PROGRAMMING PROJECT 2

SOLVING THE N-QUEEN PROBLEM

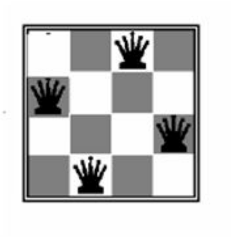
UNC CHARLOTTE

AKSHAY GUPTA

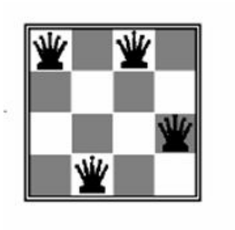
agupta52@uncc.edu | 704-421-6654
ID: 801135279

N-QUEENS PROBLEM

GOAL : Given any initial configuration i.e. a setting on ‘n’ queens on a n x n chess board, find a configuration such that no two queens are attacking each other.



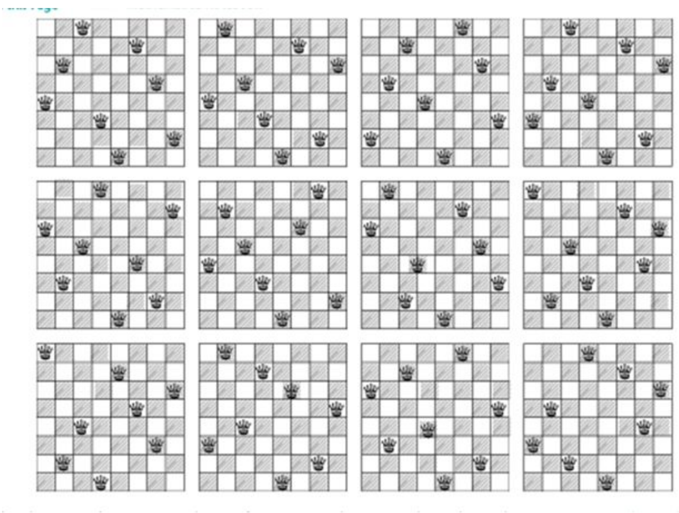
Goal configuration



Bad goal configuration

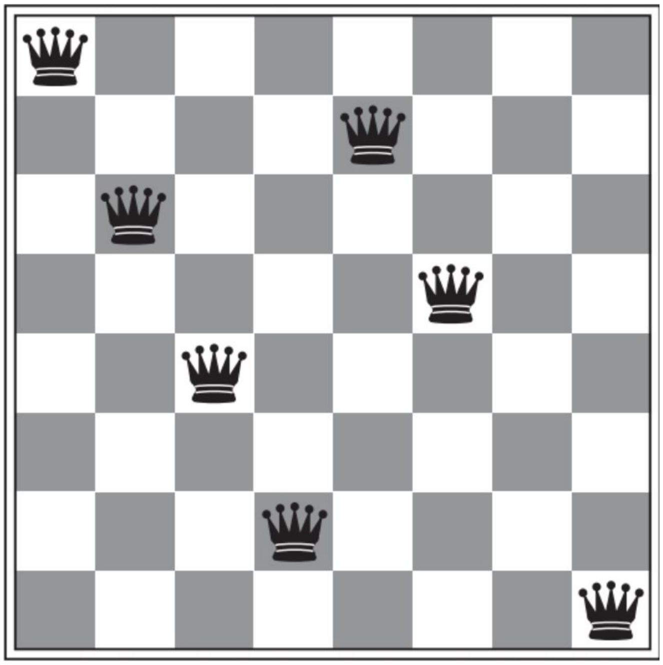
*here n = 4

12 UNIQUE SOLUTIONS FOR THE 8-QUEENS PROBLEM



N-QUEENS PROBLEM FORMULATION

N = 8



- State : Any arrangement of 8 queens on the board is a state
- Initial state: Randomly place 8 queens on the board
- Actions: Add a queen to any empty square
- Transition Model : Return a model with a queen added to the specified square
- Goal Test: 8 (n in general) queens are on the board. None Attacked.

HILL CLIMBING SEARCH ALGORITHM

The hill-climbing search algorithm (steepest-ascent version) is simply a loop that continually moves in the direction of increasing value—uphill.

There are four types of hill climbing search:

1. BASIC HILL CLIMBING

- Terminates when it reaches a “peak” where no neighbor has an equal or higher heuristic value.

2. HILL CLIMBING WITH SIDEWAYS MOVE

- Allows a limited number of sideways move when the algorithm reaches a point where all the neighbouring states have a heuristic value equal to the current state.

