AI UNIT 1:

PROBLEM SOLVING:

IDA* Algorithm:

IDA* = A* + Iterative Deepening DFS

The key feature of the IDA* algorithm is that it doesn’t keep a track of each visited node
which helps in saving memory consumption and can be used where memory is constrained.

It is path search algorithm which finds shortest path from start node to goal node in weighted graph.

It borrows the idea to use a heuristic function from A*

Its memory usage is weaker than in A*, thereby reducing memory problem in A*

IDA* is optimal in terms of space and cost

Problem: Excessive node generation

Algorithm:

Set certain threshold/f-bound

If f(node) > threshold/f-bound, prune the node

Set threshold/f-bound = minimum cost of any node that is pruned

Terminates when goal is reached.

Properties of IDA*:

Complete: Yes, similar to A*

Time: Depends strongly on the number of different values that the heuristic value can take on. If A* expands N nodes, IDA* expands 1+2+……+N=O(N2) nodes.

Space: It is DFS, it only requires space proportional to the longest path it explores.

Optimal: Yes, similar to A*.

Adversarial Search:

Adversarial search is a game-playing technique where the agents are surrounded by a competitive environment.

– A conflicting goal is given to the agents (multi agent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
– Here, game-playing means discussing those games where human intelligence and logic factor is used, excluding other factors such as luck factor.
– Tic-tac-toe, chess, checkers, etc., are such type of games where no luck factor works, only mind works.
–Mathematically, this search is based on the concept of 'Game Theory.'
– According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.

Perfect decision games:
In game theory, a sequential game has perfect information if each player, when making any decision, is perfectly informed of all the events that have previously occurred,including the "initialization event" of the game (e.g. the starting hands of each player in a card game).
– Perfect information is importantly different from complete information, which implies common knowledge of each player's utility functions, payoffs, strategies and "types". A game with perfect information may or may not have complete information.
– Chess is an example of a game with perfect information as each player can see all the pieces on the board at all times.
– Other examples of games with perfect information include tic-tac-toe, checkers, infinite chess.

Imperfect decision games:
In game theory, a game is imperfect game if each player, when making any decision, is uninformed of all the events that have previously occurred.

Imperfect information implies no idea of the move taken by the opponent it means the player is unaware of the actions chosen by other players.
Card games where each player's cards are hidden from other players such as poker and bridge are examples of games with imperfect information

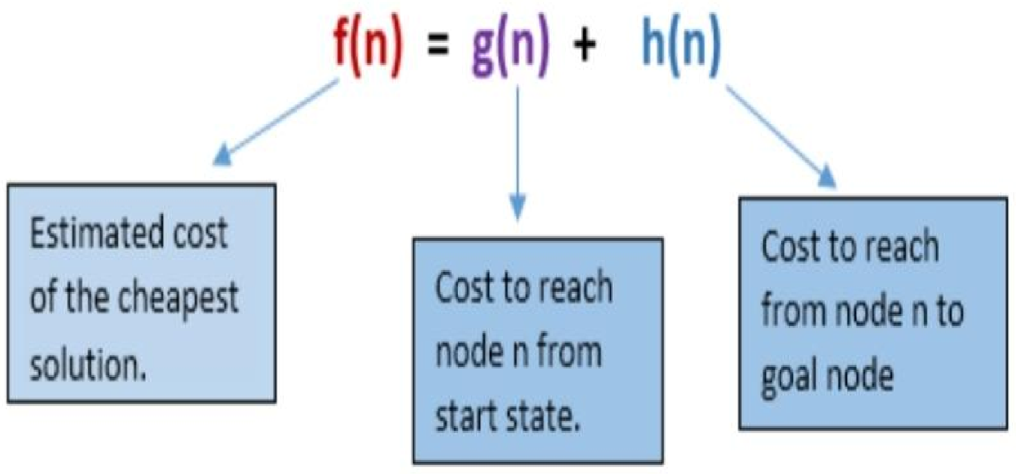
Evaluation Functions:
An Evaluation Function:
estimates how good the current board configuration is for a player.
–Typically, one figures how good it is for the player, and how good it is for the opponent,and subtracts the opponents score from the players
– Tic-tac-toe : Number of successful moves by X – Number of successful moves by O
– Chess: Value of all white pieces - Value of all black pieces
– Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
– If the board evaluation is X for a player, it's -X for the opponent.
– Many clever ideas about how to use the evaluation function.
– e.g. null move heuristic: let opponent move twice.
Example:
Evaluating chess boards,
Tic-tac-toe
The static heuristic evaluation function of Tic-tac-toe

α-β Pruning
Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
– As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree.
– Since we cannot eliminate the exponent, but we can cut it to half.

–Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning.
–This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm.
– Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
The two-parameter can be defined as:
– Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
– Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.
– The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.
– Using α-β Pruning improve search by reducing the size of the game tree.
Principle of α-β Pruning
If a move is determined worse than another move already examined, then there is no need for further examination of the node.
Rules of Thumb
α is the best (highest) found so far along the path for Max
β is the best (lowest) found so far along the path for Min
Search below a MIN node may be alpha-pruned if the its β of some MAX ancestor
Search below a MAX node may be beta-pruned if the its β of some MIN ancestor.

