Collections.reverse(ref);
        res.add(ref);
        return;
    }

    int level = hm.get(word);
    for(int i=0;i<word.length();i++){
        for(char ch ='a';ch<='z';ch++){
            char replaceChars[] = word.toCharArray();
            replaceChars[i] = ch;
            String replaceStr = new String(replaceChars);

            if(hm.containsKey(replaceStr) && hm.get(replaceStr) == level-1){
                seq.add(replaceStr);
                dfs(replaceStr,seq,res,beginWord,hm);
                seq.remove(seq.size()-1);
            }
        }
    }
}

```

```
}
```

Output:

Case 1

Case 2

+

beginWord =

"hit"

endWord =

"cog"

wordList =

["hot","dot","dog","lot","log","cog"]

Program:

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
class Solution {
```

```
    private List<Integer> orderList;
```

```
    public int[] findOrder(int numCourses, int[][] prerequisites) {
```

```
        orderList = new ArrayList<>();
```

```
        if(canFinish(numCourses, prerequisites)) {
```

```
            return orderList.stream().mapToInt(i -> i).toArray();
```

```
        } else {
```

```
            return new int[]{};
```

```
        }
```

```
    }
```

```
    public boolean canFinish(int numCourses, int[][] prerequisites) {
```

```
        ArrayList<Integer>[] adj = new ArrayList[numCourses];
```

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

```
            adj[i] = new ArrayList<>();
```

```
        }
```

```
        for (int[] pre : prerequisites) {
```

```
            adj[pre[0]].add(pre[1]);
```

```
        }
```

```
        int[] visited = new int[numCourses];
```

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

```
            if (!dfs(i, visited, adj)) {
```

```
                return false;
```

```
            }
```

```
        }
```



```

        return true;
    }

    public boolean dfs(int node, int[] visited, ArrayList<Integer>[] adj) {
        if (visited[node] == 1) {
            return false;
        }

        if (visited[node] == 2) {
            return true;
        }

        visited[node] = 1;

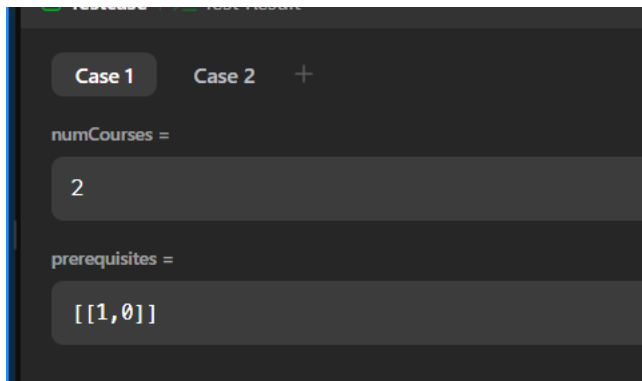
        for (int n : adj[node]) {
            if (!dfs(n, visited, adj)) {
                return false;
            }
        }

        visited[node] = 2;
        orderList.add(node);

        return true;
    }
}

```

Output:



11. Design tic tac toe:

Program:

```

class Solution {
    public boolean validTicTacToe(String[] board) {
        int[] arr=new int[2];

        boolean xwin=false,owin=false;

        int xdiag=0,odiag=0;

        for(int i=0;i<3;i++){

            int x=0,o=0;

            for(int j=0;j<3;j++){

                {

                    if(i==0){

                        if(board[i].charAt(j)=='X' && board[i+1].charAt(j)=='X' &&
board[i+2].charAt(j)=='X')xwin=true;

                        if(board[i].charAt(j)=='O' && board[i+1].charAt(j)=='O' &&
board[i+2].charAt(j)=='O')owin=true;

                    }

                    if(board[i].charAt(j)=='X'){

                        if(i==j)xdiag++;

                        arr[1]++;

                        x++;

                    }

                    else if(board[i].charAt(j)=='O'){

                        if(i==j)odiag++;

                        arr[0]++;

                        o++;

                    }

                }

            }

        }
    }
}

```

```

    }

    if(o==3 && owin)return false;

    if(owin && xwin)return false;

    if(x==3)xwin=true;

    else if(o==3)owin=true;

}

if(xdiag==3)xwin=true;

if(oddiag==3)owin=true;

if(board[0].charAt(2)=='X' && board[1].charAt(1)=='X' && board[2].charAt(0)=='X')xwin=true;

if(board[0].charAt(2)=='O' && board[1].charAt(1)=='O' && board[2].charAt(0)=='O')owin=true;

if(arr[0]>=arr[1] && xwin || (arr[1]>arr[0] && owin))return false;

if(arr[0]>arr[1] || Math.abs(arr[0]-arr[1])>1)return false;

if(xwin&&owin) return false;

else return true;

}

}

```

Output:

