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

Implement by : Anup Bharade and Sumedh Joglekar
Language : Python3
Inbuild libraries : Random, choice (If needed install these libraries)
Data Structure Used : List, Dictionary

- **N–queen formulation:**
    1. Numbers of queens needed to form the N-queen matrix are taken from user.
    2. Once the number of queens and the algorithm to be implemented is entered by the user object of QueenState() will get created.
    3. Global Variable n and num_iter will get created for number of queens and number of iterations as this parameters are required throughout the code.
    4. Default constructor of QueenState will give call to method 'random_queen_position', which will decide the random queen positions in a random row in a separate column.
    5. randrange function of random library is used to create random position of the queen within side_length(NxN) and queen_num(number of queens).
    6. Now with the use of built in function object of QueenState() is represented in matrix form.

- **Program Structure**:
  Functions and Classes:
    1. Class QueensProblem() will create the object of QueensState and initialization of object will happen.

    2. QueensState() is the main class which will create the n queen matrices. Below all methods are present in the QueensState:
        a. Default Constructor (__init__): Constructor will initialize side_length equal to the n(number of columns). Every node will keep track of the path cost, its parent and instance id. For root node path length and parent both will be '0'.
        b. random_queen_position: random_queen_position is taking object of QueensState as an input. List of queen_positions 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_positions All possible children of parent will get generated in this list. Now the objects of QueensState will be getting created for all children. List 'Children' will contain all the available children.
        d. random_child : No need Remove from code
        e. queen_attacks: This function will identify the number of queens attacking each other. For each element in queen_positions it will calculate the heuristic cost

and checks if the queens are attacking each other or not. If the queens are attacking each other, pair will get added in new list called 'attacking_pairs'.

f. num_queen_attacks: Total number of queens attacking to each other will be calculated in this function. Length of attacking_pairs which is getting returned from queen_attacks will give this attribute.

g. __str__: It will override the default definition of __str__, so that the N*N queen board will get printed.

3. steepest_ascent_hill_climb_false:
   This function will get call when steepest ascent hill climb without side moves is selected by the user. It is taking parameter as object of QueenState(), Counter, allow_sideways(which is by default false for this function call), max_sideways (which is set as 100). As we are not allowing any side way moves, we are initializing that sideways_moves=0. First function will find all the children and number of queens attacking and will store the result in children and children_num_queen_attacks resp. Now the best child will be selected from available children. It will return success if number of attacking queens==0 or the number of iterations entered by the user.

4. steepest_ascent_hill_climb_true:
   This function will also select the best child from all derived children based on the heuristic and also it will allow making side move if the heuristic value of current node is same as the heuristic value of its adjacent. Function will return success if number of attacking queens==0 or the number of iterations entered by the user. Max_sideways is set to 100.

5. random_restart_hill_climb:
   This function is for random restart algorithm. It takes parameters as allow_sideways stating if sideway moves are allowed or not, number of restarts, max permitted side way moves. Inside this function we are calling steepest_ascent_hill_climb repeatedly till it reaches to max number of restarts allowed. Global variable total is used to store the number of restarts needed for particular iteration and as it is global variable we are accessing it in the function called 'stats' in order to calculate average number of restarts required.

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

7. stats_random:
   It is a function used to analyze the stats of random restart hill climbing method. It will store the path followed by the algorithm to reach the goal. It also stores the number of successes and failures found and hence average number of steps required when it success and fails can be calculated.

- Algorithm for Hill Climbing:

1. Take number of queens and number of iterations from user
2. Generate object of QueenState()
3. Initialize all variables of QueenState with the help of default constructor.
4. Call to the specific algorithm will be made on the basis of user input(Lets consider user selects Steepest Ascent with side moves)
5. While True
6. {
7. All children of current state will get generated and will get 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 will get selected from all available heuristics which will be in 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. Now the result will have path, path length and outcome either ‘success’ or ‘failure’.
19. At last, print the path followed and average number of steps required.

- Algorithm for random restart:

1. Loop
2. Generate initial states using random generator function and pass it to steepest ascent function. (If sideway move allowed then call steepest ascent function with sideway move)
3. If result is success
4. Break
5. Now the result will have path, path length and outcome