UNC CHARLOTTE
College of Computing and Informatics

N Queens using Hill Climbing

PROJECT DOCUMENTATION REPORT

PROJECT 2

ITCS 6150 - Intelligent Systems

DEPARTMENT OF COMPUTER SCIENCE

SUBMITTED TO

Dewan T. Ahmed, Ph.D.

SUBMITTED BY:

Harshitha Keshavaraju Vijayalakshmi **Tannu D. Singh**

801084151 **801085297**

Table of Contents

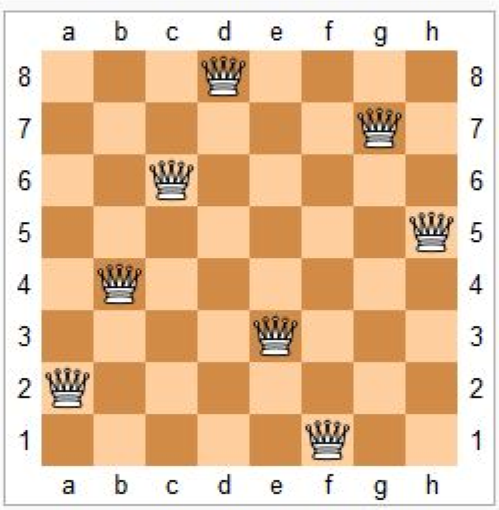
I. Problem Formulation

1.1 Introduction

N-queens

N queen puzzle is the problem of placing n chess queens on nxn dimensional chess board such that the solution exist when none of the queens share any column, rows or diagonally intersect each other in the chess board.

The solution of n queens does not exist for n=2 or n=3.

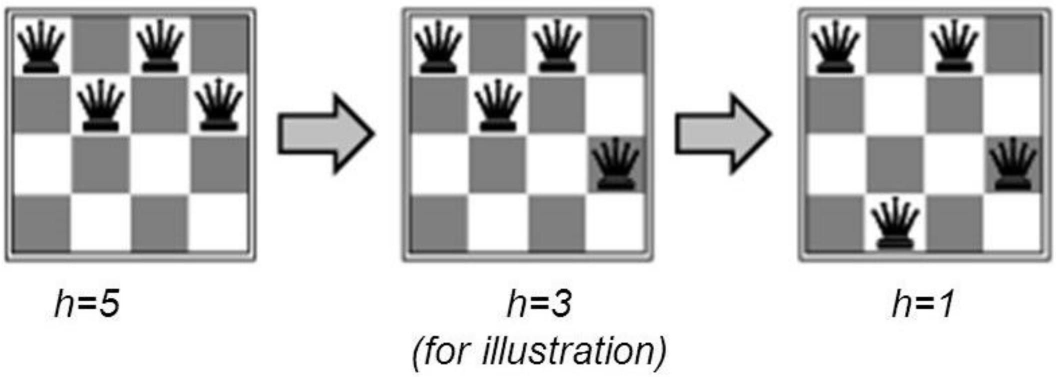


One solution to the eight queens puzzle https://leetcode.com//n-queens/description/

Constructing and counting solutions

The number of ways to display 8-queens on 8x8 board problem is ${}_{64}C_8$, but the actual number of solutions is 92. Some restrictions on the arrangement can be applied using brute force technique, such as applying per column one queen restriction which decreases the possible arrangement to

8^8. Further applying permutations and adding no diagonal intersection to the existing restriction reduces the possible number of arranging 8 queens to 8!.



Hill Climbing algorithm

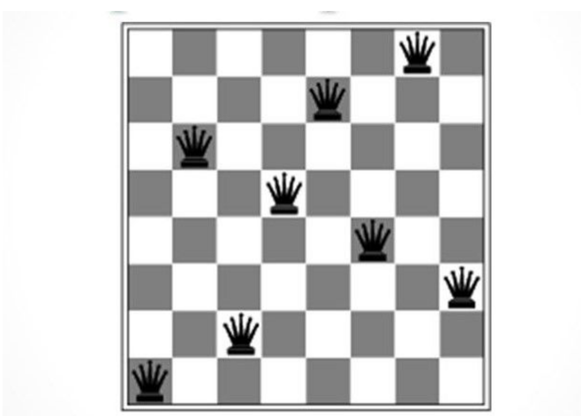
Hill climbing is a type of Local search technique which finds the optimal solution iteratively by incrementing the improvements after each iteration starting from an arbitrary solution.

Hill climbing finds the solution for problems that have well defined restrictions which is usually referred as problems in convex set where all the constraints are within convex function. For problems in other category Hill climbing will find the solution at local maxima and will get stuck at local maxima. The simple Hill climbing method uses greedy method by selection the first best neighboring node as the next incremental improvement.

To avoid getting stuck at local maxima as the solution one could use many variations with Hill Climbing such as repeated local search (random restarts), iterated local search, or memory less stochastic modifications usually known as simulated annealing.

Simple hill climbing is not suitable to be considered optimal for the following reasons:

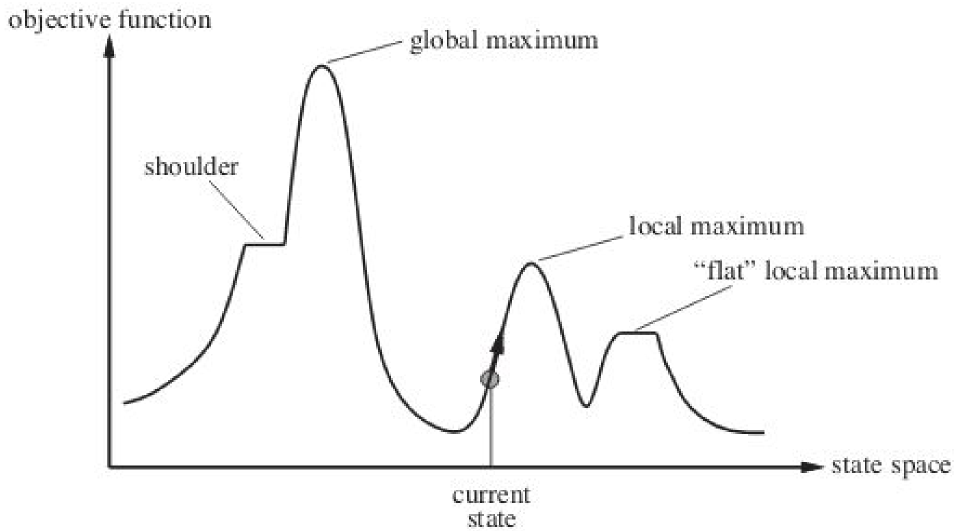
1)Local maxima



8 queens stuck at local maxima with heuristic value h=1

2)Ridges : A series of local maxima forms a ridge. It is usually difficult for simple hill climbing method to surpass the ridges and find the next optimal solution due to its greedy selection of next state.

3)Plateau: Plateaus are formed from flat local maxima, that is there is no way to move uphill or in other words no progress is possible if a flat global maxima is found. Shoulder is a variant of plateau from where progress is possible .



https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Variants of Hill Climbing

- **Steepest ascent hill climbing**

In steepest ascent hill climbing instead of selection the first best neighboring or successor node, the best among all the neighboring node is selected after evaluating every successor.The node traversal method and selecting the best successor of steepest ascent method is similar to best first search

- **Stochastic hill climbing**

Stochastic hill climbing does not evaluate all successors before deciding the path. Rather, the selection of path is completely random in which a neighboring node is selected at random and the improvement is evaluated on the currently selected node, and depending on the evaluation of the improvement or path nearest to the solution it selects the node and explore the path further.

- **Random-restart hill climbing**

Random-restart is a variant of simple hill climbing in which hill climbing search is performed iteratively by randomly selecting some condition initially, the next best state space is stored and replaced with the existing one if it is better than the previous one.It is a meta-algorithm built on top of the hill climbing algorithm. It is also known as Shotgun hill climbing.

- **Sideways move hill climbing**

Sideways move hill climbing allows iterations when a plateau is reached in the hope that the current flat local maxima is shoulder and global maxima can be reached. The iterations will go till infinity when the flat local maxima is just flat surface of local maxima.

II. Program structure

2.1 Global variables

Variable names	Variable description
Conflicts	stores the total number of rows and diagonal conflicts
OptimalSol	stores duplicate of the current state to perform many operations
CurrentMin	variable that stores the conflicts of current board so that if finds better than this it will exit
RestartsSum	stores the sum of total number of restarts
MovesSum	Stores the sum of total moves
OptimalMoves	Stores the sum of total moves to reach the solution set.

2.2 Functions and Procedures

Function/procedure	Description
RowConflict	returns the number of row conflicts for a particular queen in a particular position
DiagonalConflict	returns the number of diagonal conflicts for a particular queen in a particular position
ConflictSum	Returns the sum of rows and diagonal conflicts

OptimalSolution	This method calculates the conflicts for the current state of the board and exits whenever finds a better state.
PrintBoard	This function uses set method in python to check if the current state is a duplicate of any other state that has been traversed in past or not and returns 1 if it is true and 0 if it is not.
GenerateRandomBoard	This function helps in generating the board for restart function randomly
CheckSoluton	This function checks for the current state that whether it is a solution or not by checking whether the total number of conflicts is zero
FindMinConflict	This method finds the solution for the n-queens problem with Min-Conflicts algorithm with random restart
FillBoard	Fills the array with the board elements
NoRestartFindMinConflict	This method finds the solution for the n-queens problem with Min-Conflicts algorithm without random restart

2.3 SOURCE CODE

1. Hill Climbing steepest ascent with restart

```
import random
import copy

class NQueen(object):
    # Below function verifies whether the current state of the board is the solution(I.e with zero conflicts)
    def CheckSolution(self, a, n):
        if self.ConflictSum(a, n) == 0:
            return True
        return False


    # This method calculates all the diagonal conflicts for a particular position of the queen
    def DiagonalConflict(self, a, n):
        conflicts = 0
        d = 0
        i = 0
        while i < n:
            j = 0
            while j < n:
                if i != j:
                    d = abs(i - j)
                    if (a[i] == a[j] + d) or (a[i] == a[j] - d):
                        conflicts += 1
                j += 1
            i += 1
        return conflicts


    # This method calculates the conflicts for the current state of the board and quits whenever finds a better state.
    def OptimalSolution(self, a, n):
        conflicts = 0
        row = -1
        col = -1
        bettersol = False
        optimalSol = []
        # Sets min variable to the conflicts of current board so that if finds better than this it will quit.
        currentMin = self.ConflictSum(a, n)
        optimalSol = copy.deepcopy(a)
        # Create a duplicate array for handling different operations
        i = 0
        while i < n:
            # This iteration is for each column
            if bettersol:
                # If it finds and better state than the current, it will quit
```

```
            break
        m = optimalSol[i]
        j = 0
        while j < n:
            # This iteration is for each row in the selected column
            if j != m:
                # This condition ensures that, current queen position is not taken into consideration.
                optimalSol[i] = j
                conflicts = self.ConflictSum(optimalSol, n)
                if currentMin > conflicts:
                    # If a better state is found, that particular column and row values are stored
                    col = i
                    row = j
                    currentMin = conflicts
                    bettersol = True
                    break
            optimalSol[i] = m
            # Restoring the array to the current board position
            j += 1

        i += 1
    if col == -1 or row == -1:
        # If there is no better state found
        print("local maxima at " ,conflicts ," calling random regenerate")
        return False
    a[col] = row
    return True
    # Returns true to the main function if there is any better state found


# This method returns total number of conflicts for a particular queen position
def ConflictSum(self, a, n):
    conflicts = 0
    conflicts = self.RowConflict(a, n) + self.DiagonalConflict(a, n)
    return conflicts

# Below function generates a random state of the board
def GenerateRandomBoard(self, a, n):
    i = 0
    while( i < n):
        a[i] =random.randint(0,n-1) + 0
        i += 1


# This method calculates all the row conflicts for a queen placed in a particular cell.
def RowConflict(self, a, n):
    conflicts = 0
    i = 0
    while i < n:
        j = 0
        while j < n:
            if i != j:
                if a[i] == a[j]:
                    conflicts += 1
```

```
            j += 1
          i += 1
        return conflicts

restartsSum = 0
movesSum = 0
optimalMoves = 0

print("Please select one from the below options:")
print("1. Hill Climbing and Random Restart")
print("2.exit ")

choice = int(input("enter your choice"))
        # Randomly generate the board
if choice == 1:
    n = int(input("Please enter the value of n:"))
    if (n > 1 and n < 4) or n <= 1:
        print("*Please choose n value either greater than 3 or equals to 1 - Program Terminated")
        exit()
    a = [None] * n
    b = [None] * n
    queens = NQueen()
    queens.GenerateRandomBoard(a, n)
    b = copy.deepcopy(a)

    print("$$$$$$$$$$    Hill Climbing with Random Restart  $$$$$$$$$$$$$$$")
    while not queens.CheckSolution(a, n):
            # Executes until a solution is found
        if queens.OptimalSolution(a, n):
                # If a better state for a board is found
            movesSum += 1
            optimalMoves += 1
            continue
        else:
                # If a better state is not found
            optimalMoves = 0
            queens.GenerateRandomBoard(a, n)
                # Board is generated Randomly
            restartsSum += 1
    print("The Number of restarts: ",restartsSum)
    print("Total number of moves: ",movesSum-1)
    # Gives the total number of moves from starting point
    print("Number of moves in the solution set: ",optimalMoves)
    # Gives number of steps in the solution set.
    i = 0
    while i < n:
        j = 0
        while j < n:
            if j == a[i]:
                print(" Q ", end="")
            else:
                print(" x ", end="")
            j += 1
        print()
```

```
        i += 1
    if(restartsSum == 0):
         print("Average steps",movesSum)
    else:
        print("Average steps",movesSum/restartsSum)

if choice == 2:
    exit()
```

2.Sideways with restart

```
import random
import copy

class NQueen(object):
#generates the board randomly for random restarts
    def GenerateRamdomBoard(self, a, n):
        i = 0
        while( i < n):
            a[i] =random.randint(0,n-1) + 0
            i += 1

    def FillBoard(self, store, n):
        i = 0
        while i < n:
            store.append(i)
            i += 1
        return

    # This method calculates all the diagonal conflicts for a particular position of the queen
    def DiagonalConflict(self, a, n):
        conflicts = 0
        d = 0
        i = 0
        while i < n:
            j = 0
            while j < n:
                if i != j:
                    d = abs(i - j)
                    if (a[i] == a[j] + d) or (a[i] == a[j] - d):
                        conflicts += 1
                j += 1
            i += 1
```

```
        return conflicts
    # This method calculates all the row conflicts for a queen placed in a particular cell.
    def RowConflict(self, a, n):
        conflicts = 0
        i = 0
        while i < n:
            j = 0
            while j < n:
                if i != j:
                    if a[i] == a[j]:
                        conflicts += 1
                j += 1
            i += 1
        return conflicts

    # This method returns total number of conflicts for a particular queen position
    def ConflictSum(self, a, n):
        conflicts = 0
        conflicts = self.RowConflict(a, n) + self.DiagonalConflict(a, n)
        return conflicts


    def CheckSolution(self, a, n):
        if self.ConflictSum(a, n) == 0:
            return True
        return False


    def FindMinConflict(self, b, n, iterations):
        store = []
        self.FillBoard(store, n)
        restartsSum = 0
        movesSum = 0
        optimalMoves = 0
        row = 0
        maxSteps = iterations
        # The maximum steps that can be allowed to find a solution with this algorithm
        while not self.CheckSolution(b, n):
            # Loops until it finds a solution,
            randomSelection = random.randint(0,len(store)-1) + 0
            # Randomly selects a column from the available
            currentValue = b[store[randomSelection]]
            # This stores the current queue position in the randomly selected column
            randomValue = store[randomSelection]
            currentMin = self.FindColumnCollisions(b, n, randomValue)
            # Sets the minimum variable to the current queue conflicts
            min_compare = currentMin
            while(not store):
                store.remove(randomSelection)
            i = 0
            while i < n:
                if currentValue != i:
                    b[randomValue] = i
                    col = self.FindColumnCollisions(b, n, randomValue)
```

```
            # Calculates the conflicts  of the queen at particular position
            if col < currentMin:
                currentMin = col
                row = i
        i += 1
    if min_compare == currentMin:
        # When there is no queen with minimum conflicts than the current position
        if maxSteps != 0:
            # Checks if the maximum steps is reached
            if len(store) >= 0:
                # checks whether there are columns available in the Array List
                b[randomValue] = currentValue
                # restores the queen back to the previous position
                maxSteps -= 1
            else:
                self.FillBoard(store, n)
        else:
            # If the max steps is reached then, the board is regenerated and initiated the max steps variable
            restartsSum += 1
            optimalMoves = 0
            self.GenerateRamdomBoard(b, n)
            self.FillBoard(store, n)
            maxSteps = iterations
    else:
        # When we find the the position in the column with minimum conflicts
        movesSum += 1
        optimalMoves += 1
        b[randomValue] = row
        min_compare = currentMin
        store.clear()
        maxSteps -= 1
        self.FillBoard(store, n)
print()
i = 0
while i < n:
    j = 0
    while j < n:
        if j == b[i]:
            print(" Q ", end="")
        else:
            print(" x ", end="")
        j += 1

    print()
    i += 1
print("Total number of Random Restarts: ",restartsSum)
print("Total number of Moves: ", movesSum)
print("Number of Moves in the solution set: ", optimalMoves)

# Below function returns the conflicts of a queen in a particular column of the board

def FindColumnCollisions(self, b, n, index):
    conflicts = 0
    t = 0
```

```
        i = 0
        while i < n:
            if i != index:
                t = abs(index - i)
                if b[i] == b[index]:
                    conflicts += 1
                elif b[index] == b[i] + t or b[index] == b[i] - t:
                    conflicts += 1
            i += 1
        return conflicts



print("Please select one from the below options:")
print("1. Sideways Hill Climbing With Random Restart ")
print("2.exit ")

choice = int(input("enter your choice"))
if choice == 1:
    n = int(input("Please enter the value of n(no. of queens):"))
    a = [None] * n
    b = [None] * n
    queens = NQueen()
    queens.GenerateRamdomBoard(a, n)
    b = copy.deepcopy(a)
    iterations = 0
    print()
    print(" $$$$$$$  Sideways Hill Climbing With Random Restart $$$$$$$$ ")
    iterations= int(input("Please enter the  number of steps for iteration:"))
    queens.FindMinConflict(b, n, iterations)


if choice  == 2:
    exit()
```

