- **Search**

The problem of trying to figure out what to do when we have some sort of situation that the computer is in. some sort of environment that an agent is in, so to speak.
& we would like for that agent to be able to somehow look for a solution to that problem.

→ These problems can come in any number of different types of formats.

Ex = Solve the puzzle games, maze games also Google map for showing all the paths long-ways or shortcut there is some type of 'search algorithm' performing to complete the task.

Agent → entity that percevies its environment & acts upon that environment.

State → a configuration of the agent and its environment.

inital state → the state in which the agent begins.

actions → choices that can be made in a state.
* ACTIONS(s) returns the set of actions that can be executed in state s.

Teacher's Signature .....

- ## Search

The problem of trying to figure out what to do when we have some sort of situation that the computer is in, some sort of environment that an agent is in, so to speak.
& we would like for that agent to be able to somehow look for a solution to that problem.

→ These problems can come in any number of different types of formats.

Ex = Solve the puzzle games, maze games also Google map for showing all the paths long-ways or shortcut there is some type of 'search algorithm' performing to complete the task.

Agent → entity that percevies its environment & acts upon that environment.

State → a configuration of the agent and its environment.

inital state → the state in which the agent begins.

actions → choices that can be made in a state.
* ACTIONS(s) returns the set of actions that can be executed in state s.

transition model → a description of what state results from performing any applicable action in any state.

* RESULT(s,a) returns the state resulting from performing action a in state s.

state space → the set of all states reachable from the initial state by any sequence of actions.

goal test → way to determine whether a given state is a goal state.

path cost → numerical associated with a given path.

Search problems consist of –

(i) initial state (ii) actions (iii) transition model (iv) goal test (v) path cost function

solution → a sequence of actions that lead from initial state to a goal state.

optimal solution → a solution that has the lowest path cost among all solutions.

node → a data structure that keeps track of –
- a state
- a parent (a node that generate this node).
- an action (action applied to parent to get node).
- a path cost (from initial state to node).

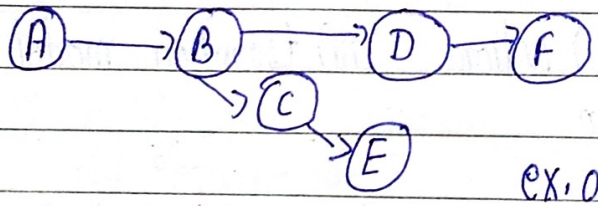
Teacher's Signature .....

Frontier → It represents all of the things that we could explore next, that we have not yet explored or visited.

- Approach to solve search problems -

→ Start with a frontier that contains the inital state.
→ Repeat:
  - if the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - if node contains goal state, return the solution.
  - Expand node, add resulting nodes to the frontier.

Find a path from A to E.



ex. 01

Frontier →

A → B → C, D → C → E

Here we have seen we are constantly adding the state in the frontier from inital state to meet the goal state by applying the actions.

What could go wrong?

(A) ⟷ (B) rest same previous question.

Frontier –
(A) → (B) → (A), (C), (D) → (A)

This will create a infinite loop.

- To tackle these kind of situations, a revised approach to search problem–

→ start with a frontier that contains the inital state.
→ start with an empty explored set.
→ Repeat:
  - if the frontier is empty, then no solution.
  - Remove a node from the frontier.
  - if node contains goal state, return the solution.
  - Add the node to the explored set.
  - Expand node. add resulting nodes to the frontier if they are not already in the frontier or the explored set.

stack → last-in first-out data type.

---

with the revised approach

(DFS to stack LIFO)

Frontier – (A) → (B) → (A), (C), (D) → (D) → (F), (C) → (E)

Explored set – (A) (B) (D) (F) (C) (E)

Teacher's Signature

We went very deep in this search tree, so to speak, all the way untill the bottom where we hit a dead end. And then, effectively backed up and explored the other route that we didn't tried before. And its this going very deep in the search tree idea, this way the algorithm ends up working when we use a stack, that we call this version of algorithm 'Depth-First Search' (DFS).

DFS → Search algorithm that always explore the deepest node in the frontier.

'Breadth-First Search' (BFS) →

- It behaves very similarly to DFS with one difference.
- Search algorithm that always expands the shallowest node in the frontier.
- It means instedd of using a stack, which DFS used where the most recent item added to the frontier is the one we'll explore next, in BFS will instead use a queue where a queue is a first-in, first out (FIFO) data-type.

X——————X——————X——————

Solving the Ex.01 with BFS.

Frontier — A → B → C, D → ~~E~~ → D → F

Explored set — A B C D E

- We use DFS to save memory.
- We use BFS to save time.

There are two different types of search algorithm -

(i) Uninformed Search (ii) Informed Search

US → Search strategy that uses no problem-specific Knowledge.
Ex = BFS, DFS

IS → search strategy that uses problem-specific Knowledge to find solutions more efficiently
Ex = greedy best-fit search (GBFS)

GBFS → search algorithm that expands the node that is closet to the goal, as estimated by a heuristic function h(n).

h(n) → that takes a state of input & returns our estimate of how close we are to the goal.

→ GBFS is not always give the optimal solution.

A* search →

- search algorithm that expands node with lowest value of $g(n) + h(n)$.

  $g(n)$ = Cost to reach goal.
  $h(n)$ = estimated cost to goal.

→ It gives us the optimal sol. because, it did find us the quickest possible way to get from the initial state.

- optimal if -

→ $h(n)$ is admissible (never overestimates the true cost)
→ $h(n)$ is consistent (for every node n & successor n' with step cost c, $h(n) \leq h(n') + c$).

- It does have a tendency to use quit a bit of memory, so, there are alternative approaches to A* that ultimately use less memory than this version of A* happens to use.

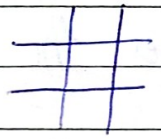
To work on the adversorial search problem (Tic tac toe) we work on different types of algorithms -

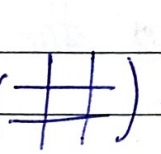
Mini max Algorithm

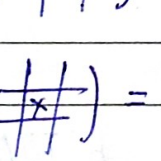
Minimax → (Tic tac toe ex.)

- MAX (x) aims to maximize score.
- MIN (o) aims to minimize score.

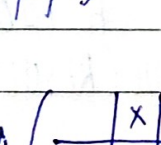
So : inital state
Player(s) : returns which player to move in state s
Actions(s) : returns legal move in state s.
Results(s,a) : returns state after action a taken in state s.
Terminal (s) : check if state is is a terminal state.
Utility (s) : final numerical value for terminal state s.

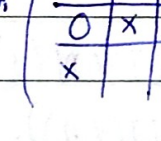
Intial state: 

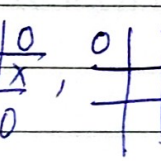
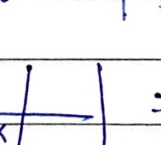
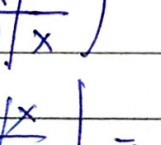
Player(s): Player() = X

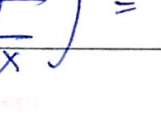
Player() = O

Utility(s);

Utility() = 1

Utility() = -1

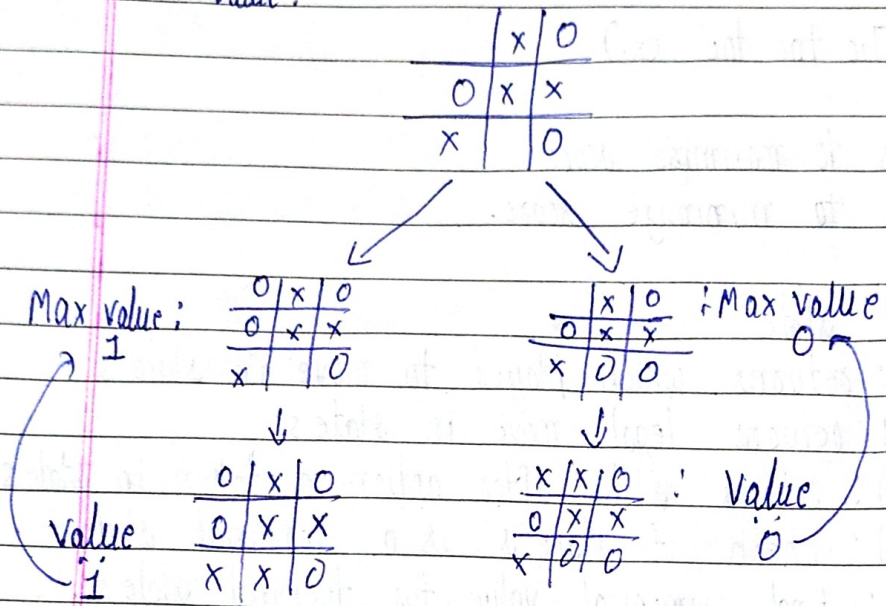
Actions(s): Action() = {  ,  }

Result(s,a): ( ,) =

Terminal(s): () = False

() = True

Min value:



- the min. player always chooses the option of smallest value.
- the max. player always chooses the option of highest value.

Given a state s:

- MAX picks action a in ACTION(s) that produces highest value of MIN-VALUE(RESULT(s,a)).
- MIN picks action a in ACTION(s) that produces smallest value of MAX-VALUE(RESULT(s,a)).

Minimax: Pseudo-Code

```
function MAX-VALUE(state):
    if TERMINAL(state)
        return UTILITY(state)
    v = -∞
    for action in ACTIONS(state):
        v = MAX(v, MIN-VALUE(RESULT(state, action)))
    return v

function MIN-VALUE(state):
    if TERMINAL(state):
        return UTILITY(state)
    v = ∞
    for action in ACTIONS(state):
        v = MIN(v, MAX-VALUE(RESULT(state, action)))
    return v
```

---

evaluation function → function that estimates the expected utility of the game for a given state.