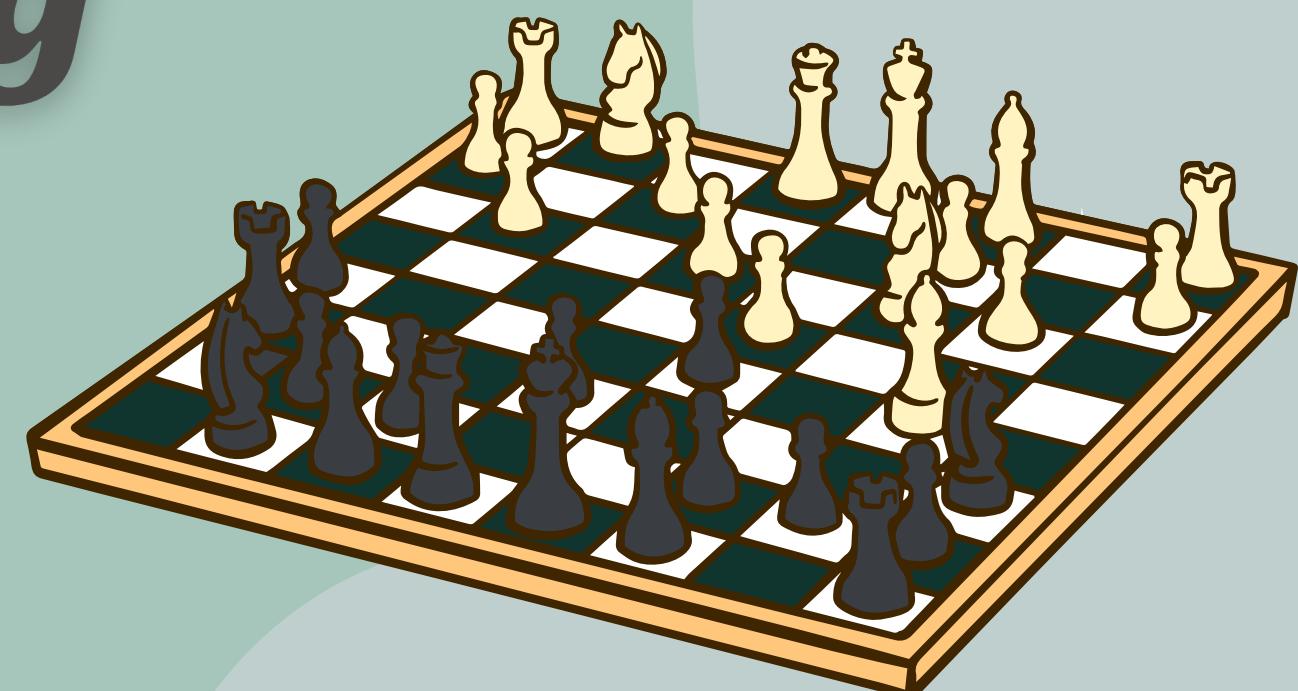


METODE OPTIMASI

Penerapan *Simulated Annealing* pada *N-Queen's* dengan *Backtracking*

KELOMPOK 1

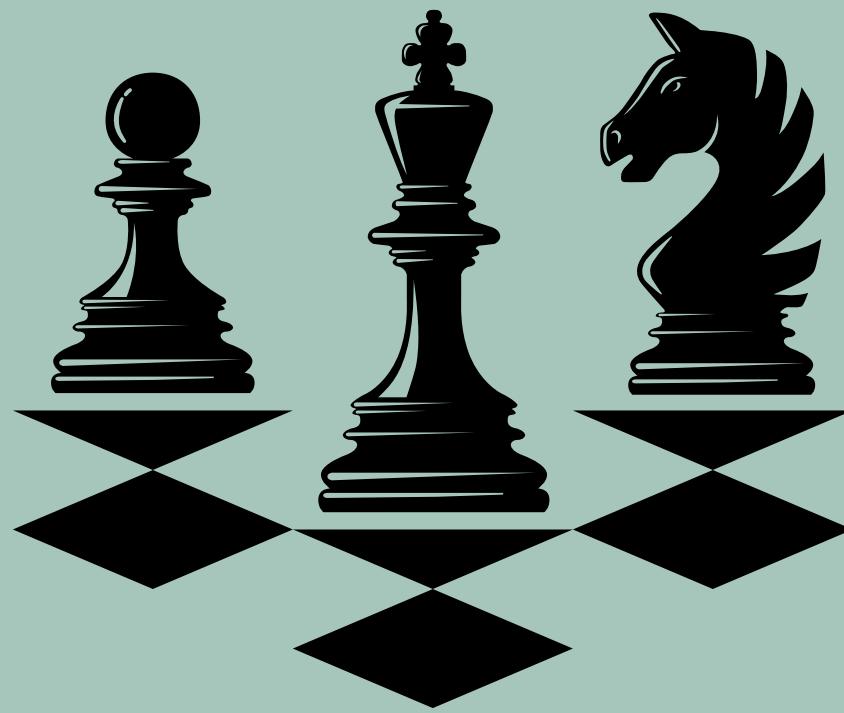