3. Hill Climbing no restart

```
import random
import time
import copy

class NQueen(object):


        def PrintBoard(self,OptimalSolution,n):
        i = 0
```

```
while i < n:
j = 0
while j < n:
if j == OptimalSolution[i]:
        print(" Q ", end="")
else:
        print(" x ", end="")
j += 1
print()
i += 1
```

This class calculates the collisions or conflicts in a row that any two queens are getting in a particular position.

```
def RowConflict(self, a, n):
conflicts = 0
i = 0
while i < n:
j = 0
while j < n:
if i != j:
        if a[i] == a[j]:
        conflicts += 1
j += 1
i += 1
return conflicts
```

This class calculates the number of collisions or conflicts diagonally that any particular queen is getting in the current cell

```
def DiagonalConflict(self, a, n):
conflicts = 0
d = 0
i = 0
while i < n:
j = 0
while j < n:
if i != j:
        d = abs(i - j)
        if (a[i] == a[j] + d) or (a[i] == a[j] - d):
        conflicts += 1
j += 1
i += 1
return conflicts
```

This class sums up the row and diagonal conflict of a particular queen in a particular cell.

```
def ConflictSum(self, a, n):
conflicts = 0
conflicts = self.RowConflict(a, n) + self.DiagonalConflict(a, n)
return conflicts
```

#This function checks for the current state that whether it is a solution or not by checking whether the total number of conflicts is zero

```
def CheckSolution(self, a, n):
```

```
            if self.ConflictSum(a, n) == 0:
            return True
            return False

# This function randomly generates the board
            def GenerateRandomBoard(self, a, n):

            i = 0
            while( i < n):
            a[i] =random.randint(0,n-1) + 0
            i += 1
# This class calls ConflictSum and takes the sum of conflicts from that function for the current state and exit the
program whenever a better state is found
            def OptimalSolution(self, a, n):

            global movesSum
            conflicts = 0
            row = -1
            col = -1

            betterSol = False
            OptimalSolution = []
# Puts CurrentMin variable to the sume of conflicts of current state to initiate exit whenever a better state is found
            CurrentMin = self.ConflictSum(a, n)
            OptimalSolution = copy.deepcopy(a)
# This code creates a new array and column Copys the current state to it so that many operations and checking can
be done on this current state
            i = 0
            while i < n:
# This traverse through each column of the new array
            if betterSol:
# Break will be initiated if a better state than current state is found
            break
            m = OptimalSolution[i]
            j = 0
            while j < n:
# This array traversal is iterated over each row against the current column
            if j != m:
#The above condition makes sure that current position of the queen is not selected for further operations
                        OptimalSolution[i] = j
                        self.PrintBoard(OptimalSolution,n)
                        movesSum +=1

# This assigns the queen to the iterated places and then calculates the sum of conflict for that particular position of
the queen
                        conflicts = self.ConflictSum(OptimalSolution, n)
                        if CurrentMin > conflicts:
# this checks for next better state and if it is found then the current values of array positions are stored
                        col = i
                        row = j
                        CurrentMin = conflicts
                        betterSol = True
                        break
            OptimalSolution[i] = m
```

