- Backtracking Algoritması İle N Vezir Probleminin Çözümü

1. `is_safe` Fonksiyonu Açıklaması

   `is_safe` fonksiyonu, bir vezir yerleştirmenin tahtanın kurallarına uygun olup olmadığını belirler. Fonksiyon, o satırdaki sütunun ve çaprazların tehdit altında olmadığından emin olduktan sonra, pozisyonun güvenli olduğunu bildirir. Bu kontrol, algoritmanın her bir adımında yapılır ve çözümün doğruluğunu garanti eder.

```
# Aynı sütundaki vezirleri kontrol et
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Sol üst çaprazı kontrol et
    for i, j in zip(range(row, -1, -1),
range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Sağ üst çaprazı kontrol et
    for i, j in zip(range(row, -1, -1),
range(col, n)):
        if board[i][j] == 1:
            return False

    return True
```

2. `solve_n_queens` Fonksiyonu Açıklaması

   Bu fonksiyon, "n-vezir" problemini çözmek için kullanılan backtracking algoritmasını uygular.

- board: Şu anda çözülmekte olan satranç tahtasını temsil eden bir 2D matris. Hücre değeri 1 olan pozisyonlar, o hücrede bir vezir bulunduğunu gösterir.
- row: Şu anda yerleştirilmekte olan vezirin satır numarası.
- n: Tahta boyutu ve aynı zamanda çözülmesi gereken vezir sayısı (örneğin, n=8 için 8x8 tahta).
- solutions: Bulunan tüm çözümlerin saklandığı bir liste. Her çözüm, bir matris kopyası olarak eklenir.
- steps: Algoritma sırasında yapılan tüm adımların saklandığı bir liste. Bu, algoritmanın ilerleyişini ve geri izleme (backtracking) işlemini görselleştirmek için kullanılır.

```
def solve_n_queens(board, row, n, solutions,
steps): # Çözüm bulundu, çözümleri kaydet
if row == n:
solutions.append(np.copy(board))

return True

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1

    # Adımları kaydet
            steps.append(np.copy(board))
                 solve_n_queens(board, row + 1, n,
    solutions, steps)
            board[row][col] = 0

    # Backtracking adımını kaydet
            steps.append(np.copy(board))


    return False
```

3. `create_chessboard_image` Fonksiyonu Açıklaması

Bu fonksiyon, verilen n x n boyutundaki bir satranç tahtasının renkli bir görselini oluşturmak için kullanılır. Satranç tahtasındaki kareler, geleneksel olarak iki farklı renkle (örneğin, açık ve koyu kahverengi) boyanır. Fonksiyon, bir renk matrisini (colors) oluşturarak bunu temsil eder.

```
# Tahta renklerini ayarla
def
create_chessboard_image(n):
colors = np.zeros((n, n, 3))
for i in range(n):
    for j in range(n):
    # Açık kahverengi
        if (i + j) % 2 == 0:
        colors[i, j] = [0.8, 0.6,
        0.4] # Koyu kahverengi
        else:
        colors[i, j] = [0.4, 0.2, 0]

return colors
```

4. `show_solution_image` Fonksiyonu Açıklaması

Bu fonksiyon, bir çözümü satranç tahtası üzerinde görsel olarak göstermek için kullanılır.

- board: Çözümü temsil eden 2D matris. 1 olan hücreler vezirlerin yerleştirildiği pozisyonları gösterir.
- n: Tahta boyutu (örneğin, 8x8 için n=8).
- solution_number: Çözüm numarası (çözüm başlığı için).

```
def show_solution_image(board, n, solution_number):
fig, axes = plt.subplots(nrows=n, ncols=n, figsize=(8, 8))
# 2D array'i 1D'ye çeviriyoruz, böylece her hücreye
kolayca erişebiliriz
axes = axes.flatten()
# Hücreler arasındaki boşluğu kaldırmak için
plt.subplots_adjust(left=0, right=1, top=1,
bottom=0, wspace=0, hspace=0) # wspace ve hspace'ı 0
yapıyoruz

for i in range(n):
    for j in range(n):
     # her bir hücreye ait ax'yi alıyoruz
        ax = axes[i*n + j]
        ax.set_xticks([])  # x eksenini kaldırıyoruz
        ax.set_yticks([])  # y eksenini kaldırıyoruz
        ax.set_aspect('equal')  # Eksen oranını
        eşitliyoruz # Tahtanın rengini ayarla
     if (i + j) % 2 == 0:
      ax.set_facecolor([0.8, 0.6, 0.4]) # Açık kahverengi
        else:
            ax.set_facecolor([0.4, 0.2, 0]) # Koyu kahverengi
        # Vezirleri yerleştir
        if board[i][j] == 1:
# Vezir görseli yolu
try:
queen_img = mpimg.imread("queen.png") # Görseli okumaya çalış
ax.imshow(queen_img, extent=(0, 1, 0, 1), aspect='auto')
except FileNotFoundError:
print("Hata: 'queen.png' görseli bulunamadı.")
sys.exit() # Programı sonlandır
plt.suptitle(f"Çözüm {solution_number}", fontsize=16, y=1.05)
plt.show()
```

5. `run_n_queens` Fonksiyonu Açıklaması

Bu fonksiyon, n-vezir problemini çözmek için kullanılan ana işlevdir. Verilen tahta boyutunda (n), tüm çözüm seçeneklerini hesaplar ve her çözümü görselleştirir.

- n: Satranç tahtasının boyutu. Örneğin, n=4 ise 4x4 bir tahta.

```
def run_n_queens(n):
board = np.zeros((n, n), dtype=int)
solutions = []
steps = []
solve_n_queens(board, 0, n, solutions, steps)
print(f"{n}x{n} tahtası için toplam {len(solutions)}
çözüm bulundu.")

for solution_number,
solution in enumerate(solutions, start=1):
    show_solution_image(solution, n,solution_number)
```

6. `is_valid_board_size` Fonksiyonu Açıklaması

Bu fonksiyon, kullanıcıdan alınan girdinin 2 nin kuvveti olup olmadığını ve minimum 4 olması gerektiğini kontrol eder.

```
def
is_valid_board_size(n): #
2'nin kuvveti kontrolü
return n >= 4 and (n & (n - 1)) == 0
```

7. `main` Fonksiyonu Açıklaması

Bu main fonksiyonu kullanıcıdan tahta boyutu girdiği bir döngüyü yönetir. Kullanıcı, 2'nin kuvveti olan (ör. 4, 8, 16) geçerli bir tahta boyutu girene kadar döngü devam eder. Giriş hatalıysa veya çok büyük bir tahta boyutu seçilirse kullanıcıya uyarı mesajı gösterilir. Kullanıcı "q" yazarak programdan çıkabilir.

```
def main():
while True:
try:
user_input = input("Kaça kaçlık bir tahta
istiyorsunuz? (4, 8, 16, ...) veya çıkmak için 'q':
").strip()
    if user_input.lower() == 'q':
    print("Programdan çıkılıyor.")
    break
    n = int(user_input)
    if n >= 16:
    print("Seçilen tahta boyutu çok büyük ve işlem
    uzun sürebilir. Lütfen daha küçük bir değer
    giriniz.")
    continue
    if is_valid_board_size(n):
    run_n_queens(n)
    break
    else:
    print("Hatalı giriş! Lütfen 2'nin kuvveti
    olan ve en az 4 olan bir değer giriniz.")
    except ValueError:
    print("Lütfen geçerli bir tam sayı giriniz!")


if __name__ == "__main__": main()
```

- Kodun Full Hâli

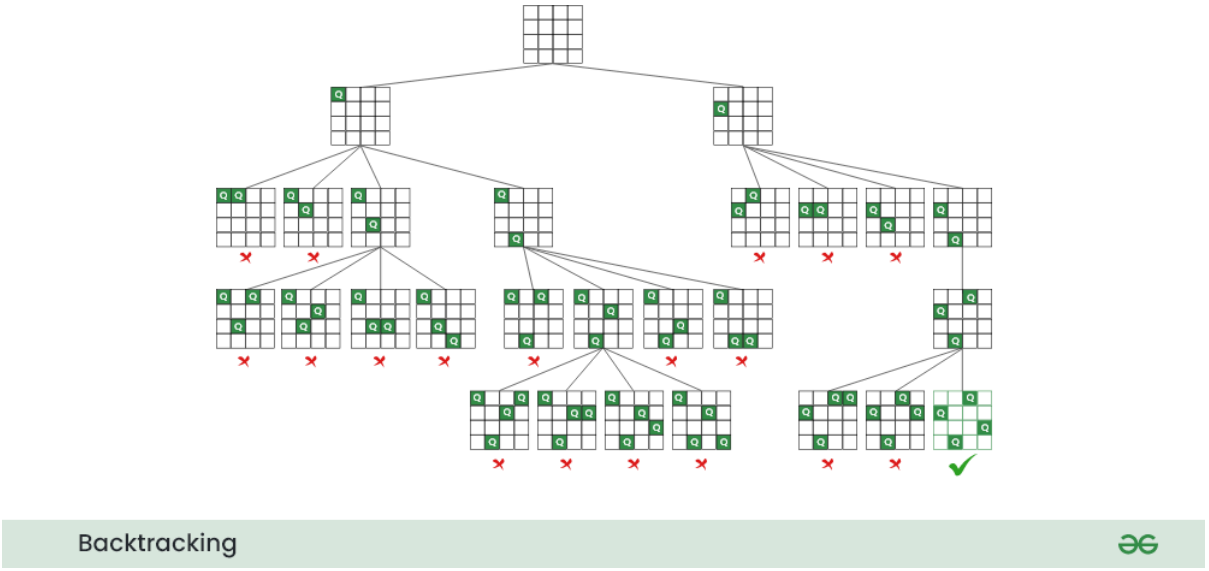
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as
mpimg def is_safe(board, row,
col, n):
# Aynı sütundaki vezirleri kontrol et
for i in range(row):
if board[i][col] == 1:
return False