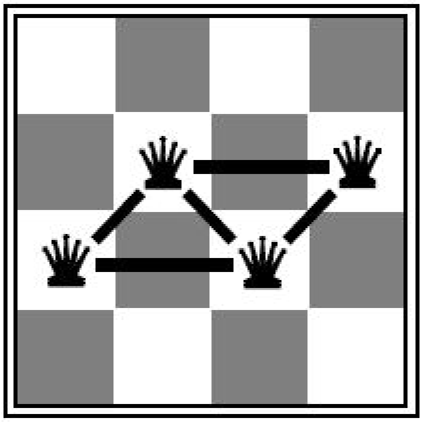
  Sideways move : Make a neighbouring state with value equal to current as the current state and apply hill climbing search on that.

3. RANDOM RESTART HILL CLIMBING

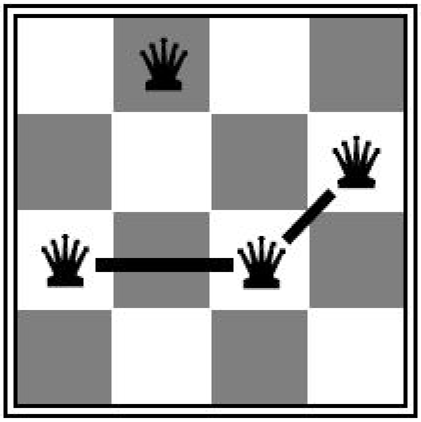
- Conducts a series of hill climbing searched until from random generated initial states, until goal is found.

HEURISTIC FUNCTION

*The heuristic cost function **h** is the number of pairs of queens that are attacking each other, either directly or indirectly.*



h = 5



h = 2

PROGRAM STRUCTURE

1. The user is prompted for an input. This number, say ‘n’ defines the dimension of the board (n x n) and of course, the no. of queens to be placed.
2. The create_initial_config(n) function creates a random initial configuration.
3. A node is created for the created for the configuration entailing the heuristic value(obtained by the function cal_heuristc(state)), the best successor and all the successors.
4. The hill climbing function is executed and returns a state.

   *The above steps run in a loop of 100 and thus return the following:
   - The average number of steps taken when its a success
   - The average number of steps taken when its a Failure
   - Success Rate
   - Failure Rate

GLOBAL VARIABLES

1. No_of_steps (int) : The number of steps taken to reach the global/local minima
2. Steps_when_success (list) : Maintains the number of steps taken for a success
3. Steps_when_failure (list) : Maintains the number of steps taken for a failure

FUNCTIONS

1. **create_intial_config(dim):** returns - a random initial configuration | argument - number of queens
2. **cal_heuristic(initial):** returns - heuristic value of a given state | argument - a state
3. **find_successors(initial):** returns - a list of all possible successors of a state | argument - a state
4. **best_neighbour(successors):** returns - best successor from a state | argument - list of successors from a state
5. **move_sideway(current):** returns - the adjacent node with the same heuristic value | argument - current node
6. **hill_climb(state):** returns - a goal state (global/local minima) | argument - an initial state

   ** works differently for different hill climbing searches*

SEARCH SEQUENCES FROM RANDOM INITIAL STATES

BASIC HILL CLIMBING:

CASE 1

```
Initial State Heuristic value: 8
[0, 0, 0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 1 Heuristic value: 8
[0, 0, 0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 2 Heuristic value: 4
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
```

```
Step : 3 Heuristic value: 3
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 4 Heuristic value: 2
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
```

CASE 2

```
Initial State Heuristic value: 7
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]

Step : 1 Heuristic value: 7
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
```

```
Step : 2 Heuristic value: 4
[0, 1, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
```

CASE 3

```
Initial State Heuristic value: 8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]

Step : 1 Heuristic value: 8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]

Step : 2 Heuristic value: 5
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
```

```
Step : 3 Heuristic value: 3
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]

Step : 4 Heuristic value: 2
[0, 0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
```

CASE 4

```
Initial State Heuristic value: 10
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]

Step : 1 Heuristic value: 10
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
```

```
Step : 2 Heuristic value: 6
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]

Step : 3 Heuristic value: 4
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]

Step : 4 Heuristic value: 3
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
```

HILL CLIMBING WITH SIDEWAYS MOVE

CASE 1

```
Initial State Heuristic value: 6
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

```
Step : 1 Heuristic value: 4
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 2 Heuristic value: 3
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 3 Heuristic value: 2
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 4 Heuristic value: 1
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
```

CASE 2

```
Initial State Heuristic value: 8

[0, 1, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 1 Heuristic value: 4
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 2 Heuristic value: 3
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Step : 3 Heuristic value: 2
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

```
Step : 4 Heuristic value: 0
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

CASE 3

```
Initial State Heuristic value: 7
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 1 Heuristic value: 4
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 2 Heuristic value: 2
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
```

```
Step : 3 Heuristic value: 1
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
```

CASE 4