Anggota

Anisa Nur Aidah (2104089)

Salma Septiani Rahman (2103873)

Syifa Fatmawati (2103840)

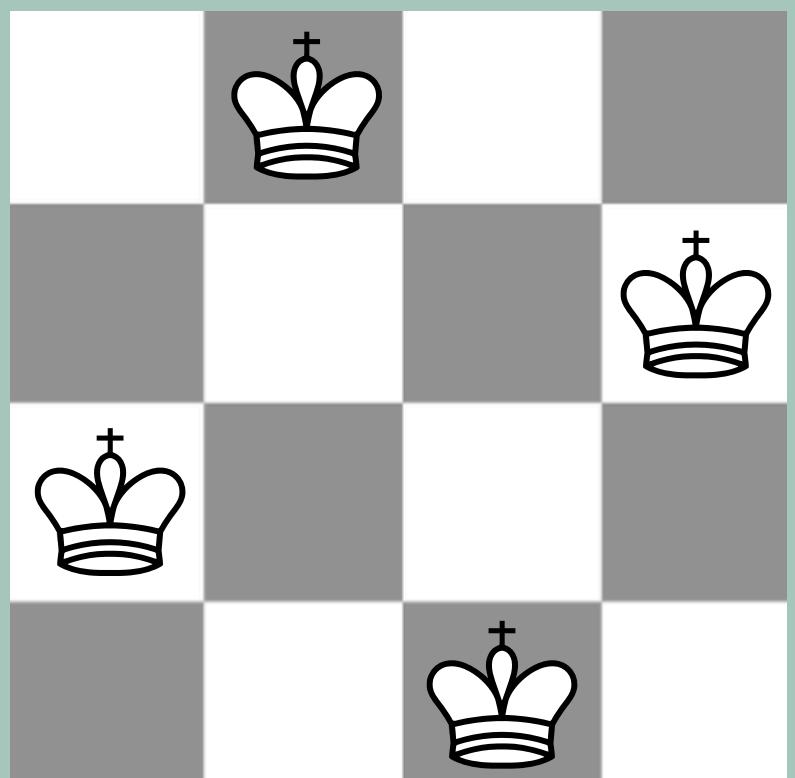
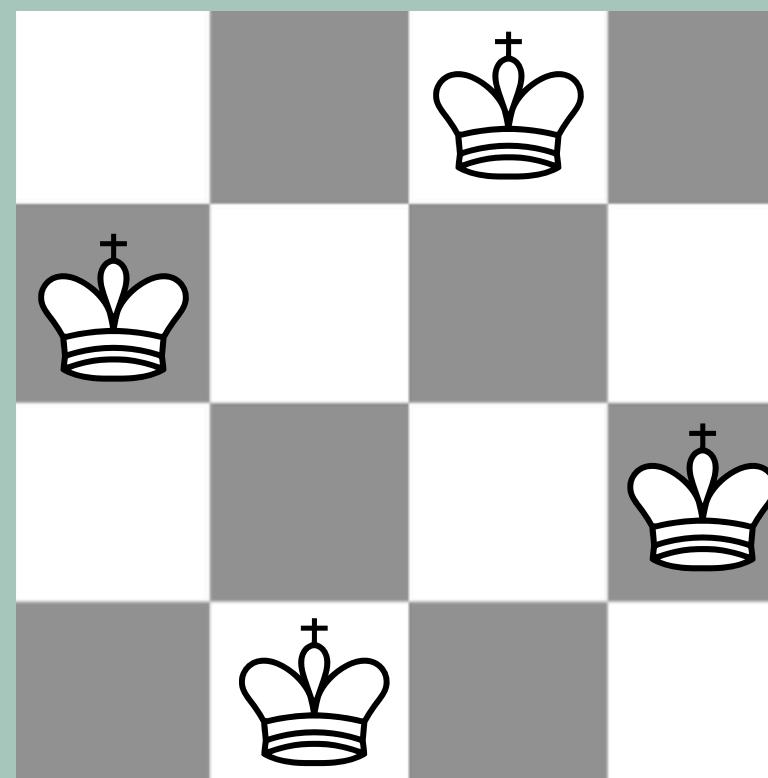


