# PRACTICAL NO.-9: Write a program for Hill climbing problem.

**Hill climbing algorithm**

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. A node of hill climbing algorithm has two components which are state and value. Hill Climbing is mostly used when a good heuristic is available. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

**Algorithm with an example trace**

1. **Algorithm**

Step 1: Evaluate the initial state, if it is goal state then return success and Stop.

Step 2: Loop Until a solution is found or there is no new operator left to apply.

Step 3: Select and apply an operator to the current state.

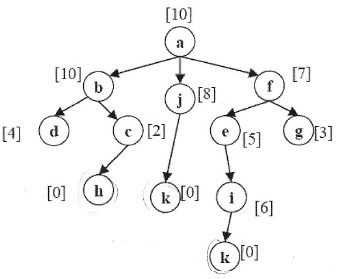
Step 4: Check new state:

* 1. If it is goal state, then return success and quit.
  2. Else if it is better than the current state then assign new state as a current state.
  3. Else if not better than the current state, then return to step2.

Step 5: Exit.

**Question: Use Hill Climbing Search to place the initial sate given below to final state for following tree.**

**Tree:**



**Code:**

sussList = {} # This dictionary holds all the nodes with their successors and their corresponding heuristic value

temp\_key\_list = [] # Holds key node whose successor is to be inputted

initial\_node = str(input("Input initial node: ")) # root node

initial\_value = eval(input(f"Input {initial\_node}'s heuristic value: ")) # holds heuristic value of root node

numberNode = eval(input(f"How many successor nodes in node '{initial\_node}': ")) # number of successor of root node

temp\_key\_list.append(initial\_node)

def nodeInput(numberNode): # Function used to input all nodes with their successor and corresponding heuristic value

new\_node = temp\_key\_list[0]

temp\_key\_list.pop(0)

new\_list = []

for i in range(numberNode):

key\_name = str(input(f"Enter {i+1}'th successor of {new\_node}: "))

key\_value = eval(input(f"Enter {key\_name}'s heuristic value: "))

temp = [key\_name,key\_value]

new\_list.append(temp)

sussList[new\_node] = new\_list

temp\_key\_list.append(key\_name)

if len(temp\_key\_list) != 0:

new\_node = temp\_key\_list[0]

new\_numberNode = eval(input(f"How many successor nodes in node {new\_node}?: "))

nodeInput(new\_numberNode)

else:

pass

def sortList(new\_list): #Function to sort the selected list in ascending order

new\_list.sort(key = lambda x: x[1])

return new\_list

def hillClimbing\_search(node,value): #Function to find shortest path using heuristic value

new\_list = list()

if node in sussList.keys():

new\_list = sussList[node]

new\_list = sortList(new\_list)

if (value > new\_list[0][1]):

value = new\_list[0][1]

node = new\_list[0][0]

hillClimbing\_search(node, value)

if (value < new\_list[0][1]):

print(f"ANSWER:\nFor given Data, the local maxima is at node '{node}' with heuristic value {value}")

else:

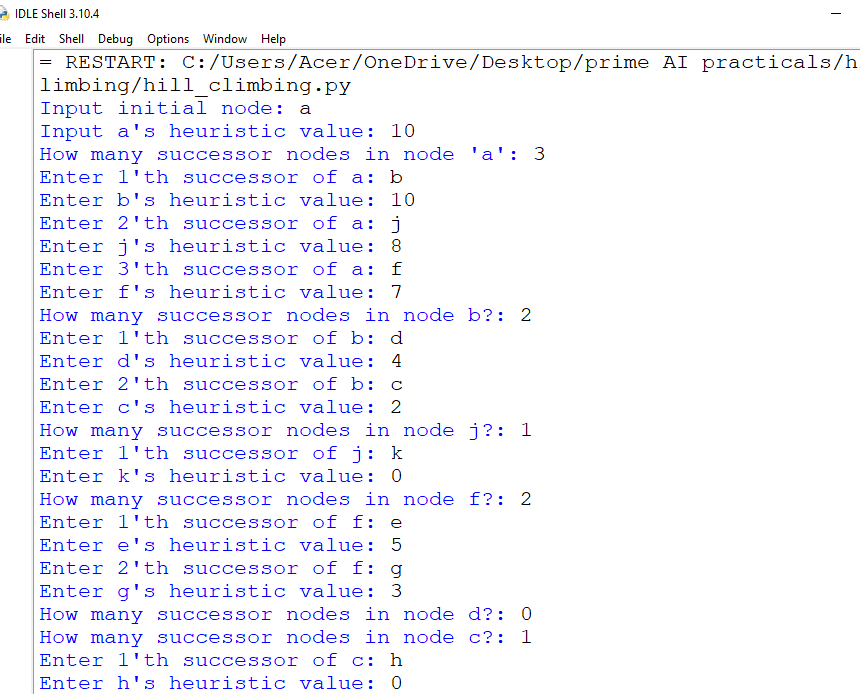
print(f"ANSWER:\nFor given Data, the local maxima is at node '{node}' with heuristic value {value}")

nodeInput(numberNode)

print("The user input is as follows: \n", sussList)

hillClimbing\_search(initial\_node, initial\_value)

**Output**

****