```
Initial State Heuristic value: 6
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 1 Heuristic value: 2
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]

Step : 2 Heuristic value: 1
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
```

SOURCE CODES

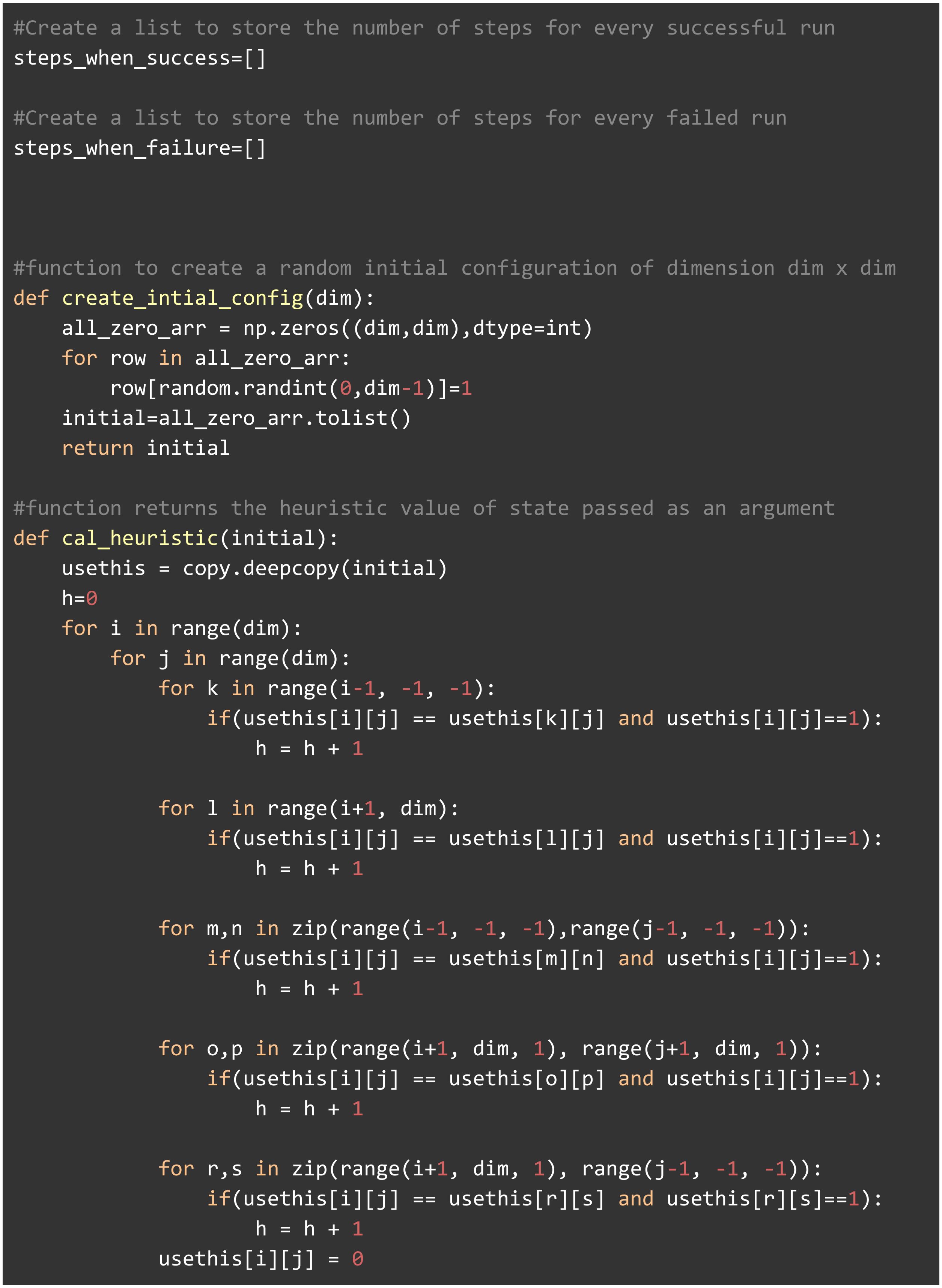
BASIC HILL CLIMBING

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 20 04:44:08 2019

@author: Akshay
"""

import numpy as np
import copy
import random


#Create a list to store the number of steps for every successful run
steps_when_success=[]

#Create a list to store the number of steps for every failed run
steps_when_failure=[]




#function to create a random initial configuration of dimension dim x dim
def create_intial_config(dim):
    all_zero_arr = np.zeros((dim,dim),dtype=int)
    for row in all_zero_arr:
        row[random.randint(0,dim-1)]=1
    initial=all_zero_arr.tolist()
    return initial

#function returns the heuristic value of state passed an  argument
def cal_heuristic(initial):
    usethis = copy.deepcopy(initial)
    h=0
    for i in range(dim):
        for j in range(dim):
            for k in range(i-1, -1, -1):
                if(usethis[i][j] == usethis[k][j] and usethis[i][j]==1):
                    h = h + 1

            for l in range(i+1, dim):
                if(usethis[i][j] == usethis[l][j] and usethis[i][j]==1):
                    h = h + 1

            for m,n in zip(range(i-1, -1, -1),range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[m][n] and usethis[i][j]==1):
                    h = h + 1

            for o,p in zip(range(i+1, dim, 1), range(j+1, dim, 1)):
                if(usethis[i][j] == usethis[o][p] and usethis[i][j]==1):
                    h = h + 1

            for r,s in zip(range(i+1, dim, 1), range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[r][s] and usethis[r][s]==1):
                    h = h + 1

            usethis[i][j] = 0
    return h

#function that returns a list of all successors of a state passed as an
argument
def find_successors(initial):
    num_dim = dim
    current = np.asarray(initial)
    childrennp = np.array([], dtype='int')

    for x,y in np.argwhere(current==1):
        temp = current.copy()
