Sinhgad College of Engineering
Department of Computer Engineering

Photo

Name of Student: Pratik Kashinath Kate

Roll No: 305A043 PRN Number: 72162555L

Class: Third Year (T.E) Div: A Batch: B

Name of Laboratory: Artificial Intelligence [Laboratory Practice II(LP2)]

List of Assignments

Sr. No.	**Title of Assignment**	**Remark**
1	Depth and Breadth First Search algorithm	
2	A star Algorithm	
3	Greedy search algorithm	
4	Solution for a Constraint Satisfaction Problem	
5	Development of elementary chatbot	
6	Implement of Expert System	

Assignment No. 1

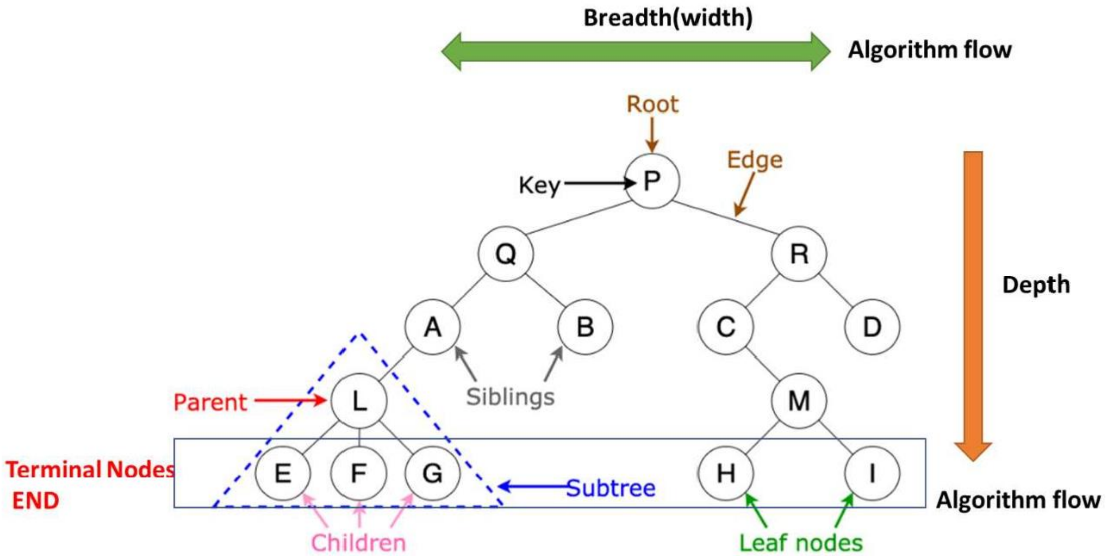
Title: Depth and Breadth First Search algorithm.

Problem Definition: Implement depth first search algorithm and Breadth First Search algorithm, use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

Concept:

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. Breadth-first search is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier.

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph. This is achieved very simply by using a LIFO queue for the frontier.



Conclusion: Thus, we have learned depth first search (DFS) & breadth first search (BFS) algorithms.

Algorithm/Code:

DFS algorithm:

```
function DEPTH-FIRST-SEARCH(problem) returns a solution, or failure
node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0  # Initialization
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  # goal state = starting state

frontier ←a LIFO queue with node as the only element # frontier = root node
explored ←an empty set #  no path
loop do
        if EMPTY?( frontier) then return failure # true when no other node is after root node
        node←POP( frontier ) # chooses the deepest node after frontier
        add node.STATE to explored # add state explored (LIFO Path)
        PATH-COST ←path-cost(explored) # path cost LIFO Queue

        for each child node in problem.Queue(node.STATE) do
        child ←CHILD-NODE(problem, node, queue) # keep all child node in queue except LIFO child node
        if LIFO.child .STATE is not in explored  then  # is leaf node is achieved?
        if problem.GOAL-TEST(child .STATE) then return SOLUTION(LIFO.child ) # solution at leaf node
        frontier ←INSERT(Lifo.child ) # Load LIFO frontier child
```

BFS Algorithm:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0  # Initialization
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  # goal state = starting state

frontier ←a FIFO queue with node as the only element # frontier  = root node
explored ←an empty set #  no path
loop do
        if EMPTY?( frontier) then return failure # true when no other node is after root node
        node←POP( frontier ) # chooses the shallowest node in frontier
        add node.STATE to explored # add state explored (FIFO Path)
        PATH-COST ←path-cost(explored) # path cost FIFO Queue

        for each action in problem.ACTIONS(node.STATE) do
        child ←CHILD-NODE(problem, node, action) # explore all child node
        if child .STATE is not in explored  then  # is leaf node is achieved?
        if problem.GOAL-TEST(child .STATE) then return SOLUTION(child) # solution at leaf node
        frontier ←INSERT(child , frontier ) # frontier pointed at child node
```

Assignment No. 2

Title: A star Algorithm.

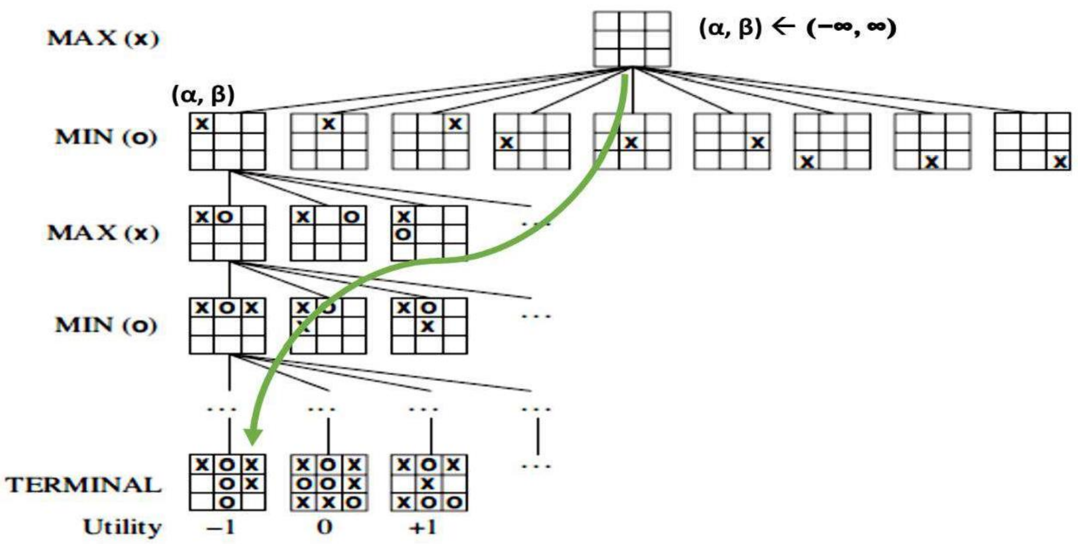
Problem Definition: Implement A star algorithm for tic tac toe game search problem

Concept: The most widely known form of best-first search is called A*("A-star search"). It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

$$f(n) = g(n) + h(n)$$

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have

f (n) = estimated cost of the cheapest solution through n.



Here:

$f(s) = g(s,a) + h(s)$

Evaluation function= Result(s,a) + Heuristic function

for MIN player h(α, β)= minimum(α, β)

Hence f(s) = Result(s,a) + minimum(α, β)

Conclusion: Thus, we have learned A star search algorithm for tic-tac-toe game.

Algorithm/Code:

Algorithm:

function TICTACTOE-SEARCH(game) returns a win, or draw

node ←a node with STATE = game.INITIAL-STATE , α ← −∞ , β ← ∞ # Initialization

frontier ←α, β pruning queue with node as the only element # frontier is at root node

explored ←an empty set # no path

loop do

if EMPTY?(frontier) then return failure # true when no other node is after root node

node←POP(frontier) # chooses the node with min value of α, β

α ←MAX-VALUE(state) # calculate value α for MAX at MAX node

β ← MIN-VALUE(state) # calculate value β for MIN at MIN node

f(s) ← Result(s,a) + minimum(α, β) # evaluation function

add node.STATE to explored # add state explored

if problem.TERMINAL-TEST(node .STATE) then return WIN(node) # solution

return UTILITY(state) # point scored by players

function MAX-VALUE(state) returns a utility value

v ←−∞

for each a in ACTIONS(state) do

v ←MAX(v, MIN-VALUE(RESULT(s, a)))

return v

function MIN-VALUE(state) returns a utility value

k ←∞

for each a in ACTIONS(state) do

k ←MIN(v, MAX-VALUE(RESULT(s, a)))

return k

Assignment No. 3

Title: Greedy search algorithm.

Problem Definition: Implement Greedy search algorithm for Kruskal's Minimal Spanning Tree Algorithm

Concept: Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function, that is, f(n) = h(n).

Minimum Spanning Tree

For a given connected tree(graph), a spanning tree of that tree is a subtree(subgraph) that connects all the node (Vertices) together without any cycle, while minimum spanning tree is having minimum path cost (Weight) from all possible spanning tree of that graph(tree).

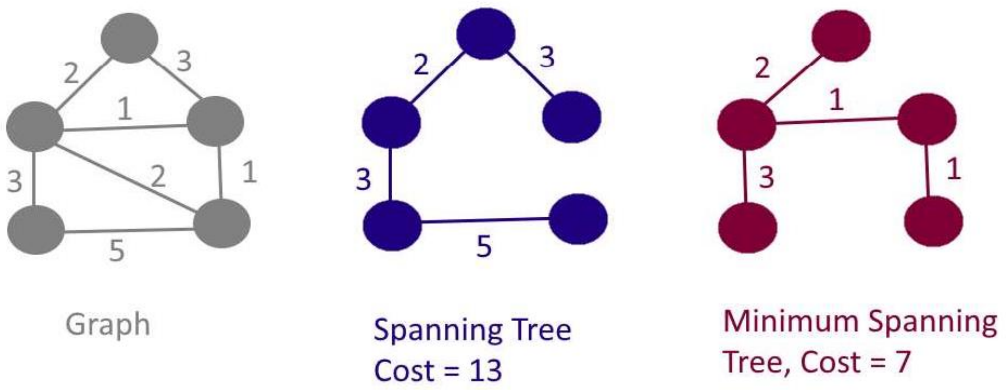
Obs: A spanning tree has (N -1) paths (Branches/edges) where N is the total number of nodes(Vertices)in the given graph(tree).

Greedy Kruskal's Minimum Spanning Tree Algorithm steps

1. Sort all the paths (Branches/edges) with increasing order of their path cost (weight).
2. Pick the path with low path cost(weight). Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this path. Else, discard it.

Here greedy based **Heuristic value (H_{Cnp})** is minimum value path cost of all connected node pair path cost.

3. Repeat step-2 until there are (N-1) edges in the spanning tree.



Conclusion: Thus, we have learned Greedy search algorithm for Kruskal's Minimal Spanning Tree Algorithm.

Algorithm/Code:

```
function GREEDY_KRUSKAL's_MST(graph) returns a solution, or failure
  node ←a node with STATE = graph.INITIAL-STATE # Initialization
  frontier ←load with starting node, N ← number of nodes # frontier is at starting node
  explored_path_pair ←an empty set # Initialization & no explored path
  spanning_tree_path ←an empty set # Initialization & no explored path
  mst_path ←an empty set # Initialization & no explored path
  HCnp ← 0,Count ← 0 # Heuristic connected node pair path cost initialize it to zero
  loop do
    if EMPTY?( frontier) then return failure # true when no other node is after starting node
      node←POP( frontier ) # chooses the node connected to frontier
      explored_path_pair ←path cost # add weights to explored set
    if TERMINAL-TEST(node.STATE) then # at terminal node
      HCnp ← Minimum(explored_path_pair )
      frontier ←load at minimum path cost # frontier is at lowest weight edge
    loop do
      node←POP( frontier ) # chooses the node without forming cycle
      spanning_tree_path ← path cost # add explored without forming cycle
     if NO_Cycle (node.STATE) then
       count ← increment counter # count number of explored path
     if count is equal to N-1 then
       return a Solution(mst_path ← spanning_tree_path )

function NO_Cycle(state) returns a true or false
      temp ← state
      x ← POP(temp)
      y ← POP(x)
      if x is equal to y then
             return false
      return True
```

Assignment No. 4

Title: Solution for a Constraint Satisfaction Problem (CSP)**.**

Problem Definition: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem.

Concept: CSP search algorithms take advantage of the structure of states and use general-purpose heuristics rather than problem-specific heuristics to enable the solution of complex problems. Where each state is atomic, or indivisible ~~a~~ black box with no internal structure.

There are three components: X, D & C

Where

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

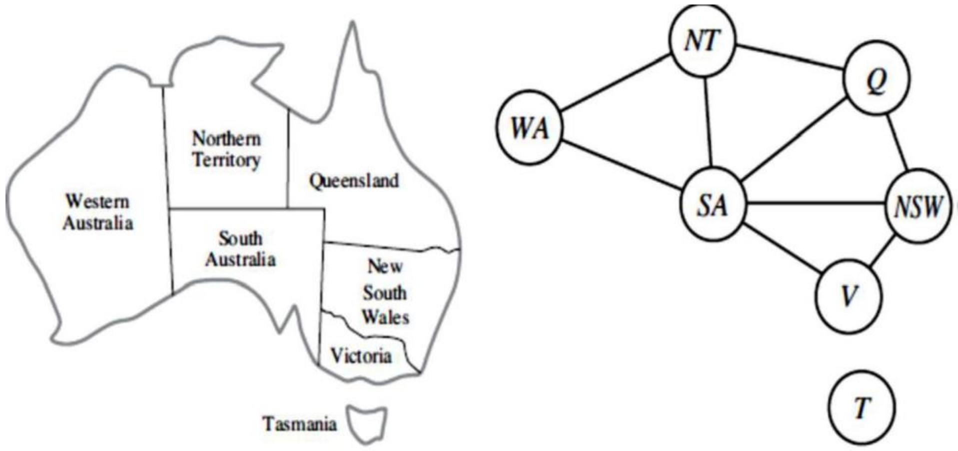
Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i

Here each constraint $\mathbf{C_i}$ consists of a pair **<scope, rel>**

where, scope is a **tuple of variables** that participate in the constraint and **rel** is a relation that defines the values that those variables can take

Example: If X_1 and X_2 both have the domain {A,B}, then the constraint saying the two variables must have **different values** can be written as $<(X_1,X_2), [(A,B), (B,A)]>$ or $<(X_1,X_2), X_1 \neq X_2>$

Example: Graph coloring problem is converted to tree data structure



Conclusion: Thus, we have learned a problem of constraint satisfaction problem.

N queen solution source code:

```
""" Python3 program to solve N Queen Problem
using Branch or Bound """

N = 8

""" A utility function to print solution """
def printSolution(board):
	for i in range(N):
		for j in range(N):
			print(board[i][j], end = " ")
		print()

""" A Optimized function to check if
a queen can be placed on board[row][col] """
def isSafe(row, col, slashCode, backslashCode,
		rowLookup, slashCodeLookup,
					backslashCodeLookup):
	if (slashCodeLookup[slashCode[row][col]] or
		backslashCodeLookup[backslashCode[row][col]] or
		rowLookup[row]):
		return False
	return True

""" A recursive utility function
to solve N Queen problem """
def solveNQueensUtil(board, col, slashCode, backslashCode,
					rowLookup, slashCodeLookup,
					backslashCodeLookup):

	""" base case: If all queens are
	placed then return True """
```

```
if(col >= N):
    return True
for i in range(N):
    if(isSafe(i, col, slashCode, backslashCode,
              rowLookup, slashCodeLookup,
              backslashCodeLookup)):

        """ Place this queen in board[i][col] """
        board[i][col] = 1
        rowLookup[i] = True
        slashCodeLookup[slashCode[i][col]] = True
        backslashCodeLookup[backslashCode[i][col]] = True

        """ recur to place rest of the queens """
        if(solveNQueensUtil(board, col + 1,
                            slashCode, backslashCode,
                            rowLookup, slashCodeLookup,
                            backslashCodeLookup)):
            return True


        """ If placing queen in board[i][col]
        doesn't lead to a solution,then backtrack """

        """ Remove queen from board[i][col] """
        board[i][col] = 0
        rowLookup[i] = False
        slashCodeLookup[slashCode[i][col]] = False
        backslashCodeLookup[backslashCode[i][col]] = False

""" If queen can not be place in any row in
this column col then return False """
return False
```

```
""" This function solves the N Queen problem using
Branch or Bound.
"""
def solveNQueens():
	board = [[0 for i in range(N)]
				for j in range(N)]

	# helper matrices
	slashCode = [[0 for i in range(N)]
					for j in range(N)]
	backslashCode = [[0 for i in range(N)]
						for j in range(N)]

	# arrays to tell us which rows are occupied
	rowLookup = [False] * N

	# keep two arrays to tell us
	# which diagonals are occupied
	x = 2 * N - 1
	slashCodeLookup = [False] * x
	backslashCodeLookup = [False] * x

	# initialize helper matrices
	for rr in range(N):
		for cc in range(N):
			slashCode[rr][cc] = rr + cc
			backslashCode[rr][cc] = rr - cc + 7

	if(solveNQueensUtil(board, 0, slashCode, backslashCode,
						rowLookup, slashCodeLookup,
						backslashCodeLookup) == False):
```

