**Lab File**

**Artificial Intelligence**

**[CSE401]**

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



**Submitted To: Submitted By:** Dr. S.K. Dubey Abhimanyu Bhatia

Associate Professor A2305219079

CSE Department, ASET 6CSE2X

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA-201301

**INDEX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment No.** | **Category**  **of Assignment** | **Code** | **Name of Experiment** | **Date of Allotment of experiment** | **Date of Evaluation** | **Max.**  **Marks** | **Marks obtained** | **Sign.**  **of Faculty** |
|  | **Mandatory Experiment** | **LR (0)** | Write a program to implement A\* algorithm in python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Write a program to implement Single Player Game |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Write a program to implement Tic-Tac-Toe game problem |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Implement Brute force solution to the Knapsack problem in Python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Implement Graph coloring problem using python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Write a program to implement BFS for water jug problem using Python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Write a program to implement DFS using Python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Tokenization of word and Sentences with the help of NLTK package |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Design an XOR truth table using Python |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | Study of SCIKIT fuzzy |  |  | **1** |  |  |

**Experiment-1**

**AIM**- Write a program to implement A\* algorithm.

**Software Used:** Repl.it

**Theory:**

It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A\* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

**Code:**

def aStarAlgo(start\_node, stop\_node):

    open\_set = set(start\_node)

    closed\_set = set()

    g = {}

    parents = {}

    g[start\_node] = 0

    parents[start\_node] = start\_node

    while len(open\_set) > 0:

        n = None

        for v in open\_set:

            if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):

                n = v

        if n == stop\_node or Graph\_nodes[n] is None:

            pass

        else:

            for (m, weight) in get\_neighbors(n):

                if m not in open\_set and m not in closed\_set:

                    open\_set.add(m)

                    parents[m] = n

                    g[m] = g[n] + weight

                else:

                    if g[m] > g[n] + weight:

                        # update g(m)

                        g[m] = g[n] + weight

                        # change parent of m to n

                        parents[m] = n

                        # if m in closed set,remove and add to open

                        if m in closed\_set:

                            closed\_set.remove(m)

                            open\_set.add(m)

        if n is None:

            print('Path does not exist!')

            return None

        if n == stop\_node:

            path = []

            c = 0

            while parents[n] != n:

                c += g[n]

                path.append(n)

                n = parents[n]

            path.append(start\_node)

            path.reverse()

            print('Path found: {}'.format(path))

            print('Cost: {}'.format(c))

            return path

        open\_set.remove(n)

        closed\_set.add(n)

    print('Path does not exist!')

    return None

def get\_neighbors(v):

    if v in Graph\_nodes:

        return Graph\_nodes[v]

    else:

        return None

def heuristic(n):

    H\_dist = {

        'A':11,

        'B':6,

        'C':99,

        'D':1,

        'E':7,

        'G':0,

    }

    return H\_dist[n]

# Describe your graph here

Graph\_nodes = {

   'A':[('B',2),('E',3)],'B':[('C',1),('G',9)],'C':None,'E':[('D',6)],'D':[('G',1)]

}

if \_\_name\_\_ == '\_\_main\_\_':

  aStarAlgo('A', 'G')

