**/\*max equilibrium array o(n)\*/**

import java.lang.Math.\*;

import java.util.stream.\*;

class GFG {

// Function to find maximum equilibrium

// sum.

static int findMaxSum(int arr[], int n)

{

int sum = IntStream.of(arr).sum();

int prefix\_sum = 0,

res = Integer.MIN\_VALUE;

for (int i = 0; i < n; i++)

{

prefix\_sum += arr[i];

if (prefix\_sum == sum)

res = Math.max(res, prefix\_sum);

sum -= arr[i];

}

return res;

}

// Driver Code

public static void main(String[] args)

{

int arr[] = { -2, 5, 3, 1,

2, 6, -4, 2 };

int n = arr.length;

System.out.print(findMaxSum(arr, n));

}

}

**/\*leaders with O(n^2)\*/**

int arr[] = new int[]{16, 17, 4, 3, 5, 2};

int size = arr.length;

        for (int i = 0; i < size; i++)

{

int j;

for (j = i + 1; j < size; j++)

{

if (arr[i] <=arr[j])

break;

}

if (j == size) // the loop didn't break

System.out.print(arr[i] + " ");

}

**/\*leaders with o(n)\*/**

int arr[] = new int[]{16, 17, 4, 3, 5, 2};

int n = arr.length;

        int max\_from\_right=0;

        for (int i = size-1; i >= 0; i--)

{

if (max\_from\_right < arr[i])

{

max\_from\_right = arr[i];

System.out.print(max\_from\_right + " ");

}

}

**/\*majority element o(n^2)\*/**

import java.util.Scanner;

public class majorityele {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        int arr[]={2,3,3,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3};

        int n= arr.length;

        int index=0,max=0,count;

        for(int i =0;i<n;i++){

            count=0;

            for(int j=0;j<n;j++){

                if(arr[i]==arr[j]){

                    count++;

                }

            }

            if(count>max){

                max=count;

                index=i;

            }

        }

        if(max>n/2)

        System.out.println("Majority element "+arr[index]);

        else

        System.out.println("No majority element");

    }

}

**/\*majority element o(n)\*/**

import java.io.\*;

import java.util.HashMap;

public class majorityele1 {

    public static void main(String[] args) {

        HashMap<Integer,Integer> map=new HashMap<>();

        int ct=0;

        int arr[]={2,4,3,4,4};

        int n=arr.length;

        for(int i=0;i<n;i++)

            map.put(arr[i], 0);

        int max=0,index=0;

        for(int i=0;i<n;i++){

            ct=map.get(arr[i])+1;

            map.put(arr[i],ct);

            if(ct>max){

                max=ct;

                index=i;

            }

        }

        if(max>n/2)

        System.out.println("Majority element "+arr[index]);

        else

        System.out.println("No majority element");

    }

}

**Majority element Boyer Moore**

import java.io.\*;

class majelt2

{

// Function to find majority element

public static int findMajority(int[] nums)

{

    int count = 0, candidate = -1;

    // Finding majority candidate

    for (int index = 0; index < nums.length; index++) {

    if (count == 0) {

        candidate = nums[index];

        count = 1;

    }

    else {

        if (nums[index] == candidate)

        count++;

        else

        count--;

    }

    }

    count = 0;

    for (int index = 0; index < nums.length; index++) {

    if (nums[index] == candidate)

        count++;

    }

    if (count > (nums.length / 2))

    return candidate;

    return -1;

}

public static void main(String[] args)

{

    int arr[] = { 1, 1, 1, 1, 2, 3, 4 };

    int majority = findMajority(arr);

    System.out.println(" The majority element is : "

                    + majority);

}

}

**Selection Sort**

import java.io.\*;

public class selectionsort

{

    void sort(int arr[])

    {

        int n = arr.length;

        for (int i = 0; i < n-1; i++)

        {

            int min\_idx = i;

            for (int j = i+1; j < n; j++)

                if (arr[j] < arr[min\_idx])

                    min\_idx = j;

            int temp = arr[min\_idx];

            arr[min\_idx] = arr[i];

            arr[i] = temp;

