TUGAS 4

8-Queens using Genetic Algorithm

Kecerdasan Buatan (F)
Kelompok 2 - Cucur Adabi

Anggota Kelompok 2 - Cucur Adabi

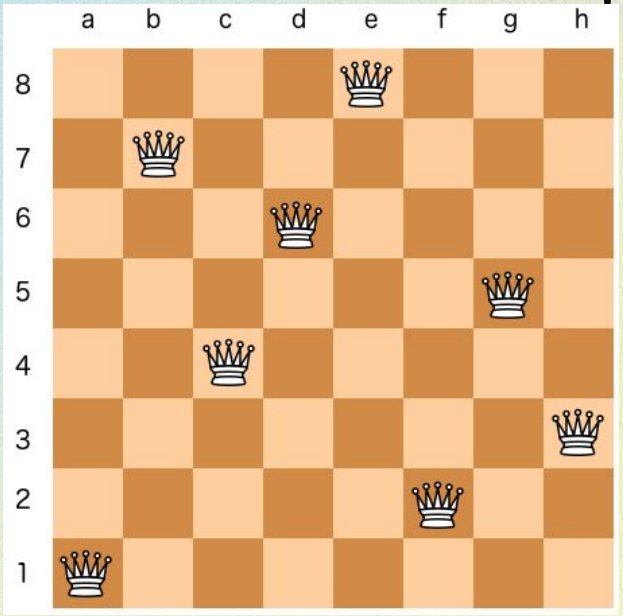
Rizky Alifiyah Rahma
5025211208

Salsabila Fatma Aripa
5025211057

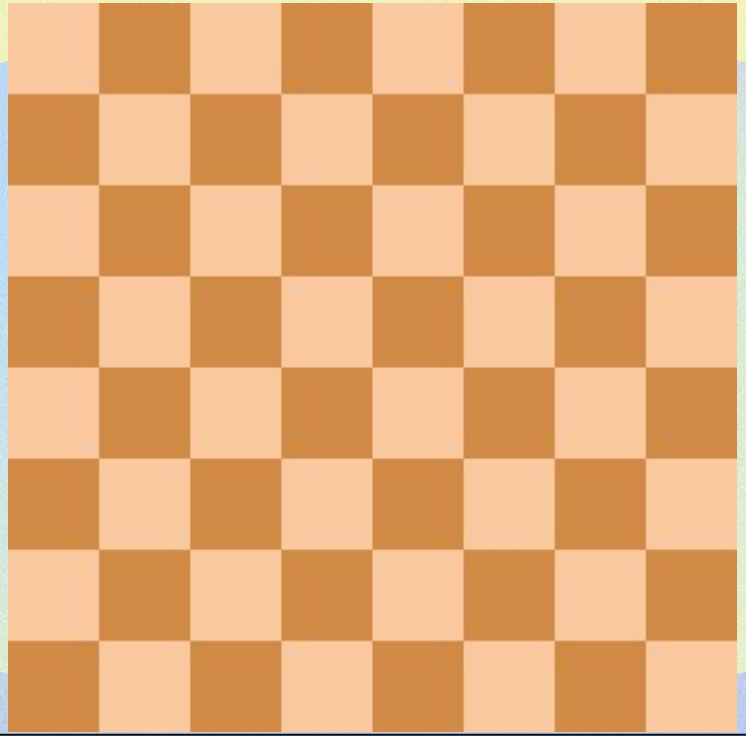
Tsabita Putri Ramadhany
5025211130

8-Queens

8 Queens merupakan sebuah problem di mana 8 queens yang diletakkan di papan catur tidak mengancam satu sama lain , baik itu secara vertikal, horizontal, dan diagonal



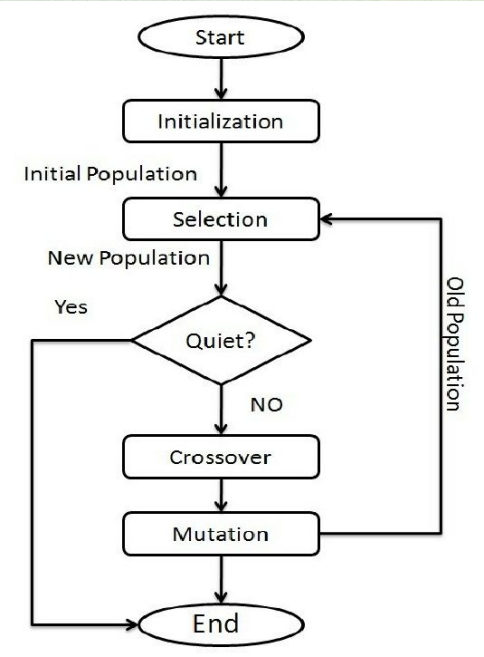
8-Queens Illustration



Genetic Algorithm

Genetic Algorithms (GA) adalah algoritma pencarian heuristik yang menggabungkan konsep seleksi alam dan genetika. GA mensimulasikan seleksi alam dan menggunakan operasi genetika untuk menciptakan generasi baru dan mencari solusi terbaik pada berbagai masalah optimisasi dan pencarian. Dengan demikian, GA adalah metode yang efektif dan berguna dalam menyelesaikan masalah optimisasi dan pencarian yang kompleks.