The array is restored to the board with current positions
            j += 1
            i += 1
            if col == -1 or row == -1:
#this checks for no further better state
            return False
            a[col] = row
            return True
#The OptimalSolutions returns the boolean value as true if the next state that is explored is better than the current state

```
# this function evaluates the solution for N queens with Minimum conflict algorithm
        def FindMinConflict(self, b, n, iterations):
# This array will sstore the This array list is for storing the columns from which a random column will be selected
        columnCopy = []
        self.FillBoard(columnCopy, n)

        randomCount = 0
        movesSum = 0
        OptimalMoves = 0
        row = 0
        maxSteps = iterations
        # The maximum steps that can be allowed to find a solution with this algorithm
        while not self.CheckSolution(b, n):
        # Loops until it finds a solution,
        randomSelection = random.randint(0,len(columnCopy)-1) + 0
        # Randomly selects a column from the available
        currentValue = b[columnCopy[randomSelection]]
        # This stores the current queue position in the randomly selected column
        randomValue = columnCopy[randomSelection]
        CurrentMin = self.FindColumnCollisions(b, n, randomValue)
        # Sets the minimum variable to the current queue conflicts
        min_compare = CurrentMin

        while(not columnCopy):
        columnCopy.remove(randomSelection)
        i = 0
        while i < n:
        if currentValue != i:
                b[randomValue] = i
                col = self.FindColumnCollisions(b, n, randomValue)
                # Calculates the conflicts of the queen at particular position
                if col < CurrentMin:
                CurrentMin = col
                row = i
        i += 1
        if min_compare == CurrentMin:
        # When there is no queen with minimum conflicts than the current position
        if maxSteps != 0:
                # Checks if the maximum steps is reached
                if len(columnCopy) >= 0:
                # checks whether there are columns available in the Array List
                b[randomValue] = currentValue
```

```
                # recolumnCopys the queen back to the previous position
                maxSteps -= 1
                else:
                self.FillBoard(columnCopy, n)
        else:
                # If the max steps is reached then, the board is regenerated and initiated the max steps variable
                randomCount += 1
                OptimalMoves = 0
                self.FillBoard(columnCopy, n)
                maxSteps = iterations
        else:
        # When we find the the position in the column with minimum conflicts
        OptimalMoves += 1
        b[randomValue] = row
        min_compare = CurrentMin
        columnCopy.clear()
        maxSteps -= 1
        self.FillBoard(columnCopy, n)
        print()
        i = 0
        while i < n:
        j = 0
        while j < n:
        if j == b[i]:
                print(" Q ", end="")
        else:
                print(" x ", end="")
        j += 1

        print()
        i += 1

        print("Number of Moves in the solution set: ", OptimalMoves)



# Below function returns the conflicts of a queen in a particular column of the board
        @classmethod
        def FindColumnCollisions(self, b, n, index):
        conflicts = 0
        t = 0
        i = 0
        while i < n:
        if i != index:
        t = abs(index - i)
        if b[i] == b[index]:
                conflicts += 1
        elif b[index] == b[i] + t or b[index] == b[i] - t:
                conflicts += 1
        i += 1
        return conflicts

# Below function fills the Array List with numbers 0 to n-1
        @classmethod
```

```
        def FillBoard(self, columnCopy, n):
        i = 0
        while i < n:
        columnCopy.append(i)
        i += 1
        return

succrate = 0
failrate = 0
executeiter = 3
totalRestart = 0
movesSum = 0

print("Please select one from the below options:")
print("1. Solve n queens with Hill Climbing without restart ")
print("2. exit")
choice = int(input("Please enter the choice:"))
if choice == 1:

        n = int(input("Please enter the value of n(no. of queens):"))
        if (n > 1 and n < 4) or n <= 1:
        print("*Please choose n value either greater than 3 or equals to 1 - Program Terminated")
        exit()
        if choice < 1 or choice > 7:
        print("*Program terminated - Wrong option selected")
        exit()
        executeiter = int(input("Please enter the number of times the reporting needs to be done"))
        while (executeiter != 0):
        a = [None] * n
        b = [None] * n
        OptimalMoves = 0
        queens = NQueen()
        queens.GenerateRandomBoard(a, n)
 # Randomly generate the board
        b = copy.deepcopy(a)
        print("**********Steepest Ascent without Random Restart*********")

        while not queens.CheckSolution(a, n):
#run till a the optimal solution is found \
        if queens.OptimalSolution(a, n):
#check for better state
        movesSum += 1
        OptimalMoves += 1
        break

        if queens.CheckSolution(a, n) == True:
        print("Solution found")
        succrate += 1
        else:
        print("Solution not found")
        failrate += 1
#Total number of moves
        print("Total moves count ",movesSum-1)
#Total number of moves in the solution
```

```
            print("Total moves count in solution set ",OptimalMoves)

            i = 0
            while i < n:
            j = 0
            while j < n:
            if j == a[i]:
                        print(" Q ", end="")
            else:
                        print(" x ", end="")
            j += 1
            print()
            i += 1
            executeiter -= 1
            else:
            print("success rate =",succrate/(succrate+failrate)*100)
            print("failure rate =",failrate/(succrate+failrate)*100)

if choice == 2:
            exit()
```