Graphical user interface, application

Description automatically generated**Output:**

Graphical user interface, text, application

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

**Experiment-2**

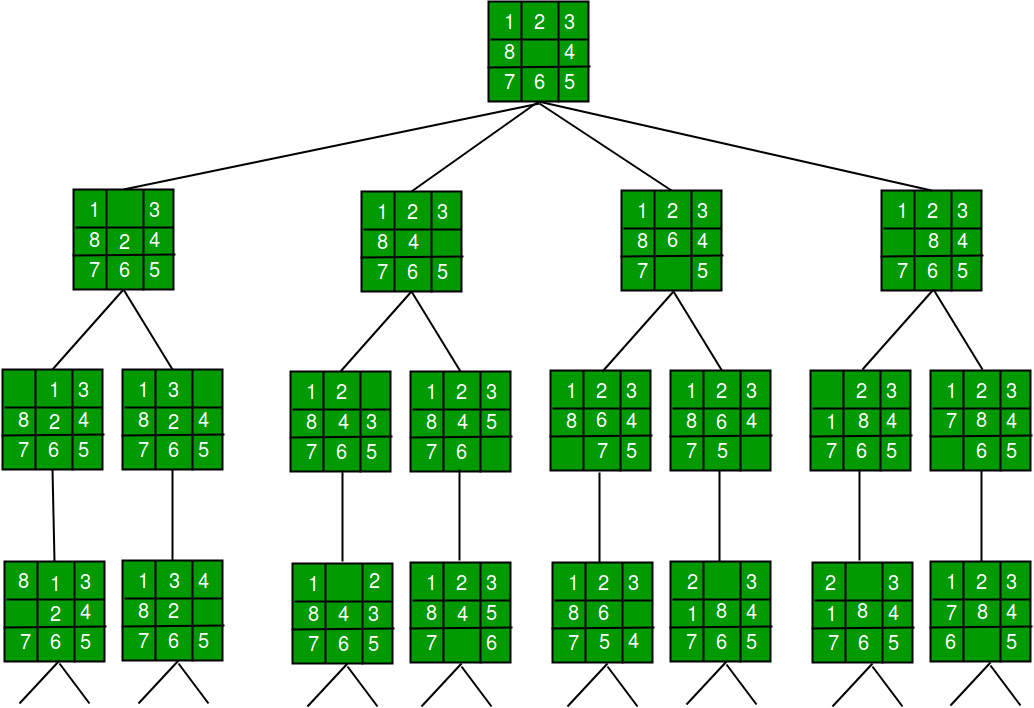
**AIM**- Write a program to implement Single player game.

**Software Used:** Repl.it

**Theory:**

We know the **eight-puzzle problem**by the name of**N puzzle problem** or **sliding puzzle problem.** **N-puzzle** that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on. In our example**N = 8.** (that is **square root of  (8+1) = 3 rows and 3 columns**).In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow **(square root of (N+1) rows  and square root of (N+1) columns).**That is if**N=15**than number of rows and columns= 4,and if **N= 24**number of rows and columns= 5.So, basically in these types of problems we have given a **initial state or initial configuration (Start state) and a Goal state or Goal Configuration.** Here We are solving a problem**of 8 puzzle**that is**a 3x3 matrix.**

**Below is the state space tree of 8 Puzzle problem:**



**Code:**

class Node:

    def \_\_init\_\_(self,data,level,fval):

        self.data = data

        self.level = level

        self.fval = fval

    def generate\_child(self):

        x,y = self.find(self.data,'\_')

        val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]

        children = []

        for i in val\_list:

            child = self.shuffle(self.data,x,y,i[0],i[1])

            if child is not None:

                child\_node = Node(child,self.level+1,0)

                children.append(child\_node)

        return children

    def shuffle(self,puz,x1,y1,x2,y2):

        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):

            temp\_puz = []

            temp\_puz = self.copy(puz)

            temp = temp\_puz[x2][y2]

            temp\_puz[x2][y2] = temp\_puz[x1][y1]

            temp\_puz[x1][y1] = temp

            return temp\_puz

        else:

            return None

    def copy(self,root):

        temp = []

        for i in root:

            t = []

            for j in i:

                t.append(j)

            temp.append(t)

        return temp

    def find(self,puz,x):

        for i in range(0,len(self.data)):

            for j in range(0,len(self.data)):

                if puz[i][j] == x:

                    return i,j

class Puzzle:

    def \_\_init\_\_(self,size):

        self.n = size

        self.open = []

        self.closed = []

    def accept(self):

        puz = []

        for i in range(0,self.n):

            temp = input().split(" ")

            puz.append(temp)

        return puz

    def f(self,start,goal):

        return self.h(start.data,goal)+start.level

    def h(self,start,goal):

        temp = 0

        for i in range(0,self.n):

            for j in range(0,self.n):

                if start[i][j] != goal[i][j] and start[i][j] != '\_':

                    temp += 1

        return temp

    def process(self):

