Student Names : Raj Jani(801077311), Amit Patil(801077053).

libraries used: Random, choice

**Solving N-queens problem by Hill-Climbing and its variants**

**N–queen formulation:**

User inputs number of queens for N-queen Matrix. After that the object of Q\_State() will be created. Global Variable n and num\_iter will get created for number of queens and number of iterations as these will be used throughout the program. Default constructor of Q\_State will call the method ‘rand\_position’, that will position the queens in random positions with each queen in a random row in a separate column. randrange function is used to create random position of the queen within side\_length(NxN) and queen\_num(number of queens).

Now with the use of built in function object of Q\_State() is represented in matrix form.

**Program Structure:**

Functions and Classes:

1. Class QueensProblem() will create the object of Q\_State and initialization of object will happen.

2. Q\_State() is the main class which will create the n queen matrix. Following methods are present in the Q\_State:

a. Default Constructor (\_\_init\_\_): Constructor will initialize s\_len equal to n i.e. number of columns. All nodes keep track of path cost, their parent and instance id. Path length and parent for root node is equal to ‘0’.

b. rand\_position: It takes object of Q\_State as input. List of q\_pos will get created randomly with the use of randrange function of library random.

c. get\_children : All valid positions(position of queen not equal to parent) will get find out and will get stored in list named new\_pos. All possible children will be generated in this list. Now the objects of Q\_State will be generated for all children. List ‘Children’ will contain all the available children.

d. q\_attacks: This function is used to check if there are queens that are attacking each other. It will check each queen in q\_pos, calculate the heuristic cost and then check if any queens attack each other or not. The pair of queens will be added to ‘att\_pairs’ if they are attacking eachother.

e. num\_q\_attacks: This will give the total number of queens that are attacking each other.this will be calculated using the Length of att\_pairs which is returned from q\_attacks function.

f. \_\_str\_\_: This will print the N\*N queen chess board by overriding the default function \_\_str\_\_

3. steepest\_without\_sidemoves: This function is called when steepest ascent (without sidemoves) is chosen by the user. It takes parameters such as an object of Q\_State(), a counter count, allow\_sidemoves(i.e. false) and max\_sidemoves (set to 100). we initialized s\_moves to 0 as we are not allowing any sidemoves.At first the function will search for all the children and store it in children array, then it will store all the queens attacking in num\_attacks\_children . Then among all the children the best child will be chosen and will return ‘success’ if the number of queens attacking is 0 or if the number of iterations entered by the user is reached.

4. steepest\_with\_sidemoves: This function will also function like previous one but here the sidemoves are allowed. Similarly from the set of children’s it will select the best child based on the heuristic cost, it will allow side moves if the heuristic of current node is same as the heuristic value of the adjacent node and will return ‘success’ if number of attacking queens is 0 or if the number of iterations entered by the user is reached. Also, the Max\_sidemoves is set to 100.

5. random\_restart : This function is called when random restart option is selected. The parameters that it takes are allow\_sidemoves(i.e. true or false) , the number of restarts as num\_restarts, max\_sidemoves(set to 100) . In this function we call steepest\_with\_sidemoves or steepest\_without\_sidemoves multiple times till it reaches the maximum no. of restarts that are allowed. We use a global variable ‘total’ which stores the total number of restarts required for an iteration and this variable is used in the ‘stats’ to calculate average number of restarts required.

6. status: It is a function used to calculate the statistics of a hill climbing method. It stores the path followed by the algorithm to reach the goal. It also stores the number of success and failures found and hence average number of steps required when it success and fails can be calculated.

7. rand\_stats: It is a function used to analyze the statistics of random restart hill climbing method. It stores the path used by the algorithm to reach goal state. Also, the number of success and failures encountered are stored, and average number of steps required for each success and fails can be calculated using this method.

**Algorithm for Hill Climbing:**

1. input number of iterations and queens(done by user)

2. Create Q\_State() object.

3. Initialize all variables using default constructor.

4. Call to the specific algorithm will be made based on user input(for example Steepest Ascent with side moves)

5. While True

6. {

7. Generate all children and store in children

8. Heuristic value(Number of queens attacking each other) of all generated nodes will get store in

children\_num\_queen\_attacks

9. Minimum value of heuristic = best\_child.

10. if (number of attacks in best\_child > number of attacks in node):

11. break;

12. elseif (number of attacks in best\_child == number of attacks in node):

13. if not allow\_sideways or sideways\_moves == max\_sideways:

14. break;

15. else:

16. Increment sideways\_moves

17. Select best node as child node

18. Results contain path, path length and outcome.

19.Print path.

20.Print number of steps.

**Algorithm for random restart:**

1. Loop, generate initial states using random generator function and pass it to steepest ascent function.

3. If (success)

4. Break;

5. Result contains path, path length and outcome.