CS3491 – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIT 1 – PROBLEM SOLVING

SYLLABUS:

Introduction to AI - AI Applications - Problem solving agents – search algorithms – uninformed search strategies – Heuristic search strategies – Local search and optimization problems – adversarial search – constraint satisfaction problems (CSP)

PART A

1. What is Artificial Intelligence? Or Define Artificial Intelligence.

- Artificial intelligence is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.
- Artificial Intelligence is the process of building intelligent machines from vast volumes of data.
- Systems learn from past learning and experiences and perform human-like tasks.
- It enhances the speed, precision, and effectiveness of human efforts.

2. How Artificial Intelligence is Defined? List the Types Of Approaches for Artificial Intelligence.

- Thinking humanly: mimicking thought based on the human mind.
- Thinking rationally: mimicking thought based on logical reasoning.
- Acting humanly: acting in a manner that mimics human behavior.
- Acting rationally: acting in a manner that is meant to achieve a particular goal.

3. What Are The Four Types Of Artificial Intelligence?

- Reactive machines: able to perceive and react to the world in front of it as it performs limited tasks.
- Limited memory: able to store past data and predictions to inform predictions of what may come next.
- Theory of mind: able to make decisions based on its perceptions of how others feel and make decisions.

- Self-awareness: able to operate with human-level consciousness and understand its own existence.

4. List the types of Artificial Intelligence-based on capabilities.

- Narrow AI
- General AI
- Super AI

5. List the types of Artificial Intelligence-based on functionalities -

- Purely Reactive or Reactive Machines
- Limited Memory
- Theory of Mind
- Self Awareness

6. Tabulate the comparison of Artificial Intelligence-based on functionalities.

Reactive Machines	**Limited Memory**	**Theory of Mind**	**Self-Awareness**
Simple classification and pattern recognition tasks	Complex classification tasks	Understands human reasoning and motives	Human-level intelligence that can by-pass human intelligence too
Great when all parameters are known	Uses historical data to make predictions	Needs fewer examples to learn because it understands motives	Sense of self-consciousness
Can't deal with imperfect information	Current state of AI	Next milestone for the evolution of AI	Does not exist yet

7. List the Four possible goals to pursue in artificial intelligence.

- Systems that think like humans.
- Systems that think rationally.
- Systems that act like humans
- Systems that act rationally

8. Point out the subfields of Artificial Intelligence.

- Machine Learning
- Deep Learning
- Neural Networks
- Cognitive Computing
- Computer Vision
- Natural Language Processing

9. *What are the advantages of Artificial Intelligence?*

- Reduced human error
- Risk avoidance
- Replacing repetitive jobs
- Digital assistance

10. *List the limitations of Artificial Intelligence*

- High cost of creation.
- No emotions.
- Box thinking.
- Can't think for itself.

11. *Differentiate Deep learning vs. machine learning*

- **Machine Learning**
  - Machine Learning is the learning in which a machine can learn on its own from examples and previous experiences.
  - It is machine learning that gives AI the ability to learn.
  - This is done by using algorithms to discover patterns and generate insights from the data they are exposed to.
- **Deep Learning**
  - Deep learning, is a subcategory of machine learning, provides AI with the ability to mimic a human brain's neural network.
  - It can make sense of patterns, noise, and sources of confusion in the data.

12. *Compare Data Science, Artificial Intelligence and Machine Learning.*

Data Science	Artificial Intelligence	Machine Learning
Data Science is used for data sourcing, cleaning, processing, and visualizing for analytical purposes.	AI combines iterative processing and intelligent algorithms to imitate the human brain's functions.	Machine Learning is a part of AI where mathematical models are used to empower a machine to learn with or without being programmed regularly.
Data Science deals with both structured and unstructured data for analytics.	AI uses decision trees and logic theories to find the best possible solution to the given problem.	Machine Learning utilizes statistical models and neural networks to train a machine.
Some of the popular tools in Data Science are Tableau, SAS2, Apache, MATLAB, Spark, and more.	Some of the popular libraries include Keras, Scikit-Learn, and TensorFlow.	As a subset of AI, Machine Learning also use the same libraries, along with tools such as Amazon Lex2, IBM Watson, and Azure ML Studio.

Data Science includes data operations based on user requirements.	AI includes predictive modeling to predict events based on the previous and current data.	ML is a subset of Artificial Intelligence.
It is mainly used in fraud detection, healthcare, BI analysis, and more.	Applications of AI include chat bots, voice assistants, and weather prediction.	Online recommendations, facial recognition, and NLP are a few examples of ML.

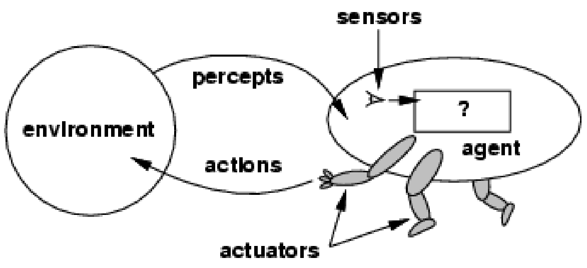
13. List the various Applications of Artificial Intelligence or AI Applications.

- AI Application in E-Commerce
  - Personalized Shopping
  - AI-powered Assistants
  - Fraud Prevention
- AI Application in Education
  - Administrative Tasks Automated to Aid Educators
  - Creating Smart Content
  - Voice Assistants
  - Personalized Learning
- AI Application in Lifestyle
  - Autonomous Vehicles
  - Spam Filters
  - Facial Recognition
  - Recommendation System
- AI Application in Navigation
- AI Application in Robotics
  - Carrying goods in hospitals, factories, and warehouses
  - Cleaning offices and large equipment
  - Inventory management
- AI Application in Human Resource
- AI Application in Healthcare
- AI Application in Agriculture
- AI Application in Gaming
- AI Application in Automobiles
- AI Application in Social Media
  - Instagram
  - Facebook
  - Twitter
- AI Application in Marketing
- AI Application in Chatbots

14. Define an Agent and list its types.

Agent

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.



Types of Agent

- **Simple reflex agents** respond directly to percepts.
- **Model-based reflex** agents maintain internal state to track aspects of the world that are not evident in the current percept.
- **Goal-based agents** act to achieve their goals, and
- **Utility-based agents** try to maximize their own expected "happiness."

15. Define an environment and task environment.

Environments

- The environment could be everything—the entire universe.

Task Environment

- A task environment specification includes the performance measure, the external environment, the actuators, and the sensors.

16. List the steps in designing an agent

- In designing an agent, the first step must always be to specify the task environment as fully as possible.
- The agent program implements the agent function.
- There exists a variety of basic agent program designs reflecting the kind of information made explicit and used in the decision process.
- The designs vary in efficiency, compactness, and flexibility.
- The appropriate design of the agent program depends on the nature of the environment.
- All agents can improve their performance through learning.

17. Define Problem Solving Agent. List the steps involved in Problem Solving Agent.

➢ **Problem Solving Agent**

- Problem-solving agent is a goal-based agent that focuses on goals using a group of algorithms and techniques to solve a well-defined problem.

- An agent may need to plan a sequence of actions that form a path to a goal state.
- Such an agent is called a problem-solving agent, and the computational process it undertakes is called search.

➢ **Steps performed by Problem-solving agent**
- Goal Formulation
- Problem Formulation
- Search
- Solution
- Execution

18. List the Components in problem formulation.

a) **Initial State:** It is the starting state or initial step of the agent towards its goal.
b) **Actions:** It is the description of the possible actions available to the agent.
c) **Transition Model:** It describes what each action does.
d) **Goal Test:** It determines if the given state is a goal state.
e) **Path cost:** It assigns a numeric cost to each path that follows the goal.

19. Define search algorithms and list the types of search algorithm.

- A search algorithm takes a search problem as input and returns a solution.
- **Types of Search Algorithm**
  - **Uninformed Search Algorithms**
    - Depth First Search
    - Depth Limited Search
    - Breadth First Search
    - Iterative Deepening Search
    - Uniform Cost Search
    - Bidirectional Search
  - **Informed (Heuristic) Search Strategies**
    - Greedy Search
    - A* Tree Search

20. Define Depth First Search. List the advantages and disadvantages of DFS.

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

- It uses last in-first-out strategy and hence it is implemented using a stack.

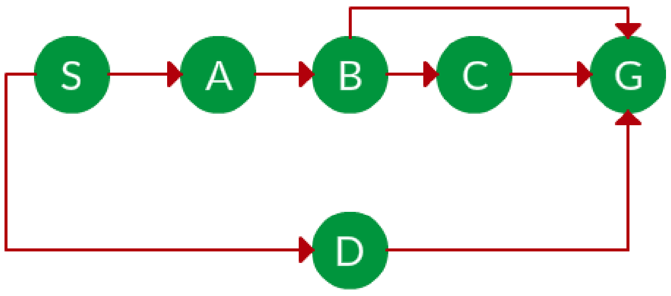
Advantage of Depth-first Search:

- DFS uses extremely little memory.
- It takes less time to reach the goal node than the BFS method.

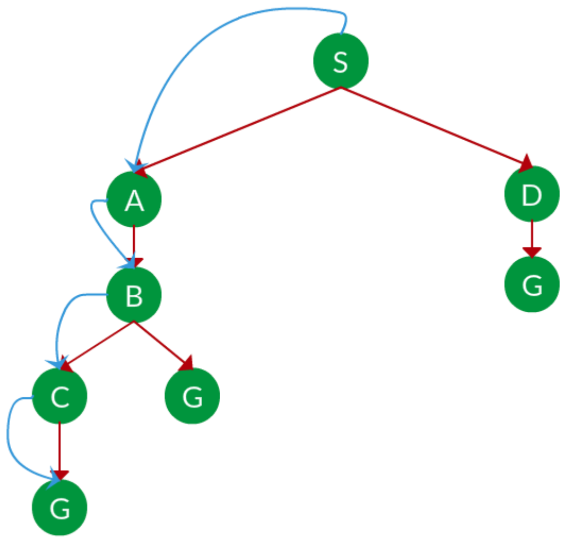
Disadvantage of Depth-first Search:

- There's a chance that many states will recur, and there's no certainty that a solution will be found.
- The DFS algorithm performs deep searching and may occasionally enter an infinite cycle.

21. Which solution would DFS find to move from node S to node G if run on the graph below?



Solution.



Path: S - > A -> B -> C -> G

22. Define Depth Limited Search. List its advantages and disadvantages.

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit.
- Depth-limited search can solve the drawback of the infinite path in the Depth-first search.

Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

23. Define Breadth-first search (BFS). List its advantages and disadvantages.

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

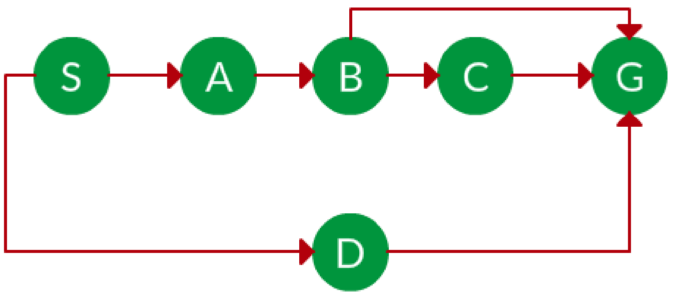
Advantages of Breadth-first Search:

- If a solution is available, BFS will provide it.
- If there are multiple answers to a problem, BFS will present the simplest solution with the fewest steps.

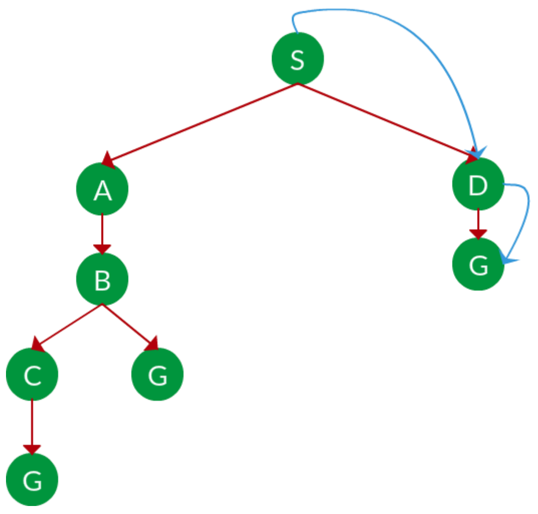
Disadvantages of Breadth-first Search:

- It necessitates a large amount of memory.
- If the solution is located far from the root node, BFS will take a long time.

24. Which solution would BFS find to move from node S to node G if run on the graph below?



Solution.



Path: S -> D -> G

25. Define iterative deepening depth-first search/ iterative deepening search

- This search is a combination of BFS and DFS, as BFS guarantees to reach the goal node and DFS occupies less memory space.
- Therefore, iterative deepening search combines these two advantages of BFS and DFS to reach the goal node.
- It gradually increases the depth-limit from 0,1,2 and so on and reach the goal node.

26. Define Uniform Cost Search. List its advantages and disadvantages.

- A searching algorithm for traversing a weighted tree or graph is uniform-cost search.
- When a separate cost is provided for each edge, this algorithm is used.
- The uniform-cost search's main purpose is to discover the shortest path to the goal node with the lowest cumulative cost.
- **Cost of a node** is defined as:

cost(node) = cumulative cost of all nodes from root

cost(root) = 0

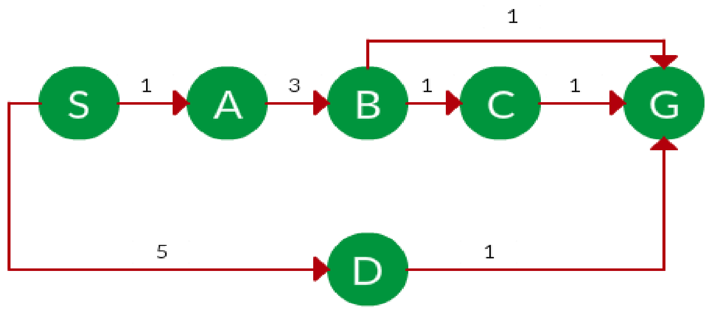
Advantages of Uniform-cost Search:

- The path with the lowest cost is chosen at each state.
- UCS is complete only if states are finite and there should be no loop with zero weight.
- UCS is optimal only if there is no negative cost.

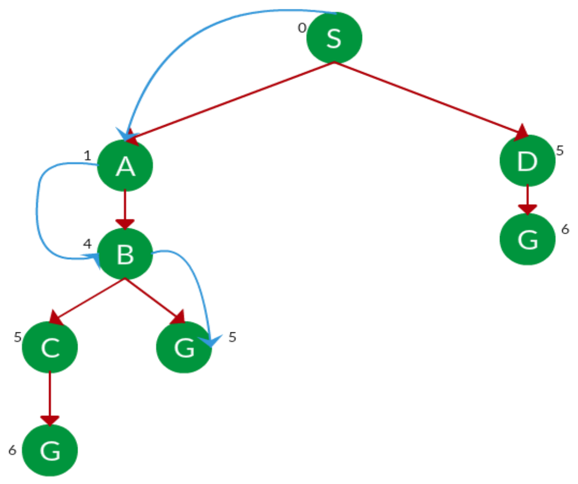
Disadvantages of Uniform-cost Search:

- It is just concerned with the expense of the path.

27. Which solution would UCS find to move from node S to node G if run on the graph below?



Solution.



Path: *S -> A -> B -> G*
Cost: *5*

28. Define Bidirectional Search.

- The strategy behind the bidirectional search is to run two searches simultaneously**--one forward search from the initial state and other from the backside of the goal**--hoping that both searches will meet in the middle.

29. Define informed (heuristic) search strategies and mention its types.

- Here, the algorithms have information on the goal state, which helps in more efficient searching.
- This information is obtained by something called a *heuristic.*

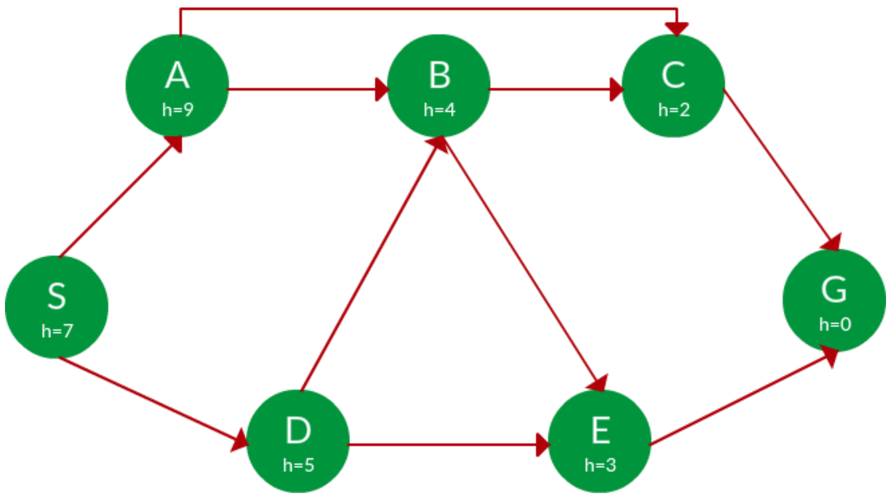
Search Heuristics:

- In an informed search, a heuristic is a *function h(n)* estimates how close a state is to the goal state.
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state.

Types of Informed search algorithms.

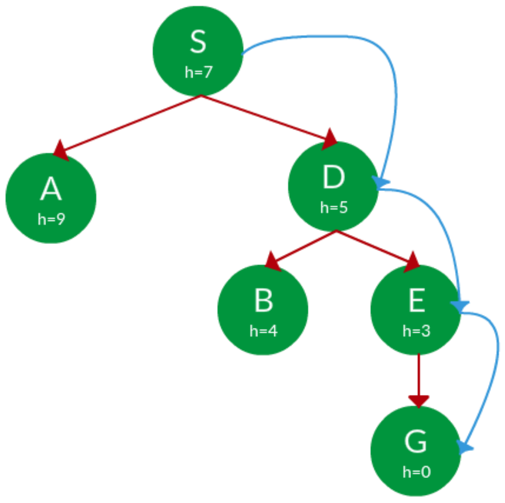
- Greedy Search
- A* Tree Search
- A* Graph Search

30. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.



Solution.

Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.

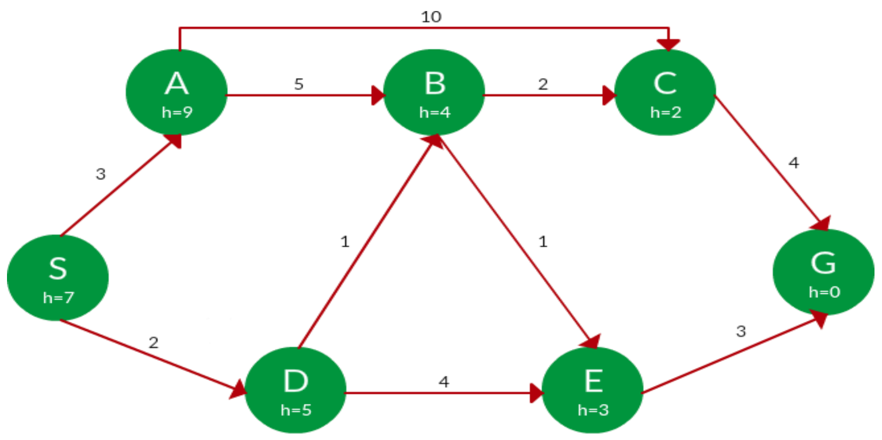


Path: S -> D -> E -> G

31. Define A* tree search.

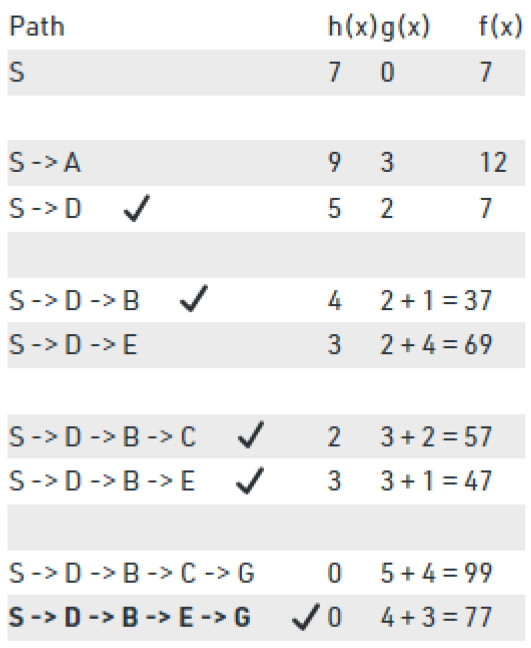
- A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search.
- In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in the greedy search, denoted by h(x). The summed cost is denoted by f(x).

32. Find the path to reach from S to G using A* search.



Solution.

- Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum.
- The entire work is shown in the table below.

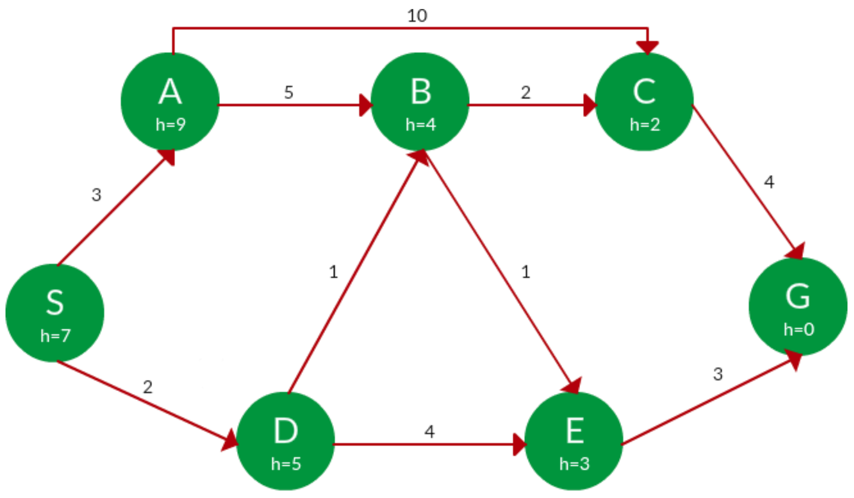


Path	h(x)	g(x)	f(x)
S	7	0	7
S -> A	9	3	12
S -> D ✓	5	2	7
S -> D -> B ✓	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C ✓	2	3 + 2 = 5	7
S -> D -> B -> E ✓	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G ✓	0	4 + 3 = 7	7

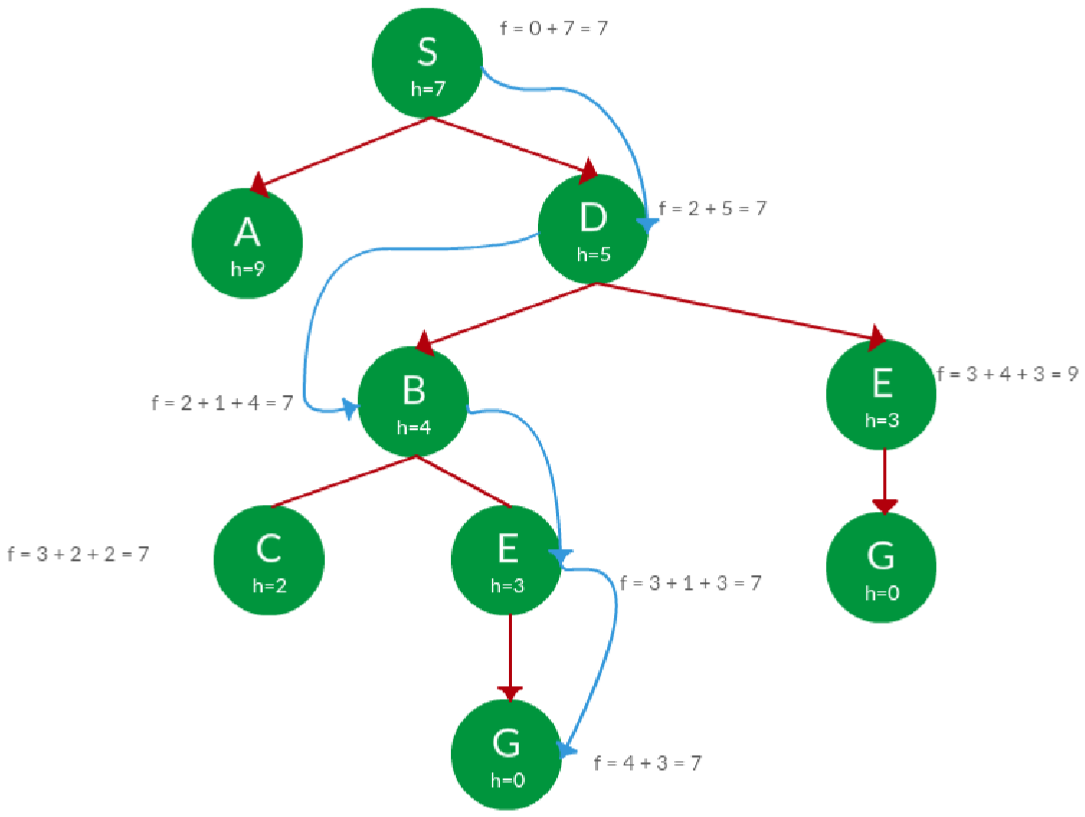
Path: S -> D -> B -> E -> G

Cost: 7

33. Use graph searches to find paths from S to G in the following graph.



Solution



Path: S -> D -> B -> E -> G

Cost: 7

34. Define Local Search Algorithm. List its types.

Local Search Algorithm - Definition

- **Local search** algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, or the set of states that have been reached.
- **"Local search algorithms" where the path cost does not matters, and only focus on solution-state needed to reach the goal node.**
- It has two key advantages:
  - o use very little memory;
  - o Often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Different types of local searches:

- Hill-climbing Search

  Types of Hill climbing search algorithm
  - Simple hill climbing
  - Steepest-ascent hill climbing
  - Stochastic hill climbing
  - Random-restart hill climbing
- Simulated Annealing
- Local Beam Search

35. What are the Limitations of Hill climbing algorithm.

- **Local Maxima:** It is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it.
- **Plateau:** It is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point.
- **Ridges:** It is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself.

36. Define 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 (multiagent).
- 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.