        temp[x,y]=0
        for k in range(y+1,dim):
            temp[x,k]=1
#            print(temp)
            childrennp = np.append(childrennp,temp)
            temp[x,k]=0
        for l in range(y-1,-1,-1):
            temp[x,l]=1
#            print(temp)
```

```
            childrennp = np.append(childrennp,temp)
            temp[x,l]=0

    childrennp = childrennp.reshape(-1, num_dim, num_dim).tolist()

    return childrennp

#this function returns the best successors from the list of successors
passed as the argument
def best_neighbour(successors):
    all_heuristics = []
    for n in successors:
        all_heuristics.append(cal_heuristic(n))
    best_h = min(all_heuristics)
    best_at = all_heuristics.index(best_h)
    return successors[best_at]

#this functions run the basic hill climbing algorithm with that state
passed to it as the initial state
def hill_climb(initial):
    global no_of_steps,steps_when_failure,steps_when_success
    current = initial.copy()
    successors = find_successors(current)
    best_successor=best_neighbour(successors)


    if cal_heuristic(current) <= cal_heuristic(best_successor):
        if cal_heuristic(current) == 0:
            steps_when_success.append(no_of_steps)
        else:
            steps_when_failure.append(no_of_steps)
        return current
    else:
        no_of_steps=no_of_steps+1
        current = best_successor
        hill_climb(current)

#taking input from the user for the number of queens
try:
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)
except:
    print("Please enter an integer value")
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)


print("Loading ....")
#runs the basic climbing 200 times
for iterations in range(201):
    no_of_steps=0
    initial = create_intial_config(dim)
    hill_climb(initial)


print("       ")
print("The Average of Steps for success", np.average(steps_when_success))
print("Success Rate",
100*(len(steps_when_success)/(len(steps_when_success)+len(steps_when_failur
e))),"%")
print("       ")
print("The Average of Steps for a failure",np.average(steps_when_failure))
print("Failure Rate",
100*(len(steps_when_failure)/(len(steps_when_success)+len(steps_when_failur
e))),"%")
```