```
        print("Solution does not exist")
        return False

    # solution found
    printSolution(board)
    return True

# Driver Cde
solveNQueens()
```

Assignment No. 5

Title: Development of elementary chatbot.

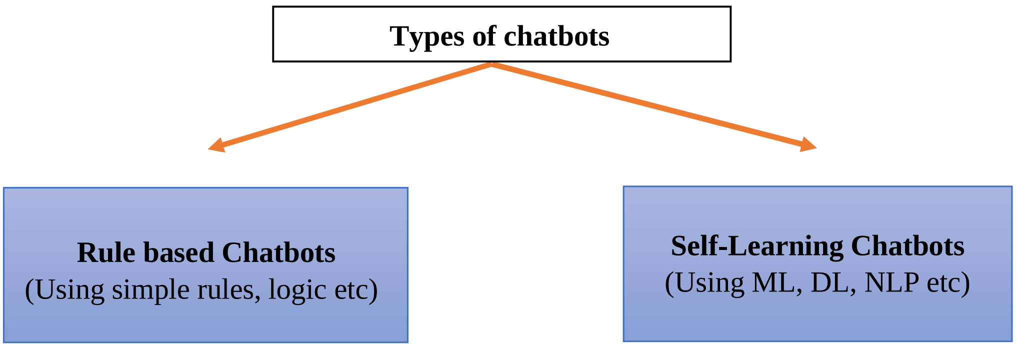
Problem Definition: Develop an elementary simple rule based chatbot for any suitable customer interaction application, using Python *tkinter* & *nltk* library.

Concept: A chatbot is an AI-based software designed to interact with humans in their natural languages. These chatbots are usually converse via auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like manner. A chatbot is arguably one of the best applications of natural language processing.

Chatbot asks for basic information of customers like name, email address, and the query. If a query is simple like product fault, booking mistake, need some information then without any human connection it can solve it automatically and if some problem is high then it passes the details to the human head.

The Rule-based approach trains a chatbot to answer questions based on a set of pre-determined rules on which it was initially trained. These set rules can either be very simple or very complex. While rule-based chatbots can handle simple queries quite well, they usually fail to process more complicated queries/requests.

As the name suggests, self-learning bots are chatbots that can learn on their own. These leverage advanced technologies like Artificial Intelligence and Machine Learning to train themselves from instances and behaviours. Naturally, these chatbots are much smarter than rule-based bots. Self-learning bots can be further divided into two categories –Retrieval Based or Generative.



Conclusion: Thus, we have learned elementary implementation of chatbots.

Chatbot source code:

Rule Based ChatBot:

```
from tkinter import *
root = Tk()
root.title("Chatbot")
def send():
    send = "\n You -> "+e.get()
    txt.insert(END, ""+send)
    user = e.get().lower()
    if(user == "hello"):
        txt.insert(END, "" +  "\n Bot -> Hi")
    elif(user == "hi" or user == "hii" or user == "hiiii"):
        txt.insert(END, "" + "\n Bot -> Hello")
    elif(e.get() == "how are you"):
        txt.insert(END, "" + "\n Bot -> fine! and you")
    elif(user == "fine" or user == "i am good" or user == "i am doing good"):
        txt.insert(END, "" + "\n Bot -> Great! how can I help you.")
    elif(user.lower() == "wish me") :
        txt.insert(END, "\n Bot -> Happy Birthday to You Master")
    elif(user.lower() == "thank you" or user.lower() == 'thanks') :
        txt.insert(END, "\n Bot -> You are Welcome")
    elif(user.lower() == "good morning") :
        txt.insert(END, "\n Bot -> Good Morning Master!!")
    elif(user.lower() == "Good Night") :
        txt.insert(END, "\n Bot -> Good Night Master!!")
    else:
        txt.insert(END, "" + "\n Bot -> Sorry! I dind't got you")
    e.delete(0, END)
txt = Text(root)
txt.grid(row=0, column=0, columnspan=2)
e = Entry(root, width=100)
e.grid(row=1, column=0)
```

```
send = Button(root, text="Send", command=send).grid(row=1, column=1)
root.mainloop()
```

NLTK Based ChatBot:

```
import nltk
from nltk.chat.util import Chat, reflections

pairs =[
    ['my name is (.*)', ['Hello !']],
    ['(hi|hello|hey|holla|hola)', ['Hey there !', 'Hi there !', 'Hey !']],
    ['how are you ?',['I am Fine! And You ?']],
    ['i am fine',['Great! \nHow May I Help You ?']],
    ['fine',['Great! \nHow May I Help You ?']],
    ['i am good',['Great! \nHow May I Help You ?']],
    ['(.*) your name ?', ['My name is Alen']],
    ['(.*) do you do ?', ['Making robo army !']],
    ['(.*) created you ?', ['--- nobody created me i am the creator']],
    ['(.*) my birthday',['Happy Birthday to you']],
    ['thank you',['You are welcome']],
    ['quit',['Bye! Nice talking To You. \nTake Care. \nHave A Nice Day']]
]

chat = Chat(pairs, reflections)
chat.converse()
```