37. What are the Elements of Game Playing search?

- **S_0:** It is the initial state from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves to be used in a state.
- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**.
- The price which the winner will get i.e.
  - **(-1):** If the PLAYER loses.
  - **(+1):** If the PLAYER wins.
  - **(0):** If there is a draw between the PLAYERS.

38 List the types of algorithms in adversarial search.

- Minimax Algorithm
- Alpha-beta Pruning

39. Define a Game Tree.

- **A game tree** is where the nodes represent the states of the game and edges represent the moves made by the players in the game.
- Players will be two namely:
  - **MIN:** Decrease the chances to win the game.
  - **MAX:** Increases his chances of winning the game.
- MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

40. Define Constraint satisfaction problems (CSP) – Definition.

- A **constraint satisfaction problem (CSP)** is a problem that requires its solution within some limitations or conditions also known as constraints.
- It consists of the following:
  - A finite set of **variables** which stores the solution (V = {V1, V2, V3, ..., Vn})
  - A set of **discrete** values known as **domain** from which the solution is picked (D = {D1, D2, D3, ...,Dn})
  - A finite set of **constraints** (C = {C1, C2, C3, ..., Cn})

41. List the CSP Algorithms and Problems.

1. CryptArithmetic (Coding alphabets to numbers.)
2. n-Queen (In an n-queen problem, n queens should be placed in an nXn matrix such that no queen shares the same row, column or diagonal.)
3. Map Coloring (coloring different regions of map, ensuring no adjacent regions have the same color)
4. Crossword (everyday puzzles appearing in newspapers)
5. Sudoku (a number grid)
6. Latin Square Problem

42. List the Types of Domains in CSP.

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example,** a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

43. List the Constraint Types in CSP.

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value $\mathbf{x_2}$ will contain a value which lies between **x1** and **x3**.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

44. Define Map Coloring / Graph Coloring Problem.

- It is a map of **Australia**; it consists of states and territories as in Figure 1.19.
- The task will be coloring each region with the colors either red, green or blue.
- In such case, no neighboring regions will have the same color.

PART B

1. Define Artificial Intelligence and its types, goals, advantages and limitations in detail.

ARTIFICIAL INTELLIGENCE:

1.1 Artificial Intelligence Definition

- Artificial Intelligence is the process of building intelligent machines from vast volumes of data.
- Systems learn from past learning and experiences and perform human-like tasks.
- It enhances the speed, precision, and effectiveness of human efforts.
- AI uses complex algorithms and methods to build machines that can make decisions on their own as shown in Figure 1.1.
- Machine Learning and Deep learning forms the core of Artificial Intelligence.

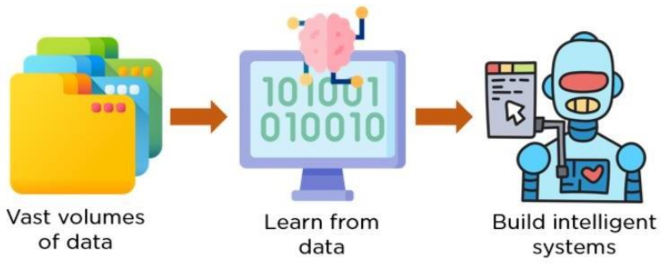


Figure 1.1 - Artificial Intelligence

1.2 Types of Artificial Intelligence

- Artificial Intelligence can be divided based on capabilities and functionalities as in figure 1.2.
- Types of Artificial Intelligence-based on capabilities -
    - Narrow AI
    - General AI
    - Super AI

- Types of Artificial Intelligence-based on functionalities -
  - Reactive Machines
  - Limited Memory
  - Theory of Mind
  - Self Awareness

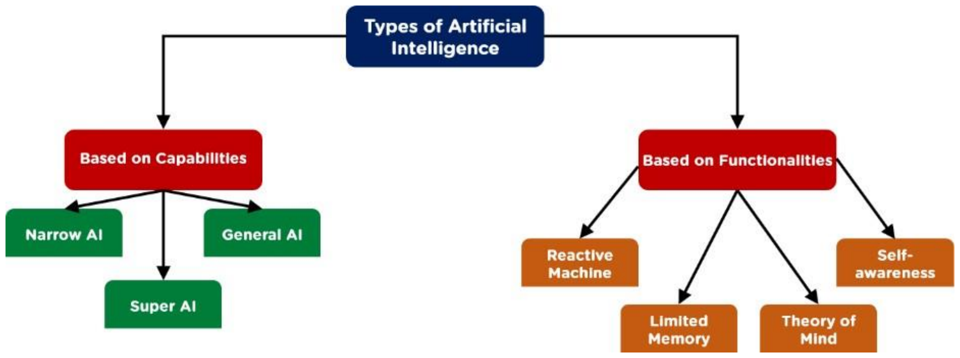


Figure 1.2 - Types of Artificial Intelligence

1.2.1 Types of Artificial Intelligence-based on capabilities -

- Narrow AI
- General AI
- Super AI

- **Narrow AI**:
  - Referred as Weak AI, focuses on one automate specific task and cannot perform beyond its limitations.
  - Example: Alexa and other smart assistants, Self-driving cars, Google search, Email spam filters, Netflix's recommendations,
- **Artificial General Intelligence (AGI)**:
  - Referred as “strong AI,” can understand and learn any intellectual task that a human being can.
  - Example: Fujitsu has built the K computer, which is one of the fastest supercomputers in the world
- **Super Intelligence**:
  - Super AI surpasses human intelligence and can perform any task better than a human.
  - Some of the critical characteristics of super AI include thinking, solving puzzles, making judgments, and decisions on its own.

1.2.2 Types of Artificial Intelligence-based on functionalities -

- Purely Reactive or Reactive Machines
- Limited Memory
- Theory of Mind
- Self Awareness

- **Purely Reactive or Reactive Machines**
  - These machines do not have any memory or data to work with, specializing in just one field of work.
  - For example, in a chess game, the machine observes the moves and makes the best possible decision to win.
- **Limited Memory**
  - These machines collect previous data and continue adding it to their memory.
  - They have enough memory or experience to make proper decisions, but memory is minimal.
  - For example, this machine can suggest a restaurant based on the location data that has been gathered.
- **Theory of Mind**
  - This kind of AI can understand thoughts and emotions, as well as interact socially. However, a machine based on this type is yet to be built.
- **Self-Aware**
  - Self-aware machines are the future generation of the new technologies. It will be intelligent, sentient, and conscious.

Table 1.1 - Comparison of types of Artificial Intelligence-based on functionalities:

Reactive Machines	**Limited Memory**	**Theory of Mind**	**Self-Awareness**
Simple classification and pattern recognition tasks	Complex classification tasks	Understands human reasoning and motives	Human-level intelligence that can by-pass human intelligence too
Great when all parameters are known	Uses historical data to make predictions	Needs fewer examples to learn because it understands motives	Sense of self-consciousness
Can't deal with imperfect information	Current state of AI	Next milestone for the evolution of AI	Does not exist yet

1.3 Four possible goals to pursue in artificial intelligence:

- Systems that think like humans.
- Systems that think rationally.
- Systems that act like humans
- Systems that act rationally
  - Refer Figure 1.3

Four Approaches from two dimensions

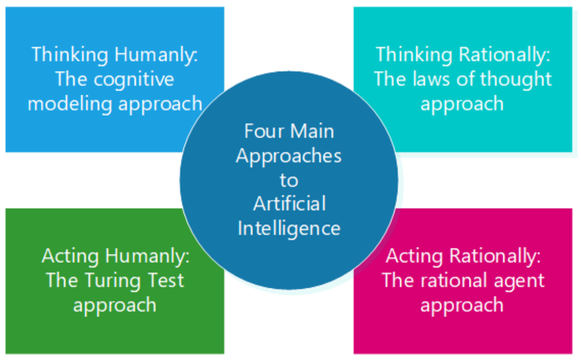

Thinking Humanly: The cognitive modeling approach
Thinking Rationally: The laws of thought approach
Four Main Approaches to Artificial Intelligence
Acting Humanly: The Turing Test approach
Acting Rationally: The rational agent approach

Figure 1.3 - Four Approaches from two dimensions

1.3.1 Acting humanly: The Turing test approach

- The **Turing test**, proposed by Alan Turing (1950), was designed as a method for determining if a machine is intelligent or not.
- The computer would need the following capabilities:
  - **natural language processing** to communicate successfully in a human language;
  - **knowledge representation** to store what it knows or hears;
  - **automated reasoning** to answer questions and to draw new conclusions;
  - **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.
- However, other researchers have proposed a **total Turing test**.
- To pass the total Turing test, a robot will need
  - **computer vision** and speech recognition to perceive the world;
  - **robotics** to manipulate objects and move about.

1.3.2 Thinking humanly: The cognitive modeling approach

- Thinking humanly is to make a system or program to think like a human.
- Can learnabout human thought in three ways:
  - **Introspection**—trying to catch own thoughts as they go by;
  - **Psychological experiments**—observing a person in action;
  - **Brain imaging**—observing the brain in action.

