**Developing a system for First-Choice Hill Climbing Local Search Algorithm**

We have taken 8-queens problem for demonstrating the working of First-choice Hill Climbing search strategy. A state is represented as an eight-digit positive integer between 1 to 8. We generate all 56 successors of a current state, and choose the first state which has a heuristic value better than the current state. The process is continued until a state with a specified value is found. Here is a possible outcome of a typical implementation of the algorithm:

For the initial state 67327364 and threshold value 27 after 4 iteration a solution was found in the following form:

Iteration: 1 Selected First Uphill State: [1, 7, 3, 2, 7, 3, 6, 4] Value: 22

Iteration: 2 Selected First Uphill State: [1, 5, 3, 2, 7, 3, 6, 4] Value: 24

Iteration: 3 Selected First Uphill State: [1, 5, 1, 2, 7, 3, 6, 4] Value: 25

Iteration: 4 Selected First Uphill State: [1, 5, 7, 2, 7, 3, 6, 4] Value: 26

Found! Id: 22 Type: s State: [1, 5, 8, 2, 7, 3, 6, 4] Value: 27

And the states are as follows:

[1, 'c', [1, 5, 7, 2, 7, 3, 6, 4], 26]

[2, 's', [2, 5, 7, 2, 7, 3, 6, 4], 25]

…

[22, 's', [1, 5, 8, 2, 7, 3, 6, 4], 27]

[23, 's', [1, 5, 7, 1, 7, 3, 6, 4], 24]

…

[56, 's', [1, 5, 7, 2, 7, 3, 6, 7], 24]

[57, 's', [1, 5, 7, 2, 7, 3, 6, 8], 26]

Analyzing each iteration, we can see an uphill successor is chosen each time which has better fitness function value than the previous state. However, the system may get stuck up frequently at local maxima for a large threshold value.

**The demonstrated Python code of First-Choice Hill Climbing Algorithm is as below:**

states = []

iteration = 0

def evalState(lst):

count = 0

#horizontal check

for i in range( len(lst) ):

for j in range(i+1,len(lst),1):

if (lst[i] == lst[j]):

count = count + 1

#diagonal\_up check

for i in range( len(lst) ):

x = lst[i]

for j in range(i+1,len(lst),1):

x = x + 1

if (x == lst[j]):

count = count + 1

#diagonal\_down check

for i in range( len(lst) ):

x = lst[i]

for j in range(i+1,len(lst),1):

x = x - 1

if (x == lst[j]):

count = count + 1

return count

def eval(L):

return 28 - evalState(L)

def genarate\_successor(L, threshold):

global states

global iteration

states = []

iteration += 1

isFound = False

stuck = True

hval = eval(L)

max\_val = hval

current\_state = [1,'c',L,hval]

states.append( current\_state )

first\_uphill = []

if hval >= threshold:

isFound = True

checkall(stuck,y,threshold)

if isFound == False:

count = 2

for i in range(1,9):

for j in range(1,9):

if(L[i-1] != j):

x = L[:]

x[i-1] = j

hval = eval(x)

successor\_state = [count,'s',x,hval]

states.append( successor\_state )

if(max\_val < hval and stuck == True):

first\_uphill = successor\_state

stuck = False

count += 1

checkall(stuck , first\_uphill , threshold )

return states

def checkall(stuck , uphill , threshold ):

if stuck == False :

if (uphill[3] >= threshold) :

print('\nFound! Id: '+str(uphill[0])+' Type: '+str(uphill[1])+' State: '+str(uphill[2])+' Value: '+str(uphill[3]) )

return states

else:

print('Iteration: '+str(iteration)+' Selected First Uphill State: '+str(uphill[2])+' Value: '+str(uphill[3]) )

genarate\_successor( uphill[2] , threshold)

else:

print('Stuckup!\n ')

def clear\_database():

global states

states = []

def execute\_hcls():

global iteration

state = input('Enter a state: ')

threshold = int(input('enter threshold value: '))

L = [ int(x) for x in str(state)]

iteration = 0

return genarate\_successor(L , threshold)

def display\_states(states):

for j in range(len(states)):

print(states[j])

def save\_states(states):

f1=open("output.txt", "w")

for j in range(len(states)):

print(states[j],file = f1)

f1.close

#Main

case = 1

states = []

while(case >= 1 and case < 5):

print('\n1. Clear database')

print('2. Execute hcls')

print('3. Display states')

print('4. Save states')

print('5. Exit')

case = int(input('\n\nEnter your choice: '))

if(case == 1):

clear\_database()

elif(case == 2):

states = execute\_hcls()

elif(case == 3):

display\_states( states )

elif(case == 4):

