Lab 4

(i)1672

class Solution {

    public int maximumWealth(int[][] accounts) {

        int max  =  0;

        int temp = 0;

        for(int i = 0; i < accounts.length; i++){

            for(int j = 0; j< accounts[i].length; j++ ){

                temp  += accounts[i][j];

            }

            if (temp > max){

                max =  temp;

            }

            temp = 0;

        }

        return max;

    }

}

(ii)867

class Solution {

    public int[][] transpose(int[][] matrix) {

        int[][] answer = new int[matrix[0].length][matrix.length];

        for(int i=0; i < matrix.length; i++){

            for (int j = 0; j < matrix[0].length; j++){

                answer[j][i]  = matrix[i][j];

            }

        }

        return answer;

    }

}

(iii)36

public boolean isValidSudoku(char[][] board) {

    Set seen = new HashSet();

    for (int i=0; i<9; ++i) {

        for (int j=0; j<9; ++j) {

            if (board[i][j] != '.') {

                String b = "(" + board[i][j] + ")";

                if (!seen.add(b + i) || !seen.add(j + b) || !seen.add(i/3 + b + j/3))

                    return false;

            }

        }

    }

    return true;

}

(iv)48

class Solution {

    public void rotate(int[][] matrix) {

        int n = matrix.length;

        int j= 0;

        while(j<n){

            reverse(matrix, 0, j, n-1, j);

            j++;

        }

        int i1 = 0;

        int j1 = 0;

        int i2 = 0;

        int j2 = 0;

        while(i1<n){

            daignol(matrix, i1, i2, j1, j2);

            i1++;

            j2++;

        }

        j2--;

        i1--;

        while(i2<n){

            i2++;

            j1++;

            if(i2>=n){

                break;

            }

            daignol(matrix, i1, i2, j1, j2);

        }

    }

    public void daignol(int[][] matrix, int i1, int j1, int i2, int j2){

        while(i2<i1){

            swap(matrix, i1, i2,j1,j2);

            i1--;

            j1++;

            i2++;

            j2--;

        }

    }

    public void reverse(int[][] matrix, int i1, int j1, int i2, int j2){

        while(i1<i2){

            swap(matrix, i1, i2,j1,j2);

            i1++;

            i2--;

        }

    }

    public void swap(int[][] matrix, int i1, int i2, int j1, int j2){

        int temp = matrix[i1][j1];

        matrix[i1][j1] = matrix[i2][j2];

        matrix[i2][j2] = temp;

    }

}

(v)54

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

            List<Integer> ans = new ArrayList<>();

    int top = 0, right = matrix[0].length-1, bottom = matrix.length-1, left = 0;

    while(top <= bottom && left <= right){

        for(int i = left; i<=right; i++){

            ans.add(matrix[top][i]);

        }

        top++;

        for(int i = top; i<=bottom; i++){

            ans.add(matrix[i][right]);

        }

        right--;

        if(top<=bottom){

            for(int i = right; i>=left; i--){

                ans.add(matrix[bottom][i]);

            }

            bottom--;

        }

        if(left<=right){

            for(int i = bottom; i>=top; i--){

                ans.add(matrix[i][left]);

            }

            left++;

        }

    }

    return ans;

}

}

(vi)130

class Solution {

    public void dfs(int i,int j,int n,int m,char board[][]){

        if(i<0 || j<0 || i>=n || j>=m || board[i][j]!='O'){

            return;

        }

        board[i][j]='\*';

        dfs(i-1,j,n,m,board);

        dfs(i+1,j,n,m,board);

        dfs(i,j-1,n,m,board);

        dfs(i,j+1,n,m,board);

    }

    public void solve(char[][] board) {

        int n=board.length;

        int m =board[0].length;

        for(int i=0;i<n;i++){

            for(int j=0;j<m;j++){

                if(i\*j==0 || i==n-1 || j==m-1 ){

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

                        dfs(i,j,n,m,board);

                    }

                }

            }

        }

        for(int i=0;i<n;i++){

            for(int j=0;j<m;j++){

                if(board[i][j]=='O')

                board[i][j]='X';

                if(board[i][j]=='\*')

                board[i][j]='O';

            }

        }

    }

}

(vii)994

class Solution {

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

        Queue<int[]> q = new LinkedList<>();

        int row = grid.length;

        int col = grid[0].length;

        int min = 0;

        int countfresh = 0;

        for(int i = 0;i<row;i++){

            for(int j=0;j<col;j++){

                if(grid[i][j] == 2){

                    q.offer(new int[]{i,j});

                }

                else if(grid[i][j] == 1){

                    countfresh++;

                }

            }

        }

        if(countfresh==0)return 0;

        int[][] path = {{1,0},{0,1},{-1,0},{0,-1}};

        int count = 0;

        while(!q.isEmpty()){

            count++;

            int size = q.size();

            while(size != 0){

            int[] t = q.remove();

            for(int i=0;i<4;i++){

                int x = t[0] + path[i][0];

                int y = t[1] + path[i][1];

                if(x<0 || y<0 || y>=col || x>=row || grid[x][y] ==0 || grid[x][y] ==2)continue;

                grid[x][y] = 2;

                q.offer(new int[]{x,y});

                countfresh--;

            }

            size--;

            }

        }

        return countfresh==0?count-1:-1;

    }

}

(viii)155

class MinStack {

    private int size, top;

    private int[] stack, minStack;

    public MinStack() {

        size = 1000;

        stack = new int[size];

        minStack = new int[size];

        top = -1;