        print("Enter the start state matrix \n")

        start = self.accept()

        print("Enter the goal state matrix \n")

        goal = self.accept()

        start = Node(start,0,0)

        start.fval = self.f(start,goal)

        self.open.append(start)

        print("\n")

        while True:

            cur = self.open[0]

            print("")

            print("  | ")

            print("  | ")

            print(" \\\'/ \n")

            for i in cur.data:

                for j in i:

                    print(j,end=" ")

                print("\n")

            if(self.h(cur.data,goal) == 0):

                break

            for i in cur.generate\_child():

                i.fval = self.f(i,goal)

                self.open.append(i)

                self.closed.append(cur)

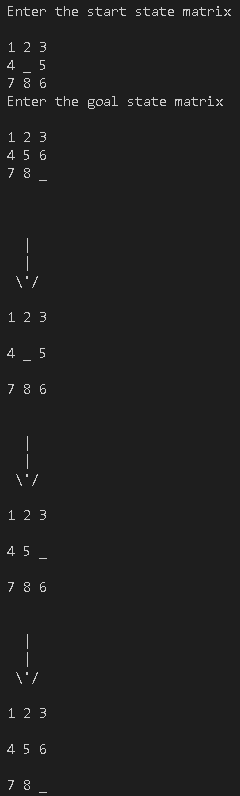
            del self.open[0]

            self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)

puz.process()

**Output:**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

**Experiment-3**

**AIM**- Write a program to implement TIC-TAC-TOE game problem.

**Software Used:** Repl.it

**Theory:**

Tic-tac-toe who takes turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. Players soon discover that the best play from both parties leads to a draw. Hence, tic-tac-toe is most often played by young children, who often have not yet discovered the optimal strategy. Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space. The game can be generalized to an m,n,k-game in which two players alternate placing stones of their own colour on an m×n board, with the goal of getting k of their own colour in a row. Tic-tac-toe is the (3,3,3)-game. Tic-tac-toe is the game where n equals 3 and d equals 2. If played properly, the game will end in a draw, making tic-tac-toe a futile game. 

**Diagram

Description automatically generated**

**Code:**

board = ["-", "-", "-",

         "-", "-", "-",

         "-", "-", "-"]

game\_still\_going = True

winner = None

current\_player = "X"

def play\_game():

  display\_board()

  while game\_still\_going:

    handle\_turn(current\_player)

    check\_if\_game\_over()

    flip\_player()

  if winner == "X" or winner == "O":

    print(winner + " won.")

  elif winner == None:

    print("Tie.")

def display\_board():

  print("\n")

  print("---" + "----"+"----"+"     -----------")

  print("|"+ board[0] + " | " + board[1] + " | " + board[2] +"|" +"     |1 | 2 | 3|")

  print("---" + "----"+"----"+"     -----------")

  print("|"+board[3] + " | " + board[4] + " | " + board[5] +"|"+ "     |4 | 5 | 6|")

  print("---" + "----"+"----"+"     -----------")

  print("|"+board[6] + " | " + board[7] + " | " + board[8] +"|"+ "     |7 | 8 | 9|")

  print("---" + "----"+"----"+"     -----------")

  print("\n")

def handle\_turn(player):

  print(player + "'s turn.")

  position = input("Choose a position from 1-9: ")

  valid = False

  while not valid:

    while position not in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:

      position = input("Choose a position from 1-9: ")

    position = int(position) - 1

    if board[position] == "-":

      valid = True

    else:

      print("You can't go there. Go again.")

