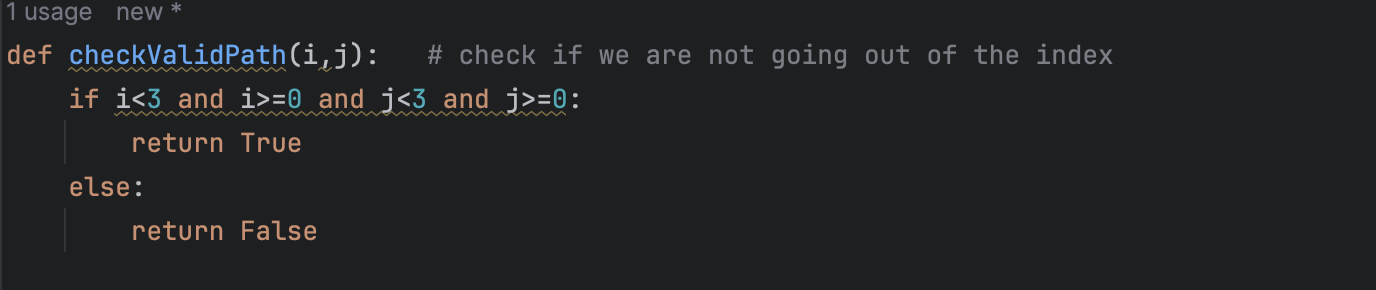
Hill Climbing: -

Presented by: - Ashutosh Samal

(2403RES15)

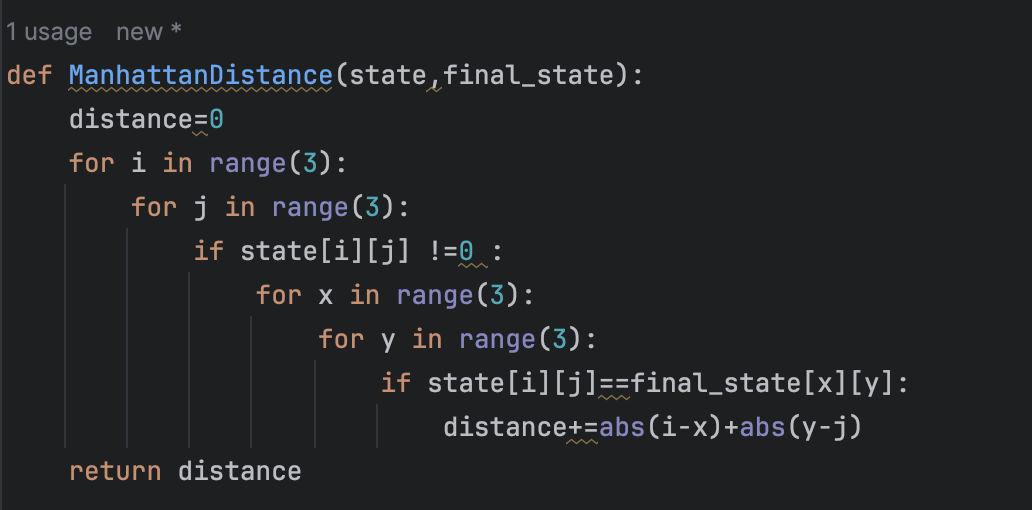
User defined Functions: -

1-



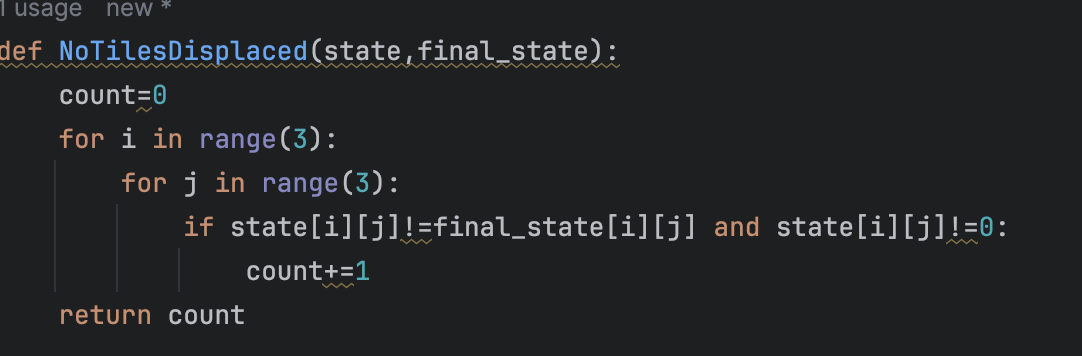
Will check if we are not getting out of bound.

2-



It will calculate Manhattan distance between given state and final\_state

3-



This will calculate number of tiles misplaced with respect to the given final\_state.

4-A screen shot of a computer program

Description automatically generated

This is generic objective function which will return the value as per the type give to the function.

1 – Manhattan distance

2- Number of tiles misplaced

5-

A computer screen shot of white text

Description automatically generated

This function takes a state, possible moves, cost\_type(1 or 2) , visited state list and the final\_state.

It returns the state which can directly be reachable from current state and not visited yet, along with their cost.

6-

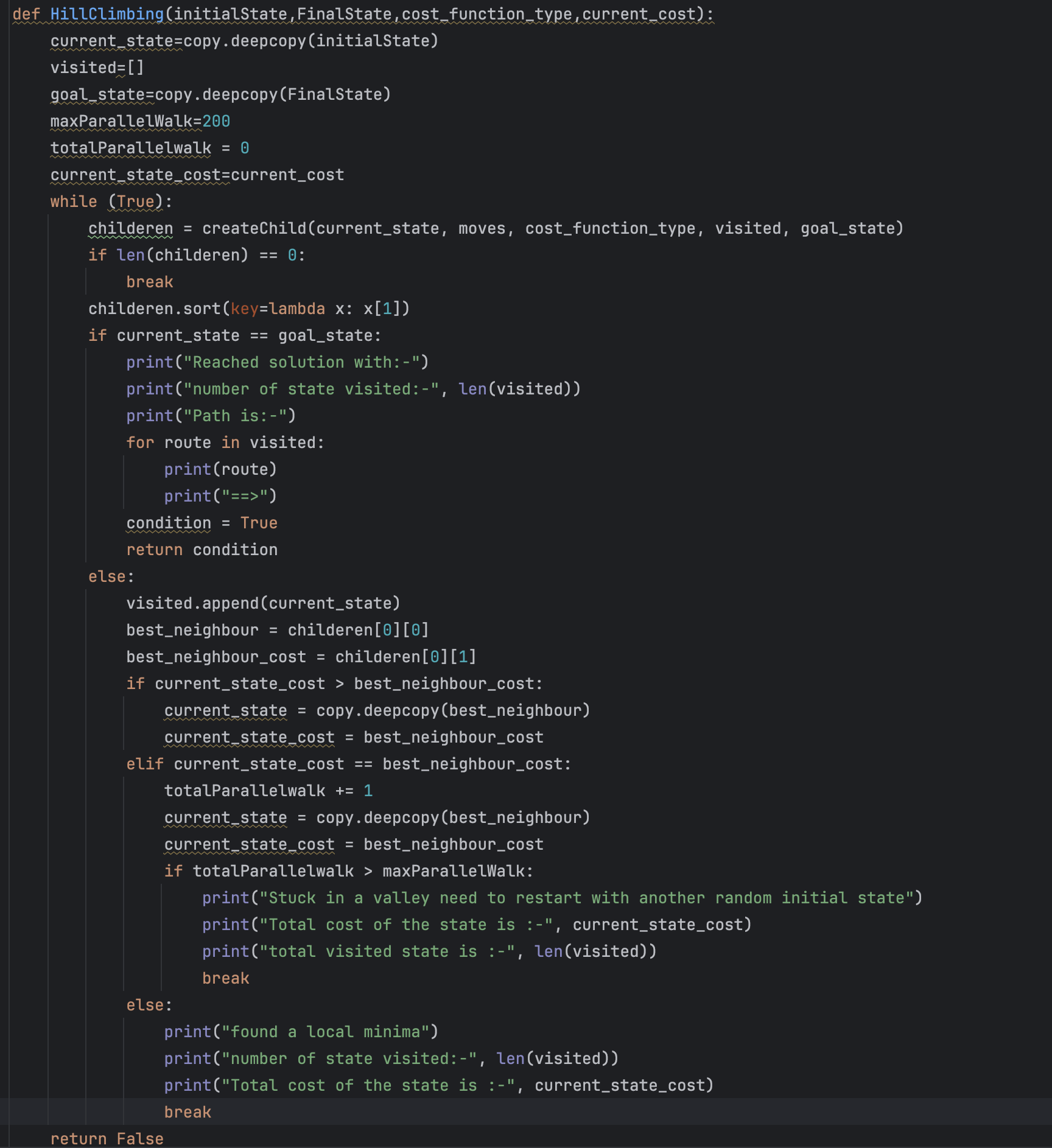
A screenshot of a computer

Description automatically generated

This 2 are helper functions.

1. Get the location of the blank space.
2. It randomizes the given state. used to generate random state from given state.

Main Hill climbing Code: -



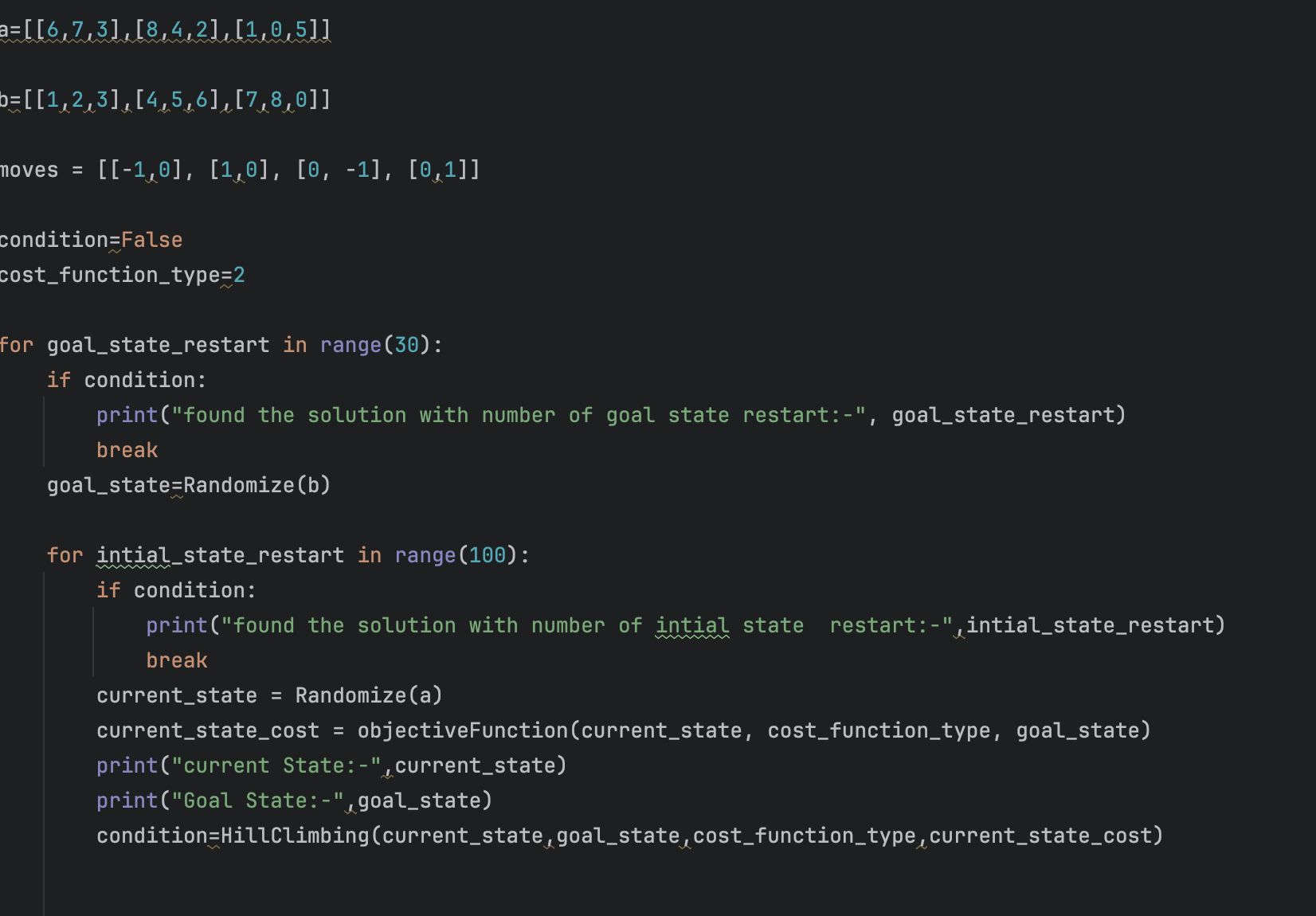
This function takes a random initial state, a random final state, cost\_funtion\_type (Manhattan or number of tiles replaced) and current state cost.