    }

    public void push(int val) {

        if(top==size-1)

            return;

        else{

            stack[++top] = val;

            if(top==0)

                minStack[top] = val;

            else

                minStack[top] = Math.min(val, minStack[top-1]);

        }

    }

    public void pop() {

        if(top==-1)

            return;

        else

            top--;

    }

    public int top() {

        if(top==-1)

            return -1;

        return stack[top];

    }

    public int getMin() {

        if(top==-1)

            return -1;

        return minStack[top];

    }

}

(ix)20

class Solution {

    public boolean isValid(String s) {

         Stack<Character> stack = new Stack<>();

        for (char x : s.toCharArray()

             ) {

            if(x=='('){

                stack.push(')');

            } else if (x=='{') {

                stack.push('}');

            } else if (x=='[') {

                stack.push(']');

            } else if (stack.isEmpty() || stack.pop()!=x) {

                return false;

            }

        }

        return stack.isEmpty();

    }

}

(x)844

class Solution {

    public boolean backspaceCompare(String s, String t) {

        Stack<Character> stack1= new Stack<>();

        Stack<Character> stack2=new Stack<>();

        for(int i=0;i<s.length();i++){

            char c= s.charAt(i);

            if(!stack1.isEmpty() && c =='#') stack1.pop();

            else{

                if(c!='#')

                    stack1.push(c);

             }

        }

        for(int i=0;i<t.length();i++){

            char d= t.charAt(i);

            if(!stack2.isEmpty() && d=='#') stack2.pop();

           else{

                if(d!='#')

                    stack2.push(d);

             }

        }

    //   String s="";

return stack1.equals(stack2);

    }

}

(xi)2390

class Solution {

    public String removeStars(String s) {

    StringBuilder ans = new StringBuilder();

         int k =-1;

         for(int i=0; i<s.length(); i++){

             if(s.charAt(i)=='\*'){

                 ans.deleteCharAt(k);

                 k--;

             }

             else{

                 k++;

                 ans.append(s.charAt(i));

             }

         }

         return ans.toString();

    }

}

(xii)682

class Solution {

    public int calPoints(String[] ops) {

        Stack<Integer> stack = new Stack<>();

        for(String s : ops){

              if( s.equals("C")){

                 stack.pop();

             }

             else if(s.equals("D")){

                 stack.push(2\* stack.peek());

             }

              else if(s.equals("+")){

                  if(stack.size() >=2){

                     int res1= stack.pop();

                     int res2= stack.pop();

                     int res= res1+res2;

                     stack.push(res2);

                     stack.push(res1);

                     stack.push(res);

                  }

             }

             else{

                 stack.push(Integer.parseInt(s));

             }

        }

        int ans =0;

         while(!stack.isEmpty()){

            ans += stack.pop();

         }

        return ans;

    }

}

(xiii)496

class Solution {

    public int[] nextGreaterElement(int[] nums1, int[] nums2) {

        int ans[]=new int[nums1.length];

        for(int i=0;i<nums1.length;i++){

            int nextG=-1;

            int j=nums2.length-1;

            while(j>=0&&nums2[j]!=nums1[i]){

                if(nums2[j]>nums1[i]){

                    nextG=nums2[j];

                }

                j--;

            }

            ans[i]=nextG;

        }

        return ans;

    }

}

(ixv)503

class Solution {

    public int[] nextGreaterElements(int[] nums) {

      int n = nums.length;

        Stack<Integer> s = new Stack<>();

        int[] arr = new int[n];

        for(int i = 2 \* n - 1; i >= 0 ; i--){

            while(!s.isEmpty() && s.peek() <= nums[i % n]){

                s.pop();

            }

            if(s.isEmpty()){

                arr[i % n] = -1;

            }

            else{

                arr[i % n] = s.peek();

            }

            s.push(nums[i % n]);

        }

        return arr;

    }

}

(xv)232

class MyQueue {

    Stack<Integer> stack1 = new Stack<>();

    Stack<Integer> stack2 = new Stack<>();

    public MyQueue() {

    }

    public void push(int x) {

        while (!stack1.isEmpty()) {

            stack2.push(stack1.pop());

        }

        stack1.push(x);

        while (!stack2.empty()) {

            stack1.push(stack2.pop());

        }

    }

    public int pop() {

        int removed = stack1.pop();

        return removed;

    }

    public int peek() {

        int top = stack1.peek();

        return top;

    }

    public boolean empty() {

        return stack1.isEmpty();

    }

}

(xvi)225

class MyStack {

    Queue<Integer> queue;

    public MyStack() {

        this.queue=new LinkedList<Integer>();

    }

    public void push(int x) {

        queue.add(x);

        for(int i=0;i<queue.size()-1;i++)

            queue.add(queue.poll());

    }

    public int pop() {

        return queue.poll();

    }

    public int top() {

        return queue.peek();

    }

    public boolean empty() {

        return queue.isEmpty();

    }

}

(xvii)2073)

class Solution {

    public int timeRequiredToBuy(int[] tickets, int k) {

     int seconds = 0;

        for(int i = 0; i < tickets.length; i++){

            if(tickets[i] < tickets[k]) seconds += tickets[i];

            else{

                if(i <= k) seconds += tickets[k];

                else seconds += tickets[k] - 1;

            }

        }

        return seconds;

    }

}