Cognitive Science

- Cognitive science is the interdisciplinary study of mind and intelligence, embracing philosophy, psychology, artificial intelligence, neuroscience, linguistics, and anthropology.

1.3.3 Thinking rationally: The "laws of thought" approach

- The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking"—that is, irrefutable reasoning processes.
- His **syllogisms** provided patterns for argument structures that always yielded correct conclusions when given correct premises.
- These laws of thought were supposed to govern the operation of the mind; the study initiated the field called **logic**.
- By 1965, programs could solve *any* solvable problem described in logical notation. The so-called **logicist** tradition.
- The theory of **probability** allows rigorous reasoning with uncertain information.
- In principle, it allows the construction of a comprehensive model of rational thought, leading from raw perceptual information to an understanding of how the world works to predictions about the future.

1.3.4 Acting rationally: The rational agent approach

- An **agent** is just something that acts to do.
- Computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals.
- A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

1.4 Subfields of Artificial Intelligence

- **Machine Learning**
  - Machine Learning is the learning in which a machine can learn on its own from examples and previous experiences.
  - It is machine learning that gives AI the ability to learn.
  - This is done by using algorithms to discover patterns and generate insights from the data they are exposed to.

  Deep Learning
  - Deep learning, is a subcategory of machine learning, provides AI with the ability to mimic a human brain's neural network.
  - It can make sense of patterns, noise, and sources of confusion in the data.
- **Neural Networks**
  - Artificial Neural Networks (ANNs) are one of the most important tools in Machine Learning to find patterns within the data, which are far too complex for a human to figure out and teach the machine to recognize.

- **Cognitive Computing:**
  - The ultimate goal of cognitive computing is to imitate the human thought process in a computer model.
  - Using self-learning algorithms, pattern recognition by neural networks, and natural language processing, a computer can mimic the human way of thinking.
  - Here, computerized models are deployed to simulate the human cognition process.
- **Computer Vision:**
  - Computer vision works by allowing computers to see, recognize, and process images, the same way as human vision does, and then it provides an appropriate output.
  - The computer must understand what it sees, and then analyze it, accordingly.
- **Natural Language Processing:**
  - Natural language processing means developing methods that help us communicate with machines using natural human languages like English.

1.5 Advantages of Artificial Intelligence

- **Reduced human error:**
  - With humans involved in the tasks where precision is required, there will always be a chance of error. However, if programmed properly, machines do not make mistakes and easily perform repetitive tasks without making many errors, if not at all.
- **Risk avoidance:**
  - Replacing humans with intelligent robots is one of the biggest advantages of Artificial Intelligence.
- **Replacing repetitive jobs:**
  - Our day-to-day work includes many repetitive tasks that we have to do every day without any change. Now, machines have replaced these tasks so that humans can spend this time doing creative things.
- **Digital assistance:**
  - With digital assistants to interact with users 24/7, organizations can save the need for human resources and deliver faster service to customers.

1.6 Limitations of Artificial Intelligence

- High cost of creation.
- No emotions.
- Box thinking.
- Can't think for Itself.

2. Compare and differentiate between Data Science vs Artificial Intelligence Vs. Machine Learning.

2.1 Deep learning vs. machine learning

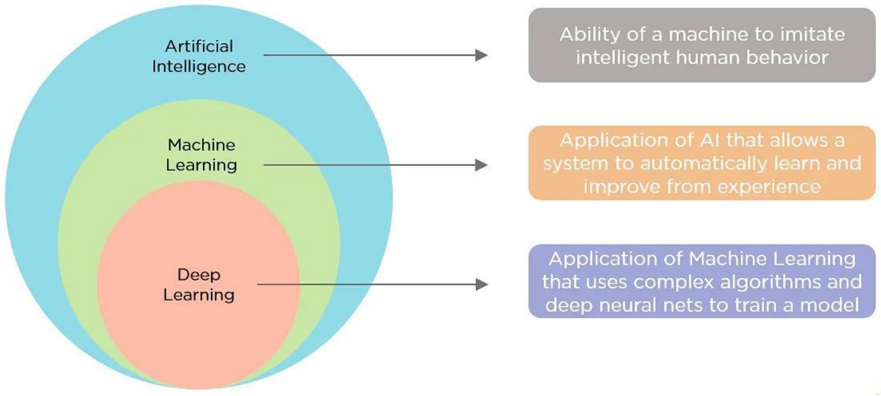

Artificial Intelligence
Machine Learning
Deep Learning
Ability of a machine to imitate intelligent human behavior
Application of AI that allows a system to automatically learn and improve from experience
Application of Machine Learning that uses complex algorithms and deep neural nets to train a model

Figure 1.4 - Deep learning vs. machine learning

- “Artificial intelligence is a set of algorithms and intelligence to try to mimic human intelligence.
- Machine learning is one of them, and deep learning is one of those machine learning techniques as shown in figure 1.4.”
- **Machine Learning**
    - Machine Learning is the learning in which a machine can learn on its own from examples and previous experiences.
    - It is machine learning that gives AI the ability to learn.
    - This is done by using algorithms to discover patterns and generate insights from the data they are exposed to.
- **Deep Learning**
    - Deep learning, is a subcategory of machine learning, provides AI with the ability to mimic a human brain’s neural network.
    - It can make sense of patterns, noise, and sources of confusion in the data.

2.2 Data Science vs. Artificial Intelligence vs. Machine Learning

- Data Science, Machine Learning, and Artificial Intelligence are interconnected, but each one of them uniquely serves a different purpose as mentioned in table 1.2

Table 1.2 - Comparison of Data Science, Artificial Intelligence and Machine Learning:

Data Science	Artificial Intelligence	Machine Learning
Data Science is used for data sourcing, cleaning, processing, and visualizing for analytical purposes.	AI combines iterative processing and intelligent algorithms to imitate the human brain's functions.	Machine Learning is a part of AI where mathematical models are used to empower a machine to learn with or without being programmed regularly.
Data Science deals with both structured and unstructured data for analytics.	AI uses decision trees and logic theories to find the best possible solution to the given problem.	Machine Learning utilizes statistical models and neural networks to train a machine.
Some of the popular tools in Data Science are Tableau, SAS2, Apache, MATLAB, Spark, and more.	Some of the popular libraries include Keras, Scikit-Learn, and TensorFlow.	As a subset of AI, Machine Learning also use the same libraries, along with tools such as Amazon Lex2, IBM Watson, and Azure ML Studio.
Data Science includes data operations based on user requirements.	AI includes predictive modeling to predict events based on the previous and current data.	ML is a subset of Artificial Intelligence.
It is mainly used in fraud detection, healthcare, BI analysis, and more.	Applications of AI include chat bots, voice assistants, and weather prediction.	Online recommendations, facial recognition, and NLP are a few examples of ML.

3. Explain the history of artificial intelligence.

- The important events and milestones in the evolution of artificial intelligence include the following:

  1943:
  - The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943).
  - Inspired by the mathematical modeling work of Pitts's advisor Nicolas they drew on three sources:
    - knowledge of the basic physiology and function of neurons in the brain;
    - a formal analysis of propositional logic;
    - Turing's theory of computation.

  1949:
  - Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons.
  - His rule, now called Hebbian learning, remains an influential model to this day.

1950:

- Alan Turing publishes *Computing Machinery and Intelligence.* In the paper, Turing—proposes to answer the question 'can machines think?' and introduces the Turing Test to determine if a computer can demonstrate the same intelligence as a human. The value of the Turing test has been debated ever since.

1956:

- John McCarthy coins the term 'artificial intelligence' at the first-ever AI conference at Dartmouth College. Later that year, Allen Newell, J.C. Shaw, and Herbert Simon create the Logic Theorist, the first-ever running AI software program.

1958:

- In 1958, John McCarthy had defined the high-level language Lisp, which was to become the dominant AI programming language for the next 30 years.

1959:

- At IBM, Nathaniel Rochester and his colleagues produced some of the first AI programs.
- Herbert Gelernter (1959) constructed the Geometry Theorem Prover, which was able to prove theorems that many students of mathematics would find quite tricky.

1967:

- Frank Rosenblatt builds the Mark 1 Perceptron, the first computer based on a neural network that 'learned' though trial and error.

1980s:

- Neural networks which use a back propagation algorithm to train itself become widely used in AI applications.

1997:

- IBM's Deep Blue beats then world chess champion Garry Kasparov, in a chess match (and rematch).

2011:

- IBM Watson beats champions Ken Jennings and Brad Rutter at *Jeopardy!*

2015:

- Baidu'sMinwa supercomputer uses a special kind of deep neural network called a convolution neural network to identify and categorize images with a higher rate of accuracy than the average human.

2016:

- DeepMind'sAlphaGo program, powered by a deep neural network, beats Lee Sodol, the world champion Go player, in a five-game match.

- The first "robot citizen," a humanoid robot named Sophia, is created by Hanson Robotics and is capable of facial recognition, verbal communication and facial expression.

2018:

- Google releases natural language processing engine BERT, reducing barriers in translation and understanding by ML applications.