4. Sideways with No restart

```
import random
import copy

class NQueen(object):
   # This method calculates all the row conflicts for a queen placed in a particular cell.
   def RowConflict(self, a, n):

      Conflict = 0
      i = 0
      while i < n:
         j = 0
         while j < n:
            if i != j:
               if a[i] == a[j]:
                  Conflict += 1
            j += 1
         i += 1
      return Conflict

   # This method calculates all the diagonal conflicts for a particular position of the board
   def DiagonalConflict(self, a, n):
      Conflict = 0
      d = 0
      i = 0
      while i < n:
         j = 0
         while j < n:
            if i != j:
```

```
                d = abs(i - j)
                if (a[i] == a[j] + d) or (a[i] == a[j] - d):
                    Conflict += 1
            j += 1
        i += 1
    return Conflict

# This method returns total number of Conflict for a particular queen position
def ConflictSum(self, a, n):
    Conflict = 0
    Conflict = self.RowConflict(a, n) + self.DiagonalConflict(a, n)
    return Conflict

def printBoard(self,b,n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            if j == b[i]:
                print(" Q ", end="")
            else:
                print(" x ", end="")
            j += 1
        print()
        i += 1



# Below function generates a random state of the board
def GenerateRandomBoard(self, a, n):
    i = 0
    while( i < n):
        a[i] =random.randint(0,n-1) + 0
        i += 1


# Below function verifies whether the current state of the board is the solution
def CheckSolution(self, a, n):
    if self.ConflictSum(a, n) == 0:
        return True
    return False

# Below method finds the solution for the n-queens problem with Min-Conflicts algorithm
def FindMinConflict(self, b, n, iterations):
    store = []
    self.FillBoard(store, n)
    movesSum = 0
    optimalMoves = 0
    row = 0
    maxSteps = iterations
    # The maximum steps that can be allowed to find a solution with this algorithm
    while not self.CheckSolution(b, n):
        # Loops until it finds a solution
        randomSelection = random.randint(0,len(store)-1) + 0
```

```
    # Randomly selects a column from the available
    currentValue = b[store[randomSelection]]
    # This stores the current queue position in the randomly selected column
    randomValue = store[randomSelection]
    currentMin = self.FindColumnCollision(b, n, randomValue)
    # Sets the minimum variable to the current queue Conflict
    min_compare = currentMin
    while(not store):
        store.remove(randomSelection)
    i = 0
    while i < n:
        if currentValue != i:
            b[randomValue] = i

            col = self.FindColumnCollision(b, n, randomValue)
            # Calculates the Conflict of the queen at particular position
            if col < currentMin:
                currentMin = col
                row = i
        i += 1
    if min_compare == currentMin:
        # When there is no queen with minimum conflicts than the current position
        if maxSteps != 0:
            # Checks if the maximum steps is reached
            if len(store) >= 0:
                # checks whether there are columns available in the Array List
                b[randomValue] = currentValue

                # restores the queen back to the previous position
                maxSteps -= 1

            else:
                self.FillBoard(store, n)
        else:
            break
    else:
        # When we find the the position in the column with minimum conflicts
        movesSum += 1
        optimalMoves += 1
        b[randomValue] = row
        min_compare = currentMin
        store.clear()
        maxSteps -= 1
        self.FillBoard(store, n)
print()
i = 0

while i < n:
    j = 0
    while j < n:
        if j == b[i]:
            print(" Q ", end="")
        else:
            print(" x ", end="")
```

```
            j += 1

        print()
        i += 1
    print("Total number of Moves: ", movesSum)
    print("Number of Moves in the solution set: ", optimalMoves)


def noRestartFindMinConflict(self, b, n, iterations):
    store = []
    self.FillBoard(store, n)
    global succrate
    global failrate
    global movesSum
    optimalMoves = 0
    row = 0
    maxSteps = iterations
    # The maximum steps that can be allowed to find a solution with this algorithm
    while not self.CheckSolution(b, n):
        # Loops until it finds a solution,
        randomSelection = random.randint(0,len(store)-1) + 0
        # Randomly selects a column from the available
        currentValue = b[store[randomSelection]]
        # This stores the current queue position in the randomly selected column
        randomValue = store[randomSelection]
        currentMin = self.FindColumnCollision(b, n, randomValue)
        # Sets the minimum variable to the current queue Conflict
        min_compare = currentMin
        while(not store):
            store.remove(randomSelection)
        i = 0
        while i < n:
            if currentValue != i:
                b[randomValue] = i
                col = self.FindColumnCollision(b, n, randomValue)
                # Calculates the Conflict of the queen at particular position
                if col < currentMin:
                    currentMin = col
                    row = i
            i += 1

        if min_compare == currentMin:
            # When there is no queen with minimum conflicts than the current position
            if maxSteps != 0:
                # Checks if the maximum steps is reached
                if len(store) > 0:
                    # checks whether there are columns available in the Array List
                    b[randomValue] = currentValue
                    # restores the queen back to the previous position
                    maxSteps -= 1

                else:
                    self.FillBoard(store, n)
            else:
```

```
                break
        else:
            # When we find the the position in the column with minimum conflicts

            optimalMoves += 1
            b[randomValue] = row
            min_compare = currentMin
            store.clear()
            maxSteps -= 1
            self.FillBoard(store, n)

    print()
    i = 0
    while i < n:
        j = 0
        while j < n:
            if j == b[i]:
                print(" Q ", end="")
            else:
                print("x ", end="")
            j += 1
        print()
        i += 1
    print(" total moves ",movesSum)
    print("Number of Moves in the solution set: ", optimalMoves)

def noRestartFindMinConflict(self, b, n, iterations):
    store = []
    self.FillBoard(store, n)
    movesSum = 0
    optimalMoves = 0
    global succstep, failstep,c

    row = 0
    maxSteps = iterations
    # The maximum steps that can be allowed to find a solution with this algorithm
    while not self.CheckSolution(b, n):
        # Loops until it finds a solution,
        randomSelection = random.randint(0,len(store)-1) + 0
        # Randomly selects a column from the available
        currentValue = b[store[randomSelection]]
        # This stores the current queue position in the randomly selected column
        randomValue = store[randomSelection]
        currentMin = self.FindColumnCollision(b, n, randomValue)
        # Sets the minimum variable to the current queue Conflict
        min_compare = currentMin
        while(not store):
            store.remove(randomSelection)
        i = 0
        while i < n:
            if currentValue != i:
                b[randomValue] = i
                col = self.FindColumnCollision(b, n, randomValue)
                # Calculates the Conflict of the queen at particular position
```

```
            if col < currentMin:
                currentMin = col
                row = i
        i += 1

        self.printBoard(b,n)
        print("\n")
        c+=1
    if min_compare == currentMin:
        # When there is no queen with minimum conflicts than the current position
        if maxSteps != 0:

            # Checks if the maximum steps is reached
            if len(store) > 0:
                # checks whether there are columns available in the Array List
                b[randomValue] = currentValue
                # restores the queen back to the previous position
                maxSteps -= 1
            else:
                self.FillBoard(store, n)
        else:
            break
    else:
        # When we find the the position in the column with minimum conflicts
        movesSum += 1
        optimalMoves += 1
        b[randomValue] = row
        min_compare = currentMin
        store.clear()
        maxSteps -= 1
        self.FillBoard(store, n)

print()
i = 0
while i < n:
    j = 0
    while j < n:
        if j == b[i]:
            print(" Q ", end="")
        else:
            print(" x ", end="")
        j += 1
    print()
    i += 1
print("Total number of Moves: ", movesSum)
print("Number of Moves in the solution set: ", optimalMoves)

if queens.CheckSolution(b,n):
    succstep+=movesSum
else:
    failstep+=movesSum
```

```
    # Below function returns the Conflict of a queen in a particular column of the board

    def FindColumnCollision(self, b, n, index):
        """ generated source for method FindColumnCollision """
        Conflict = 0
        global c
        t = 0
        i = 0
        while i < n:
            if i != index:
                t = abs(index - i)
                if b[i] == b[index]:
                    Conflict += 1
                elif b[index] == b[i] + t or b[index] == b[i] - t:
                    Conflict += 1
            i += 1


        return Conflict

    # Below function fills the Array List with numbers 0 to n-1
    def FillBoard(self, store, n):
        """ generated source for method FillBoard """
        i = 0
        while i < n:
            store.append(i)
            i += 1
        return

movesSum = 0
optimalMoves = 0
succrate = 0
failrate = 0



print("Please select one from the below options:")
print("1. Min Conflict method without random restart")
print("2.exit")
choice = int(input("Please enter the choice:"))

if choice == 1:
    successrate=0
    failurerate=0
    c=0
    succstep=0
    failstep=0
    n = int(input("Please enter the value of n:"))

    if (n > 1 and n < 4) or n <= 1:
        print("*Please choose n value either greater than 3 or equals to 1 - Program Terminated")
        exit()
    a = [None] * n
```

```
    b = [None] * n
    queens = NQueen()
    queens.GenerateRandomBoard(a, n)
        # Randomly generate the board
    b = copy.deepcopy(a)
    print(" $$$$$$$$  Min Conflict Sideways Without Random Restart  $$$$$$$$")
    print("Enter the number of times to execute")
    notimes = int(input("Please enter the value:"))
    print("Please enter the maximum number of steps for iteration:")
    iterations = int(input("Please enter the value:"))
    while (notimes!=0):
        queens.noRestartFindMinConflict(b, n, iterations)

        movesSum += 1
        if queens.CheckSolution(b,n):

            print('Solution Found')
            successrate+=1
        else:
            print('No solution Found')
            failurerate+=1
        print(notimes)
        notimes = notimes - 1
        queens.GenerateRandomBoard(b,n)
    if(failstep == 0):
        print(" average success steps",succstep)
    else:
         print(" average failure steps",succstep/c)

    print('successrate- '+str(successrate))
    print('failurerate- '+str(failurerate))
    print('Success moves- '+str(succstep))
    print("moves",c)

if choice == 2:
    exit()
```