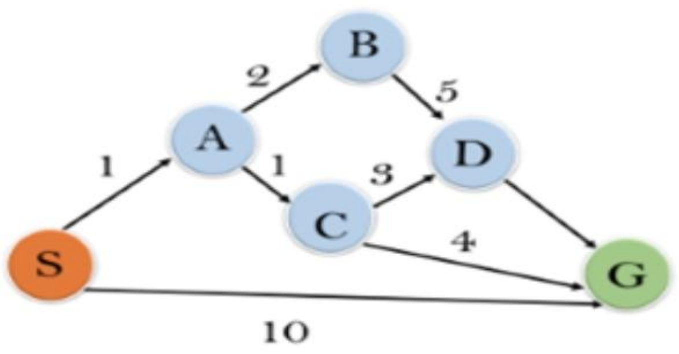
A*search

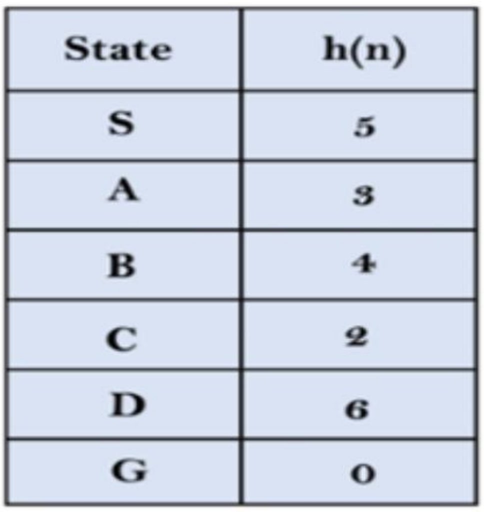
A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
– A* search algorithm finds the shortest path through the search space using the heuristic function.
– This search algorithm expands less search tree and provides optimal result faster.
– In A* search algorithm, we use search heuristic as well as the cost to reach the node.
–Hence we can combine both costs as following, and this sum is called as a fitness number.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

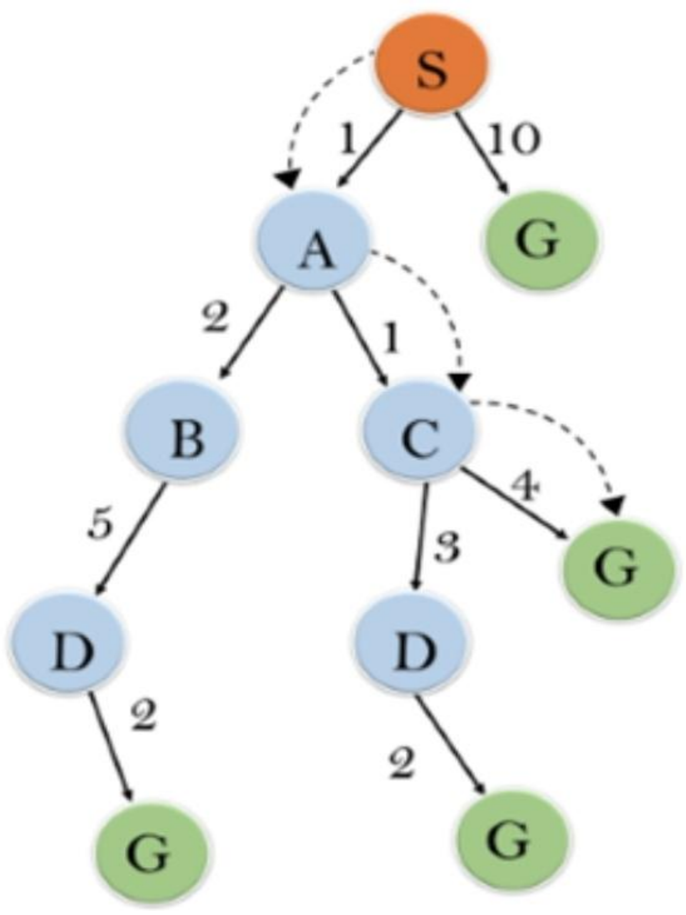
Algorithm of A* search:
Step1: Place the starting node in the OPEN list.
Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
Step 6: Return to Step 2.
Advantages:
A* search algorithm is the best algorithm than other search algorithms.
A* search algorithm is optimal and complete.
This algorithm can solve very complex problems.
Disadvantages:
It does not always produce the shortest path as it mostly based on heuristics and approximation.
A* search algorithm has some complexity issues.
The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.
Example:
Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.
In this search example, we are using two lists which are OPEN and CLOSED Lists. Following are the iteration for traversing the above example.





State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



Start State: {(S, 5)}
Iteration1: {(S--> A, 4), (S-->G, 10)}
Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}
Iteration 3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}
Iteration 4 will give the final result, as S--->A--->C--->G it provides the optimal path with cost 6
So A* avoid expanding paths that are already expensive
Optimal: A* search algorithm is optimal if it follows below two conditions
Admissible: the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
Consistency: Second required condition is consistency for only A* graph-search.