Assignment - 7

Aim

8-Queens Matrix is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix using Python. Create a backtracking scenario and use HPC architecture (Preferably BBB) for computation of next placement of a queen.

Learning Objective

To understand:

· Basics of the Backtracking Stratergy for problem solving.

· HPC(High Performance Computing) Architecture.

· Working and implementation of N-Queens problem

Learning Outcome

We will be able to

·Understand the internals of the HPC Architecture by designing the 8-Queens problem using backtracking stratergy.

Software And Hardware Requirements

· Latest 64-BIT Version of Linux Operating System.

· Python Interpreter

Mathematical Model

Let y=f(x) be the solution to the above problem.
Let S be a system such that S={s,e,X,Y,DD,NDD,se,fe,f1,P,sharedMem | ø}
s: Start state of system i.e Y= ø
e:End state of system when i.e. when Y=X
X: Set of inputs X={x_1} where $x_1 \in$ N where N is a even Number ≥ 4
Y: Set of Outputs i.e $\{Y_1..Y_n\}$ i.e i.e The N-Queens Matrix where Y_i=$\{y_1..y_n\}$
f1: Function to Accept Input of N from user
f2: Function to Place Queen
f3: Function to display result of multiplication
f3: X→Y se: Success state is when we get correct placement of queens without conflicts
fe: Failure State is when we do not get correct placement of queens i.e with conflicts

Deterministic Data

Value of N
Non Deterministic Data
Placement of Queens

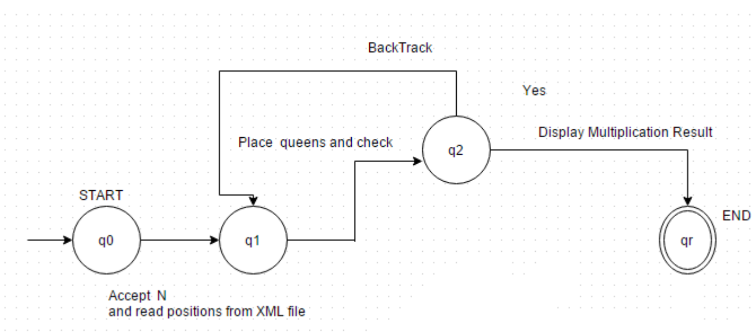
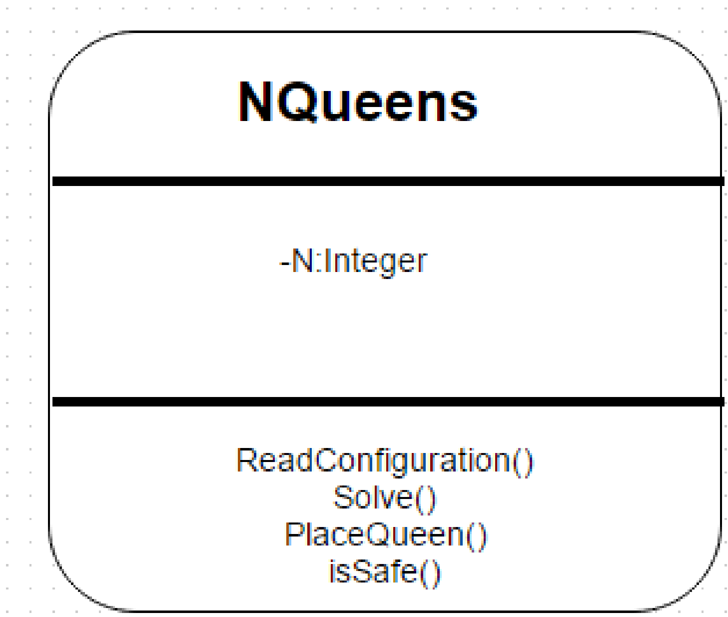


Figure 1: State Disagram

UML-CLASS DIAGRAM



Theory

N-Queens Problem

The eight queens problem is the problem of placing eight queens on an 88 chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). More generally, the n queens problem places n queens on an nn chessboard.

PsuedoCode

SolveQueens (Integer boardSize, Queequeen[boardSize]);// i $\leftarrow$ 0
//Begin by placing the queen number 0
while i $\leq$ boardSize
queen[i].row ¡- queen[i].row + 1

//Place queen[i] to next row /* If queen[i] exceeds the row count, reset the queen and re-place queen[i-1]

if(queen[i].row $\geq$ boardSize)
queen[i] $\leftarrow$ -1;
i $\leftarrow$ i - 1;
else
//While the queen[i] is under attack move it down the row
while(isUnderAttack(queen[i])
queen[i].row ¡- queen[i] + 1;
//if queen[i] exceeds the row count, reset it, re-place queen[i-1]
if(queen[i].row ¿= boardSize)
queen[i].row $\leftarrow$ -1
i $\leftarrow$ i - 1;
else
i++;
end while

Backtracking Algorithmic Stratergy

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons each partial candidate c ("backtracks") as soon as it determines that c cannot possibly be completed to a valid solution.
The classic textbook example of the use of backtracking is the eight queens puzzle, that asks for all arrangements of eight chess queens on a standard chessboard so that no queen attacks any other. In the common backtracking approach, the partial candidates are arrangements of k queens in the first k rows of the board, all in different rows and columns. Any partial solution that contains two mutually attacking queens can be abandoned.
Backtracking can be applied only for problems which admit the concept of a "partial candidate solution" and a relatively quick test of whether it can possibly be completed to a valid solution. It is useless, for example, for locating a given value in an unordered table. When it is applicable, however, backtracking is often much faster than brute force enumeration of all complete candidates, since it can eliminate a large number of candidates with a

single test.
Backtracking is an important tool for solving constraint satisfaction problems, such as crosswords, verbal arithmetic, Sudoku, and many other puzzles. It is often the most convenient (if not the most efficient technique for parsing, for the knapsack problem and other combinatorial optimization problems.