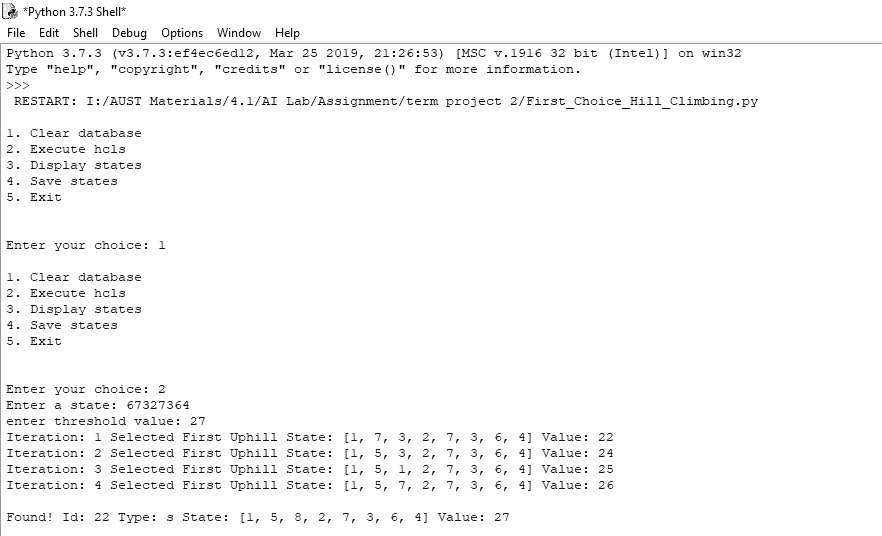
save\_states(states)

else:

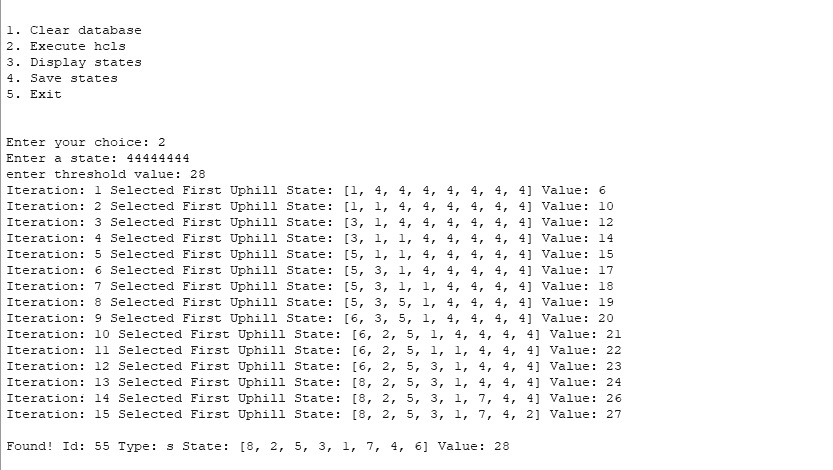
break

**A sample input and output is as below:**

1. Input and output for test case 1:

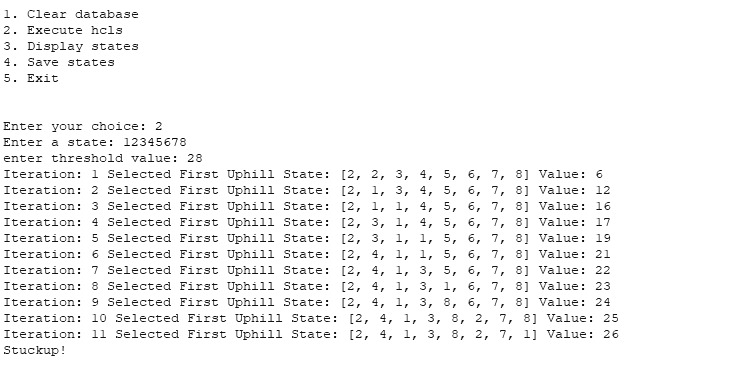


1. Input and output for test case 2:

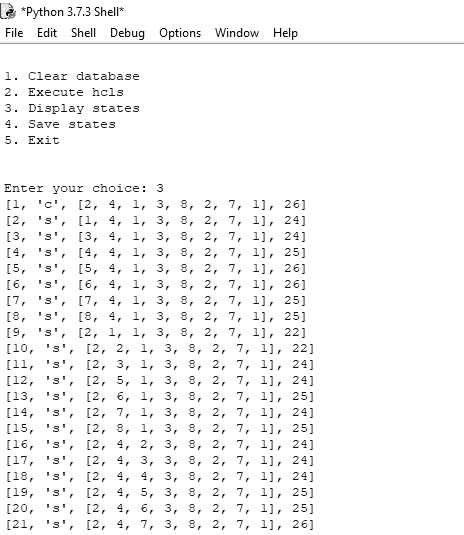


1. Input and output for test case 3:

(In this test case the system got stuck up at local maxima)



1. Displaying states of test case 3:



1. Displaying states of test case 3 in output.txt file:

