**Program**:

|  |  |
| --- | --- |
| import java.util.ArrayList;  import java.util.HashMap;  import java.util.Iterator;  import java.util.Random;  import java.util.HashMap;  import java.util.SortedSet;  import java.util.TreeSet;  import java.util.HashSet;  public class NQueens {    static final int maxIters = 10000;  public NQueens() {  }  public static Solution solve(Board board) {  int iters = 0;  HashMap<Integer,Integer> fitnessMap = new HashMap<Integer,Integer>();  while (board.fitness() < board.maxFitness && iters < maxIters) {  Iterator<Board> iter = board.makeNeighborhood().iterator();  while (iter.hasNext()) {  Board b = iter.next();  if ( b.fitness() > board.fitness() ) {  board = b;  fitnessMap.put(iters, board.fitness());  }  }  Iters++; }  return new Solution(board, iters, fitnessMap);  }  public static ArrayList<Solution> solve(int iters) {  ArrayList<Solution> solutions = new ArrayList<Solution>();  for (int i=0; i<iters; i++) {  Board b = new Board(8);  Solution solution = solve(b);  solutions.add(solution); }  return solutions; }  public static int schedule(int iter) {  return 100/(iter+1); }  public static Solution annealing(Board board) {  int iters = 0;  Random r = new Random();  HashMap<Integer,Integer> fitnessMap = new HashMap<Integer,Integer>();  while (board.fitness() < board.maxFitness && iters <= maxIters) {  ArrayList<Board> neighbors = board.makeNeighborhood();  Board n = neighbors.get(r.nextInt( neighbors.size() ));  this.state = state;  this.maxFitness = ((size-1) \* size) / 2;  }  public void printBoard() {  for (int row=0; row<size; row++) {  for (int col=0; col<size; col++) {  if (state[row] == col) {  System.out.print("1 ");  } else {  System.out.print("0 ");  }  }  System.out.println("");  }    System.out.println("number of pairs of attacking Queens via cols: " +  countCol());  System.out.println("number of pairs attacking via diagonal: " +  countDiagonal());  System.out.println("cost: " + cost() + ",fitness: " + fitness());  }  public int fitness() {  return maxFitness - cost();  }    public int countDiagonal() {    int attacking = 0;  for (int i = 0; i<size; i++) {  for (int j = i+1; j<size; j++) {  if (state[i] + (j-i) == state[j]) {  attacking++;  }  if (state[i] + (i-j) == state[j]) {  attacking++;  }  }  }  return attacking;  }  public int countCol() {  int count = 0;  for (int i=0; i<size; i++) {  for (int j=i+1; j<size; j++) {  if (state[i] == state[j]) {  Count++;  }  }  }  return count; } | if (n.fitness() >= board.fitness()) {  board = n; }  else {  int deltaF = n.fitness() - board.fitness();  double prob = Math.exp(deltaF/(float)schedule(iters));  if (r.nextInt(10000) <= (int)Math.floor(prob\*10000)) {  board = n;  }  }  if (iters % 50 == 0) {  fitnessMap.put(iters, board.fitness());  }  iters++;  }  return new Solution(board, iters, fitnessMap);  }    public static void main(String[] args) {    Board board = new Board(8);  board.printBoard();  System.out.println();  Solution s = solve(board);  Solution s = annealing(board);  s.printSolution();  }  }  class Board{  int maxFitness;  private int size;  int[] state = new int[size];  Random rand = new Random();  public Board(int s) {  int[] state = new int[s];  for (int i=0; i<s; i++) {  state[i] = rand.nextInt(s);  }  construct(state);  }  public Board(int[] state) {  construct(state);  }    private void construct(int[] state) {  this.size = state.length;  public int cost() {  return countCol() + countDiagonal();  }  private Board createNeighbor(int row) {  int[] board = new int[size];  for (int i=0; i<size; i++) {  board[i] = this.state[i];  }  board[row] = rand.nextInt(size);  return new Board(board);  }  public ArrayList<Board> makeNeighborhood() {  ArrayList<Board> nHood = new ArrayList<Board>(size);  for (int i=0; i<size; i++) {  nHood.add( createNeighbor(i) );  }  return nHood;  }  }  class Solution {  Board board;  int iterations;  HashMap<Integer,Integer> fitnessMap;    public Solution(Board b, int iters, HashMap<Integer,Integer> fm) {  this.board = b;  this.iterations = iters;  this.fitnessMap = fm;  }    public void printSolution() {  board.printBoard();  System.out.println("total iterations: " + iterations);    System.out.println();  System.out.println("iteration | fitness");  SortedSet<Integer> fs = new TreeSet<Integer>(fitnessMap.keySet());  for (Integer f : fs) {  System.out.println(" " + f + "\t\t" + fitnessMap.get(f));  }  }  } |

**Output**:



