**Task 4 N-Queen’s Problem Backtracking**

**Code**

**def is\_safe(board, row, col, n):**

**# Check if there is a queen in the same column**

**for i in range(row):**

**if board[i][col] == 1:**

**return False**

**# Check upper-left diagonal**

**for i, j in zip(range(row, -1, -1), range(col, -1, -1)):**

**if board[i][j] == 1:**

**return False**

**# Check upper-right diagonal**

**for i, j in zip(range(row, -1, -1), range(col, n)):**

**if board[i][j] == 1:**

**return False**

**return True**

**def solve\_n\_queens\_backtracking(n):**

**board = [[0] \* n for \_ in range(n)]**

**solutions = []**

**def backtrack(row):**

**nonlocal solutions**

**if row == n:**

**# Found a solution, append a copy to solutions**

**solutions.append([row[:] for row in board])**

**return**

**for col in range(n):**

**if is\_safe(board, row, col, n):**

**board[row][col] = 1**

**backtrack(row + 1)**

**board[row][col] = 0**

**backtrack(0)**

**return solutions**

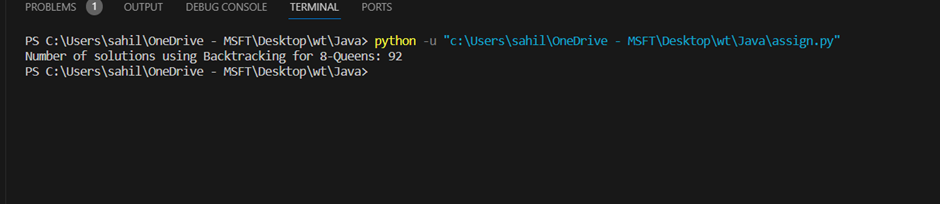
**# Example usage:**

**n = 8**

**solutions\_backtracking = solve\_n\_queens\_backtracking(n)**

**print(f"Number of solutions using Backtracking for {n}-Queens: {len(solutions\_backtracking)}")**

**Output**

****

**N-Queen’s Problem branch and bound:**

**Code**

**def solve\_n\_queens\_branch\_and\_bound(n):**

**board = [[0] \* n for \_ in range(n)]**

**solutions = []**

**def is\_safe(board, row, col, n):**

**# Check if there is a queen in the same column**

**for i in range(row):**

**if board[i][col] == 1:**

**return False**

**# Check upper-left diagonal**

**for i, j in zip(range(row, -1, -1), range(col, -1, -1)):**

**if board[i][j] == 1:**

**return False**

**# Check upper-right diagonal**

**for i, j in zip(range(row, -1, -1), range(col, n)):**

**if board[i][j] == 1:**

**return False**

**return True**

**def bound(board, row, n):**

**for i in range(row, n):**

**for j in range(n):**

**if is\_safe(board, i, j, n):**

**return True**

**return False**

**def backtrack(row):**

**nonlocal solutions**

**if row == n:**

**# Found a solution, append a copy to solutions**

**solutions.append([row[:] for row in board])**

**return**

**for col in range(n):**

**if is\_safe(board, row, col, n):**

**board[row][col] = 1**

**if bound(board, row + 1, n):**

**backtrack(row + 1)**

**board[row][col] = 0**

**backtrack(0)**

**return solutions**

**# Example usage:**

**n = 8**

**solutions\_branch\_and\_bound = solve\_n\_queens\_branch\_and\_bound(n)**

**print(f"Number of solutions using Branch and Bound for {n}-Queens: {len(solutions\_branch\_and\_bound)}")**

**Output**



**Task 5 Develop an elementary chatbot for any suitable customer interaction application.**

**Code**

class CustomerServiceChatbot:

def \_\_init\_\_(self):

self.responses = {

"hello": "Hi there! How can I help you today?",

"how are you": "I'm just a computer program, but thanks for asking!",

"bye": "Goodbye! If you have more questions, feel free to ask.",

"help": "You can ask about our products, services, or any other related queries.",

}

def get\_response(self, user\_input):

user\_input = user\_input.lower()

return self.responses.get(user\_input, "I'm sorry, I don't understand. Type 'help' for assistance.")

def start\_chat(self):

print("Customer Service Chatbot: Hi! How can I assist you today? Type 'bye' to end the conversation.")

while True:

user\_input = input("You: ")

if user\_input.lower() == "bye":

print("Customer Service Chatbot: Goodbye! Have a great day.")

break

response = self.get\_response(user\_input)

print("Customer Service Chatbot:", response)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

chatbot = CustomerServiceChatbot()

chatbot.start\_chat()

**Output**