        }

    }

    void printArray(int arr[])

    {

        int n = arr.length;

        for (int i=0; i<n; ++i)

            System.out.print(arr[i]+" ");

        System.out.println();

    }

    public static void main(String args[])

    {

        selectionsort ob = new selectionsort();

        int arr[] = {64,25,12,22,11};

        ob.sort(arr);

        System.out.println("Sorted array");

        ob.printArray(arr);

    }

}

**Quick Sort**

public class quicksort {

    static void Pivotrecursion(int[] arr,int low,int high){

        if(low<high){

        int pivotp=getpivot(arr,low,high);

        Pivotrecursion(arr, pivotp+1, high);

        Pivotrecursion(arr, low, pivotp-1);

        }

    }

    static int getpivot(int []arr,int low, int high){

        int pivtelt=arr[high];

        int pivotp=low;

        for(int i=low;i<=high;i++){

            if(arr[i]<pivtelt){

                int temp=arr[i];

                arr[i]=arr[pivotp];

                arr[pivotp]=temp;

                pivotp++;

            }

        }

        int tmp=arr[pivotp];

        arr[pivotp]=arr[high];

        arr[high]=tmp;

        return pivotp;

    }

    public static void main(String[] args) {

        int[] arr= new int[]{20,81,43,98,82,28,66};

        Pivotrecursion(arr,0,arr.length-1);

        for(int i=0;i<arr.length;i++)

            System.out.print(arr[i]+" ");

    }

}

**Sorted Unique Permutation**

**Method 1 : O(N\*N!)**

// Java program to print all the permutation

// of the given String.

//include <algorithm>

//include <String>

import java.util.\*;

class GFG{

// Count of total permutations

static int total = 0;

static void permute(int i, String s)

{

// Base case

if (i == (s.length() - 1))

{

System.out.print(s + "\n");

total++;

return;

}

char prev = '\*';

// Loop from j = 1 to length of String

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

{

char []temp = s.toCharArray();

if (j > i && temp[i] == temp[j])

continue;

if (prev != '\*' && prev == s.charAt(j))

{

continue;

}

// Swap the elements

temp = swap(temp,i,j);

prev = s.charAt(j);

// Recursion call

permute(i + 1, String.valueOf(temp));

}

}

static char[] swap(char []arr, int i, int j)

{

char temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

return arr;

}

static String sortString(String inputString)

{

// Convert input string to char array

char tempArray[] = inputString.toCharArray();

// Sort tempArray

Arrays.sort(tempArray);

// Return new sorted string

return new String(tempArray);

}

// Driver code

public static void main(String[] args)

{

String s = "abca";

// Sort

s = sortString(s);

// Function call

permute(0, s);

System.out.print("Total distinct permutations = " +

total +"\n");

}

}

**Method 2 : O(N!)**

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

public class StringPermutation {

    public List<String> permute(char input[]) {

        Map<Character, Integer> countMap = new TreeMap<>();

        for (char ch : input) {

            countMap.compute(ch, (key, val) -> {

                if (val == null) {

                    return 1;

                } else {

                    return val + 1;

                }

            });

        }

        char str[] = new char[countMap.size()];

        int count[] = new int[countMap.size()];

        int index = 0;

        for (Map.Entry<Character, Integer> entry : countMap.entrySet()) {

            str[index] = entry.getKey();

            count[index] = entry.getValue();

            index++;

        }

        List<String> resultList = new ArrayList<>();

        char result[] = new char[input.length];

        permuteUtil(str, count, result, 0, resultList);

        return resultList;

    }

    public void permuteUtil(char str[], int count[], char result[], int level, List<String> resultList) {

        if (level == result.length) {

            resultList.add(new String(result));

            return;

        }

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

            if(count[i] == 0) {

                continue;

            }

            result[level] = str[i];

            count[i]--;

            permuteUtil(str, count, result, level + 1, resultList);

            count[i]++;

        }

    }

    private void printArray(char input[]) {

        for(char ch : input) {

            System.out.print(ch);

        }

        System.out.println();