2.3 SAMPLE OUTPUT

1. Hill climbing with restart

```
Please select one from the below options:
1. Hill Climbing and Random Restart
2.exit
enter your choice1
Please enter the value of n:8
$$$$$$$$$$    Hill Climbing with Random Restart  $$$$$$$$$$$$$$$
local maxima at  4  calling random regenerate
local maxima at  10  calling random regenerate
local maxima at  8  calling random regenerate
local maxima at  6  calling random regenerate
local maxima at  8  calling random regenerate
The Number of restarts:  5
Total number of moves:  18
Number of moves in the solution set:  4
 x  x  Q  x  x  x  x  x
 x  x  x  x  x  Q  x  x
 x  x  x  x  x  x  x  Q
 Q  x  x  x  x  x  x  x
 x  x  x  x  Q  x  x  x
 x  x  x  x  x  x  Q  x
 x  Q  x  x  x  x  x  x
 x  x  x  Q  x  x  x  x
Average steps 3.8
>>> |
```

2. Sideways hill climbing with restart

```
Please select one from the below options:
1. Sideways Hill Climbing With Random Restart
2.exit
enter your choice1
Please enter the value of n:8

 $$$$$$$  Sideways Hill Climbing With Random Restart $$$$$$$$
Please enter the  number of steps for iteration:100

 x  x  x  Q  x  x  x  x
 x  x  x  x  x  Q  x  x
 x  x  x  x  x  x  x  Q
 x  Q  x  x  x  x  x  x
 x  x  x  x  x  x  Q  x
 Q  x  x  x  x  x  x  x
 x  x  Q  x  x  x  x  x
 x  x  x  x  Q  x  x  x
Total number of Random Restarts:  2
Total number of Moves:  14
Average Restarts  7.0
Number of Moves in the solution set:  5
>>> |
```

3. Hill climbing without random restart

Please select one from the below options:
1. Solve n queens with Hill Climbing without restart
2. exit
Please enter the choice:1
Please enter the value of n:8
Please enter the number of times the reporting needs to be done:10
**********Steepest Ascent without Random Restart*********

```
Q x x x x x x x
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x x Q x x x x
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x x x Q x x x
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x x x x Q x x
x x x x Q x x x
```

x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x x x x x x x Q
x x x x Q x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x Q x x x x x x
Q x x x x x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x Q x x x x x x
x Q x x x x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x Q x x x x x x
x x Q x x x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x

x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x Q x x x x x x
x x x Q x x x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
x Q x x x x x x
x x x x x Q x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
Solution not found
Total moves count 12
Total moves count in solution set 1
x Q x x x x x x
x x x x x Q x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x x x Q x x x
x x x x x x x Q
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x Q x x x x x x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x

x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x x Q x x x x x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x x x Q x x x x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x x x x Q x x x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x x x x x x Q x
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
x x x x x x x Q
x Q x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x

x Q x x x x x x
x x x x x Q x x
Q x x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
Solution not found
Total moves count 21
Total moves count in solution set 1
x x x x x Q x x
Q x x x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x Q x x x x x x
x Q x x x x x x
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
Q x x x x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x x Q x
x x x x x x Q x
Q x x x x x x x
x x x x Q x x x
x Q x x x x x x
Q x x x x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x x Q x
x x x x x x Q x
Q x x x x x x x
x x x x Q x x x
x x x Q x x x x
Q x x x x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x x Q x
x x x x x x Q x

Q x x x x x x x
x x x x Q x x x
Solution not found
Total moves count 25
Total moves count in solution set 1
x x x Q x x x x
Q x x x x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x x Q x
x x x x x x Q x
Q x x x x x x x
x x x x Q x x x
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
x x x x x x Q x
x x x x x x Q x
x x x x x x Q x
x x x x Q x x x
x x x x Q x x x
x x x x x Q x x
x x x x Q x x x
Solution not found
Total moves count 27
Total moves count in solution set 1
Q x x x x x x x
x x x x x x Q x
x x x x x x Q x
x x x x x x Q x
x x x x Q x x x
x x x x Q x x x
x x x x x Q x x
x x x x Q x x x
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x
x x Q x x x x x
x x x x Q x x x
x Q x x x x x x
Q x x x x x x x
Solution not found

Total moves count 29

Total moves count in solution set 1

Q x x x x x x x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x
x x Q x x x x x
x x x x Q x x x
x Q x x x x x x
Q x x x x x x x

**********Steepest Ascent without Random Restart*********

x Q x x x x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x x Q
x x Q x x x x x
x x x x x x Q x
Q x x x x x x x
x x x Q x x x x
x x Q x x x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x x Q
x x Q x x x x x
x x x x x x Q x
Q x x x x x x x
x x x Q x x x x
x x x Q x x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x x Q
x x Q x x x x x
x x x x x x Q x
Q x x x x x x x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x x Q
x x Q x x x x x
x x x x x x Q x
Q x x x x x x x
x x x Q x x x x

Solution not found
Total moves count 34
Total moves count in solution set 1

```
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x x Q
x x Q x x x x x
x x x x x x Q x
Q x x x x x x x
x x x Q x x x x
```

