LP1 Assignment AIR A4

N-Queens using backtracking

Date - 19th October, 2020.

Assignment Number - AIR A4

Title

N-Queens using backtracking

Problem Definition

Implement backtracking for N-Queens

Learning Objectives

- To learn and implement backtracking

Learning Outcomes

I will be able to learn and implement backtracking for N-Queens

Software Packages and Hardware Apparatus Used

- Operating System : 64-bit Ubuntu 18.04
- Programming Language : Python 3
- Jupyter Notebook Environment : Google Colaboratory

Programmers' Perspective

S = {s; e; X; Y; Fme; Ff; DD; NDD}

S = {s; e; X; Y; Fme; Ff; DD; NDD}

s = start state

- s = {n}
    - Where n is the number of queens

e = end state

- e = {2D Representation of the nxn Chess Board with all the queens placed}

X = {X1}

- X1 = s

Y = {Y1}

- Y1 = e

Fme = {f0}

- f0 = function to perform backtracking

Ff = {f1,f2,f3,f4, f5, f6} where

- f1 = function for Agent : Perception
- f2 = function for Agent : Cognition
- f3 = function for Agent : Action
- f4 = function for Agent : Goal
- f5 = function to place a queen
- f6 = function to unplace a queen

DD = integer array of size n

NDD = No non deterministic data

Concepts related Theory

N Queens

- The N queens puzzle is the problem of placing N chess queens on an 8×8 chessboard so that no two queens threaten each other, where N is a natural number.
- Thus, a solution requires that no two queens share the same row, column, or diagonal.

Backtracking

- Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems

- It incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

Mathematical Model

for a given n, N-Queens has been defined as a Constraint Satisfaction Problem

<X,D,C>

N' is a set of Whole Numbers less than n

$N' = \{x \mid x \in W \text{ and } x<n \}$

Set of variable representing n Queens

$X = \{Q_i\} \ \forall \ i \in N'$

Set of Domain for each Queen

$D = \{D_i \mid D_i \in N'\} \ \forall \ i \in N'$

Set of Constraints Two types of constraints for any two queens Each queen is placed in a different column at the time of input itself, therefore we don't need to define a constraint for it explicitly.

$C = \{C^1_{ij}, C^2_{ij}\} \ \forall \ i,j \in N'$

Constraint C^1_{ij} :

For two queens Q_i and Q_j, no two queens can be on the same row.

- $C^1_{ij} = <t^1_{ij}, R^1_{ij}>$
    - $t^1_{ij} = \{Q_i,Q_j\}$
    - $R^1_{ij} = \{x \neq y \mid x=Q_i \text{ and } y=Q_j\}$

Constraint C^2_{ij} :

For two queens Q_i and Q_j, no two queens can be on the same diagonal.

- $C^2_{ij} = <t^2_{ij}, R^2_{ij}>$
    - $t^2_{ij} = \{i,j,Q_i,Q_j\}$
    - $R^2_{ij} = \{ |x-y| \neq |x'-y'| \mid x=i, x'=Q_i, x'=j, x'=Q_j \}$

Initial State

Given n,
Where n is the number of queens to be placed

Perception

Given n number of queens

- Consider an integer array of size n.
- Each element represents a queen
- Index of the element represents column of the queen
- Value of the element represents row of the queen
- If the queen is unplaced, value of the element is -1

Cognition

N Queens is being solved using backtracking

With each step we need to check for three constraints

1. If two Queens are in the same row
2. If two Queens are in the same normal diagonal
3. If two Queens are in the same backward diagonal

Since we are placing/unplacing a queen one column at a time, we need not check for it

To Lookup these constraints in real time, we use three boolean arrays

The size of the boolean array to look up row is n

The size of the other two is (2*n-1)

The index of the lookup of normal diagonal is (row+col)

The index of the lookup of backward diagonal is (row-col+n-1)

The value 'n-1' is to lookup of backward diagonal to facilitate zero-based indexing

Action

Algorithm for solving N queens using backtracking

Algorithm: N-Queen(j)

- n is the number of queens
- queens[] is an integer array which stores row number of a queen
- queens' indexes represent the column number of a queen
- j is the column in which we will be placing the queen, starting from 0
- rowLookUp is a boolean array which is True for rows in which a new queen cannot be placed

- diagonalNormalLookUp is a boolean array which is True for normal diagonal in which a new queen cannot be placed
- diagonalBackwardLookUp is a boolean array which is True for backward diagonals in which a new queen cannot be placed
- Possible(i,j) returns true if rowLookUp(i)=False and diagonalNormalLookUp(i+j)=False and diagonalBackwardLookUp(i-j+n-1)=False