  board[position] = player

def check\_if\_game\_over():

  check\_for\_winner()

  check\_for\_tie()

def check\_for\_winner():

  global winner

  row\_winner = check\_rows()

  column\_winner = check\_columns()

  diagonal\_winner = check\_diagonals()

  if row\_winner:

    winner = row\_winner

  elif column\_winner:

    winner = column\_winner

  elif diagonal\_winner:

    winner = diagonal\_winner

  else:

    winner = None

def check\_rows():

  global game\_still\_going

  row\_1 = board[0] == board[1] == board[2] != "-"

  row\_2 = board[3] == board[4] == board[5] != "-"

  row\_3 = board[6] == board[7] == board[8] != "-"

  if row\_1 or row\_2 or row\_3:

    game\_still\_going = False

  if row\_1:

    return board[0]

  elif row\_2:

    return board[3]

  elif row\_3:

    return board[6]

  else:

    return None

def check\_columns():

  global game\_still\_going

  column\_1 = board[0] == board[3] == board[6] != "-"

  column\_2 = board[1] == board[4] == board[7] != "-"

  column\_3 = board[2] == board[5] == board[8] != "-"

  if column\_1 or column\_2 or column\_3:

    game\_still\_going = False

  if column\_1:

    return board[0]

  elif column\_2:

    return board[1]

  elif column\_3:

    return board[2]

  else:

    return None

def check\_diagonals():

  global game\_still\_going

  diagonal\_1 = board[0] == board[4] == board[8] != "-"

  diagonal\_2 = board[2] == board[4] == board[6] != "-"

  if diagonal\_1 or diagonal\_2:

    game\_still\_going = False

  if diagonal\_1:

    return board[0]

  elif diagonal\_2:

    return board[2]

  else:

    return None

def check\_for\_tie():

  global game\_still\_going

  if "-" not in board:

    game\_still\_going = False

    return True

  else:

    return False

def flip\_player():

  global current\_player

  if current\_player == "X":

    current\_player = "O"

  elif current\_player == "O":

    current\_player = "X"

play\_game()

**Output:**

**Calendar

Description automatically generatedCalendar

Description automatically generatedCalendar

Description automatically generatedX won:**

**Calendar

Description automatically generatedText

Description automatically generatedO won:**

Text

Description automatically generated with low confidence

**Text, calendar

Description automatically generatedCalendar

Description automatically generatedCalendar

Description automatically generated**Calendar

Description automatically generated**Draw:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

**Experiment-4**

**AIM**- Implement Brute force solution to the Knapsack problem in Python.

**Software Used:** Repl.it

**Theory:­**

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Brute force is a straightforward approach to solving a problem, usually directly based on the problem’s statement and definitions of the concepts involved. If there are n items to choose from, then there will be 2 n possible combinations of items for the knapsack. An18 2 item is either chosen or not chosen. A bit string of 0’s and 1’s is generated which is of length n. If the I th symbol of a bit string is 0, then the i th item is not chosen and if it is 1, the i th item is chosen.

Therefore, the complexity of the Brute Force algorithm is O (n2 n ). Since the complexity of this algorithm grows exponentially, it can only be used for small instances of the KP. Otherwise, it does not require much programming effort in order to be implemented.

**Code:**

import itertools

def findsubsets(s, n):

    return list(itertools.combinations(s, n))

item=[0,1,2,3]

w=[2,5,10,5]

c=[20,30,50,10]

sub=[]

weight=[]

cost=[]

items=4

prof=0

capacity=12

subset=[]

for i in range(1,items+1):

    subset.append(findsubsets(item,i))

for i in subset:

    for j in i:

        sub.append(j)

        sumw=0

        sumc=0

        for k in j:

            sumw=sumw+w[k]

            sumc=sumc+c[k]

        weight.append(sumw)

        cost.append(sumc)

print("ITEM WEIGHTS: ",w,"\nITEM COSTS: ",c )

print("\nITEMS\t\tWEIGHT\t\tCOST\t\tVALIDITY ")

for i in range(len(sub)):

    if(len(sub[i])<3):

        print(sub[i],"\t\t",weight[i],"\t\t",cost[i],"\t\t",end="")

    else:

        print(sub[i],"\t",weight[i],"\t\t",cost[i],"\t\t",end="")

    if(weight[i]<=capacity):

        print('Valid')

    else:

        print('Invalid')

    if(weight[i]<=capacity and cost[i]>prof):

        prof=cost[i]

        profweight=weight[i]

        profsub=sub[i]

print("\nMax Profit: ",prof,"\nWEIGHT: ",profweight,"\nITEMS: ",profsub)

**Output:**

A picture containing text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

**Experiment-5**

**AIM**- Implement Graph coloring problem using python.

**Software Used:** Repl.it

**Theory:**

In graph theory, graph colouring is a special case of graph labelling; it is an assignment of labels traditionally called "colours" to elements of a graph subject to certain constraints. In its simplest form, it is a way of colouring the vertices of a graph such that no two adjacent vertices are of the same colour; this is called a vertex colouring. Similarly, an edge colouring assigns a colour to each edge so that no two adjacent edges are of the same colour, and a face colouring of a planar graph assigns a colour to each face or region so that no two faces that share a boundary have the same colour. Vertex colouring is the starting point of graph colouring. Other colouring problems can be transformed into a vertex version. For example, an edge colouring of a graph is just a vertex colouring of its line graph, and a face colouring of a plane graph is just a vertex colouring of its dual.

Diagram

Description automatically generated

**Code:**

def addEdge(adj, v, w):

    adj[v].append(w)

    adj[w].append(v)

    return adj

def gColoring(adj, V):

    result = [-1] \* V

    result[0] = 0;

    available = [False] \* V

    for u in range(1, V):

        for i in adj[u]:

            if (result[i] != -1):

                available[result[i]] = True

        cr = 0

        while cr < V:

            if (available[cr] == False):

                break

            cr += 1

        result[u] = cr

        for i in adj[u]:

            if (result[i] != -1):

                available[result[i]] = False

    for u in range(V):

        print("Vertex", u, " --->  Color", result[u])

    print("Chromatic number-",max(result)+1)

if \_\_name\_\_ == '\_\_main\_\_':

    g1 = [[] for i in range(6)]

    g1 = addEdge(g1, 0, 1)

    g1 = addEdge(g1, 0, 2)

    g1 = addEdge(g1, 0, 3)

    g1 = addEdge(g1, 1, 3)

    g1 = addEdge(g1, 2, 3)

    g1 = addEdge(g1, 3, 4)

    g1 = addEdge(g1, 2, 4)

    g1 = addEdge(g1, 4, 5)

    g1 = addEdge(g1, 3, 5)

    print("Graph color ")

    gColoring(g1, 6)

**Text

Description automatically generated with medium confidenceOutput:**

Diagram

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

**Experiment-6**

**AIM**- Write a program to implement BFS for water jug problem using Python.

**Software Used:** Repl.it

**Theory:**

In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 5 gallons of water and the other has the capacity to hold 12 gallons of water. There is no other measuring equipment available, and the jugs also do not have any kind of marking on them. So, the agent’s task here is to fill the 12-gallon jug with 7 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all its neighbours. In the next step, the neighbours of the nearest node of A are explored and process continues in the further steps. The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once, and no node is visited twice.

**Code:**

print('Abhimanyu Bhatia\nA2305219086\n')

print ("Water Jug Problem Solution:")

x\_capacity = int(input("Enter capacity of jug 1"))

y\_capacity = int(input("Enter capacity of jug 2"))

end = int(input("Enter target volume:"))

def bfs(start, end, x\_capacity, y\_capacity):

path = []

front = []

front.append(start)

visited = []

visited.append(start)

while(not (not front)):

current = front.pop()

x = current[0]

y = current[1]

path.append(current)

if x == end or y == end:

print ("Found!")

n=len(path)

(a,b)=path[n-1]

if(a!=end and b==end):

path.append([0,end])

visited.append([0,end])

path.append([end,0])

visited.append([end,0])

print(path)

return

# rule 1

if current[0] < x\_capacity and ([x\_capacity, current[1]] not in visited):

front.append([x\_capacity, current[1]])

visited.append([x\_capacity, current[1]])

# rule 2

if current[1] < y\_capacity and ([current[0], y\_capacity] not in visited):

front.append([current[0], y\_capacity])

visited.append([current[0], y\_capacity])

# rule 3

if current[0] > 0 and ([0, current[1]] not in visited):

front.append([0, current[1]])

visited.append([0, current[1]])

# rule 4

if current[1] > 0 and ([current[0], 0] not in visited):

front.append([current[0], 0])

visited.append([current[0], 0])