HILL CLIMBING WITH SIDEWAYS MOVE

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 24 23:52:35 2019

@author: Akshay
"""


import numpy as np
import copy
import random
```

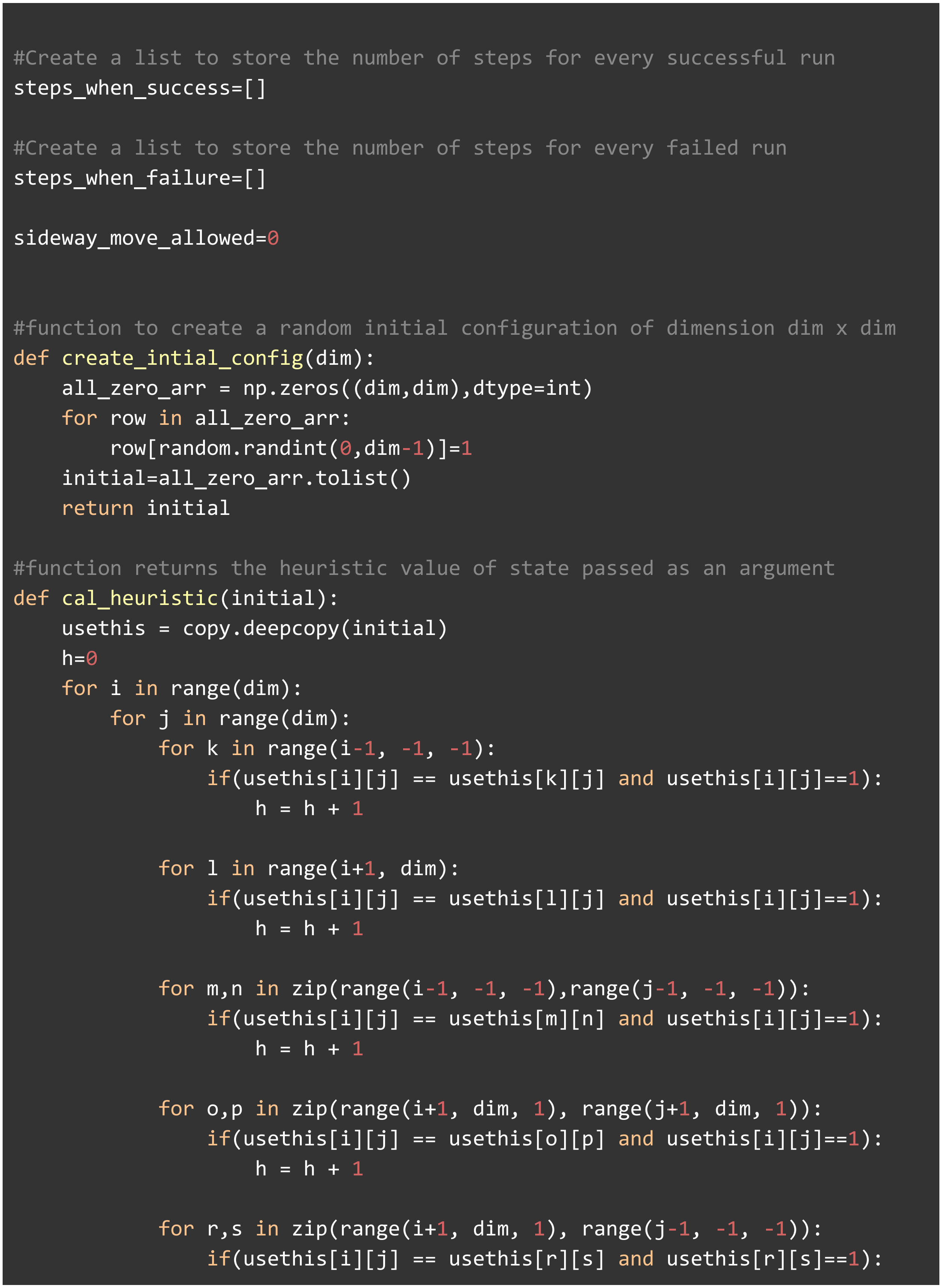


```python
#Create a list to store the number of steps for every successful run
steps_when_success=[]

#Create a list to store the number of steps for every failed run
steps_when_failure=[]



#function to create a random initial configuration of dimension dim x dim
def create_intial_config(dim):
    all_zero_arr = np.zeros((dim,dim),dtype=int)
    for row in all_zero_arr:
        row[random.randint(0,dim-1)]=1
    initial=all_zero_arr.tolist()
    return initial

#function returns the heuristic value of state passed as an argument
def cal_heuristic(initial):
    usethis = copy.deepcopy(initial)
    h=0
    for i in range(dim):
        for j in range(dim):
            for k in range(i-1, -1, -1):
                if(usethis[i][j] == usethis[k][j] and usethis[i][j]==1):
                    h = h + 1

            for l in range(i+1, dim):
                if(usethis[i][j] == usethis[l][j] and usethis[i][j]==1):
                    h = h + 1

            for m,n in zip(range(i-1, -1, -1),range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[m][n] and usethis[i][j]==1):
                    h = h + 1

            for o,p in zip(range(i+1, dim, 1), range(j+1, dim, 1)):
                if(usethis[i][j] == usethis[o][p] and usethis[i][j]==1):
                    h = h + 1

            for r,s in zip(range(i+1, dim, 1), range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[r][s] and usethis[r][s]==1):
                    h = h + 1
            usethis[i][j] = 0
    return h

#function that returns a list of all successors of a state passed as an argument
def find_successors(initial):
    num_dim = dim
    current = np.asarray(initial)
    childrennp = np.array([], dtype='int')

    for x,y in np.argwhere(current==1):
        temp = current.copy()
        temp[x,y]=0
        for k in range(y+1,dim):
            temp[x,k]=1
            childrennp = np.append(childrennp,temp)
            temp[x,k]=0
        for l in range(y-1,-1,-1):
            temp[x,l]=1
            childrennp = np.append(childrennp,temp)
            temp[x,l]=0

    childrennp = childrennp.reshape(-1, num_dim, num_dim).tolist()

    return childrennp

#this function returns the best successors from the list of successors passed as the argument
def best_neighbour(successors):
    all_heuristics = []
    for n in successors:
        all_heuristics.append(cal_heuristic(n))
    best_h = min(all_heuristics)
    best_at = all_heuristics.index(best_h)
    return successors[best_at]