    }

    public static void main(String args[]) {

        StringPermutation sp = new StringPermutation();

        sp.permute("AABC".toCharArray()).forEach(s -> System.out.println(s));

    }

}

**Manuevering**

class manuevering {

    static int numberOfPaths(int m, int n){

        if (m == 1 || n == 1)

            return 1;

        return numberOfPaths(m - 1, n)+ numberOfPaths(m, n - 1);

    }

    public static void main(String args[])

    {

        System.out.println(numberOfPaths(3, 3));

    }

}

**Combinations**

import java.io.\*;

    static void combinationUtil(int arr[], int n, int r, int index, int data[], int i)

    {

        if (index == r)

        {

            for (int j=0; j<r; j++)

                System.out.print(data[j]+" ");

            System.out.println("");

        return;

        }

        if (i >= n)

        return;

        data[index] = arr[i];

        combinationUtil(arr, n, r, index+1, data, i+1);

        combinationUtil(arr, n, r, index, data, i+1);

    }

public static void main (String[] args) {

        int arr[] = {1, 2, 3};

        int r = 2;

        int n = arr.length;

        int data[]=new int[r];

        combinationUtil(arr, n, r, 0, data, 0);

    }

}

**Josephus trap**

import java.io.\*;

class josephustrap {

    static int josephus(int n, int k)

    {

        if (n == 1)

            return 1;

        else

            return (josephus(n - 1, k) + k - 1) % n + 1;

    }

    public static void main(String[] args)

    {

        int n = 5;

        int k = 2;

        System.out.println("The chosen place is " + josephus(n, k));

    }

}

**Rate in maze**

public class RatMazeSolving{

     static int sol[][], cont=0;

     static boolean MazeSolve(int maze[][],int x,int y){

          if(x==maze.length-1 && y==maze[0].length-1){

               sol[x][y]=1;

               return true;

          }

          if(ispassible(maze,x,y)){

               sol[x][y]=1;

               if(MazeSolve(maze,x,y+1))

                    return true;

               if(MazeSolve(maze,x+1,y))

                    return true;

               sol[x][y]=0;

          }

               return false;

     }

     static boolean ispassible(int maze[][], int x, int y){

     cont++;

          if(x>=0 && y>=0 && x<maze.length && y<maze[0].length && maze[x][y]==1)

           return true;

       return false;

     }

     public static void main(String[] args) {

          int maze[][]={ {1, 1, 1, 1, 0},

                         {0, 0, 0, 1, 1},

                         {1, 1, 1, 1, 1},

                         {1, 0, 0, 0, 1},

                         {1, 1, 1, 1, 1}};

          sol= new int[maze.length][maze[0].length];

          if(MazeSolve(maze, 0,0))

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

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

                         System.out.print(" "+sol[i][j]+" ");

                    System.out.println();

               }

          else

          System.out.println("Solution is not possible");

     }

}

**N Queens**

public class NQueens {

        static int N = 4;

        static boolean isSafe(int board[][], int row, int col){

            int i, j;

            for (j = col; j >=0; j--)

                if (board[row][j] == 1)

                    return false;

            for (i=row,j=col; i>=0 && j>=0; i--,j--)

                if (board[i][j] == 1)

                    return false;

            for (i = row, j = col; j >= 0 && i < N;i++,j--)

                if (board[i][j] == 1)

                    return false;

            return true;

        }

        static boolean solveNQUtil(int board[][], int col){

            if (col >= N)

                return true;

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

                if (isSafe(board, i, col)) {

                    board[i][col] = 1;

                    if (solveNQUtil(board, col+1))

                        return true;

                }

                board[i][col] = 0;

            }

            return false;

        }

        public static void main(String args[])  {

            int board[][] = new int[N][N];

            if (solveNQUtil(board, 0)){

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

                 for (int j = 0; j < N; j++)

                    System.out.print(" " + board[i][j]+ " ");

                System.out.println();

                }

            }

            else

                System.out.print("Solution does not exist");

        }

}

**Warnsdorff’s Algorithm**

**public class KnightTour {**

**public static void main(String[] args) {**

**int chess\_board\_size = 8;**

**KnightTour knightTour = new KnightTour(chess\_board\_size);**

**knightTour.solveKnightTourProblem();**

**}**

**int BOARD\_SIZE;**

**int[][] visited;**

**int[] xMoves = { 2, 1, -1, -2, -2, -1, 1, 2 };**

**int[] yMoves = { 1, 2, 2, 1, -1, -2, -2, -1 };**

**public KnightTour(int chessBoardSize) {**

**this.BOARD\_SIZE = chessBoardSize;**

**this.visited = new int[BOARD\_SIZE][BOARD\_SIZE];**

**this.initializeBoard();**

**}**

**private void initializeBoard() {**

**for (int i = 0; i < BOARD\_SIZE; i++)**

**for (int j = 0; j < BOARD\_SIZE; j++)**

**this.visited[i][j] = Integer.MIN\_VALUE;**

**}**

**public void printSolution() {**

**for (int i = 0; i < BOARD\_SIZE; i++) {**

**for (int j = 0; j < BOARD\_SIZE; j++) {**

**System.out.print(visited[i][j] + "\t");**

**}**

**System.out.println();**

**}**

**}**

**public void solveKnightTourProblem() {**

**visited[0][0] = 0;**

**// start knight tour from top left corner square (0, 0)**

**if( solveProblem(1, 0, 0)) {**

**printSolution();**

**} else {**

**System.out.println("No feasible solution found...");**

**}**

**}**

**public boolean solveProblem(int moveCount, int x, int y) {**

**// Base Case : We were able to move to each square exactly once**

**if (moveCount == BOARD\_SIZE \* BOARD\_SIZE) {**

**return true;**

**}**

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

**int nextX = x + xMoves[i];**

**int nextY = y + yMoves[i];**

**// check if new position is a valid and not visited yet**

**if ( isValidMove(nextX, nextY) && visited[nextX][nextY] == Integer.MIN\_VALUE) {**

**visited[nextX][nextY] = moveCount;**

**if ( solveProblem(moveCount + 1, nextX, nextY) ) {**

**return true;**

**}**

**// BACKTRACK !!!**

**visited[nextX][nextY] = Integer.MIN\_VALUE;**

**}**

**}**

**return false;**

**}**

**public boolean isValidMove(int x, int y) {**

**if (x < 0 || x >= BOARD\_SIZE || y < 0 || y >= BOARD\_SIZE) {**

**return false;**

**} else {**

**return true;**

**}**

**}**

**}**

**Hamiltonian Cycle**

class HamiltonianCycle

{

final int V = 5;

int path[];

boolean isSafe(int v, int graph[][], int path[], int pos)

{

if (graph[path[pos - 1]][v] == 0)

return false;

for (int i = 0; i < pos; i++)

if (path[i] == v)

return false;

return true;

}

boolean hamCycleUtil(int graph[][], int path[], int pos)

{

if (pos == V)

{

if (graph[path[pos - 1]][path[0]] == 1)

return true;

else

return false;

}

for (int v = 1; v < V; v++)

{

if (isSafe(v, graph, path, pos))

{

path[pos] = v;

if (hamCycleUtil(graph, path, pos + 1) == true)

return true;

path[pos] = -1;

}

}

return false;

}

int hamCycle(int graph[][])

{

path = new int[V];

for (int i = 0; i < V; i++)

path[i] = -1;

path[0] = 0;

if (hamCycleUtil(graph, path, 1) == false)

{

System.out.println("\nSolution does not exist");

return 0;

}

printSolution(path);

return 1;

}

void printSolution(int path[])

{

System.out.println("Solution Exists: Following" +

" is one Hamiltonian Cycle");

for (int i = 0; i < V; i++)

System.out.print(" " + path[i] + " ");

System.out.println(" " + path[0] + " ");

}

// driver program to test above function

public static void main(String args[])

{

HamiltonianCycle hamiltonian =

new HamiltonianCycle();

int graph1[][] = {{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 1, 1, 1, 0},

};

hamiltonian.hamCycle(graph1);

int graph2[][] = {{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 0},

{0, 1, 1, 0, 0},

};

hamiltonian.hamCycle(graph2);

}

}