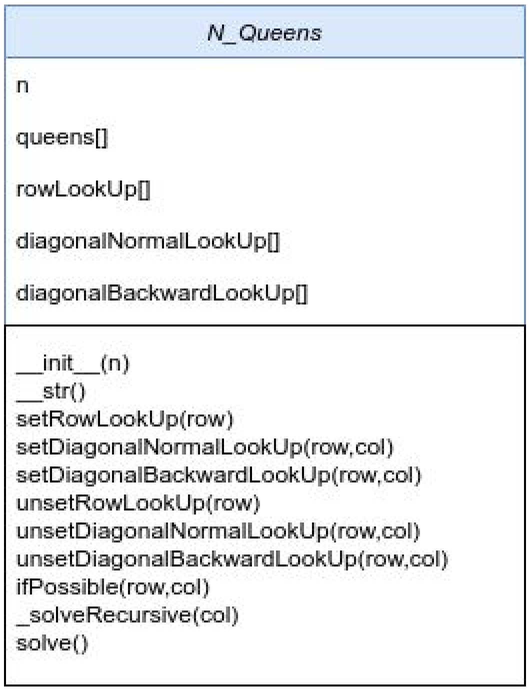
Steps:

```
1. if j=n then
    1. return true
2. for i <- 1 to n do
    1. if Possible(i,j) then do
        1. queens[j] <- i
        2. rowLookUp[i] <- True
        3. diagonalNormalLookUp[i+j] <- True
        4. diagonalBackwardLookUp[i-j+n-1] <- True
        5. if N-Queen(j+1) then
            1. return true
        6. queens[j] <- -1
        7. rowLookUp[i] <- False
        8. diagonalNormalLookUp[i+j] <- False
        9. diagonalBackwardLookUp[i-j+n-1] <- False
3. return false
```

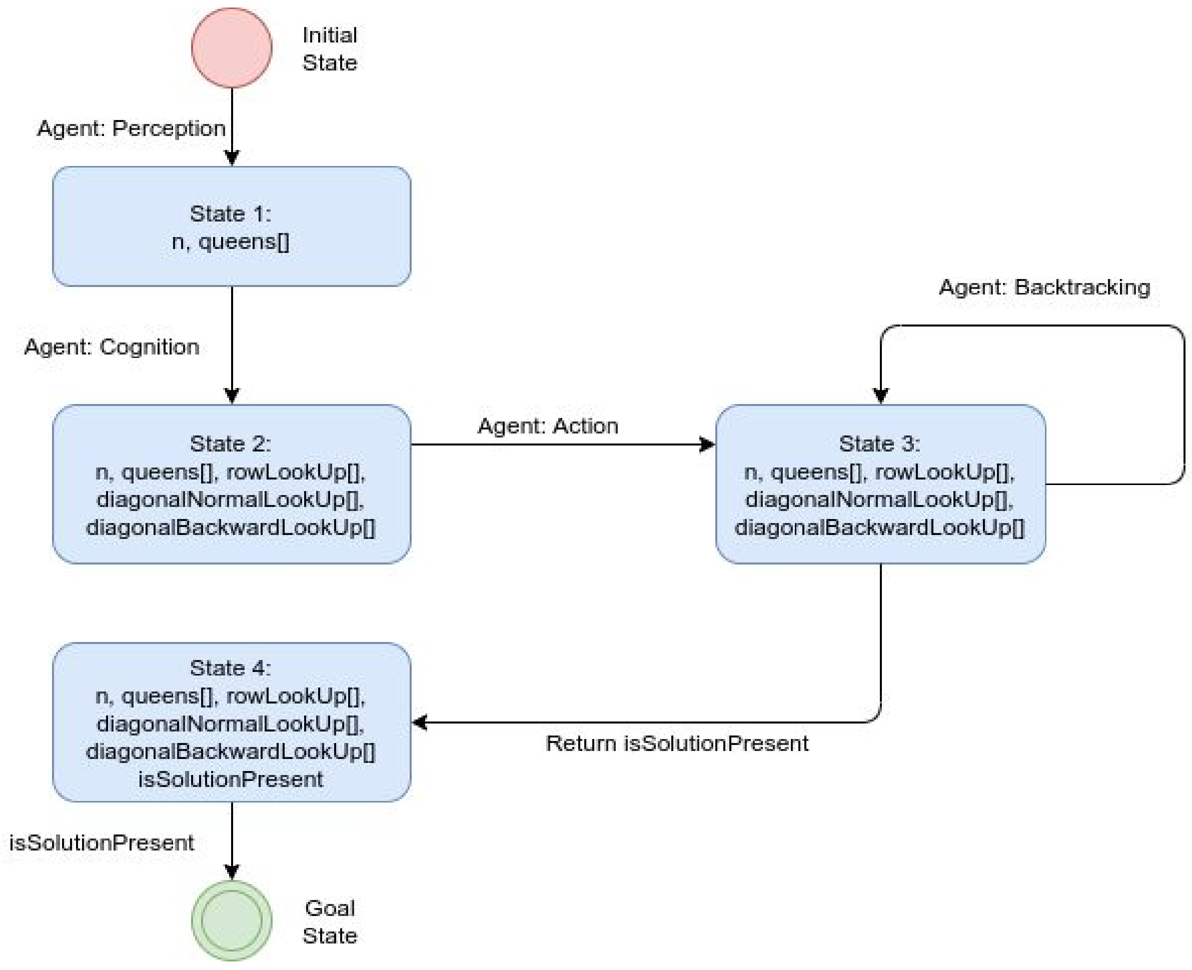
Goal State

- Boolean value isSolutionPresent
- 2D Representation of the nxn Chessboard

Class Diagram



State Diagram


Initial State
Agent: Perception
State 1: n, queens[]
Agent: Cognition
Agent: Backtracking
State 2: n, queens[], rowLookUp[], diagonalNormalLookUp[], diagonalBackwardLookUp[]
Agent: Action
State 3: n, queens[], rowLookUp[], diagonalNormalLookUp[], diagonalBackwardLookUp[]
State 4: n, queens[], rowLookUp[], diagonalNormalLookUp[], diagonalBackwardLookUp[] isSolutionPresent
Return isSolutionPresent
isSolutionPresent
Goal State

