Coding practice Problems(19/11/2024)

Q 1) Next Permutation

```
class Solution {
    public void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i>=0 && nums[i] >= nums[i + 1]){
            i--;
        }
        if (i != -1) {
            int j= nums.length-1;
            while (j>=0 && nums[i] >= nums[j]) {
                j--;
            }
            swap(nums, i, j);
        }
        int start = i + 1;
        int end = nums.length - 1;
        while (start < end) {
            swap(nums, start, end);
            start++;
            end--;
        }
    }
    public static void swap(int[] nums, int a, int b) {
        int temp = nums[a];
        nums[a] = nums[b];
        nums[b] = temp;
```

```
    }
}
```

Output:

Q 2) Spiral Matrix

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int top= 0;
        int bottom = matrix.length-1;
        int left = 0;
        int right = matrix[0].length-1;

        Integer[] arr = new Integer[matrix.length *  matrix[0].length];
        int runningIndex= 0;
        while(runningIndex!= arr.length){
            for(int n = left; n <= right && (runningIndex!= arr.length); n++){
                arr[runningIndex] = matrix[top][n];
                runningIndex++;
            }
            for(int n = top; n <bottom && (runningIndex!= arr.length); n++){
                arr[runningIndex] = matrix[n+1][right];
                runningIndex++;
```

```
        }
        for(int n = right-1; n>= left && (runningIndex!= arr.length); n--){
            arr[runningIndex]=matrix[bottom][n];
            runningIndex++;
        }
        for(int n = bottom-1; n >= top+1 && (runningIndex!= arr.length); n--){
            arr[runningIndex] = matrix[n][left];
            runningIndex++;
        }
        top++;
        bottom--;
        left++;
        right--;
      }
      return Arrays.asList(arr);
   }
}
```

Output:

Q 3) Longest Substring without Repeating Characters

```
class Solution {
   public int lengthOfLongestSubstring(String s) {
```

```
        int n = s.length();
        int maxLength = 0;
        int[] charIndex = new int[128];
        Arrays.fill(charIndex, -1);
        int left = 0;
        for (int right = 0; right < n; right++) {
            if (charIndex[s.charAt(right)] >= left) {
                left = charIndex[s.charAt(right)] + 1;
            }
            charIndex[s.charAt(right)] = right;
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```

Output:

Q 4) Remove Linked List Element

```
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;
    }
}
```

Time Complexity:

Output:

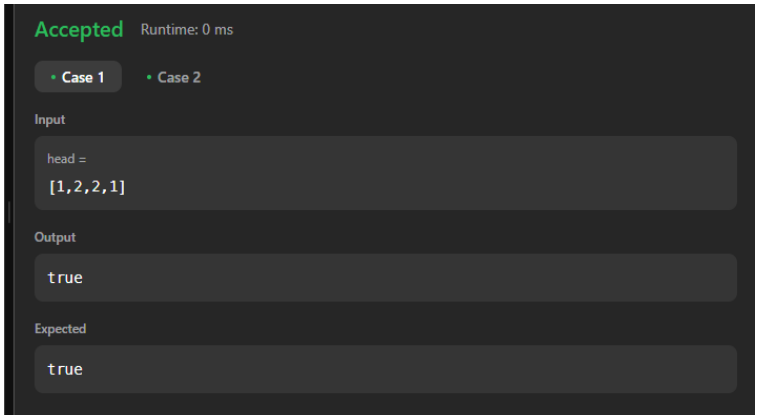
Q 5) Palindrome Linked List

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        ListNode slow=head,fast=head,prev,temp;
        while(fast!=null && fast.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        prev=slow;
```