#function to make a sideways move when the algorithm hits a shoulder
def move_sideway(current):
    global no_of_steps
    for i in range(100):
        current = Node(current.best_successor)
        no_of_steps=no_of_steps + 1
        if current.heuristic < cal_heuristic(current.best_successor):
```

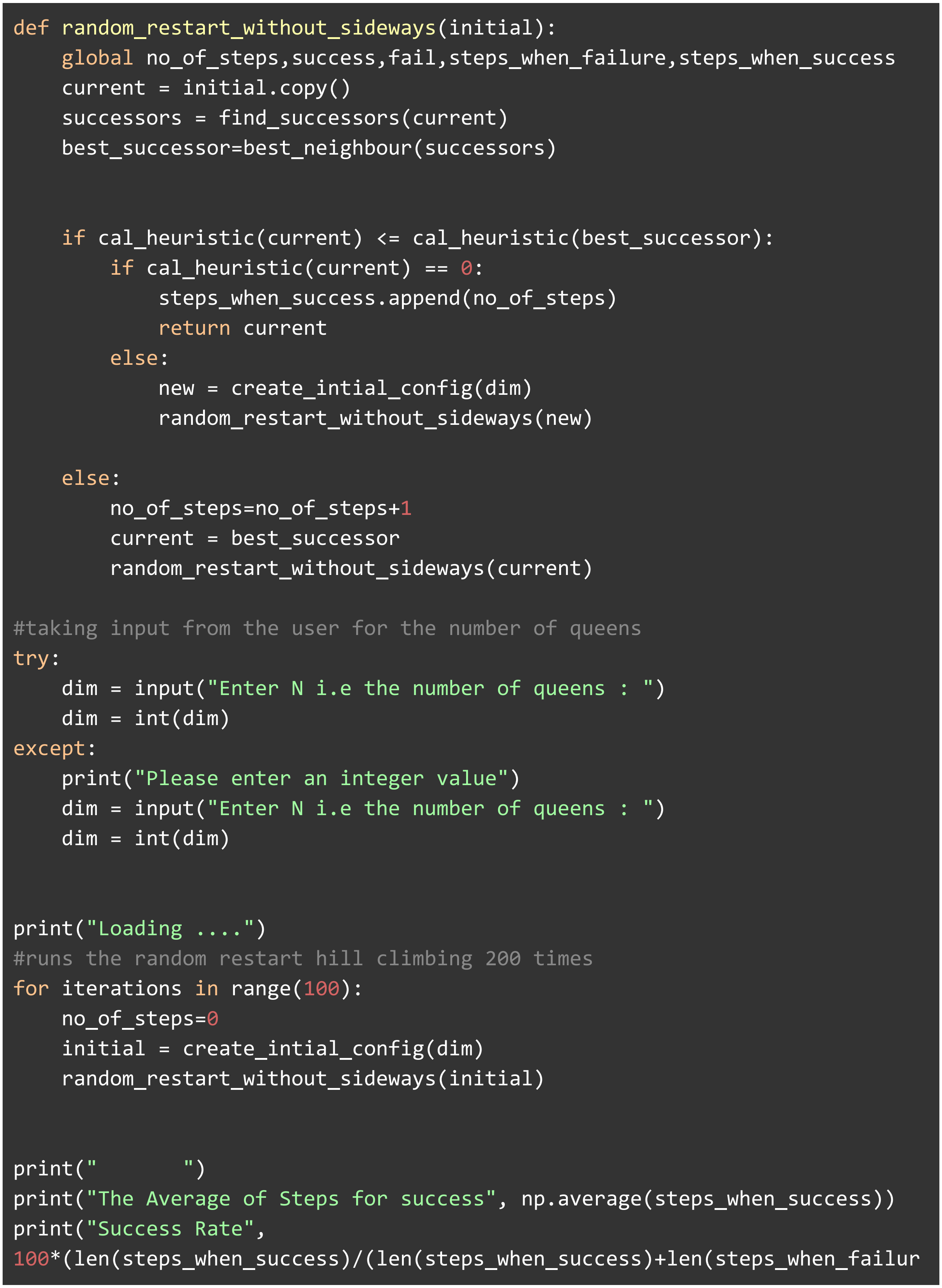
```python
            return current

#class definition for a node
class Node:
    def __init__(self,state):
        self.state = state
        self.heuristic = cal_heuristic(state)
        self.successors = find_successors(state)
        self.best_successor = best_neighbour(self.successors)