1. First get the child nodes(neighbors) and their cost using the function createChild()
2. Sort it based on the cost, so the 1st element is the child with minimum cost
3. Now check if the current state is goal state, if yes then print the path and return True.
4. If not, then check: -

* If the current cost > best neighbor cost, then update the current state with this neighbor (Steepest ascent)
* If they are equal then it might be valley or shoulder, I have limited parallel/vertical walk to 200 so that we can get out of shoulder and stop if it’s a valley. If number of parallel walks exceeds then exit
* If current cost< best neighbor cost, then it’s a local minima so exit.

1. Repeat this until a break statement is not triggered and return False in the end which means we didn’t find any solution.

Now I am running this function with random initial state and random final state like this:



Initialized moves list for possible moves, 2 states a and b which was given in the assignment.

Condition is set to false which a flag to see if we have found a solution or not.

And finally, a cost function type (1 or 2)

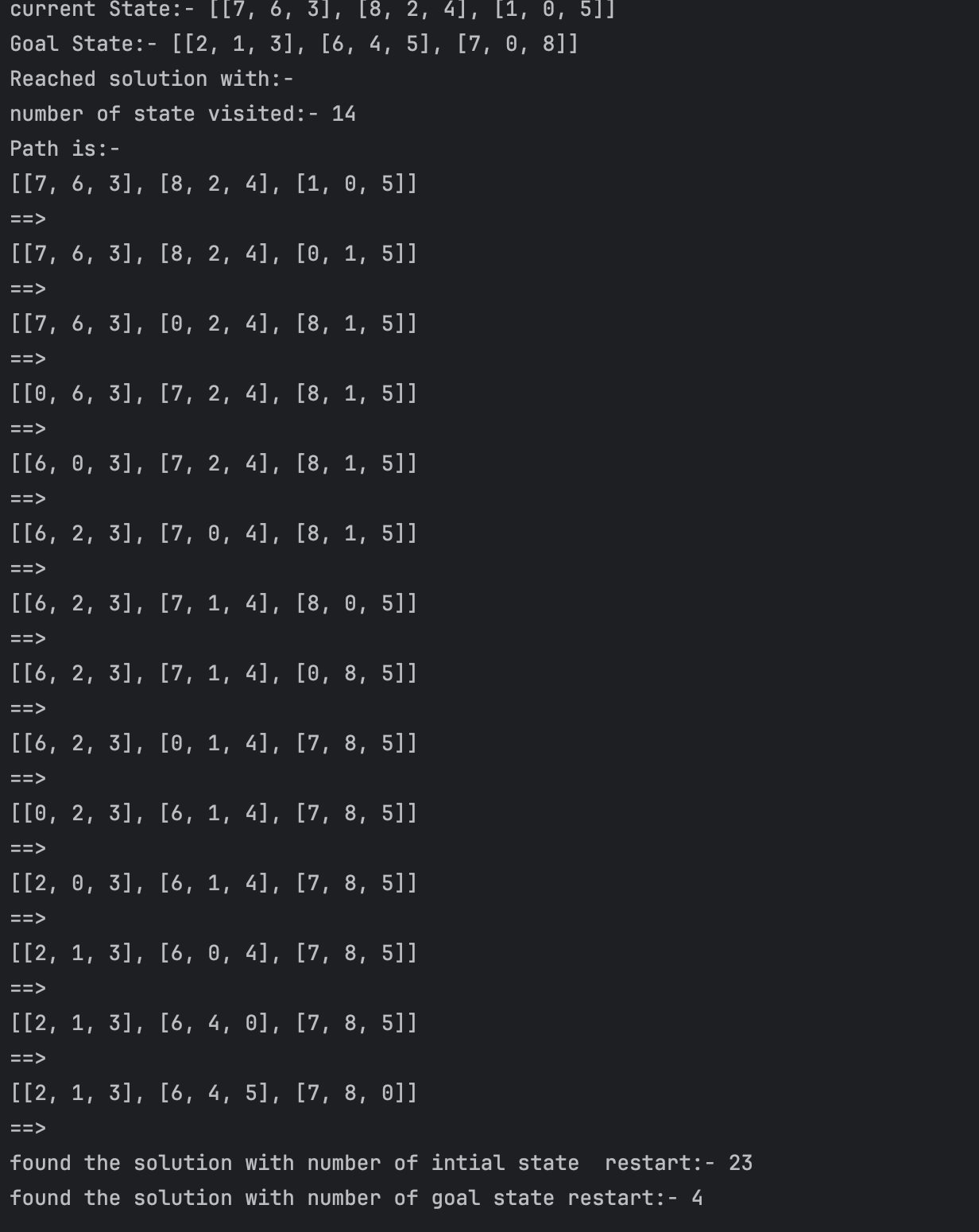
1st loop runs only 30 times ( we can obviously play around this ) and it randomize the goal state and for each goal state in 2nd loop I randomize initial state 100 times .

Inside the 2nd loop we call the HillClimbing function to compute HillClimbing for that initial state and final state and get out of the loop as soon as the condition becomes true which is the output of the Hill Climbing function.

It Prints for which random goal and final state we got the solution.