# Sol üst çaprazı kontrol et
for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
    if board[i][j] == 1:
        return False

# Sağ üst çaprazı kontrol et
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(board, row, n, solutions, steps):
if row == n:
# Çözüm bulundu, çözümleri kaydet
solutions.append(np.copy(board))
return True
for col in range(n):
    if is_safe(board, row, col, n):
        board[row][col] = 1
         # Adımları kaydet
        steps.append(np.copy(board))
        solve_n_queens(board, row + 1, n, solutions, steps)
        board[row][col] = 0
        # Backtracking adımını kaydet
        steps.append(np.copy(board))

return False
```

```
# Tahta renklerini ayarla
def create_chessboard_image(n):
colors = np.zeros((n, n, 3))
for i in range(n):
for j in range(n):
if (i + j) % 2 == 0:
colors[i, j] = [0.8, 0.6, 0.4] # Açık kahverengi
else:
colors[i, j] = [0.4, 0.2, 0] # Koyu kahverengi
return colors

def show_solution_image(board, n, solution_number):
fig, axes = plt.subplots(nrows=n, ncols=n, figsize=(8, 8))
axes = axes.flatten() # 2D array'i 1D'ye çeviriyoruz
plt.subplots_adjust(left=0, right=1, top=1, bottom=0,
wspace=0, hspace=0) # Hücreler arasındaki boşluğu kaldır
for i in range(n):
    for j in range(n):
        ax = axes[i * n + j]
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_aspect('equal')

        # Tahtanın rengini ayarla
        if (i + j) % 2 == 0:
        # Açık kahverengi
            ax.set_facecolor([0.8, 0.6, 0.4])
        else:
        # Koyu kahverengi
            ax.set_facecolor([0.4, 0.2, 0])
        # Vezirleri yerleştir
        if board[i][j] == 1:
# Vezir görseli yolu
try:
queen_img = mpimg.imread("queen.png") # Görseli okumaya çalış
ax.imshow(queen_img, extent=(0, 1, 0, 1), aspect='auto')
except FileNotFoundError:
print("Hata: 'queen.png' görseli bulunamadı.")
sys.exit() # Programı sonlandır
plt.suptitle(f"Çözüm {solution_number}", fontsize=16, y=1.05)
plt.show()
```

```
def run_n_queens(n):
board = np.zeros((n, n), dtype=int)
solutions = []
steps = []
solve_n_queens(board, 0, n, solutions, steps)
print(f"{n}x{n} tahtası için toplam {len(solutions)} çözüm
bulundu.")

for solution_number, solution in enumerate(solutions,
start=1):
    show_solution_image(solution, n, solution_number)

# 2'nin kuvveti kontrolü
def
is_valid_board_size(n):
return n >= 4 and (n & (n - 1)) == 0

def main():
while True:
try:
user_input = input("Kaça kaçlık bir tahta istiyorsunuz?
(4, 8, 16, ...) veya çıkmak için 'q': ").strip()
if user_input.lower() == 'q':
print("Programdan çıkılıyor.")
break
        n = int(user_input)
        if n >= 16:
            print("Seçilen tahta boyutu çok büyük ve işlem
uzun sürebilir. Lütfen daha küçük bir değer giriniz.")
            continue
        if is_valid_board_size(n):
            run_n_queens(n)
            brea
        k else:
            print("Hatalı giriş! Lütfen 2'nin kuvveti olan ve
en az 4 olan bir değer giriniz.")
    except ValueError:
        print("Lütfen geçerli bir tam sayı