PsuedoCode

procedure bt(c)
if reject(P,c) then return
if accept(P,c) then output(P,c)
s ? first(P,c)
while s $\neq$? do
bt(s)
s ? next(P,s)

Program Code and Output

queensserver.py

```
import socket, pickle

import sys
import numpy as num_p
from xml.dom import minidom
xmlfilename = minidom.parse('data.xml')
itemlist = xmlfilename.getElementsByTagName('Item')



s = socket.socket()         # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12348                # Reserve a port for your
    service.
s.bind((host, port))        # Bind to the port
```

```
s.listen(3)                    # Now wait for client
   connection.



while True:
        c, addr = s.accept()      # Establish connection
            with client.
        print 'Got connection from', addr

        c.send(pickle.dumps(itemlist))

        op = c.recv(10000)
        print "Soln is "
        print op

        c.close()
'''
[bbuhariwala@localhost B1]$ python queens_server.py
Got connection from ('127.0.0.1', 34474)
Soln is
Q**********Q*********Q*****Q****Q**********Q**Q
   *********Q****
^CTraceback (most recent call last):
  File "queens_server.py", line 22, in <module>
    c, addr = s.accept()      # Establish connection
       with client.
  File "/usr/lib64/python2.7/socket.py", line 202, in
     accept
    sock, addr = self._sock.accept()
KeyboardInterrupt
[bbuhariwala@localhost B1]$


'''
```

queensclient.py

```
import socket, pickle, sys

import numpy as num_p

sys.setrecursionlimit(1000000000)

placed_first=0
class nQueens:

    def __init__(self, dimension_of_board):
        self.dimension = dimension_of_board
        self.columns = [] * self.dimension
        self.num_of_places = 0
        self.num_of_backtracks = 0

    def placeFirst(self):
        self.columns.append(placed_first)

    def place(self,row_start=0):
        if len(self.columns) == self.dimension:
            print('Solution found! The board dimension
               was: ' + str(self.dimension))
            print(str(self.num_of_places) + ' total
               places were made.')
            print(str(self.num_of_backtracks) + ' total
                backtracks were executed.')
            print(self.columns)
            board = [[0 for x in range(8)] for x in
               range(8)]
            board[0][self.columns[0]]=1;
            board[1][self.columns[1]]=1;
            board[2][self.columns[2]]=1;
            board[3][self.columns[3]]=1;
            board[4][self.columns[4]]=1;
            board[4][self.columns[5]]=1;
            board[6][self.columns[6]]=1;
```

```
            board[7][self.columns[7]]=1;
            print board
            for i in range(8):
                for j in range(8):
                    print str(board[i][j])+" ",
                print "\n"
            return board
        else:
            for row in range(row_start, self.dimension)
                :

                if self.isSafe(len(self.columns), row)
                    is True:
                     self.columns.append(row)
                     self.num_of_places += 1
                    return self.place()
            else:
                row_last = self.columns.pop()
                self.num_of_backtracks += 1
                return self.place(row_start=row_last +
                    1)

    def isSafe(self, col, row):
        for row_threat in self.columns:
            col_threat = self.columns.index(row_threat)
            if row == row_threat or col == self.columns
                .index(row_threat):
                return False
            elif row_threat + col_threat == row + col
                or row_threat - col_threat == row - col:
                return False
        return True

n = 8

HOST = socket.gethostname()
HOST ='localhost'
PORT = 12348
```

```
socket = socket.socket(socket.AF_INET, socket.
   SOCK_STREAM)
socket.connect((HOST, PORT))
data = socket.recv(10000)

itemlist = pickle.loads(data)
print itemlist
dataList = []
for s in itemlist:
    dataList.append(s.getAttribute("index").encode("utf
       -8"))
print dataList


for s in dataList:

    if(s!=' '):
        placed_first=int(s)
        break

print "First queen placed in first row at: ",
   placed_first
queens = nQueens(8)
queens.placeFirst()
queens.place(0)

#convert board to numpy array for pretty printing
board = num_p.array([['*'] * n] * n)
for queen in queens.columns:
    board[queens.columns.index(queen), queen] = 'Q'
socket.send(board)

'''
[root@localhost B1]# python queens_client.py
[<DOM Element: Item at 0x1e12200>, <DOM Element: Item
   at 0x1e55710>, <DOM Element: Item at 0x1e557a0>, <
   DOM Element: Item at 0x1e55830>, <DOM Element: Item
```

```
 at 0x1e558c0>, <DOM Element: Item at 0x1e55950>, <
 DOM Element: Item at 0x1e559e0>, <DOM Element: Item
 at 0x1e55a70>]
['0', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
First queen placed in first row at:  0
Solution found! The board dimension was: 8
112 total places were made.
105 total backtracks were executed.

[0, 4, 7, 5, 2, 6, 1, 3]

[[1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 1, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0]]

[root@localhost B1]# ^C
[root@localhost B1]#
'''
```

Result

Thus,we have designed and implemented Nqueens problem using Backtracking Stratergy in python language and tested the same.