# rule 5

if current[1] > 0 and current[0]+current[1]>=x\_capacity and (x\_capacity, current[1]-(x\_capacity-current[0]) not in visited):

front.append([x\_capacity, current[1]-(x\_capacity-current[0])])

visited.append([x\_capacity, current[1]-(x\_capacity-current[0])])

# rule 6

if current[0] > 0 and current[0]+current[1]>=y\_capacity and ([current[0]-(y\_capacity-current[1]),y\_capacity] not in visited):

front.append([current[0]-(y\_capacity-current[1]),y\_capacity])

visited.append([current[0]-(y\_capacity-current[1]),y\_capacity])

# rule 7

if current[1]>0 and current[0]+current[1]<=x\_capacity and ([current[0]+current[1],0] not in visited):

front.append([current[0]+current[1],0])

visited.append([current[0]+current[1],0])

# rule 8

if current[0]>0 and current[0]+current[1]<=y\_capacity and ([0,current[0]+current[1]] not in visited):

front.append([0,current[0]+current[1]])

visited.append([0,current[0]+current[1]])

print ("Solution Not found")

start = [0, 0]

bfs(start, end, x\_capacity, y\_capacity)

**Output:**

Text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrollment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 7**

**AIM**- Write a program to implement DFS using Python

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

Depth First Traversal (or Search) for a graph is like Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. In the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don’t mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.

Diagram

Description automatically generated

**CODE:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def addEdge(self,u,v):

        self.graph[u].append(v)

    def DFSUtil(self,v,visited):

        visited[v]= True

        print (v)

        for i in self.graph[v]:

            if visited[i] == False:

                self.DFSUtil(i, visited)

    def DFS(self,v):

        visited = [False]\*(len(self.graph))

        self.DFSUtil(v,visited)

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

print ("Following is DFS from (starting from vertex 2)")

g.DFS(2)

**OUTPUT:**

Text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrolment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 8**

**AIM**- Tokenization of word and Sentences with the help of NLTK package

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analysing linguistic structure, and more.

**CODE:**

import nltk

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

newwords=[]

textw="Basketball is a team sport in which two teams, most commonly of five players each, opposing one another on a rectangular court, compete with the primary objective of shooting a basketball (approximately 9.4 inches (24 cm) in diameter) through the defender's hoop (a basket 18 inches (46 cm) in diameter mounted 10 feet (3.048 m) high to a backboard at each end of the court, while preventing the opposing team from shooting through their own hoop. A field goal is worth two points, unless made from behind the three-point line, when it is worth three. After a foul, timed play stops and the player fouled or designated to shoot a technical foul is given one, two or three one-point free throws. The team with the most points at the end of the game wins, but if regulation play expires with the score tied, an additional period of play (overtime) is mandated."

wordsw=word\_tokenize(textw)

for word in wordsw:

    if word.isalnum():

        newwords.append(word)

    else:

        continue

print("Words:", newwords)

print("No. of words: ",len(textw))

print("\n")

newwords=[]

textp="The earliest commercial Japanese animations date to 1917. A characteristic art style emerged in the 1960s with the works of cartoonist Osamu Tezuka and spread in following decades, developing a large domestic audience. Anime is distributed theatrically, through television broadcasts, directly to home media, and over the Internet. In addition to original works, anime are often adaptations of Japanese comics (manga), light novels, or video games. It is classified into numerous genres targeting various broad and niche audiences."

words=word\_tokenize(textp)

for word in words:

    if word.isalnum():

        continue

    else:

        newwords.append(word)

print("punctuations: ", newwords)

print("No. of Punctuations: ",len(newwords))

print("\n")

from nltk.tokenize import sent\_tokenize

texts="The National Football League (NFL) is a professional American football league that consists of 32 teams, divided equally between the American Football Conference (AFC) and the National Football Conference (NFC). The NFL is one of the major North American professional sports leagues and the highest professional level of American football in the world.[5] Each NFL season begins with a three-week preseason in August, followed by the eighteen-week regular season which runs from early September to the end of December (or in some cases early January), with each team playing seventeen games and having one bye week."

print("sentences: ",sent\_tokenize(texts))

print("No. of sentences: ",len(sent\_tokenize(texts)))

**Text

Description automatically generatedOUTPUT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrolment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 9**

**AIM**- Design an XOR truth table using Python

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

A Logic gate is an elementary building block of any digital circuits. It takes one or two inputs and produces output based on those inputs. Outputs may be high (1) or low (0). Logic gates are implemented using diodes or transistors. The XOR gate gives an output of 1 if either both inputs are different, it gives 0 if they are same.