2020:

- Baidu releases its Linear Fold AI algorithm to scientific and medical teams working to develop a vaccine during the early stages of the SARS-CoV-2 pandemic.
- OpenAI releases natural language processing model GPT-3, which is able to produce text modeled after the way people speak and write.

2022:

- The National Institute of Standards and Technology releases the first draft of its AI Risk Management Framework, voluntary U.S. guidance "to better manage risks to individuals, organizations, and society associated with artificial intelligence."
- DeepMind unveils Gato, an AI system trained to perform hundreds of tasks, including playing Atari, captioning images and using a robotic arm to stack blocks.

4. List and explain the various Applications of Artificial Intelligence or AI Applications.

APPLICATIONS OF AI:

4.1. AI Application in E-Commerce
4.2. AI Application in Education
4.3. AI Application in Lifestyle
4.4. AI Application in Navigation
4.5. AI Application in Robotics
4.6. AI Application in Human Resource
4.7. AI Application in Healthcare
4.8. AI Application in Agriculture
4.9. AI Application in Gaming
4.10. AI Application in Automobiles
4.11. AI Application in Social Media
4.12. AI Application in Marketing
4.13. AI Application in Chatbots
4.14. AI Application in Finance

4.1. AI Application in E-Commerce

- **Personalized Shopping**
  - Artificial Intelligence technology is used to create recommendation engines.

  - These recommendations are made in accordance with their browsing history, preference, and interests.
  - It helps in improving the relationship with the customers and the loyalty towards the brand.
- **AI-powered Assistants**
  - Virtual shopping assistants and chatbots help improve the user experience while shopping online.
  - Natural Language Processing is used to make the conversation sound as human and personal as possible.
- **Fraud Prevention**
  - Credit card frauds and fake reviews are two of the most significant issues that E-Commerce companies deal with.
  - Many customers prefer to buy a product or service based on customer reviews.
  - AI can help identify and handle fake reviews.

4.2. AI Application in Education

- **Administrative Tasks Automated to Aid Educators**
  - Artificial Intelligence can help educators with non-educational tasks like task-related duties like facilitating and automating personalized messages to students, back-office tasks like grading paperwork, arranging and facilitating parent and guardian interactions, routine issue feedback facilitating, managing enrollment, courses, and HR-related topics.
- **Creating Smart Content**
  - Digitization of content like video lectures, conferences, and text book guides can be made using Artificial Intelligence.
  - Artificial Intelligence helps create a rich learning experience by generating and providing audio and video summaries and integral lesson plans.
- **Voice Assistants**
  - Without even the direct involvement of the lecturer or the teacher, a student can access extra learning material or assistance through Voice Assistants.
- **Personalized Learning**
  - Using top AI technologies, hyper-personalization techniques can be used to monitor students' data thoroughly, and habits, lesson plans, reminders, study guides, flash notes, frequency or revision, etc., can be easily generated.

4.3. AI Application in Lifestyle

- **Autonomous Vehicles**
  - Automobile manufacturing companies like Toyota, Audi, Volvo, and Tesla use machine learning to train computers to think and evolve like humans when it comes to driving in any environment and object detection to avoid accidents.
- **Spam Filters**
  - The email has AI that filters out spam emails sending them to spam or trash folders.
  - The popular email provider, Gmail, has managed to reach a filtration capacity of approximately 99.9%.
- **Facial Recognition**
  - Devices like our phones, laptops, and PCs use facial recognition techniques by using face filters to detect and identify in order to provide secure access.
  - Facial recognition is a widely used Artificial Intelligence application even in high security-related areas in several industries.
- **Recommendation System**
  - Various platforms in daily lives like e-commerce, entertainment websites, social media, video sharing platforms, like youtube, etc., all use the recommendation system to get user data and provide customized recommendations to users to increase engagement.

4.4. AI Application in Navigation

- The technology uses a combination of Convolution Neural Network and Graph Neural Network, which makes lives easier for users by automatically detecting the number of lanes and road types behind obstructions on the roads.
- AI is heavily used by Uber and many logistics companies to improve operational efficiency, analyze road traffic, and optimize routes.

4.5. AI Application in Robotics

- Robotics is another field where artificial intelligence applications are commonly used.
- Robots powered by AI use real-time updates to sense obstacles in its path and pre-plan its journey instantly.
- It can be used for -
  - Carrying goods in hospitals, factories, and warehouses
  - Cleaning offices and large equipment
  - Inventory management

4.6. AI Application in Human Resource

- Artificial Intelligence helps with blind hiring.
- AI drive systems can scan job candidates' profiles, and resumes to provide recruiters an understanding of the talent pool they must choose from.

4.7. AI Application in Healthcare

- AI applications are used in healthcare to build sophisticated machines that can detect diseases and identify cancer cells.
- Artificial Intelligence can help analyze chronic conditions with lab and other medical data to ensure early diagnosis.
- AI uses the combination of historical data and medical intelligence for the discovery of new drugs.

4.8. AI Application in Agriculture

- Artificial Intelligence is used to identify defects and nutrient deficiencies in the soil.
- This is done using computer vision, robotics, and machine learning applications, AI can analyze where weeds are growing.
- AI bots can help to harvest crops at a higher volume and faster pace than human laborers.

4.9. AI Application in Gaming

- AI can be used to create smart, human-like NPCs to interact with the players.
- It can also be used to predict human behavior using which game design and testing can be improved.
- The Alien Isolation games released in 2014 uses AI to stalk the player throughout the game. The game uses two Artificial Intelligence systems - 'Director AI' that frequently knows your location and the 'Alien AI,' driven by sensors and behaviors that continuously hunt the player.

4.10. AI Application in Automobiles

- Artificial Intelligence is used to build self-driving vehicles.
- AI can be used along with the vehicle's camera, radar, cloud services, GPS, and control signals to operate the vehicle.
- AI can improve the in-vehicle experience and provide additional systems like emergency braking, blind-spot monitoring, and driver-assist steering.

4.11. AI Application in Social Media

- Instagram
- Facebook
- Twitter

4.12. AI Application in Marketing

- Artificial intelligence (AI) applications are popular in the marketing domain as well.
- AI can provide users with real-time personalization's based on their behavior and can be used to edit and optimize marketing campaigns to fit a local market's needs.

4.13. AI Application in Chatbots

- AI chatbots can comprehend natural language and respond to people online who use the "live chat" feature that many organizations provide for customer service.
- As AI continues to improve, these chatbots can effectively resolve customer issues, respond to simple inquiries, improve customer service, and provide 24/7 support.

4.14. AI Application in Finance

- Whether it's personal finance, corporate finance, or consumer finance, the highly evolved technology that is offered through AI can help to significantly improve a wide range of financial services.
- Artificial intelligence can also detect changes in transaction patterns and other potential red flags that can signify fraud, which humans can easily miss, and thus saving businesses and individuals from significant loss.
- Aside from fraud detection and task automation, AI can also better predict and assess loan risks.

5. Explain in detail about agents and its types in detail.

AGENTS

5.1 Agents and Environments

Agent

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators as in Figure 1.5.
- **Example:**
  - A human agent has eyes, ears, and other organs for sensors and hands, legs, vocaltract, and so on for actuators.
  - A robotic agent might have cameras and infrared range findersfor sensors and various motors for actuators.
  - A software agent receives file contents, network packets, and human input (keyboard/mouse/touch screen/voice) as sensory inputsand acts on the environment by writing files, sending network packets, and displaying information or generating sounds.

Environments

- The environment could be everything—the entire universe.

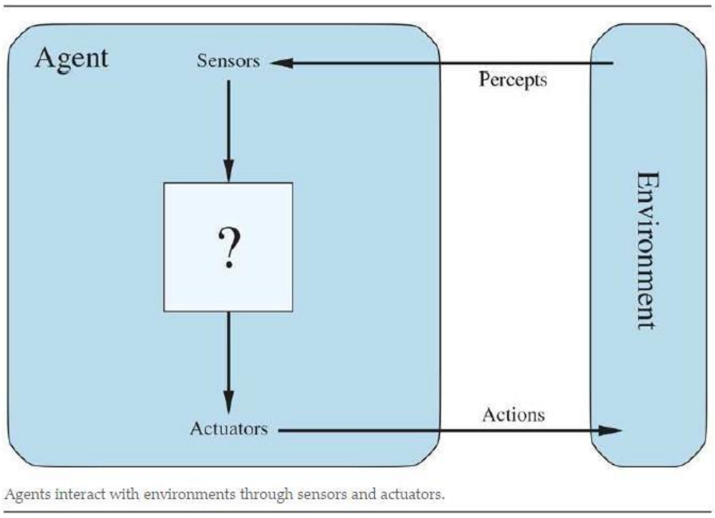

Agent
Sensors
Percepts
?
Environment
Actuators
Actions
Agents interact with environments through sensors and actuators.

