**Terminal Lab**

**COMSATS University Islamabad**

Sahiwal Campus



***Usama Sarwar***

FA17-BS(CS)-090-B

***Mr. Waqar***

Artificial Intelligence

January 15, 2021

Table of Contents

[1. Code 1](#_Toc61602253)

[1.1 Output 1](#_Toc61602254)

[2. Code 3](#_Toc61602255)

[2.1 Output 5](#_Toc61602256)

[3. Code 6](#_Toc61602257)

[3.1 Output 6](#_Toc61602258)

[4. Code 6](#_Toc61602259)

[4.1 Output 7](#_Toc61602260)

[5. Code 8](#_Toc61602261)

[5.1 Output **Error! Bookmark not defined.**](#_Toc61602262)

# Code

class AStarGraph(object):

    # Define a class board like grid with two barriers

    def init(self):

        self.barriers = []

        t=int(input("Enter number of boxes you want to include in barrier"))

        for i in range(t):

            x=int(input("Enter barrier x-axis"))

            y=int(input("Enter barrier y-axis"))

            self.barriers.append([x,y])

    def heuristic(self, start, goal):

        # Use Chebyshev distance heuristic if we can move one square either

        # adjacent or diagonal

        D = 1

        D2 = 1

        dx = abs(start[0] - goal[0])

        dy = abs(start[1] - goal[1])

        return D \* (dx + dy) + (D2 - 2 \* D) \* min(dx, dy)

    def get\_vertex\_neighbours(self, pos):

        n = []

        # Moves allow link a chess king

        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (-1, 1), (1, -1), (-1, -1)]:

            x2 = pos[0] + dx

            y2 = pos[1] + dy

            if x2 < 0 or x2 > 7 or y2 < 0 or y2 > 7:

                continue

            n.append((x2, y2))

        return n

    def move\_cost(self, a, b):

        for barrier in self.barriers:

            if b in barrier:

                return 100  # Extremely high cost to enter barrier squares

        return 1  # Normal movement cost

def AStarSearch(start, end, graph):

    G = {}  # Actual movement cost to each position from the start position

    F = {}  # Estimated movement cost of start to end going via this position

    # Initialize starting values

    G[start] = 0

    F[start] = graph.heuristic(start, end)

    closedVertices = set()

    openVertices = set([start])

    cameFrom = {}

    while len(openVertices) > 0:

        # Get the vertex in the open list with the lowest F score

        current = None

        currentFscore = None

        for pos in openVertices:

            if current is None or F[pos] < currentFscore:

                currentFscore = F[pos]

                current = pos

        # Check if we have reached the goal

        if current == end:

            # Retrace our route backward

            path = [current]

            while current in cameFrom:

                current = cameFrom[current]

                path.append(current)

            path.reverse()

            return path, F[end]  # Done!

        # Mark the current vertex as closed

        openVertices.remove(current)

        closedVertices.add(current)

        # Update scores for vertices near the current position

        for neighbour in graph.get\_vertex\_neighbours(current):

            if neighbour in closedVertices:

                continue  # We have already processed this node exhaustively

            candidateG = G[current] + graph.move\_cost(current, neighbour)

            if neighbour not in openVertices:

                openVertices.add(neighbour)  # Discovered a new vertex

            elif candidateG >= G[neighbour]:

                continue  # This G score is worse than previously found

            # Adopt this G score

            cameFrom[neighbour] = current

            G[neighbour] = candidateG

            H = graph.heuristic(neighbour, end)

            F[neighbour] = G[neighbour] + H

    raise RuntimeError("A\* failed to find a solution")

if \_name\_ == "\_main\_":

    graph = AStarGraph()

    RS=int(input("Enter x-axis of current position"))

    CS= int(input("Enter y-axis of current position"))

    RG=int(input("Enter x-axis of goal position"))

    CG= int(input("Enter y-axis of goal position"))

    result, cost = AStarSearch((RS, CS), (RG, CG), graph)

    print("route  ", result)

    print("cost ", cost)

## Output



# Code

###### alpha-beta pruning

Maximum, Minimum = 1000, -1000

# Returns optimal value for current player

# (Initially called for root and maximizer)

def minimax(depth, node, maxP,

            values, alpha, beta):

    # Terminating condition. i.e

    # leaf node is reached

    if depth == 3:

        return values[node]

    if maxP:

        best = Minimum

        # Recur for left and right children

        for i in range(0, 2):

            val = minimax(depth + 1, node \* 2 + i,

                          False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

    else:

        best = Maximum

        # Recur for left and

        # right children

        for i in range(0, 2):

            val = minimax(depth + 1, node \* 2 + i, True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

scr=[]

x=int(input("Enter total no of leaf"))

for i in range(x):

    y=int(input("Enter node"))

    scr.append(y)

depth=int(input("Enter depth value"))

node=int(input("Enter node"))

print("Optimal value is :",minimax(depth, node, True, scr, Minimum, Maximum))

Min-Max

#min-max

import math

def minimax(cdepth,nodeval,maxterm,src,tdepth):

    if(cdepth==tdepth):

        return src[nodeval]

    if(maxterm):

        return max(minimax(cdepth+1,nodeval\*2,False,src,tdepth),

                   minimax(cdepth+1,nodeval\*2+1,False,src,tdepth))

    else:

        return min(minimax(cdepth+1, nodeval\*2, True, src, tdepth),

                   minimax(cdepth+1, nodeval\*2+1, True, src, tdepth))

src=[]

x=int(input("Enter total number of leaf node"))

for i in range(x):

    y = int(input("Enter value of leaf node"))

    src.append(y)

tdepth=math.log(len(src),2)

cdepth=int(input("Enter current depth value"))

nodeval=int(input("Enter node value"))