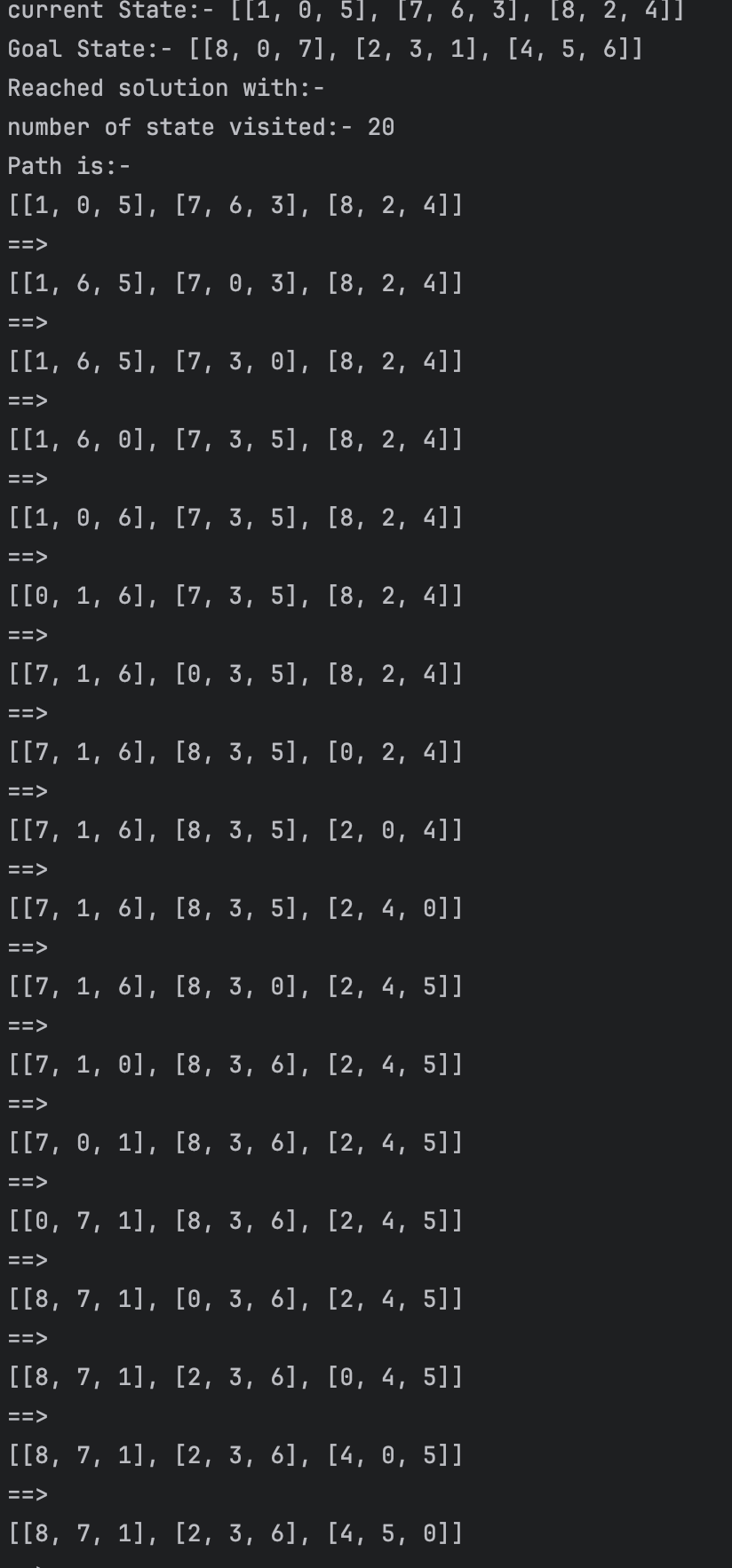
Sample Output is: -

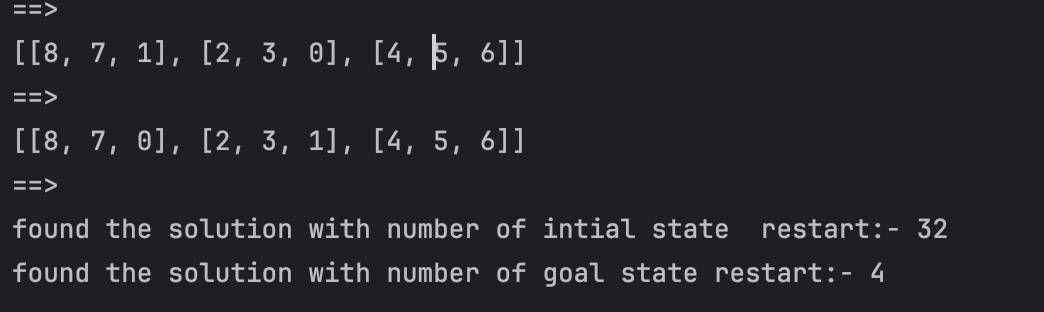
For Manhattan distance (type=2 in my code)



We are able to get solution after restarting goal state 4 times and initial state 23 times for that particular goal state loop.

For number of tiles misplaced (Type=1 in my code): -





In this also we got the solution.

As we are randomizing states, we will get different result in each run. But I observe most of the time Manhattan distance performed better but not by much.

Observation: -

Here choosing the best neighbor and completely discarding other node might result in not converging in few cases and sometime stucking in local minima as we are greedily doing descent.

Instead, we can randomly select any of the neighbor to visit but with a probability inversely proportional to the cost. In this way we are not discarding any bad neighbor and selecting them with less probability.

Randomness and probability are very powerful when it comes to searching as we search randomly in the neighborhood but after assigning a probability to the states so that bad states are less likely be selected but not completely discarded.