Figure 1.5: Agents and Environments

5.2 Basic Terminologies

Agent Function

- The agent function for an agent specifies the action taken by the agent in response to any percept sequence.

Performance Measure

- The performance measure evaluates the behavior of the agent in an environment.

Rational Agent

- A rational agent acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far.

Task Environment

- A task environment specification includes the performance measure, the external environment, the actuators, and the sensors.

5.3 Steps in designing an agent

- In designing an agent, the first step must always be to specify the task environment as fully as possible.
- The agent program implements the agent function.
- There exists a variety of basic agent program designs reflecting the kind of information made explicit and used in the decision process.
- The designs vary in efficiency, compactness, and flexibility.
- The appropriate design of the agent program depends on the nature of the environment.
- All agents can improve their performance through learning.

5.4 Types of Agent

- **Simple reflex agents** respond directly to percepts. Refer fig 1.6
- **Model-based reflex** agents maintain internal state to track aspects of the world that are not evident in the current percept. Refer fig 1.7
- **Goal-based agents** act to achieve their goals Refer fig 1.8
- **Utility-based agents** try to maximize their own expected "happiness." Refer fig 1.9

Simple reflex agents

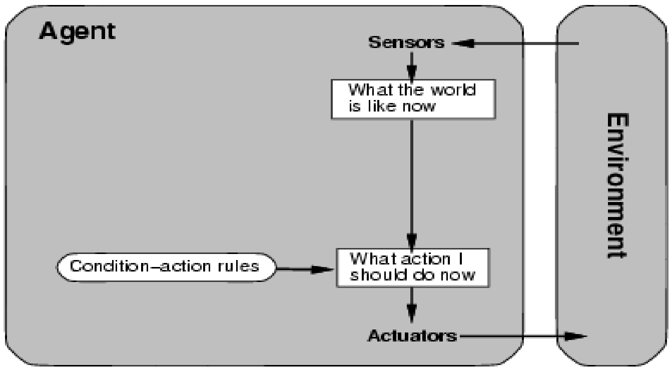


Figure 1.6: Simple reflex agents

Model-based reflex

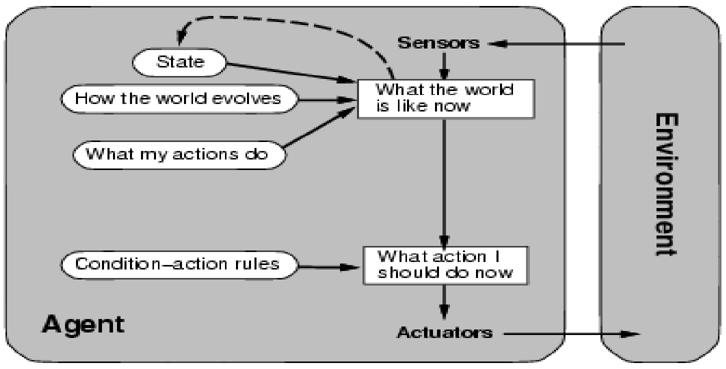


Figure 1.7: Model-based reflex

Goal-based agents

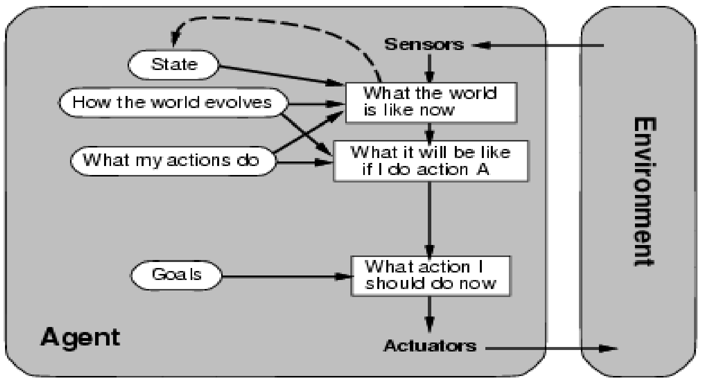

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure 1.8: Goal-based agents

Utility-based agents

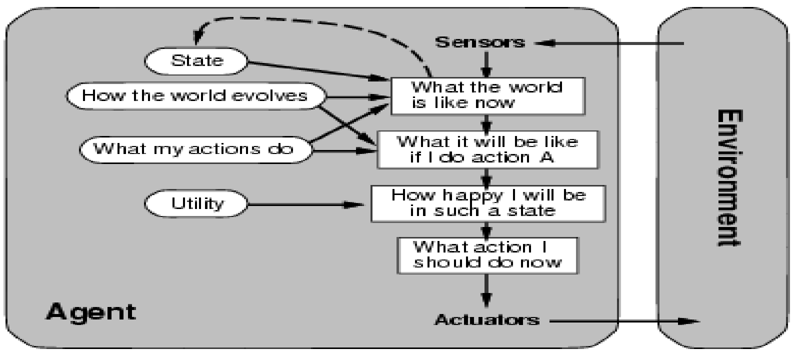

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 1.9: Utility-based agents

Learning agents:

It allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow. A learning agent can be divided into four conceptual components, as shown in figure 1.10:

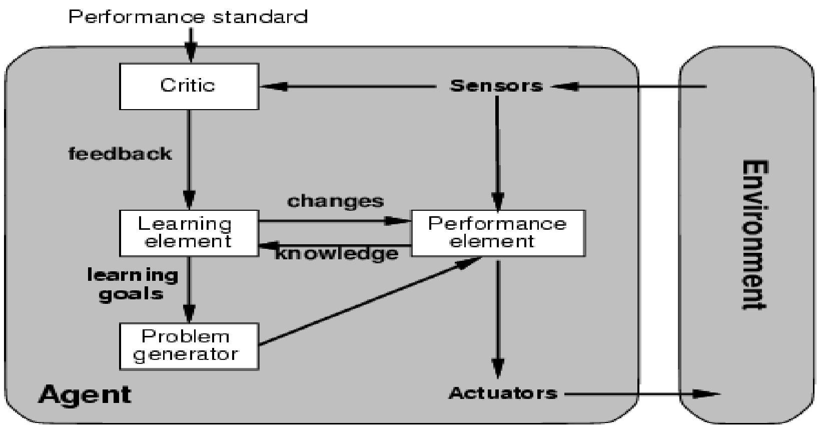


Figure 1.10: Learning agents

6. Explain in detail about problem solving agent with example in detail.

6.1 Problem Solving Agent Definition

- Problem-solving agent is a goal-based agent that focus on goals using a group of algorithms and techniques to solve a well-defined problem.
- An agent may need to plan a sequence of actions that form a path to a goal state.
- Such an agent is called a problem-solving agent, and the computational process it undertakes is called search.

6.2 Steps performed by Problem-solving agent

- **Goal Formulation:**
  - It is the first and simplest step in problem-solving.
  - It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal

formulation is based on the current situation and the agent's performance measure

- **Problem Formulation:**
  - It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal.
  - **Components** in problem formulation:
    1. **Initial State:** It is the starting state or initial step of the agent towards its goal.
    2. **Actions:** It is the description of the possible actions available to the agent.
    3. **Transition Model:** It describes what each action does.
    4. **Goal Test:** It determines if the given state is a goal state.
    5. **Path cost:** It assigns a numeric cost to each path that follows the goal.

    The problem-solving agent selects a cost function, which reflects its performance measure.
  - Initial state, actions, and transition model together define the state-space of the problem implicitly.
  - State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions.
  - The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.
- **Search:**
  - It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- **Solution:**
  - It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:**
  - It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

6.3 Types of problem approaches:

6.3.1 Toy Problem or Standardized Problem:

- It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.

Example Problem 1 - 8 Puzzle Problem:

- Consider a 3x3 matrix with movable tiles numbered from 1 to 8 with a blank space.
- The tile adjacent to the blank space can slide into that space.
- The objective is to reach a specified goal state similar to the goal state, as shown in the below figure 1.11
- The task is to convert the current state into goal state by sliding digits into the blank space

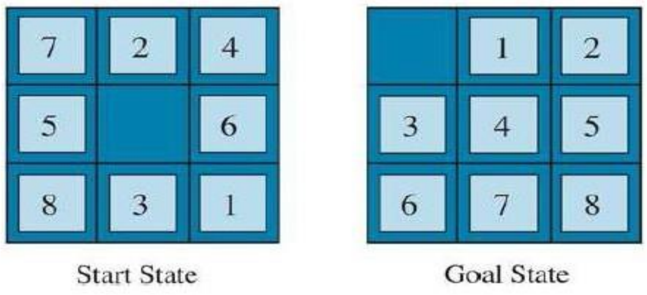


Figure 1.11: Agents and Environments

The problem formulation is as follows:

- **States:**
    - It describes the location of each numbered tiles and the blank tile.
- **Initial State:**
    - Can start from any state as the initial state.
- **Actions:**
    - Actions of the blank space is defined, i.e., either **left, right, up or down**
- **Transition Model:**
    - It returns the resulting state as per the given state and actions.
- **Goal test:**
    - It identifies whether have reached the correct goal-state.
- **Path cost:**
    - The path cost is the number of steps in the path where the cost of each step is 1.