Source Code (Modified using PCAG)

```
class N_Queens_PCAG:


 '''CALLING THE FOUR STATES'''
 #Parameterized Constructor
 def __init__(self, n):
   self.perception(n)
   self.cognition(n)
   isSolutionPresent = self.action()
   self.goal(isSolutionPresent)
```

```python
'''PERCEPTION'''
def perception(self, n):
  self.n = n
  self.queens = [-1] * n
  print('\nBefore Solving : ',self)
  return n


'''COGNITION'''
def cognition(self, n):
  self.rowLookUp = [False] * n
  self.diagonalNormalLookUp = [False] * (n+n-1)
  self.diagonalBackwardLookUp = [False] * (n+n-1)


'''ACTION'''
def action(self):
  return self.backtrack(0)


'''BACKTRACKING FUNCTION'''
def backtrack(self, col):

  #Return True if N Queens have been placed
  if col == self.n:
      return True

  #For each cell in the current column
  for row in range(self.n):

    #If It is possible to place a Queen
    if self.ifPossible(row, col):

      #Place the Queen and Set the lookups
      self.placeQueen(row, col)
```

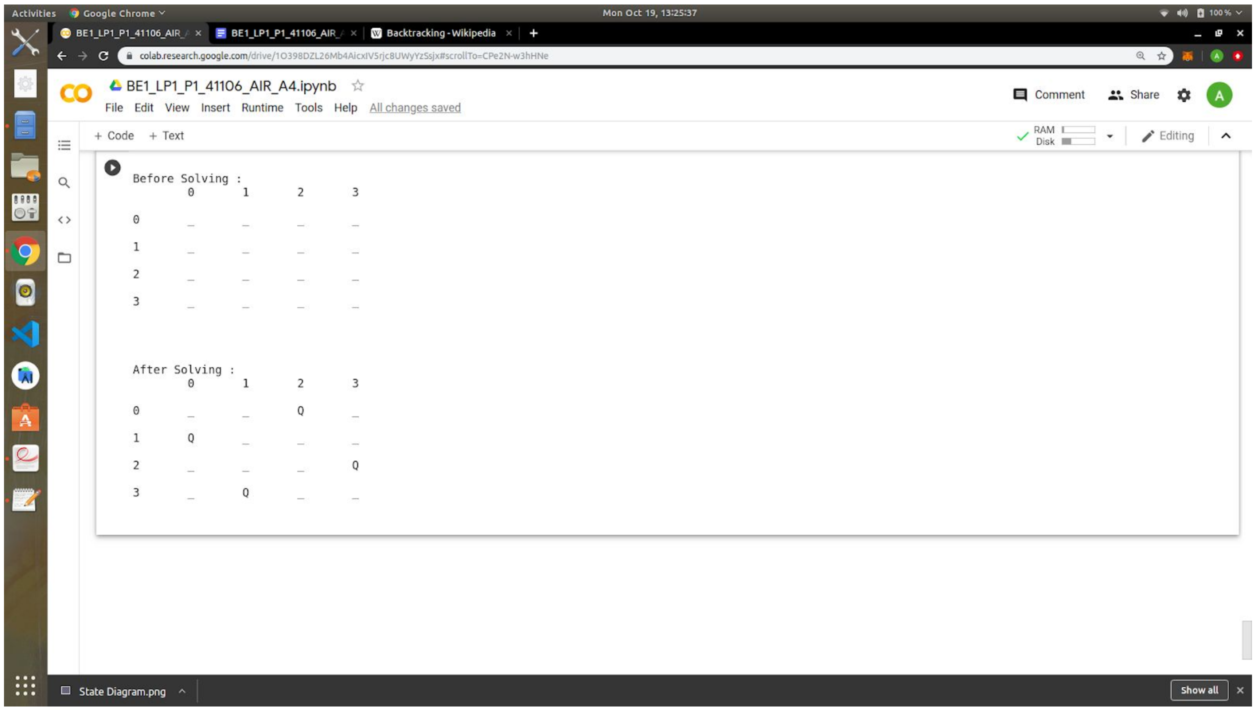
```
        #Try to place the next queen in the next column
        if self.backtrack(col + 1) == True:
          return True

        #Unplace the Queen and unset the lookups
        self.unplaceQueen(row, col)

    #Return False if No queens were placed in the Column
    return False


  '''GOAL'''
  def goal(self, isSolutionPresent):
    if not isSolutionPresent:
      print('Solution is not present')
    print('\n\nAfter Solving : ',self)


  '''STRING REPRESENTATION'''
  #Print Object as a board
  def __str__(self):
    res = '\n\t'
    for num in range(self.n):
      res += (str(num) + "\t")
    res += '\n\n'
    board =  [[ '_' for j in range(self.n)] for i in range(self.n)]
    for col, row in enumerate(self.queens):
      if(row!=-1):
        board[row][col] = 'Q'
    for index, i in enumerate(range(self.n)):
      res += str(index) + "\t"
      for j in range(self.n):
        res += (str(board[i][j]) + "\t")
      res += '\n\n'
    return res
```

```
'''HELPER FUNCTIONS'''
#Set and Unset Functions for the Lookups
def setDiagonalNormalLookUp(self,row,col):
      self.diagonalNormalLookUp[row+col] = True

def unsetDiagonalNormalLookUp(self,row,col):
      self.diagonalNormalLookUp[row+col] = False

def setDiagonalBackwardLookUp(self,row,col):
      self.diagonalBackwardLookUp[row-col+self.n-1] = True

def unsetDiagonalBackwardLookUp(self,row,col):
      self.diagonalBackwardLookUp[row-col+self.n-1] = False

def setRowLookUp(self,row):
      self.rowLookUp[row] = True

def unsetRowLookUp(self,row):
      self.rowLookUp[row] = False

def ifPossible(self, row, col):
  return ( self.rowLookUp[row] == False and
          self.diagonalNormalLookUp[row+col] == False and
          self.diagonalBackwardLookUp[row-col+self.n-1] == False )

def placeQueen(self, row, col):
  self.queens[col] = row
  self.setRowLookUp(row)
  self.setDiagonalNormalLookUp(row,col)
  self.setDiagonalBackwardLookUp(row,col)

def unplaceQueen(self, row, col):
  self.queens[col] = -1
  self.unsetRowLookUp(row)
  self.unsetDiagonalNormalLookUp(row,col)
  self.unsetDiagonalBackwardLookUp(row,col)
```

```
def execute(n):
 puzzle = N_Queens_PCAG(n)

execute(4)
```

