DAY 7- LEETCODE PROBLEMS

1.Next Permutation:

class Solution {

    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--;

        }

    }

}

2. Longest substring without repeating characters

class Solution:

    def lengthOfLongestSubstring(self, s: str) -> int:

        n=len(s)

        s1=set()

        max\_len=0

        i,j=0,0

        while i<n and j<n:

            if(s[j] not in s1):

                s1.add(s[j])

                max\_len=max(max\_len,j-i+1)

                j+=1

            else:

                s1.remove(s[i])

                i+=1

        return max\_len

Time complexity: O(n)

3. Remove linked list elements

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: O(n)

4. Palindrome linked list

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;

}

}

Time complexity: O(n)

5.Minimum path sum:

class Solution {

    public int minPathSum(int[][] grid) {

        int m=grid.length;

        int 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(mxn)

6. [Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/):

class Solution {

    public boolean isValidBST(TreeNode root) {

        return validroot(root,Long.MIN\_VALUE,Long.MAX\_VALUE);

    }

    private boolean validroot(TreeNode node,long minval,long maxval){

        if (node==null){

            return true;

        }

        if(node.val<=minval || node.val>=maxval){

            return false;

        }

        return validroot(node.left,minval,node.val) && validroot(node.right,node.val,maxval);

    }

}

Time complexity: O(n)

7. Word Ladder:

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 transformedWord = new String(wordChars);

if (wordSet.contains(transformedWord)) {

neighbors.add(transformedWord);

}

}

wordChars[i] = originalChar;

}

return neighbors;

}

}

8.Course Scheduler

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;

    }

}