Table

Description automatically generated

**CODE:**

print("X xor Y:")

print("------------------------")

print("|  X  |  Y  |  X xor Y |")

print("------------------------")

for X in [0,1]:

    for Y in [0,1]:

        print("| ",X," | ",Y," |    ",X^Y,"   |")

        print("------------------------")

print("\n X xor Y xor Z:")

print("---------------------------------")

print("|  X  |  Y  |  Z  |X xor Y xor Z|")

print("---------------------------------")

for X in [0,1]:

    for Y in [0,1]:

        for Z in [0,1]:

            print("| ",X," | ",Y," | ",Z," |     ",X^Y^Z,"     |")

            print("---------------------------------")

**OUTPUT:**

A picture containing text, electronics, keyboard, computer

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrolment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 10**

**AIM**- Study of SCIKIT fuzzy

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

Scikit-fuzzy is also known as SKFuzzy. It is a Fuzzy logic toolbox for Python.This package implements many useful tools for projects involving fuzzy logic, also known as grey logic. Scikit-Fuzzy is a collection of fuzzy logic algorithms intended for use in the [SciPy](http://scipy.org/) Stack, written in the Python computing language. Scikit-fuzzy is a fuzzy logic Python package that works with numpy arrays.

**ANALYSIS:**

Classification based on fuzzy technique can clearly classify combinational pixels. It is an efficient approach for classifying land cover which is not visible clearly. Fuzzy classification has an upper hand for representing real world through sharp objects and crisp classes.

**Supervised classification:**

While performing classification using supervised method the only data required as an input to the system is the training data set. This is due to the reason that for such cases prior information about the classes are already present in the image to be classified; hence its name is supervised classification. There are three basic steps involve in supervised classification method: identifying training sites, creating signatures, and classifying the image. There are numerous supervised classification techniques that exists currently, some common techniques are: Maximum likelihood classifier, parallelepiped classifier.

**Unsupervised classification:**

For unsupervised classification as the name suggests requires no training data. In this method the unknown pixels in the image are examined thoroughly. The examined pixels are then aggregated into finite number of classes which are based on the cluster present in the image. Unsupervised classification can be used in situations where less information is present prior to classification. The grouping of data with similar characteristics is called clustering. Some of the common unsupervised classification methods are K Means, Simple one pass clustering, minimum distribution angle.

It has been observed that a fuzzy approach gives better results when prior knowledge about the classes is present. Thus, fuzzy based supervised classification is preferred to fuzzy based unsupervised classification. Fuzzy based supervised classification also gives higher accuracy rate as compared to fuzzy based unsupervised classification. However, supervised classification has one drawback i.e., it overlooks uncertain attributes which may occur in due course of time and concentrates more on instances in the training sets. Unsupervised classification also suffers from a drawback i.e., classes formed may not be appropriate or informative enough or useful. In the research carried out by Manibhusan the satellite images were classified using fuzzy logic Supervised method of classification was implemented. After classification of images the results proved that the accuracy of fuzzy classification was better than crisp classification. This was due to the main reason that fuzzy operators can solve overlapping problems better than crisp operators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Abhimanyu Bhatia | | Enrolment No. | A2305219079 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Viva Component) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP)** | | | |
| Programme | B. Tech CSE | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | Semester | 6 |
| Student Name | Abhimanyu Bhatia | Enrollment No. | A2305219079 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Clarity of the Subject (H) | 4 |  |  |
| Quality of theoretical Discussion (I) | 6 |  |  |
| Total | 10 |  |  |