```
        slow=slow.next;
        prev.next=null;
        while (slow!=null){
            temp=slow.next;
            slow.next=prev;
            prev=slow;
            slow=temp;
        }
        fast =head;
        slow=prev;
        while(slow!=null){
            if(fast.val!=slow.val) return false;
            fast=fast.next;
            slow=slow.next;
        }
        return true;
    }
}
```

Time Complexity: O(n)

Output:

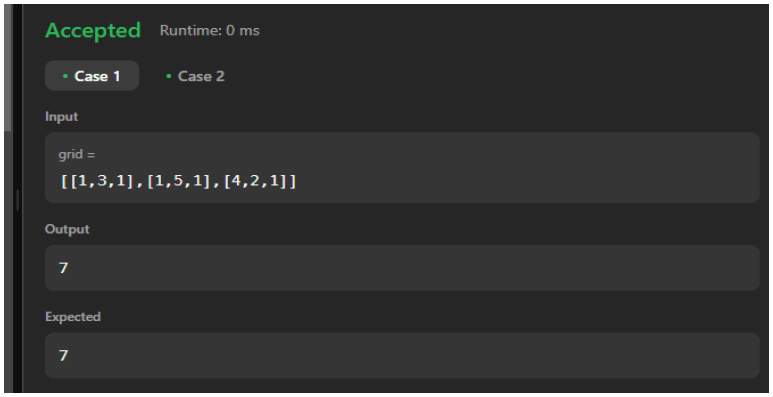


Q 6) Minimum Path Sum

```
class Solution {
    public int minPathSum(int[][] grid) {
        int m=grid.length,n=grid[0].length;
        for (int j=1;j<n;j++){
            grid[0][j]+=grid[0][j-1];
        }
        for (int i=1;i<m;i++){
            grid[i][0]+=grid[i-1][0];
        }
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                grid[i][j]+=Math.min(grid[i-1][j],grid[i][j-1]);
            }
        }
        return grid[m-1][n-1];
    }
}
```

Time Complexity: O(n*m)

Output:



Q 7) Valid Binary Search Tree

```
class Solution {
```

```
    public boolean isValidBST(TreeNode root) {
        return valid(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean valid(TreeNode node, long minimum, long maximum) {
        if (node == null) return true;
        if (!(node.val > minimum && node.val < maximum)) return false;
        return valid(node.left, minimum, node.val) && valid(node.right, node.val, maximum);
    }
}
```

Time Complexity: O(n)

Output:

Q 8) Word Ladder

```
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set <String> set=new HashSet <>(wordList);
        if (!set.contains(endWord)) return 0;
        Queue<String> queue=new LinkedList<>();
        queue.add(beginWord);
        Set<String> visited=new HashSet<>();
```

```
        queue.add(beginWord);
        int changes=1;
        while(!queue.isEmpty()){
            int size=queue.size();
            for(int i=0;i<size;i++){
                String word=queue.poll();
                if(word.equals(endWord)) return changes;
                for(int j=0;j<word.length();j++){
                    for(int k='a';k<='z';k++){
                        char arr[]=word.toCharArray();
                        arr[j]=(char)k;
                        String str=new String(arr);
                        if(set.contains(str) && !visited.contains(str)){
                            queue.add(str);
                            visited.add(str);
                        }
                    }
                }
            }
            ++changes;
        }
        return 0;
    }
}
```

Output:

Q 9) Word Ladder II