Genetic Algorithm Flowchart



Ringkasan Algoritma

1. Inisialisasi populasi p secara acak
2. Menentukan kecocokan populasi
3. Ulangi hingga konvergensi:
    a) Pilih induk dari populasi
    b) Crossover dan hasilkan populasi baru
    c) Lakukan mutasi pada populasi baru
    d) Hitung kecocokan untuk populasi baru

Implementasi

```python
from operator import indexOf
import random

# Membuat kromosom secara acak dengan ukuran size yang diinputkan
def random_chromosome(size):
    return [random.randint(0, size - 1) for _ in range(size)]


# Menghitung nilai fitness dari kromosom. Nilai fitness menunjukkan seberapa baik
# kromosom dapat menyelesaikan masalah n-queen. Semakin kecil nilai fitness, semakin baik kromosom tersebut.
def fitness(chromosome, maxFitness):
    horizontal_collisions = (
        sum([chromosome.count(queen) - 1 for queen in chromosome]) / 2
    )
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * (2 * n - 1)
    right_diagonal = [0] * (2 * n - 1)
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2 * n - 1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i] - 1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i] - 1
        diagonal_collisions += counter

    # 28-(2+3)=23
    return int(maxFitness - (horizontal_collisions + diagonal_collisions))

# Melakukan operasi crossover antara 2 kromosom yang diberikan
def crossover(x, y):
    n = len(x)
    child = [0] * n
    for i in range(n):
        c = random.randint(0, 1)
        if c < 0.5:
            child[i] = x[i]
        else:
            child[i] = y[i]
    return child


# Melakukan mutasi pada suatu kromosom.
def mutate(x):
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(0, n - 1)
    x[c] = m
    return x


# Menghitung probabilitas dari suatu kromosom.
def probability(chromosome, maxFitness):
    return fitness(chromosome, maxFitness) / maxFitness

```

```python
# Memilih kromosom secara acak dari populasi dengan probabilitas tertentu (Roulette-wheel selection).
def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"


# Melakukan proses evolusi dengan menggunakan algoritma genetika.
def genetic_queen(population, maxFitness):
    mutation_probability = 0.1
    new_population = []
    sorted_population = []
    probabilities = []
    for n in population:
        f = fitness(n, maxFitness)
        probabilities.append(f / maxFitness)
        sorted_population.append([f, n])

    sorted_population.sort(reverse=True)

    # Elitism
    new_population.append(sorted_population[0][1])  # the best gen
    new_population.append(sorted_population[-1][1])  # the worst gen

    for i in range(len(population) - 2):

        chromosome_1 = random_pick(population, probabilities)
        chromosome_2 = random_pick(population, probabilities)
```

```
        # Creating two new chromosomes from 2 chromosomes
        child = crossover(chromosome_1, chromosome_2)

        # Mutation
        if random.random() < mutation_probability:
            child = mutate(child)

        new_population.append(child)
        if fitness(child, maxFitness) == maxFitness:
            break
    return new_population


# Mencetak kromosom beserta nilai fitnessnya.
def print_chromosome(chrom, maxFitness):
    print(
        "Chromosome = {},  Fitness = {}".format(str(chrom), fitness(chrom, maxFitness))
    )


# Mencetak papan catur berdasarkan kromosom yang diberikan.
def print_board(chrom):
    board = []

    for x in range(nq):
        board.append(["x"] * nq)

    for i in range(nq):
        board[chrom[i]][i] = "Q"

    def print_board(board):
        for row in board:
            print(" ".join(row))

    print()
    print_board(board)
```

```python
if __name__ == "__main__":
    POPULATION_SIZE = 500

    while True:
        # say N = 8
        nq = int(input("Silakan masukkan jumlah ratu yang Anda inginkan (0 untuk keluar): "))
        if nq == 0:
            break

        maxFitness = (nq * (nq - 1)) / 2  # 8*7/2 = 28
        population = [random_chromosome(nq) for _ in range(POPULATION_SIZE)]

        generation = 1
        while (
            not maxFitness in [fitness(chrom, maxFitness) for chrom in population]
            and generation < 200
        ):

            population = genetic_queen(population, maxFitness)
            if generation % 10 == 0:
                print("=== Generasi {} ===".format(generation))
                print(
                    "Maximum Fitness = {}".format(
                        max([fitness(n, maxFitness) for n in population])
                    )
                )
            generation += 1

        fitnessOfChromosomes = [fitness(chrom, maxFitness) for chrom in population]

        bestChromosomes = population[
            indexOf(fitnessOfChromosomes, max(fitnessOfChromosomes))
        ]
```

```
171         if maxFitness in fitnessOfChromosomes:
172             print("\nDiselesaikan pada generasi {}!".format(generation - 1))
173 
174             print_chromosome(bestChromosomes, maxFitness)
175 
176             print_board(bestChromosomes)
177 
178         else:
179             print(
180                 "\nJawaban tidak ditemukan {}. Jawaban terbaik yang ditemukan adalah:".format(
181                     generation - 1
182                 )
183             )
184             print_board(bestChromosomes)
```

Hasil

```
Silakan masukkan jumlah ratu yang Anda inginkan (0 untuk keluar): 8
=== Generasi 10 ===
Maximum Fitness = 27
=== Generasi 20 ===
Maximum Fitness = 27
=== Generasi 30 ===
Maximum Fitness = 27
=== Generasi 40 ===
Maximum Fitness = 27
=== Generasi 50 ===
Maximum Fitness = 27
=== Generasi 60 ===
Maximum Fitness = 27
=== Generasi 70 ===
Maximum Fitness = 27
Diselesaikan pada generasi 75!
Chromosome = [6, 3, 1, 7, 5, 0, 2, 4],  Fitness = 28

x x x x x Q x x
x x Q x x x x x
x x x x x x Q x
x Q x x x x x x
x x x x x x x Q
x x x x Q x x x
Q x x x x x x x
x x x Q x x x x
```

Terima Kasih