What is the N-Queen's Problem?

Masalah N-ratu adalah menempatkan sejumlah N ratu pada papan catur berukuran $N \times N$ sehingga tidak ada ratu yang berada di baris yang sama, kolom yang sama, atau diagonal yang sama.

Contoh:

$N = 4$, maka solusi seperti pada gambar di samping



N-Queen's Problem

Himpunan:

B = Representasi papan catur berukuran $N \times N$

Parameter:

N = Banyaknya ratu yang ditempatkan di papan catur

N_{ij} = Banyaknya ratu yang ditempatkan di baris ke- i dan kolom ke- j pada papan catur B

Variabel keputusan:

$N_{ij} = \begin{cases} 1; & \text{jika ratu ditempatkan di baris ke-}i \text{ dan kolom ke-}j \\ 0; & \text{jika ratu tidak ditempatkan di baris ke-}i \text{ dan kolom ke-}j \end{cases}$





N-Queen's Problem

Fungsi Objektif:

N-Queen's Problem tidak memiliki fungsi objektif yang spesifik untuk meminimalkan atau memaksimalkan. Dalam masalah ini hanya berfokus pada mencari konfigurasi yang valid di mana semua kendala terpenuhi.



N-Queen's Problem

Kendala:

1. Tepat satu ratu di setiap baris

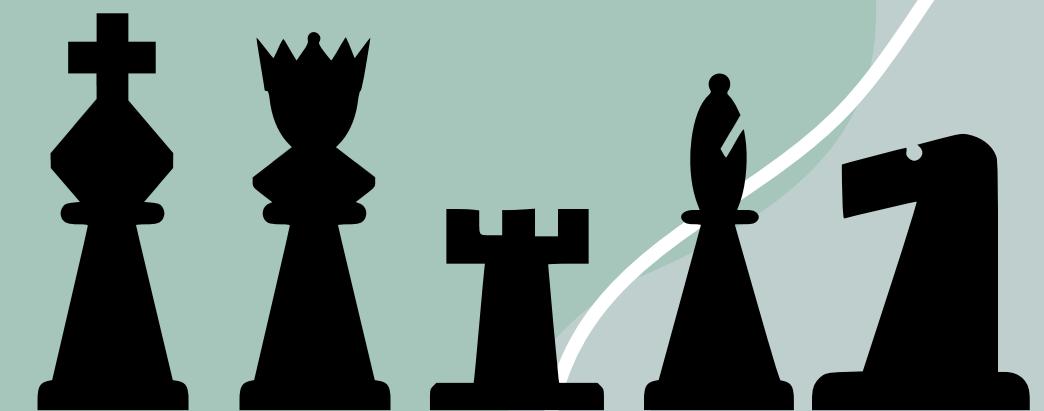
asumsikan N_{ij} untuk setiap kotak di baris tertentu (i). Jumlahnya harus sama dengan 1. Ditulis sebagai berikut:

$$\sum N_{ij} = 1 \text{ untuk semua } i \in \{1, \dots, n\}$$

2. Hanya satu ratu di setiap kolom

asumsikan N_{ij} untuk setiap kotak di kolom tertentu (j). Jumlahnya harus kurang dari atau sama dengan 1 karena ada kolom yang kosong. Ditulis sebagai berikut:

$$\sum N_{ij} \leq 1 \text{ untuk semua } j \in \{1, \dots, n\}$$



N-Queens Problem

3. Tidak ada ratu yang menyerang secara diagonal

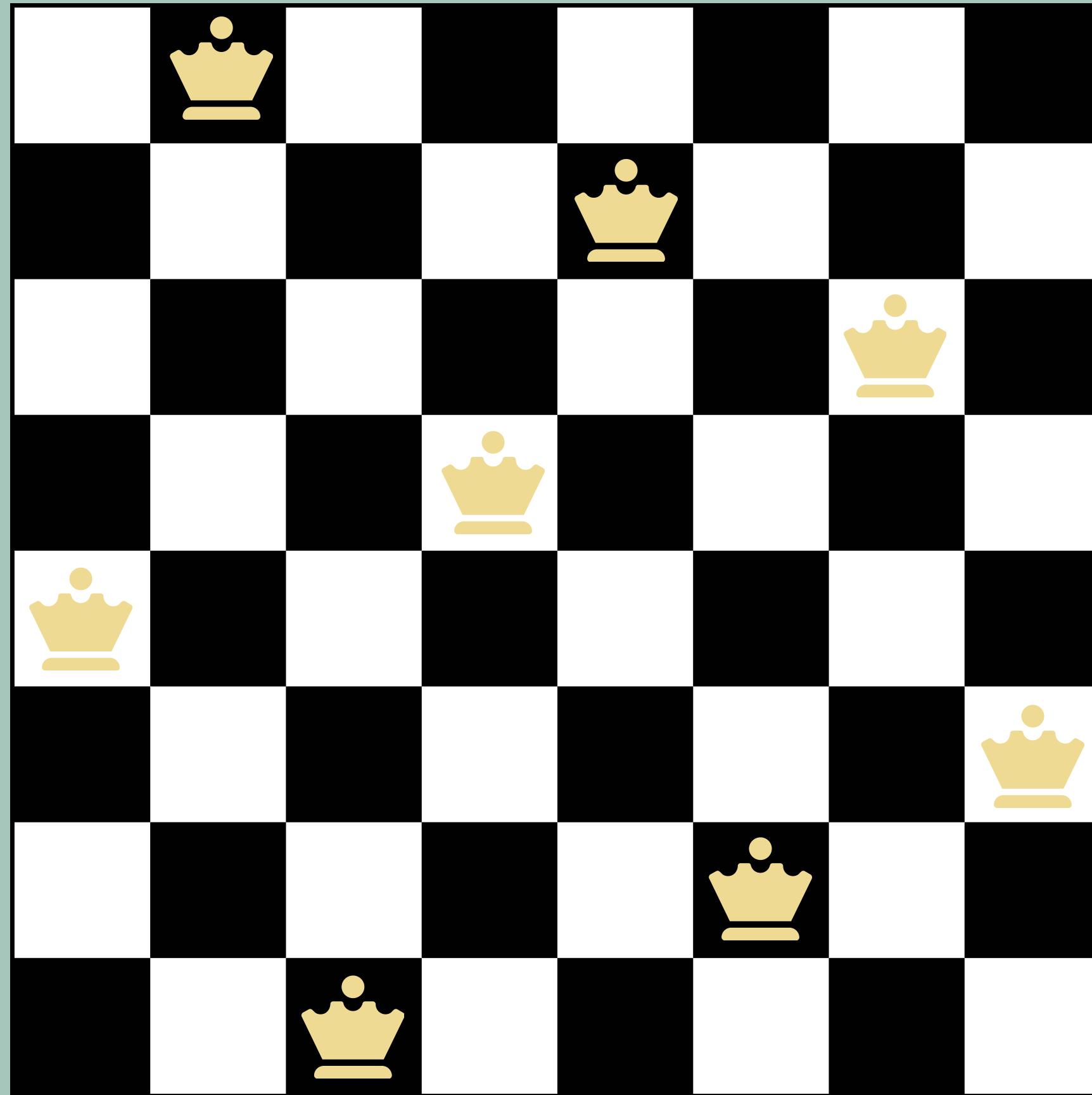
Ratu dapat menyerang secara diagonal, sehingga perlu dipastikan tidak ada dua ratu yang berbagi diagonal yang sama.

- Diagonal kanan: cari seluruh kemungkinan posisi pada diagonal kanan ($i + j$) dan cek apakah sudah ada ratu yang ditempatkan di diagonal tersebut. Secara matematis dapat ditulis:

$$\sum N_{i+j} \leq 1 \text{ untuk semua } i \in \{1, \dots, n\}$$

- Diagonal kiri: cari seluruh kemungkinan posisi pada diagonal kiri ($i - j$) dan cek apakah sudah ada ratu yang ditempatkan di diagonal tersebut. Secara matematis dapat ditulis:

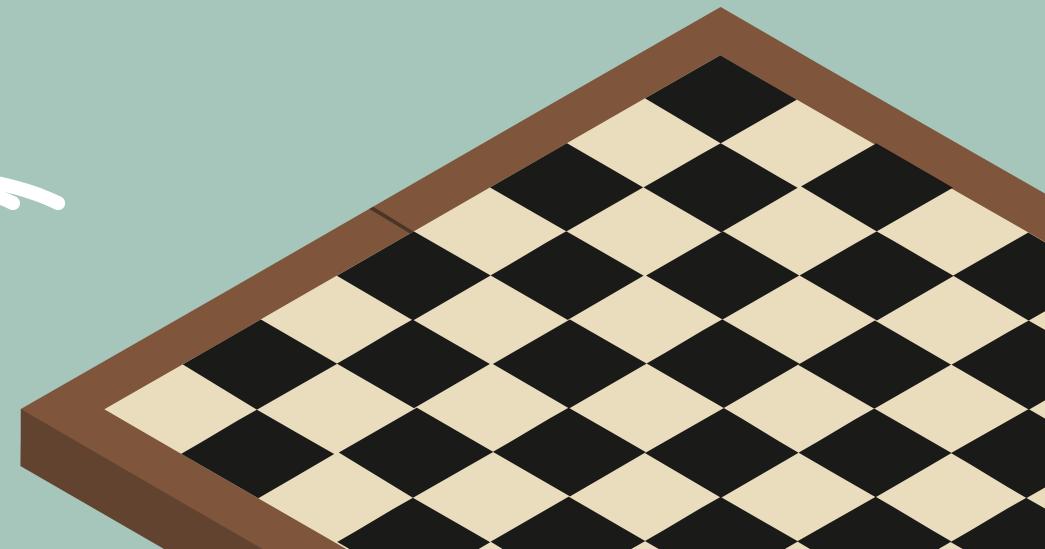
$$\sum N_{i-j} \leq 1 \text{ untuk semua } i \in \{1, \dots, n\}$$



Simulasi
penempatan
Queen

Deskripsi Masalah

Diberikan sebuah papan catur yang berukuran 8×8 dan 8 buah ratu. Bagaimanakah menempatkan 8 buah ratu (Q) itu pada petak-petak papan catur sedemikian sehingga tidak ada dua ratu atau lebih yang terletak pada satu baris yang sama, atau pada satu kolom yang sama, atau pada satu diagonal yang sama?



Simulated Annealing dengan Backtracking

Masalah N-Queen's dapat diselesaikan dengan menggunakan algoritma *backtracking*. Ideanya adalah untuk mengeksplorasi secara sistematis semua kemungkinan konfigurasi ratu di papan sambil memastikan tidak ada dua ratu yang saling mengancam.

Backtracking mampu menghasilkan solusi yang optimum, tetapi waktu komputasinya lama. Untuk mengatasi kelemahan yang dimiliki oleh Backtracking, digunakan Algoritma *Simulated Annealing* karena SA cepat dalam komputasi.

Algoritma Backtracking

Masukan : jumlah ratu (N)

Keluaran : jumlah solusi dari masalah penempatan ratu berdasarkan aturan yang berlaku

Langkah 1:

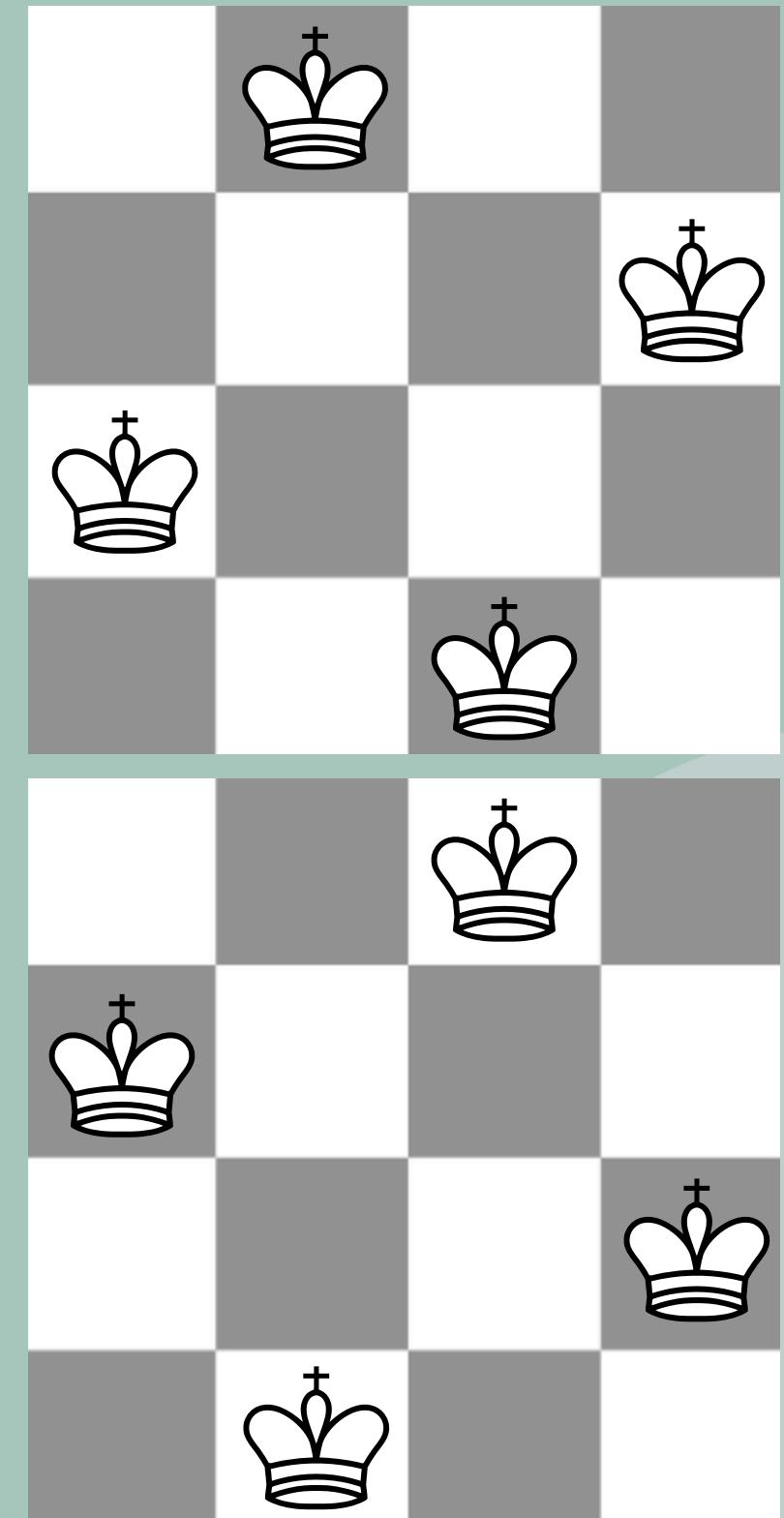
tempatkan ratu ke-1 di baris pertama

Langkah 2:

tempatkan ratu ke-2 di baris berikutnya dengan syarat tidak berada di kolom yang sama dan berada di satu garis diagonal yang sama dengan ratu ke-1

Langkah 3:

apabila tidak ada posisi yang memungkinkan di baris berikutnya, kembali ke baris sebelumnya dan tempatkan ratu di posisi yang tersedia di baris tersebut dan ulangi



Algoritma Simulated Annealing

Langkah-langkah Algoritma Simulated Annealing (SA):

1. Inisialisasi solusi awal
 - Menggunakan “Backtracking” untuk menghasilkan solusi awal untuk masalah N-Queens.
 - Jika solusi awal tidak ditemukan, maka lakukan pencarian awal lagi.
2. Hitung biaya awal
 - Menghitung biaya (cost) dari solusi awal dengan menggunakan fungsi “cost”.
 - simpan biaya awal ke dalam variabel cost_answer.
3. Inisialisasi parameter simulated annealing
 - Tetapkan suhu awal (t) dengan nilai TEMPERATURE.
 - Tetapkan faktor pendukung (sch) dengan nilai 0,99.

Algoritma Simulated Annealing

Langkah-langkah Algoritma Simulated Annealing (SA):

4. Posisi Iterasi Simulated Annealing

Ulangi proses selama suhu t lebih besar dari 0:

- Kurangi suhu dengan mengalikan t dengan faktor pendinginan (sch).
- Salin solusi saat ini (answer) ke variabel successor.
- Pilih secara acak baris (row) dan kolom (col) dalam rentang N-Queens.

Pindahkan Queen di baris Terpilih:

- Cari posisi queen di kolom terpilih dalam baris tersebut.
- Tetapkan queen pada posisi kolom baru secara acak dalam baris yang sama.

Algoritma Simulated Annealing

Langkah-langkah Algoritma Simulated Annealing (SA):

4. Posisi Iterasi Simulated Annealing

Hitung Delta Biaya:

- Hitung perbedaan biaya antara solusi baru (successor) dan solusi saat ini (cost_answer).

Penerimaan Solusi Baru:

- Jika delta lebih kecil dari 0 (solusi baru lebih baik), terima solusi baru.
- Jika delta lebih besar dari 0, terima solusi baru dengan probabilitas yang dihitung menggunakan fungsi $\exp(-\delta / T)$.

Algoritma Simulated Annealing

Langkah-langkah Algoritma Simulated Annealing (SA):

4. Posisi Iterasi Simulated Annealing

Pembaruan Solusi:

- Jika solusi baru diterima, perbarui answer dengan successor dan perbarui cost_answer.

Periksa Solusi Optimal:

- Jika cost_answer sama dengan 0 (solusi optimal ditemukan), cetak solusi dan tandai solution_found sebagai True, kemudian keluar dari loop.