**********Steepest Ascent without Random Restart*********

```
Q x x x x x x x
x x x x x x x Q
Q x x x x x x x
Q x x x x x x x
x x x x x x x Q
x x x x x Q x x
x x x x x x x Q
x x x x Q x x x
x Q x x x x x x
x x x x x x x Q
Q x x x x x x x
Q x x x x x x x
x x x x x x x Q
x x x x x Q x x
x x x x x x x Q
x x x x Q x x x
```

Solution not found
Total moves count 37
Total moves count in solution set 1

```
x Q x x x x x x
x x x x x x x Q
Q x x x x x x x
Q x x x x x x x
x x x x x x x Q
x x x x x Q x x
x x x x x x x Q
x x x x Q x x x
```

**********Steepest Ascent without Random Restart*********

```
x Q x x x x x x
x x x x x x x Q
x x Q x x x x x
```

Q x x x x x x x
x x Q x x x x x
x x x x x x Q x
x x x x x x x Q
Q x x x x x x x
Solution not found
Total moves count 39
Total moves count in solution set 1
x Q x x x x x x
x x x x x x x Q
x x Q x x x x x
Q x x x x x x x
x x Q x x x x x
x x x x x x Q x
x x x x x x x Q
Q x x x x x x x
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x Q x x x x x x
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x Q x x x x x
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x x x Q x x x
x Q x x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x x x x Q x x
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x x x x x Q x
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x x x x x x Q
x Q x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
x x x Q x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
Solution not found
Total moves count 48

Total moves count in solution set 1
x x x Q x x x x
Q x x x x x x x
x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x x x Q x x
**********Steepest Ascent without Random Restart*********
Q x x x x x x x
x x x x x Q x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x x x x x x Q x
x x x x x Q x x
x x x Q x x x x
Solution not found
Total moves count 50
Total moves count in solution set 1
Q x x x x x x x
x x x x x Q x x
x x x x x x Q x
x x x x Q x x x
x x x Q x x x x
x x x x x x Q x
x x x x x Q x x
x x x Q x x x x
success rate = 0.0
failure rate = 100.0

4. Sideways with no restart approach

Please select one from the below options:
1. Min Conflict method without random restart
2.exit
Please enter the choice:1
Please enter the value of n(no.of queens):8

$$$$$$$$ Min Conflict Sideways Without Random Restart $$$$$$$$
Enter the number of times to execute
Please enter the value:1
Please enter the maximum number of steps for iteration:
Please enter the value:100

```
Q x x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x


Q x x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x


x x Q x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x


x x x Q x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x
```

x x x x Q x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x Q x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x x Q x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x x x Q
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x

Q x x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x Q x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x Q x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x

x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x Q x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
Q x x x x x x x
x x x Q x x x x
x x x x x Q x x

x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x Q x x x x x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x Q x x x x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x Q x x x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x Q x x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x Q x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x Q x x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x x Q
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
Q x x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x Q x x x x x x
x x x x Q x x x

Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x Q x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x

x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x Q x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
Q x x x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x Q x x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x Q x x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x Q x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x Q x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x Q x x

x Q x x x x x x
x x x x Q x x x

x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x x Q x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x x x Q

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x Q x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x Q x x x x x
x x x x x x Q x

x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x Q x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x Q x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x Q x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
Q x x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x Q x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x Q x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x

x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x Q x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x

x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

Q x x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

Q x x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x Q x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x Q x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x

x x Q x x x x x

x x x x Q x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x Q x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x x Q x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x x x x x x x Q
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
Q x x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x Q x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x Q x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x Q x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x

Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x Q x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x

x Q x x x x x x
Q x x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x

x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x Q x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x Q x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x Q x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
Q x x x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x Q x x x x x x

x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x Q x x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x Q x x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x Q x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x

x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x Q x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
Q x x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x Q x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x Q x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x Q x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x

x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x Q x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x Q x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x Q x x x x x

x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x Q x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x Q x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x Q x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x

x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
x x x x x x x Q
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
Q x x x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x Q x x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x Q x x x x x x

x Q x x x x x x

x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x Q x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x Q x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x Q x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x x Q x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x

x x x x x x Q x
x x x Q x x x x
x x x x x Q x x
x x x x x x x Q

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
Q x x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x Q x x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x Q x x x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x Q x x x x

x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x Q x x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x Q x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
Q x x x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x Q x x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x Q x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x Q x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x

Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x x x Q x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x x x x Q x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x x x Q x x x
Q x x x x x x x
x x x x x x Q x
x x x x x x x Q
x x x x x Q x x
x x Q x x x x x

this goes on to give output

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x

x x x x x x Q x
x x x Q x x x x
x x x x x x x Q
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
Q x x x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x Q x x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x Q x x x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x Q x x x x x
x x x x x Q x x

x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x Q x x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x x Q x x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x x x Q x
x x x x x Q x x
x x Q x x x x x

x Q x x x x x x
x x x x Q x x x
x x Q x x x x x
Q x x x x x x x
x x x x x x Q x
x x x x x x x Q
x x x x x Q x x
x x Q x x x x x

```
 x Q x x x x x x
 x x x x Q x x x
 x x Q x x x x x
 Q x x x x x x x
 x x x x x x Q x
 x x x x x x x Q
 x x x x x Q x x
 x x Q x x x x x
```

Total number of Moves: 1

Number of Moves in the solution set: 1

No solution Found

1

average failure steps 0.0

successrate- 0

failurerate- 1

Success moves- 0

moves 808

CONCLUSION

As a conclusion we have implemented the hill climbing approach to solve the n queen problem using the hill climbing algorithm. We have implemented two approach with restart and the same approach without restart.

we were able identify that both hill climbing steepest and sideways without restart approach would not succeed many times as it would get stuck in a local maxima. where as the same 2 approaches used with restart would lead us to the definite solution, therefore we use the random restart approach this resolves the problem

REFERENCES

1. http://www.d.umn.edu/~jrichar4/8puz.html
2. https://www.cs.princeton.edu/courses/
3. http://stackoverflow.com
4. http://www.github.com