#this functions run the hill climbing algorithm with sideways move allowed with that state passed to it as the initial state
def hill_climb(state):
    global no_of_steps
    current = Node(state)

    if current.heuristic < cal_heuristic(current.best_successor):
        if current.heuristic == 0:
            steps_when_success.append(no_of_steps)

        else:
            steps_when_failure.append(no_of_steps)
        return current.state
    if current.heuristic == cal_heuristic(current.best_successor):
        try:
            current = move_sideway(current)
            hill_climb(current.state)
        except:
            steps_when_failure.append(no_of_steps)

    else:
        current = Node(current.best_successor)
        no_of_steps = no_of_steps + 1
        hill_climb(current.state)


#taking input from the user for the number of queens
try:
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)
except:
    print("Please enter an integer value")
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)

print("Loading ....")
#runs the basic climbing 200 times
for iterations in range(150):
    no_of_steps=0
    initial = create_intial_config(dim)
    current = initial.copy()
    hill_climb(current)

print("       ")
print("The Average of Steps for success", np.average(steps_when_success))
print("Success Rate",
100*(len(steps_when_success)/(len(steps_when_success)+len(steps_when_failure))),"%")
print("       ")
print("The Average of Steps for a failure",np.average(steps_when_failure))
print("Failure Rate",
100*(len(steps_when_failure)/(len(steps_when_success)+len(steps_when_failure))),"%")
```

RANDOM-RESTART WITHOUT SIDEWAYS MOVE

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 20 04:44:08 2019

@author: Akshay
"""


import numpy as np
import copy
import random
```



```
#Create a list to store the number of steps for every successful run
steps_when_success=[]

#Create a list to store the number of steps for every failed run
steps_when_failure=[]

sideway_move_allowed=0


#function to create a random initial configuration of dimension dim x dim
def create_intial_config(dim):
    all_zero_arr = np.zeros((dim,dim),dtype=int)
    for row in all_zero_arr:
        row[random.randint(0,dim-1)]=1
    initial=all_zero_arr.tolist()
    return initial

#function returns the heuristic value of state passed as an argument
def cal_heuristic(initial):
    usethis = copy.deepcopy(initial)
    h=0
    for i in range(dim):
        for j in range(dim):
            for k in range(i-1, -1, -1):
                if(usethis[i][j] == usethis[k][j] and usethis[i][j]==1):
                    h = h + 1

            for l in range(i+1, dim):
                if(usethis[i][j] == usethis[l][j] and usethis[i][j]==1):
                    h = h + 1

            for m,n in zip(range(i-1, -1, -1),range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[m][n] and usethis[i][j]==1):
                    h = h + 1

            for o,p in zip(range(i+1, dim, 1), range(j+1, dim, 1)):
                if(usethis[i][j] == usethis[o][p] and usethis[i][j]==1):
                    h = h + 1

            for r,s in zip(range(i+1, dim, 1), range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[r][s] and usethis[r][s]==1):
                    h = h + 1

            usethis[i][j] = 0
    return h

#function that returns a list of all successors of a state passed as an
argument
def find_successors(initial):
    num_dim = dim
    current = np.asarray(initial)
    childrennp = np.array([], dtype='int')

    for x,y in np.argwhere(current==1):
        temp = current.copy()
        temp[x,y]=0
        for k in range(y+1,dim):
            temp[x,k]=1
#             print(temp)
            childrennp = np.append(childrennp,temp)
            temp[x,k]=0
        for l in range(y-1,-1,-1):
            temp[x,l]=1
#             print(temp)
            childrennp = np.append(childrennp,temp)
            temp[x,l]=0

    childrennp = childrennp.reshape(-1, num_dim, num_dim).tolist()

    return childrennp

#this function returns the best successors from the list of successors
passed as the argument
def best_neighbour(successors):
    all_heuristics = []
    for n in successors:
        all_heuristics.append(cal_heuristic(n))
    best_h = min(all_heuristics)
    best_at = all_heuristics.index(best_h)
    return successors[best_at]

#this functions run the random restart hill climbing algorithm with that
state passed to it as the initial state
```



```
def random_restart_without_sideways(initial):
    global no_of_steps,success,fail,steps_when_failure,steps_when_success
    current = initial.copy()
    successors = find_successors(current)
    best_successor=best_neighbour(successors)


    if cal_heuristic(current) <= cal_heuristic(best_successor):
        if cal_heuristic(current) == 0:
            steps_when_success.append(no_of_steps)
            return current
        else:
            new = create_intial_config(dim)
            random_restart_without_sideways(new)

    else:
        no_of_steps=no_of_steps+1
        current = best_successor
        random_restart_without_sideways(current)

#taking input from the user for the number of queens
try:
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)
except:
    print("Please enter an integer value")
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)


print("Loading ....")
#runs the random restart hill climbing 200 times
for iterations in range(100):
    no_of_steps=0
    initial = create_intial_config(dim)
    random_restart_without_sideways(initial)


print("       ")
print("The Average of Steps for success", np.average(steps_when_success))
print("Success Rate",
100*(len(steps_when_success)/(len(steps_when_success)+len(steps_when_failur
e))),"%")
```

RANDOM RESTART W/SIDE-WAYS MOVE

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 24 23:52:35 2019

@author: Akshay
"""

import numpy as np
import copy
import random

#Create a list to store the number of steps for every successful run
steps_when_success=[]

#Create a list to store the number of steps for every failed run
steps_when_failure=[]



#function to create a random initial configuration of dimension dim x dim
def create_intial_config(dim):
    all_zero_arr = np.zeros((dim,dim),dtype=int)
    for row in all_zero_arr:
        row[random.randint(0,dim-1)]=1
    initial=all_zero_arr.tolist()
    return initial

#function returns the heuristic value of state passed an a argument
def cal_heuristic(initial):
    usethis = copy.deepcopy(initial)
    h=0
    for i in range(dim):
        for j in range(dim):
            for k in range(i-1, -1, -1):
                if(usethis[i][j] == usethis[k][j] and usethis[i][j]==1):
                    h = h + 1

            for l in range(i+1, dim):
                if(usethis[i][j] == usethis[l][j] and usethis[i][j]==1):
                    h = h + 1

            for m,n in zip(range(i-1, -1, -1),range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[m][n] and usethis[i][j]==1):
                    h = h + 1

            for o,p in zip(range(i+1, dim, 1), range(j+1, dim, 1)):
                if(usethis[i][j] == usethis[o][p] and usethis[i][j]==1):
                    h = h + 1

            for r,s in zip(range(i+1, dim, 1), range(j-1, -1, -1)):
                if(usethis[i][j] == usethis[r][s] and usethis[r][s]==1):
                    h = h + 1

            usethis[i][j] = 0
    return h

#function that returns a list of all successors of a state passed as an
argument
def find_successors(initial):
    num_dim = dim
    current = np.asarray(initial)
    childrennp = np.array([], dtype='int')

    for x,y in np.argwhere(current==1):
        temp = current.copy()
        temp[x,y]=0
        for k in range(y+1,dim):
            temp[x,k]=1
            childrennp = np.append(childrennp,temp)
            temp[x,k]=0
        for l in range(y-1,-1,-1):
            temp[x,l]=1
            childrennp = np.append(childrennp,temp)
            temp[x,l]=0

    childrennp = childrennp.reshape(-1, num_dim, num_dim).tolist()
    return childrennp

#this function returns the best successors from the list of successors
passed as the argument
def best_neighbour(successors):
    all_heuristics = []
    for n in successors:
        all_heuristics.append(cal_heuristic(n))
    best_h = min(all_heuristics)
    best_at = all_heuristics.index(best_h)
    return successors[best_at]

#function to make a sideways move when the algorithm hits a shoulder
def move_sideway(current):
    global no_of_steps
    for i in range(100):
        current = Node(current.best_successor)
        no_of_steps=no_of_steps+1
        if current.heuristic < cal_heuristic(current.best_successor):
            return current

#Class definition for a node
class Node:
    def __init__(self,state):
        self.state = state
        self.heuristic = cal_heuristic(state)
        self.successors = find_successors(state)
        self.best_successor = best_neighbour(self.successors)


 #this functions run the random restart hill climbing algorithm with
sideways move allowed with that state passed to it as the initial state
def hill_climb(state):
    global no_of_steps
    current = Node(state)

    if current.heuristic < cal_heuristic(current.best_successor):
        if current.heuristic == 0:
            steps_when_success.append(no_of_steps)

        else:
            new = create_intial_config(dim)
            hill_climb(new)
        return current.state
    if current.heuristic == cal_heuristic(current.best_successor):
        try:
            current = move_sideway(current)
            hill_climb(current.state)
        except:
            new = create_intial_config(dim)
            hill_climb(new)

    else:
        current = Node(current.best_successor)
        no_of_steps =+ 1
        hill_climb(current.state)


#taking input from the user for the number of queens
try:
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)
except:
    print("Please enter an integer value")
    dim = input("Enter N i.e the number of queens : ")
    dim = int(dim)

print("Loading ....")
#runs the random restart hill climbing with sideways move allowed 200 times
for iterations in range(50):
    no_of_steps=0
    initial = create_intial_config(dim)
    current = initial.copy()
    hill_climb(current)

print("       ")
print("The Average of Steps for success", np.average(steps_when_success))
print("Success Rate",
100*(len(steps_when_success)/(len(steps_when_success)+len(steps_when_failur
e))),"%")
print("       ")
```