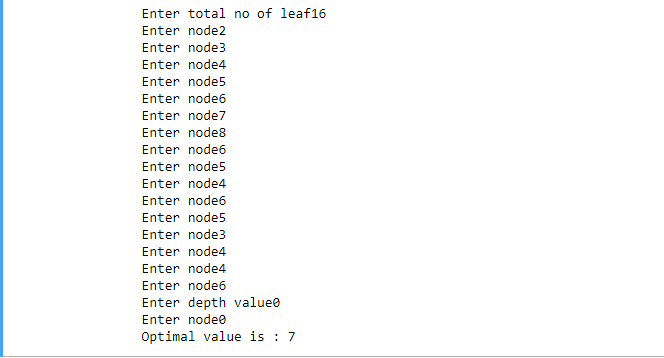
maxterm=True

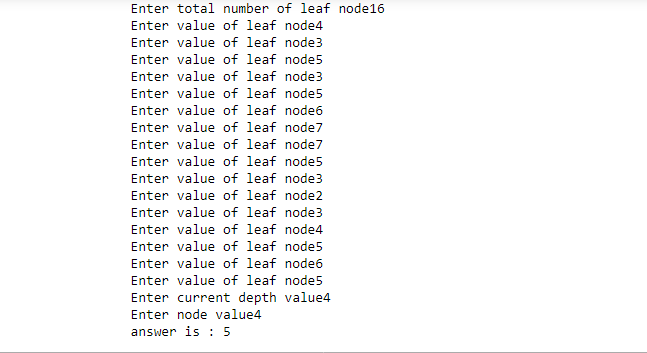
print("answer is :", end=" ")

answer=minimax(cdepth,nodeval,maxterm,src,tdepth)

print(answer)

## Output





# Code

<aiml version="1.0.1" encoding="UTF-8">

    <category>

        <pattern>HELLO</pattern>

        <template>

            hello user!

        </template>

    </category>

    <category>

        <pattern>WHAT IS YOUR NAME</pattern>

        <template>

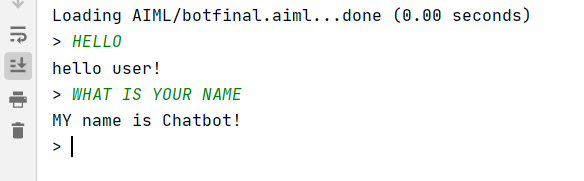
            MY name is Chatbot!

        </template>

    </category>

</aiml>

## Output



# Code

CLIPS> (deftemplate students (slot name)(slot age))

CLIPS> (assert (students (name Hamza) (age 22)))

<Fact-1>

CLIPS> (facts)

f-0     (initial-fact)

f-1     (students (name Hamza) (age 22))

For a total of 2 facts.

CLIPS> retract(0)

retract

CLIPS> (retract 0)

CLIPS> (facts)

f-1     (students (name Hamza) (age 22))

For a total of 1 fact.

CLIPS> (assert (students (name ali) (age 24)))

<Fact-2>

CLIPS> (assert (students (name usman) (age 26)))

<Fact-3>

CLIPS> (assert (students (name sana) (age 44)))

<Fact-4>

CLIPS> (assert (students (name umer) (age 66)))

<Fact-5>

CLIPS> (facts)

f-1     (students (name Hamza) (age 22))

f-2     (students (name ali) (age 24))

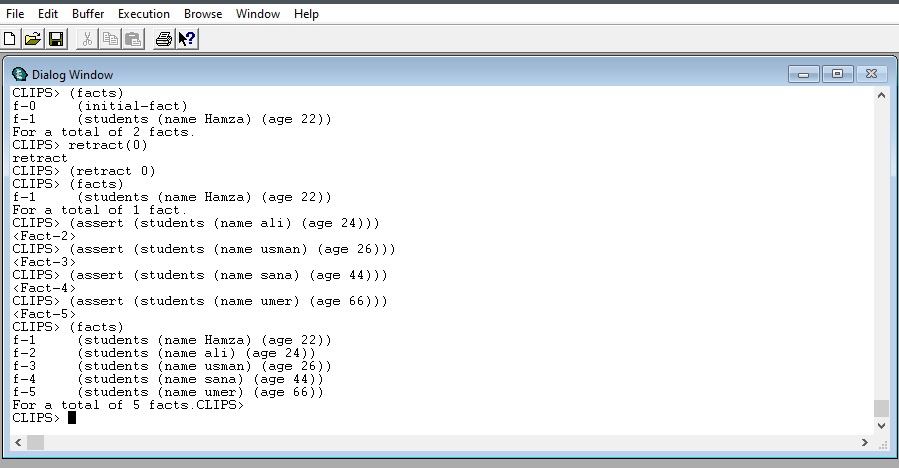
f-3     (students (name usman) (age 26))

f-4     (students (name sana) (age 44))

f-5     (students (name umer) (age 66))

For a total of 5 facts

## Output



# Code

CLIPS> (assert (weight heavy))

<Fact-1>

CLIPS> (assert (horse <= 86))

<Fact-2>

CLIPS> (defrule checkweight

(weight heavy)

=> (printout t"Horse Power"clrf))

CLIPS> run

run

CLIPS> (run)

Horse PowerclrfCLIPS> (defrule checkhorsepower

(horse <= 86)

=> (printout t"Low mileage"))

CLIPS> run

run

CLIPS> (run)

Low mileageCLIPS> (reset)

CLIPS> (assert (weight heavy))

<Fact-1>

CLIPS> (assert (horse <= 86))

<Fact-2>

CLIPS> (defrule checkweight

(weight heavy)

=> (printout t"Horse Power"clrf))

CLIPS>