Lab Task: 09

***_____786**
_____**

Name: Muhammad Sherjeel Akhtar

Roll No: 20p-0101

Subject: Artificial Intelligence Lab

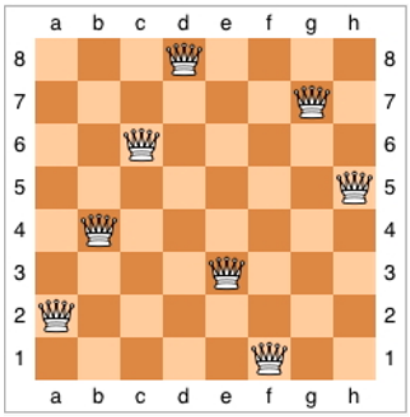
Submitted To Respected
Ma’am: Hurmat Hidayat

Section: BCS-6C

Problem:

8-Queen Problem using Hill Climbing

The objective of the 8-queens problem is to place eight queens on a chessboard in such a way that no two queens can capture each other, which means no two queens can be on the same row, column, or diagonal.



The first observation is that, in any solution, no two queens can occupy the same column, and consequently no column can be empty. At the start we can therefore assign a column to each queen and reduce the problem to the simpler task of finding an appropriate row.

Answer:

Importing The Random Library:

```
import random
```

Visaully:

Importing The Library

```
In [30]: import random
```

Creating An Array For The Representation Of Chessboard

```
board = [[0 for x in range(8)] for y in range(8)]
```

Visually:

Creating An Array For The Representation Of Chessboard

```
In [31]: board = [[0 for x in range(8)] for y in range(8)]
```

Randomly Placing A Queen On The Board Using place_queen() Function:

```
def place_queen():
    row = random.randint(0, 7)
    col = random.randint(0, 7)
    board[row][col] = 1
```

Visually:

Randomly Placing A Queen On The Board Using place_queen() Function

```
In [32]: def place_queen():
             row = random.randint(0, 7)
             col = random.randint(0, 7)
             board[row][col] = 1
```

Calculating The Objective Function Using The attacking_queens() Function Defined Below¶

```
def attacking_queens(board):
    count = 0
    for i in range(8):
        for j in range(8):
            if board[i][j] == 1:
                for k in range(8):
                    if k != i and board[k][j] == 1:
                        count += 1
                    if k != j and board[i][k] == 1:
                        count += 1
                    if i+k < 8 and j+k < 8 and board[i+k][j+k] == 1:
                        count += 1
                    if i+k < 8 and j-k >= 0 and board[i+k][j-k] == 1:
                        count += 1
                    if i-k >= 0 and j+k < 8 and board[i-k][j+k] == 1:
                        count += 1
                    if i-k >= 0 and j-k >= 0 and board[i-k][j-k] == 1:
                        count += 1
    return count // 2
```

Visually:

Calculating The Objective Function Using The attacking_queens() Function Defined Below

In [33]:
```
def attacking_queens(board):
    count = 0
    for i in range(8):
        for j in range(8):
            if board[i][j] == 1:
                for k in range(8):
                    if k != i and board[k][j] == 1:
                        count += 1
                    if k != j and board[i][k] == 1:
                        count += 1
                    if i+k < 8 and j+k < 8 and board[i+k][j+k] == 1:
                        count += 1
                    if i+k < 8 and j-k >= 0 and board[i+k][j-k] == 1:
                        count += 1
                    if i-k >= 0 and j+k < 8 and board[i-k][j+k] == 1:
                        count += 1
                    if i-k >= 0 and j-k >= 0 and board[i-k][j-k] == 1:
                        count += 1
    return count // 2
```

Search For The Solution Using Hill Climbing By The hill_climbing() Function

```
def hill_climbing(board):
    for i in range(8):
        place_queen()
    current_cost = attacking_queens(board)
    while True:
        candidates = []
        for i in range(8):
            for j in range(8):
                if board[i][j] == 1:
                    for k in range(8):
                        if k != i:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[k][j] = 1
                            candidates.append(temp)
                        if k != j:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i][k] = 1
                            candidates.append(temp)
                        if i+k < 8 and j+k < 8:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i+k][j+k] = 1
                            candidates.append(temp)
                        if i+k < 8 and j-k >= 0:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i+k][j-k] = 1
                            candidates.append(temp)
                        if i-k >= 0 and j+k < 8:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i-k][j+k] = 1
                            candidates.append(temp)
                        if i-k >= 0 and j-k >= 0:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i-k][j-k] = 1
                            candidates.append(temp)
        min_cost = current_cost
        for candidate in candidates:
            cost = attacking_queens(candidate)
            if cost < min_cost:
                min_cost = cost
                board = candidate
        if min_cost >= current_cost:
            break
        current_cost = min_cost
    return board
```

Visually:

Search For The Solution Using Hill Climbing By The hill_climbing() Function

In [34]:
```
def hill_climbing(board):
    for i in range(8):
        place_queen()
    current_cost = attacking_queens(board)
    while True:
        candidates = []
        for i in range(8):
            for j in range(8):
                if board[i][j] == 1:
                    for k in range(8):
                        if k != i:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[k][j] = 1
                            candidates.append(temp)
                        if k != j:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i][k] = 1
                            candidates.append(temp)
                        if i+k < 8 and j+k < 8:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i+k][j+k] = 1
                            candidates.append(temp)
                        if i+k < 8 and j-k >= 0:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i+k][j-k] = 1
                            candidates.append(temp)
                        if i-k >= 0 and j+k < 8:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i-k][j+k] = 1
                            candidates.append(temp)
                        if i-k >= 0 and j-k >= 0:
                            temp = [x[:] for x in board]
                            temp[i][j] = 0
                            temp[i-k][j-k] = 1
                            candidates.append(temp)
        min_cost = current_cost
        for candidate in candidates:
            cost = attacking_queens(candidate)
            if cost < min_cost:
                min_cost = cost
                board = candidate
        if min_cost >= current_cost:
            break
        current_cost = min_cost
    return board
```

Calling The hill_climbing() Function With An Argument board

```
solution = hill_climbing(board)
```

Printing The Final Solution¶

```
for row in solution:
    print(row)
```

Visually:

Calling The hill_climbing() Function With An Argument board

In [35]:
```
solution = hill_climbing(board)
```

Printing The Final Solution

In [36]:
```
for row in solution:
    print(row)
```

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

Check Randomness By Calling Again The Last Cell:

For Visualizing The Random Placement Run The Below Cell

In [40]:
```
solution = hill_climbing(board)
for row in solution:
    print(row)
```

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

In [ ]:
```

```

For Visualizing The Random Placement Run The Below Cell

In [41]:
```
solution = hill_climbing(board)
for row in solution:
    print(row)
```

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

FIN.