giriniz!") if ____name__ == "_main_": main()
```

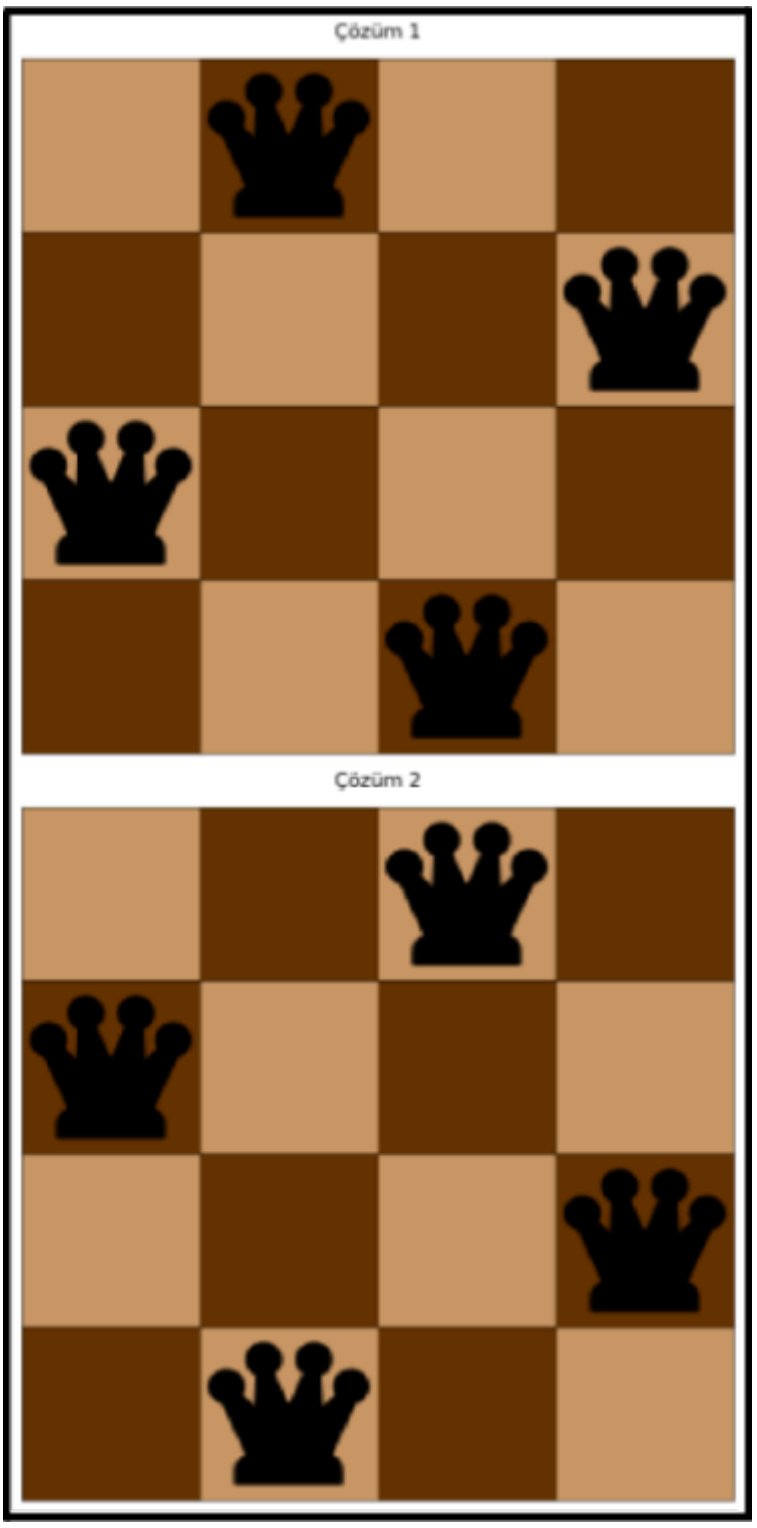
- Geriye Dönme (Backtracking) Adımı

Algoritma, diğer olasılıkları kontrol etmek için geriye doğru hareket eder. Eğer bir çözüm bulunmuşsa, diğer tüm sütun ve satırlarda farklı kombinasyonlarla çözüm arar. Bu, başka çözüm seçeneklerini de gösterebilir.



("Görsel Geeks For Geeks' den alınmıştır.")

- 4 x 4 Bir Tahta İçin Oluşan Çıktı



KAYNAKÇA

https://www.geeksforgeeks.org/backtracking-algorithms/

https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/

https://en.wikipedia.org/wiki/Eight_queens_puzzle

https://www.geeksforgeeks.org/python-programming-language-tutorial/

https://docs.python.org/3/tutorial/index.html

https://www.youtube.com/watch?v=xFv_Hl4B83A

https://www.youtube.com/watch?v=Ph95IHmRp5M