Table of Contents

Prog 1 : Write down two intelligent programs for TIC-TAC-TOE problem.

Theory :

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper-and-pencil game for two players, X and O, who take 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.
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, 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.

Intelligent Program 1

- A 9 element vector representing the board is taken. We use 2 to indicate a blank, 3 to indicate an X and 5 to indicate an O.
- We make use of Make2, Posswin(p), Go(n).

Intelligent Program 2

- The board representation changes to :

8	3	4
1	5	9
6	7	2

- Rest of the algorithm remains the same.

Findings and Learnings

We learn about the increase in complexity, use of generalisations, clarity of knowledge etc.

Prog 2 : Write down program to implement Breadth first search and depth first search for water jug problem

Theory:

Water Jug Problem

You are given two jugs, a 4-gallon one and a 3-gallon one.Neither has any measuring mark on it.There is a pump that can be used to fill the jugs with water.How can you get exactly 2 gallons of water into the 4-gallon jug.

The problem has the following production system:

Rule	**State**	**Process**
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}
5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

Breadth First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbours.
BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse and Michael Burke, in their (rejected) Ph.D. thesis on the Plankalkül programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, and later developed by C. Y. Lee into a wire routing algorithm (published 1961).

Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

A version of depth-first search was investigated in the 19th century by French mathematician Charles Pierre Trémaux as a strategy for solving mazes.

Findings and Learnings

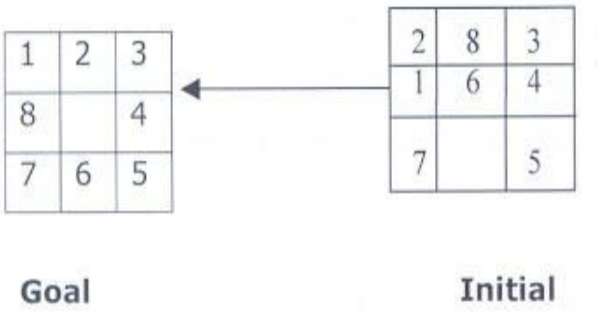
We learn how to use production systems to explore the state space via brute force graph searching algorithms.

Prog 3 : Implement Best first search for 8-Puzzle problem

Theory

8 Puzzle Problem

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.



The program is to change the initial configuration into the goal configuration. A solution to the problem is an appropriate sequence of moves, such as “move tiles 5 to the right, move tile 7 to the left ,move tile 6 to the down, etc”.

Best First Search

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
Judea Pearl described best-first search as estimating the promise of node n by a "heuristic evaluation function f(n) which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

In this program we use the heuristic function as :

$$h(n) = \Sigma \{+1 \; if \; state(x,y) \; matches \; GOAL(x,y) \; otherwise \; -1\}$$

Findings and Learning

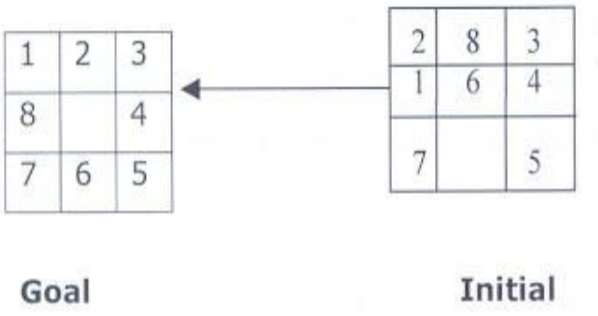
We learn how to use heuristic functions to solve complex problems using a best first search.

Prog 4 : Implement the steps of A* Algorithm for 8-Puzzle problem.

Theory

8 Puzzle Problem

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.



The program is to change the initial configuration into the goal configuration. A solution to the problem is an appropriate sequence of moves, such as “move tiles 5 to the right, move tile 7 to the left ,move tile 6 to the down, etc”.

A* Algorithm

A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node. At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes :

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

where n is the last node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must

be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

In our program :

g(n) = depth of the node

h(n) = sum of differences of relative position of each tile from goal to n

Finding and Learning

We learn how to use admissible heuristic functions to solve complex problems using a A* algorithm.

Prog 5 : Prolog programs for computing factorial of a number, area and circumference of a circle

Theory

Prolog is a general-purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel. Prolog was one of the first logic programming languages, and remains the most popular among such languages today, with several free and commercial implementations available. The language has been used for theorem proving, expert systems, term rewriting, type inference, and automated planning, as well as its original intended field of use, natural language processing. Modern Prolog environments support the creation of graphical user interfaces, as well as administrative and networked applications. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Findings and Learnings

We learn how to use prolog to form rule based logical queries.