```
class Solution {
    List<List<String>> res;
    Map<Integer, Set<String>> map;
    Set<String> set;
    int goal;
    String es;
    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {
        res = new LinkedList<>();
        set = new HashSet<>(wordList);
        es = endWord;
        if(!set.contains(endWord)) return(res);
        int step = 0;
        Queue<String> q = new LinkedList<>();
        q.offer(beginWord);
        Set<String> seen = new HashSet<>();
        seen.add(beginWord);
        map = new HashMap<>();
        boolean found = false;
        while(!q.isEmpty()){
```

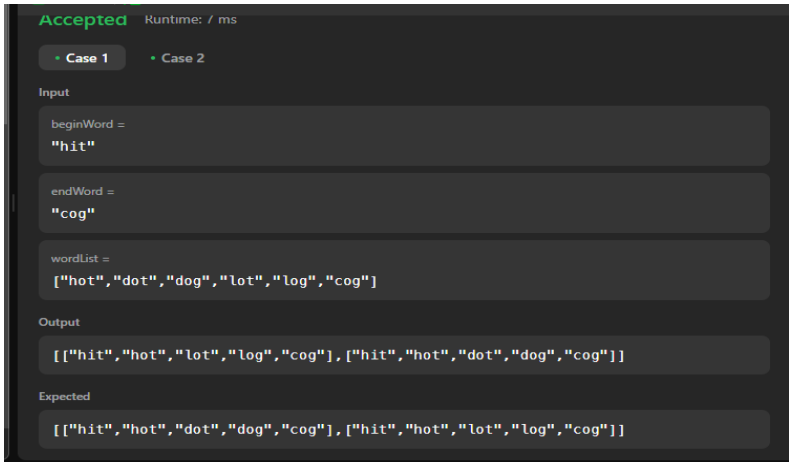
```
        int sz = q.size();
        map.put(step, new HashSet<>());
        while(sz-- > 0){
            String cs = q.poll();
            map.get(step).add(cs);
            if(cs.equals(endWord)){
                found = true;
                break;
            }
            Set<String> nbrs = nb(cs, set, seen);
            for(String ns: nbrs){
                q.offer(ns);
            }
        }
        if(found) break;
        step++;
    }
    if(!found) return(res);
    goal = step;
    List<String> list = new LinkedList<>();
    list.add(es);
    dfs(es, goal, list);
    return(res);
}

public void dfs(String s, int level, List<String> list){
    if(level == 0){
```

```
            List<String> copy = new LinkedList<>(list);
            Collections.reverse(copy);
            res.add(copy);
            return;
        }
        Set<String> nbrs = new HashSet<>();
        for(String nss: map.get(level - 1)){
            if(isnb(nss, s)) nbrs.add(nss);
        }
        for(String ns: nbrs){
            list.add(ns);
            dfs(ns, level - 1, list);
            list.remove(list.size() - 1);
        }
    }
    public boolean isnb(String s1, String s2){
        int n = s1.length();
        int d = 0;
        for(int i = 0; i < n; i++){
            if(s1.charAt(i) != s2.charAt(i)) d++;
        }
        return(d == 1);
    }
    public Set<String> nnb(String s, Set<String> set, int dep){
        Set<String> res = new HashSet<>();
        int n = s.length();
        for(int i = 0; i < n; i++){
```

```
            int chi = s.charAt(i) - 'a';
            for(int j = 0; j < 26; j++){
                if(j == chi) continue;
                String ns = s.substring(0, i) + (char)('a' + j) + s.substring(i + 1);
            }
        }
        return(res);
    }
    public Set<String> nb(String s, Set<String> set, Set<String> seen){
        Set<String> res = new HashSet<>();
        int n = s.length();
        for(int i = 0; i < n; i++){
            int chi = s.charAt(i) - 'a';
            for(int j = 0; j < 26; j++){
                if(j == chi) continue;
                String ns = s.substring(0, i) + (char)('a' + j) + s.substring(i + 1);
                if(set.contains(ns) && seen.add(ns)) res.add(ns);
            }
        }
        return(res);
    }
}
```

Time Complexity: O(m*n)

Output:



Q 10) Course Schedule

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        int counter = 0;
        if (numCourses <= 0) {
            return true;
        }
        int[] inDegree = new int[numCourses];
        List<List<Integer>> graph = new ArrayList<>();
        for (int i = 0; i < numCourses; i++) {
            graph.add(new ArrayList<>());
        }
        for (int[] edge : prerequisites) {
            int parent = edge[1];
            int child = edge[0];
            graph.get(parent).add(child);
            inDegree[child]++;
        }
        Queue<Integer> sources = new LinkedList<>();
```

```
        for (int i = 0; i < numCourses; i++) {
            if (inDegree[i] == 0) {
                sources.offer(i);
            }
        }
        while (!sources.isEmpty()) {
            int course = sources.poll();
            counter++;
            for (int child : graph.get(course)) {
                inDegree[child]--;
                if (inDegree[child] == 0) {
                    sources.offer(child);
                }
            }
        }
        return counter == numCourses;
    }
}
```

Time Complexity: O(n)

Output: