DSA – DAY 7 CODING PROBLEMS

Date:19/11/24

1.Spiral Matrix Code:

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
ArrayList<Integer> a=new ArrayList<Integer>();        int
rows = matrix.length, cols = matrix[0].length;        int
left = 0, right = cols-1, top = 0, bottom = rows-1;
while(left<=right && top<=bottom){            for(int
i=left;i<=right;i++){
                a.add(matrix[top][i]);
            }
            top++;

            for(int i=top;i<=bottom;i++){
                a.add(matrix[i][right]);
            }           right--;
if(top<=bottom){                for(int
i=right;i>=left;i--){
                    a.add(matrix[bottom][i]);
                }
                bottom--;
            }
            if(left<=right){
                for(int i=bottom;i>=top;i--){
                    a.add(matrix[i][left]);
```

```
            }
left++;
          }
       }
       return a;
    }
}
```

2.Palindrome Linked List Code:

```
class Solution {
    public boolean isPalindrome(ListNode head) {
if(head==null || head.next==null){          return
true;
       }
       Stack<Integer> stack=new Stack<Integer>();
ListNode temp=head;        int len=0;
while(head!=null){
          len++;
          head=head.next;
       } int
       i=0;
       while(temp!=null && i<len/2){
stack.push(temp.val);          temp=temp.next;
          i++;
       }
```

```
        if((len%2)!=0){
temp=temp.next;
        }
        while(temp!=null){
            if((!stack.isEmpty())&&stack.pop()!=temp.val){
return false;
            }
            temp=temp.next;
        }
        return true;
    }
}
```

3. Remove Linked List Element

Code:

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode ans=new ListNode();
        ListNode curr=ans;
while (head!=null) {     if
(head.val!=val) {
curr.next=head;
curr=curr.next;
            }
            head=head.next;
```

```
        }
        curr.next=null;
return ans.next;
    }
}
```

4.Next Permutation Code:

```
class Solution {
    public void nextPermutation(int[] nums) {
int ind1=-1;        int ind2=0;         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--;
        }
    }
}
```

5.Longenst Substring without repeating character Code:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
int[] count = new int[128];        int max =0;
int j=0;
        for(int i=0;i<s.length();i++){
count[s.charAt(i)]++;
while(count[s.charAt(i)]>1){
count[s.charAt(j)]--;                j++;
```

```
            }
            max = Math.max(max,(i-j)+1);
        }
        return max;
    }
}
```

Output:

6.Minimum Path Sum Code:

```
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];
    }
}
```

7. Validate binary search tree

Code:

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

   public boolean isValidBST(TreeNode root, long minVal, long maxVal) {
if (root == null) return true;
      if (root.val >= maxVal || root.val <= minVal) return false;        return
isValidBST(root.left, minVal, root.val) && isValidBST(root.right, root.val,
maxVal);
   }
}
```

8.Course Schedule

Code: class

```
CourseSchedule {
   public boolean canFinish(int numCourses, int[][] prerequisites) {
List<List<Integer>> graph = new ArrayList<>();        for (int i = 0; i <
numCourses; i++) {            graph.add(new ArrayList<>());
      }
      for (int[] prerequisite : prerequisites) {
int course = prerequisite[0];           int pre
```

```
= prerequisite[1];
graph.get(pre).add(course);
        }
        int[] visited = new int[numCourses];
for (int i = 0; i < numCourses; i++) {
if (hasCycle(graph, visited, i)) {
return false;
            }
        }
        return true;
    }
    private boolean hasCycle(List<List<Integer>> graph, int[] visited, int course) {
if (visited[course] == 1) {            return true;
        }
        if (visited[course] == 2) {
return false;
        }
        visited[course] = 1;
        for (int neighbor : graph.get(course)) {
if (hasCycle(graph, visited, neighbor)) {
return true;
            }
        }
        visited[course] = 2;
return false;
    }
```

```
}
```

9.Word Ladder Code:

```
class Solution {
   public int ladderLength(String beginWord, String endWord, List<String>
wordList) {
      if(wordList.size() == 0)  return 0;
      HashMap<String, List<String>> connection = new HashMap<>();
wordList.add(beginWord);
      for(String s : wordList) {
         connection.put(s, new ArrayList<String>());
      }
      for(String s1 : wordList) {
for(String s2 : wordList) {
if(canTransform(s1,s2)){
connection.get(s1).add(s2);
connection.get(s2).add(s1);
            }
         }
      }
      Queue<String> queue = new LinkedList();
queue.add(beginWord);
      int dist = 0;
```

```
      Set<String> visited = new HashSet();
while(!queue.isEmpty()){           int size
= queue.size();           dist++;
for(int i=0;i<size;i++){               String cur
= queue.poll();
if(cur.equals(endWord)) {
return dist;
            }
            for(String s : connection.get(cur)) {
if(!visited.contains(s)) {
visited.add(s);
                  queue.add(s);
               }
            }
         }
      }
      return 0;
   }

   public boolean canTransform(String s1, String s2) {
int count = 0;
      for(int i=0;i<s1.length();i++){
if(s1.charAt(i) != s2.charAt(i)){
count++;
         }
      }
```

```
        return count == 1;
    }

}
```

14.Word Ladder-II Code:

```
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);
                }
            }
        }
    }
}
```