Lab #7 (extra lab): Genetic Algorithm

The main aim of this lab is to solve the problem of 8 Queen using **Genetic Algorithm.**

The problem statement is as follows: Consider an N × N chessboard. Place N queens on the board such that no two queens are attacking each other. The queen is the most powerful piece in chess and can attack from any distance horizontally, vertically, or diagonally. Thus, a solution must place the queens such that no two queens are in the same row, the same column, or along the same diagonal.

In this lab, the problem is solved using a **complete-state formulation** (N=8), which means we start with **all 8 queens on the board**. We represent the 8 × 8 chessboard as a matrix. In addition, we assume that each Queen is placed on a different column. Therefore, we try to move the Queen to different row (each by one row) to reach a goal state. The heuristic is measured by using:

- *h = the number of pairs of attacking queens*

(Use source code from the previous Lab)

Add the following methods in **Node.java** class:

```
// get the row of a queen at index i
     public int getRow(int i) {
          return state[i].getRow();
     }

     // set the row of a queen at index i
     public void setRow(int i, int row) {
          state[i].setRow(row);

     }
```

Then implement the following methods in **GA_NQueenAlgo** for Genetic Algorithm

```
public class GA_NQueenAlgo {
     public static final int POP_SIZE = 100;//Population size
     public static final double MUTATION_RATE = 0.03;
     public static final int MAX_ITERATIONS = 1000;
     List<Node> population = new ArrayList<Node>();
     Random rd = new Random();

     // initialize the individuals of the population
     public void initPopulation() {
          for (int i = 0; i < POP_SIZE; i++) {
               Node ni = new Node();
               ni.generateBoard();
               population.add(ni);
          }
     }
```

```
	public Node execute() {
		// Enter your code here
		return null;
	}
	// Mutate an individual by selecting a random Queen and
//move it to a random row.
	public void mutate(Node node) {
		// Enter your code here
		return null;
	}
//Crossover x and y to preproduce a child
	public Node preproduce(Node x, Node y) {
		// Enter your code here
		return null;
	}

	// Select K individuals from the population at random and
	//select the best out of these to become a parent.
	public Node getParentByTournamentSelection() {
		// Enter your code here
		return null;
	}
	//Select a random parent from the population
	public Node getParentByRandomSelection() {
		// Enter your code here
		return null;
	}

}
```

Pseudocode for execute is decribed as follows:

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for $i = 1$ **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNESS-FN)
      y ← RANDOM-SELECTION(*population*, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *new_population*
    population ← *new_population*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to FITNESS-FN

---

function REPRODUCE(x, y) **returns** an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, $c + 1$, n))