Algoritma Simulated Annealing

Langkah-langkah Algoritma Simulated Annealing (SA):

5. Cek Solusi

- Jika setelah semua iterasi solusi optimal tidak ditemukan (`solution_found`) masih `False`, cetak “Failed”.

Implementasi Simulated Annealing dan Backtracking

Gambar 1. Program

```
import random
from math import exp
import time
from copy import deepcopy

N_QUEENS = 8
TEMPERATURE = 4000

def threat_calculate(n):
    '''Combination formula. It is choosing two queens in n queens'''
    if n < 2:
        return 0
    if n == 2:
        return 1
    return (n - 1) * n / 2
```

Gambar 1. Program

```
def is_safe(board, row, col, N):
    '''Check if placing a queen at (row, col) is safe'''
    # Check the column
    for i in range(row):
        if board[i][col] == 1:
            return False
    # Check the left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    # Check the right diagonal
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j] == 1:
            return False
    return True
```

Gambar 1. Program

```
def backtracking(N):
    '''Solve the N Queens problem and return the board'''
    board = [[0 for _ in range(N)] for _ in range(N)]
    if solve_util(board, 0, N):
        return board
    else:
        print("No solution exists")
        return None

def solve_util(board, row, N):
    '''Utility function to solve N Queens problem recursively'''
    if row == N:
        return True # All queens are placed successfully
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = 1
            if solve_util(board, row + 1, N):
                return True
            board[row][col] = 0 # Backtrack if no solution found
    return False

def print_solution(board):
    '''Print the chess board'''
    for row in board:
        print(' '.join('Q' if val == 1 else '.' for val in row))
```

Gambar 1. Program

Gambar 1. Program

```
def simulated_annealing():
    '''Simulated Annealing'''
    solution_found = False
    answer = backtracking(N_QUEENS)

    if answer is None:
        return

    # To avoid recounting when can not find a better state
    cost_answer = cost(answer)

    t = TEMPERATURE
    sch = 0.99
```

Gambar 1. Program

```
while t > 0:  
    t *= sch  
    successor = deepcopy(answer)  
    row = random.randrange(0, N_QUEENS)  
    col = random.randrange(0, N_QUEENS)  
    while True:  
        if successor[row][col] == 1:  
            successor[row][col] = 0  
            break  
        row = random.randrange(0, N_QUEENS)  
        col = random.randrange(0, N_QUEENS)  
        successor[row][col] = 1 # Move the queen  
    delta = cost(successor) - cost_answer  
    if delta < 0 or random.uniform(0, 1) < exp(-delta / t):  
        answer = deepcopy(successor)  
        cost_answer = cost(answer)
```

Gambar 1. Program

```
if cost_answer == 0:  
    solution_found = True  
    print_solution(answer)  
    break  
if not solution_found:  
    print("Failed")  
  
def main():  
    start = time.time()  
    simulated_annealing()  
    print("Runtime in second:", time.time() - start)  
  
if __name__ == "__main__":  
    main()
```

Gambar 2. Output

```
Q . . . . .  
. . . Q . .  
. . . . . Q  
. . . . Q . .  
. . Q . . .  
. . . . . Q  
. Q . . . .  
. . . Q . . .  
  
Runtime in second: 0.002162456512451172
```

Setelah program dijalankan, hasil dari posisi ratu yang mungkin adalah sebagai berikut:

ratu 1 berada di baris 1 kolom 1

ratu 2 berada di baris 2 kolom 5

ratu 3 berada di baris 3 kolom 8

ratu 4 berada di baris 4 kolom 6

ratu 5 berada di baris 5 kolom 3

ratu 6 berada di baris 6 kolom 7

ratu 7 berada di baris 7 kolom 2

ratu 8 berada di baris 8 kolom 4

dengan waktu komputasi 0,0022 detik

Thank You