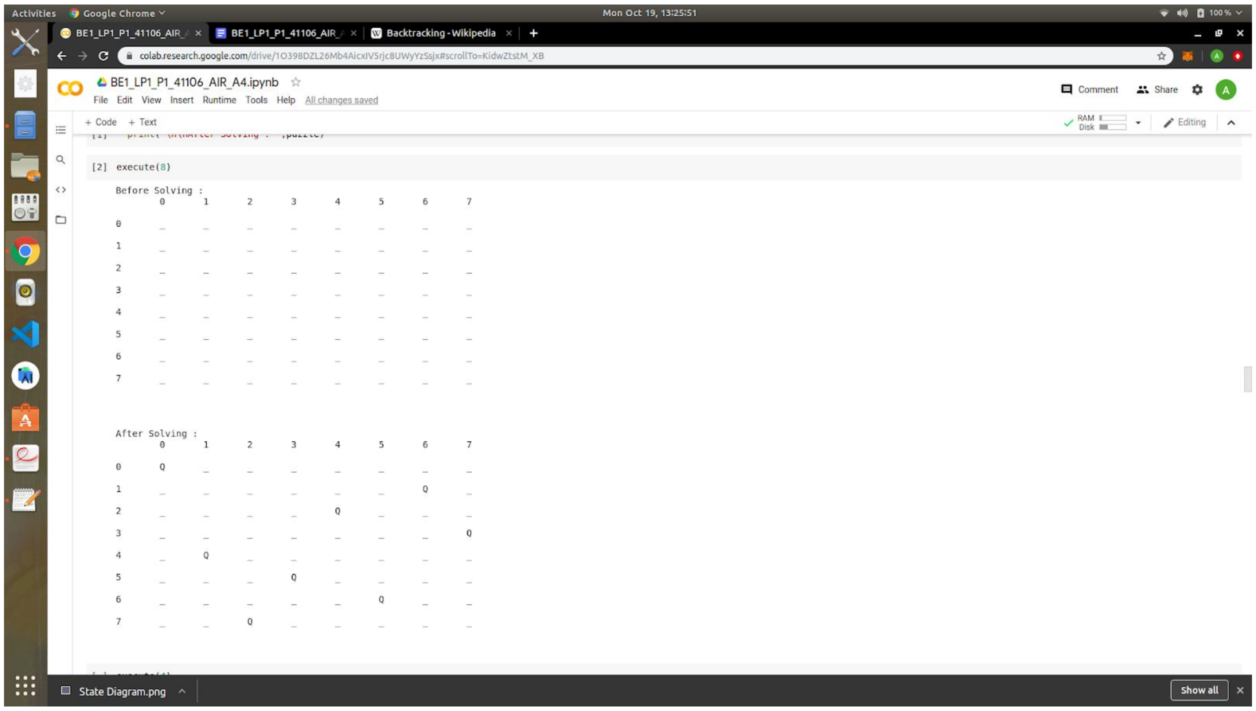
Output Screenshots

Case 1 : n=4


BE1_LP1_P1_41106_AIR_A4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Editing

```
Before Solving :
        0       1       2       3

0       _       _       _       _

1       _       _       _       _

2       _       _       _       _

3       _       _       _       _



After Solving :
        0       1       2       3

0       _       _       Q       _

1       Q       _       _       _

2       _       _       _       Q

3       _       Q       _       _
```

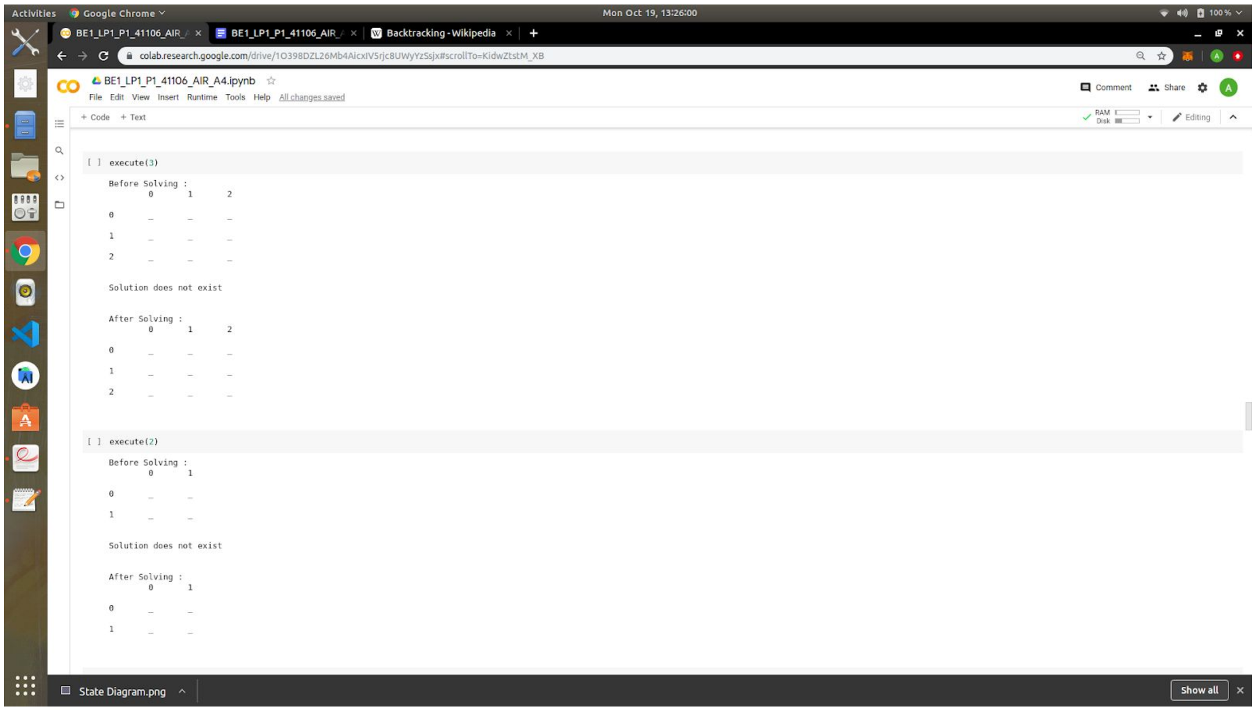
Case 2 : n=8



```
[2] execute(8)

    Before Solving :
          0     1     2     3     4     5     6     7

    0     _     _     _     _     _     _     _     _
    1     _     _     _     _     _     _     _     _
    2     _     _     _     _     _     _     _     _
    3     _     _     _     _     _     _     _     _
    4     _     _     _     _     _     _     _     _
    5     _     _     _     _     _     _     _     _
    6     _     _     _     _     _     _     _     _
    7     _     _     _     _     _     _     _     _


    After Solving :
          0     1     2     3     4     5     6     7

    0     Q     _     _     _     _     _     _     _
    1     _     _     _     _     _     _     Q     _
    2     _     _     _     _     Q     _     _     _
    3     _     _     _     _     _     _     _     Q
    4     _     Q     _     _     _     _     _     _
    5     _     _     _     Q     _     _     _     _
    6     _     _     _     _     _     Q     _     _
    7     _     _     Q     _     _     _     _     _
```

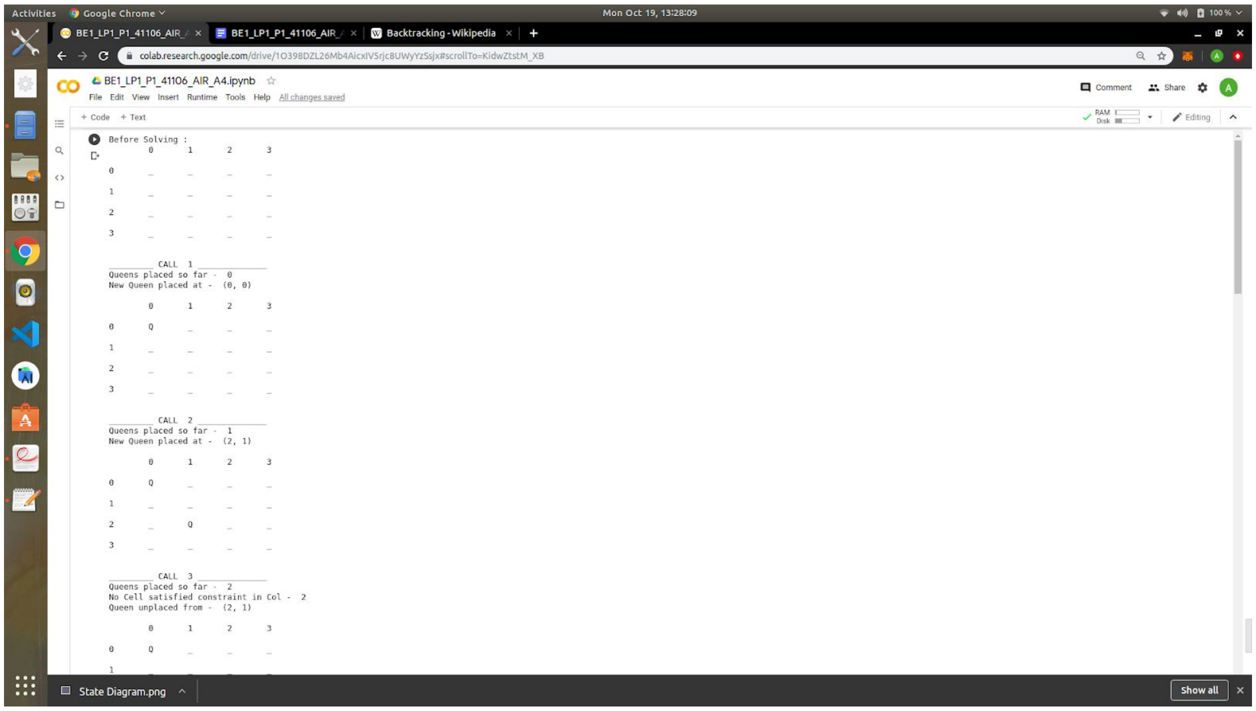
Case 3 : n=2

Case 4 : n=3

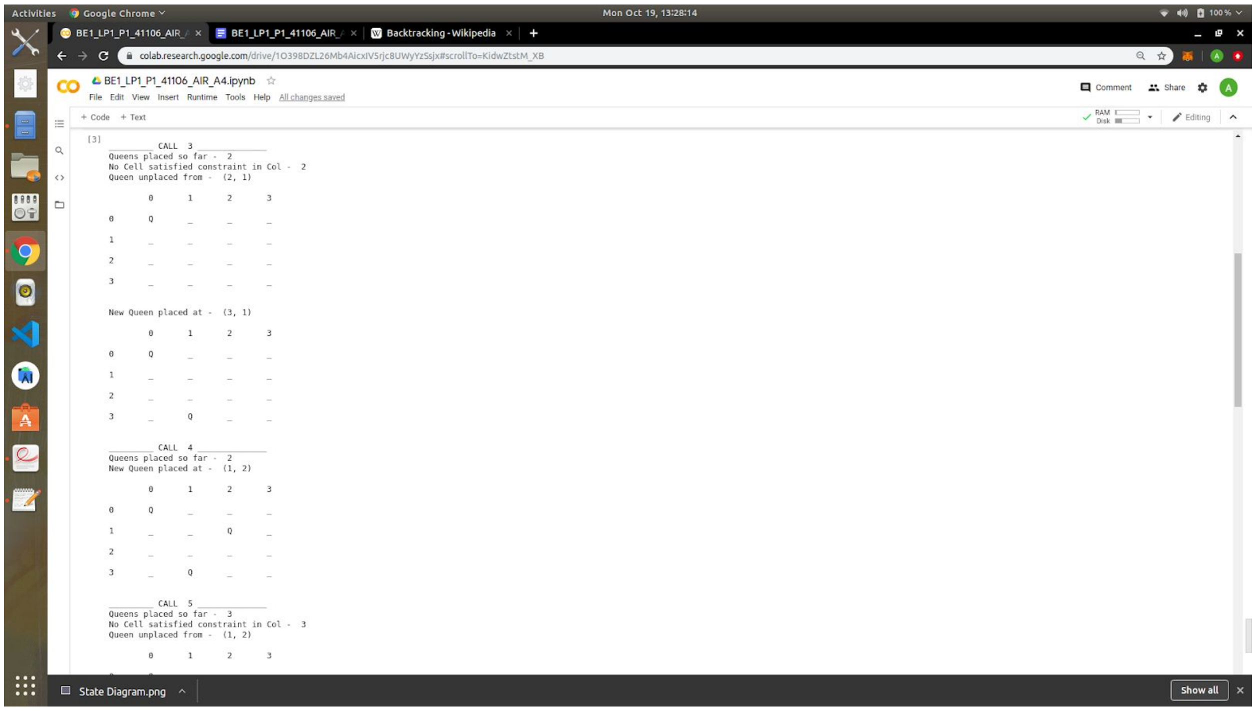


```
[ ] execute(3)

    Before Solving :
          0     1     2

    0     _     _     _
    1     _     _     _
    2     _     _     _

    Solution does not exist


    After Solving :
          0     1     2

    0     _     _     _
    1     _     _     _
    2     _     _     _


[ ] execute(2)

    Before Solving :
          0     1

    0     _     _
    1     _     _

    Solution does not exist


    After Solving :
          0     1

    0     _     _
    1     _     _
```

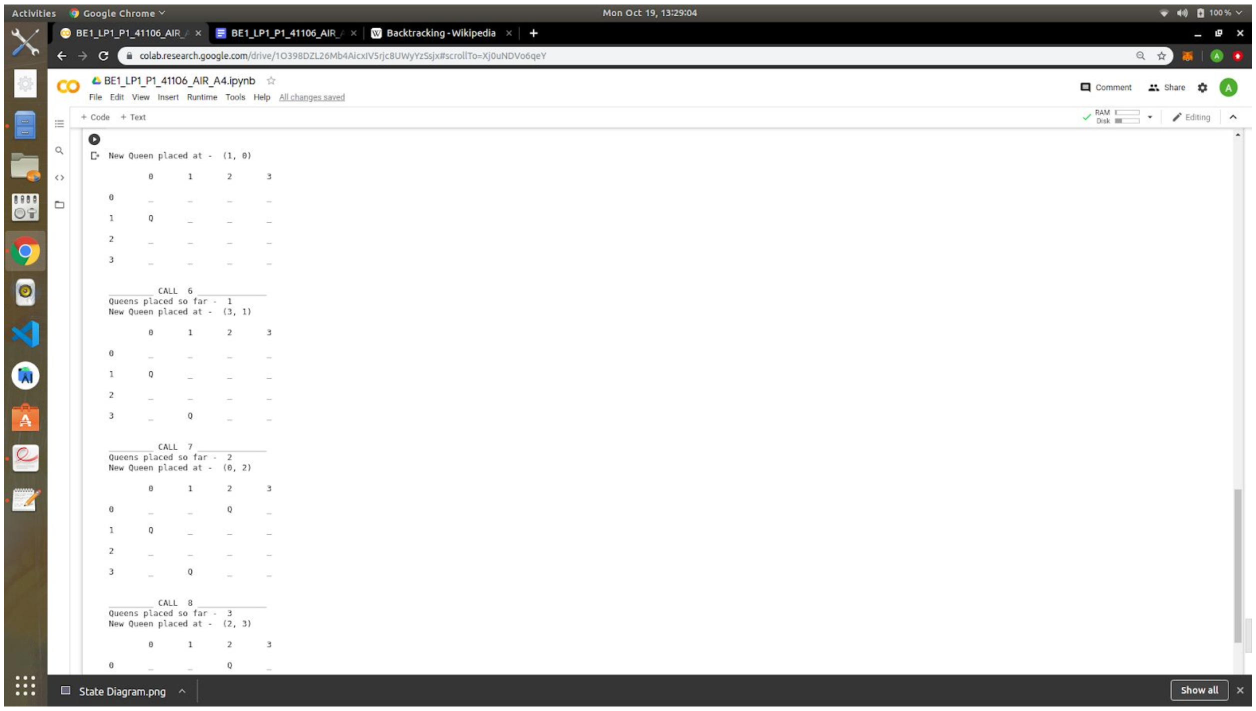
Case 4 : n=4 (Showing all steps)


Activities
Google Chrome
Mon Oct 19, 13:28:09
BE1_LP1_P1_41106_AIR_
BE1_LP1_P1_41106_AIR_
Backtracking - Wikipedia
colab.research.google.com/drive/1O398DZL26Mb4AicxIV5rjc8UWyYzSsjx#scrollTo=KidwZtstM_XB
BE1_LP1_P1_41106_AIR_A4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
State Diagram.png
Show all

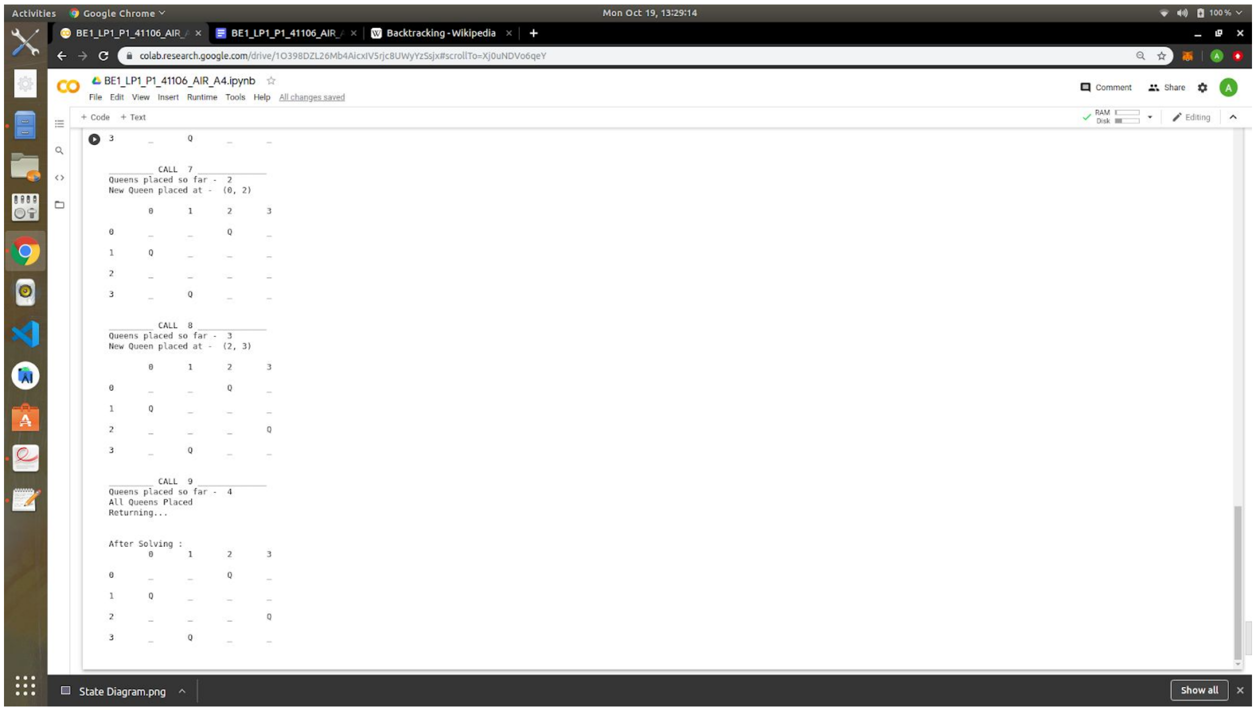
```
Before Solving :
        0       1       2       3

0       _       _       _       _

1       _       _       _       _

2       _       _       _       _

3       _       _       _       _


__________ CALL  1 ______________
Queens placed so far -  0
New Queen placed at -  (0, 0)

        0       1       2       3

0       Q       _       _       _

1       _       _       _       _

2       _       _       _       _

3       _       _       _       _


__________ CALL  2 ______________
Queens placed so far -  1
New Queen placed at -  (2, 1)

        0       1       2       3

0       Q       _       _       _

1       _       _       _       _

2       _       Q       _       _

3       _       _       _       _


__________ CALL  3 ______________
Queens placed so far -  2
No Cell satisfied constraint in Col -  2
Queen unplaced from -  (2, 1)

        0       1       2       3

0       Q       _       _       _

1
```


Activities
Google Chrome
Mon Oct 19, 13:28:14
BE1_LP1_P1_41106_AIR_
BE1_LP1_P1_41106_AIR_
Backtracking - Wikipedia
colab.research.google.com/drive/1O398DZL26Mb4AicxIV5rjc8UWyYzSsjx#scrollTo=KidwZtstM_XB
BE1_LP1_P1_41106_AIR_A4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
State Diagram.png
Show all

```
[3]
__________ CALL  3 ______________
Queens placed so far -  2
No Cell satisfied constraint in Col -  2
Queen unplaced from -  (2, 1)

        0       1       2       3

0       Q       _       _       _

1       _       _       _       _

2       _       _       _       _

3       _       _       _       _


New Queen placed at -  (3, 1)

        0       1       2       3

0       Q       _       _       _

1       _       _       _       _

2       _       _       _       _

3       _       Q       _       _


__________ CALL  4 ______________
Queens placed so far -  2
New Queen placed at -  (1, 2)

        0       1       2       3

0       Q       _       _       _

1       _       _       Q       _

2       _       _       _       _

3       _       Q       _       _


__________ CALL  5 ______________
Queens placed so far -  3
No Cell satisfied constraint in Col -  3
Queen unplaced from -  (1, 2)

        0       1       2       3
```



```
New Queen placed at -  (1, 0)

         0       1       2       3

0        _       _       _       _

1        Q       _       _       _

2        _       _       _       _

3        _       _       _       _

__________ CALL  6 ______________
Queens placed so far -  1
New Queen placed at -  (3, 1)

         0       1       2       3

0        _       _       _       _

1        Q       _       _       _

2        _       _       _       _

3        _       Q       _       _

__________ CALL  7 ______________
Queens placed so far -  2
New Queen placed at -  (0, 2)

         0       1       2       3

0        _       _       Q       _

1        Q       _       _       _

2        _       _       _       _

3        _       Q       _       _

__________ CALL  8 ______________
Queens placed so far -  3
New Queen placed at -  (2, 3)

         0       1       2       3

0        _       _       Q       _
```



```
3        _       Q       _       _

__________ CALL  7 ______________
Queens placed so far -  2
New Queen placed at -  (0, 2)

         0       1       2       3

0        _       _       Q       _

1        Q       _       _       _

2        _       _       _       _

3        _       Q       _       _

__________ CALL  8 ______________
Queens placed so far -  3
New Queen placed at -  (2, 3)

         0       1       2       3

0        _       _       Q       _

1        Q       _       _       _

2        _       _       _       Q

3        _       Q       _       _

__________ CALL  9 ______________
Queens placed so far -  4
All Queens Placed
Returning...


After Solving :
         0       1       2       3

0        _       _       Q       _

1        Q       _       _       _

2        _       _       _       Q

3        _       Q       _       _
```

Conclusion

I have successfully designed and implemented backtracking for N_Queens using Agents Perception, Cognition, Action and Goal