Note: The 8-puzzle problem is a type of sliding-block problem which is used for testing new search algorithms in artificial intelligence.

Example Problem 2 – Grid World Problem:

- A **grid world** problem is a two-dimensional rectangular array of square cells in which agents can move from cell to cell.
- Typically the agent can move to any obstacle-free adjacent cell—horizontally or vertically and in some problems diagonally.

- Cells can contain objects, which the agent can pick up, push, or otherwise act upon; a wall or other impassible obstacle in acell prevents an agent from moving into that cell.
- The **vacuum world** canbe formulated as a grid world problem as shown in figure 1.12:

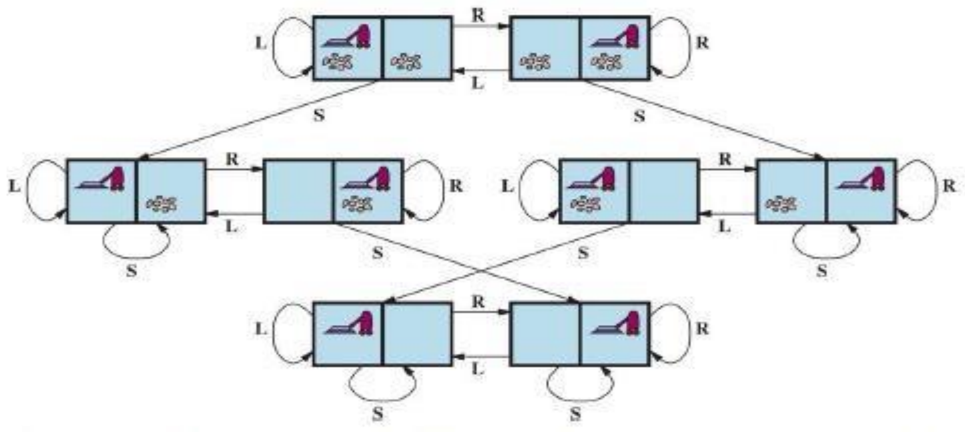


Figure 1.12: Vacuum World

The problem formulation is as follows:

- **States:**
  - For the vacuum world, the objects are the agent and any dirt. In the simple two-cell version, the agent can be in either of the two cells, and each cell can either contain dirt or not.
- **Initial state:**
  - Any state can be designated as the initial state.
- **Actions:**
  - In the two-cell world three actions are: *suck*, move *left*, and move *Right*, *upward* and *downward*, or *forward, backward, turnright*, and *turnleft*.
- **Transition model:**
  - *Suck* removes any dirt from the agent's cell;
  - *Forward* moves the agent ahead one cell in the direction it is facing, unless it hits a wall, in which case the action has no effect.
  - *Backward* moves the agent in the opposite direction
  - *Turnright* and *turnleft* change the direction it is facing by.
- **Goal states:**
  - The states in which every cell is clean.
- **Action cost:**
  - Each action costs 1.

6.3.2 Real-world Problem:

- It is real-world based problems which require solutions.

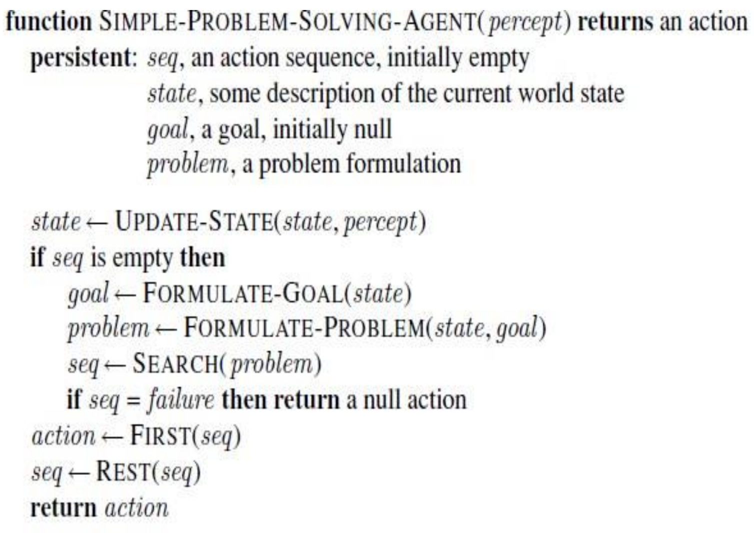
Example Problem 1 – Route-Finding Problem:

- **Route-finding problem** is defined in terms of specified locations and transitions along edges between them.

The problem formulation is as follows:

- Consider the airline travel problems that must be solved by a travel-planning Web site:
- **States:**
    - Each state obviously includes a location (e.g., an airport) and the current time.
- **Initial state:**
    - The user's home airport.
- **Actions:**
    - Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model:**
    - The state resulting from taking a flight will have the flight's Destination as the new location and the flight's arrival time as the new time.
- **Goal state:**
    - A destination city.
- **Action cost:**
    - A combination of monetary cost, waiting time, flight time, customs and Immigration procedures, seat quality, time of day, type of airplane, frequent-flyer reward Points, and so on.

<u>**Algorithm – Problem solving agent**</u>



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

7. Explain in detail about search algorithms in detail with example.

7.1 Search Algorithm

- A search algorithm takes a search problem as input and returns a solution.

7.2 Types of Search Algorithm

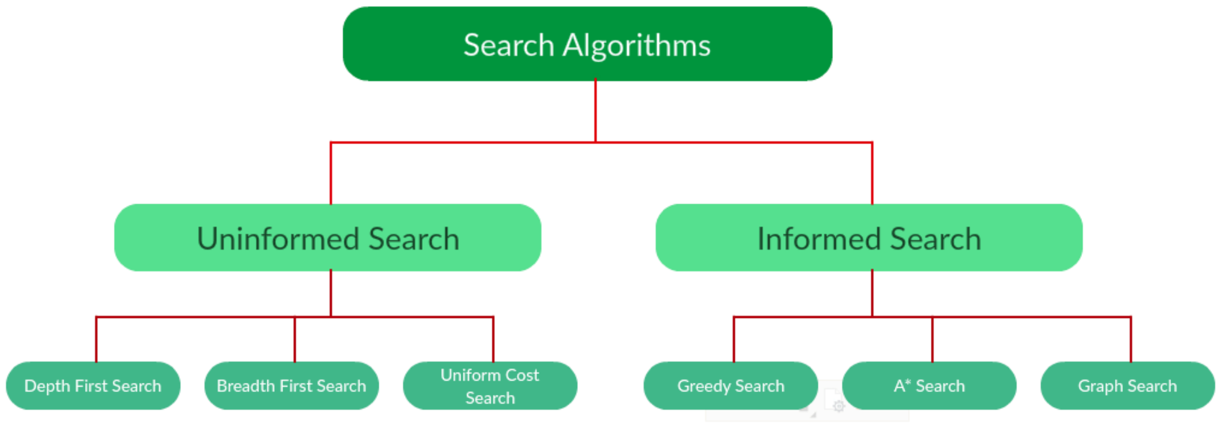


Figure 1.13: Types of Search Algorithms

7.2.1 UNINFORMED SEARCH ALGORITHMS:

- Uninformed search is also called **Blind search**.
- These algorithms can only generate the successors and differentiate between the goal state and non goal state.

- These type of search does not maintain any internal state, that's it is also known as Blind search.
- Types of uninformed search algorithms are shown in Figure 1.13.
  - 7.2.1.1 Depth First Search
  - 7.2.1.2 Depth Limited Search
  - 7.2.1.3 Breadth First Search
  - 7.2.1.4 Iterative Deepening Search
  - 7.2.1.5 Uniform Cost Search
  - 7.2.1.6 Bidirectional Search

Each of these algorithms will have:

- A problem **graph,** containing the start node S and the goal node G.
- A **strategy,** describing the manner in which the graph will be traversed to get to G.
- A **fringe,** which is a data structure used to store all the possible states (nodes) from the current states.
- A **tree,** that results while traversing to the goal node.
- A solution **plan,** which the sequence of nodes from S to G.

7.2.1.1 DEPTH FIRST SEARCH:

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node and explores as far as possible along each branch before backtracking.
- It uses last in-first-out strategy and hence it is implemented using a stack.

  d= the depth of the search tree = the number of levels of the search tree.

  n^i = number of nodes in level i
- ***DFS Example - Refer Figure 1.14***

The performance measure of DFS

- **Completeness:** DFS does not guarantee to reach the goal state.
- **Optimality:** It does not give an optimal solution as it expands nodes in one direction deeply.
- **Space complexity**: It needs to store only a single path from the root node to the leaf node. Therefore, DFS has **O(bm)** space complexity where b is the **branching factor(i.e., total no. of child nodes, a parent node have)** and m is the **maximum length of any path.**
- **Time complexity**: DFS has **$O(b^m)$** time complexity.

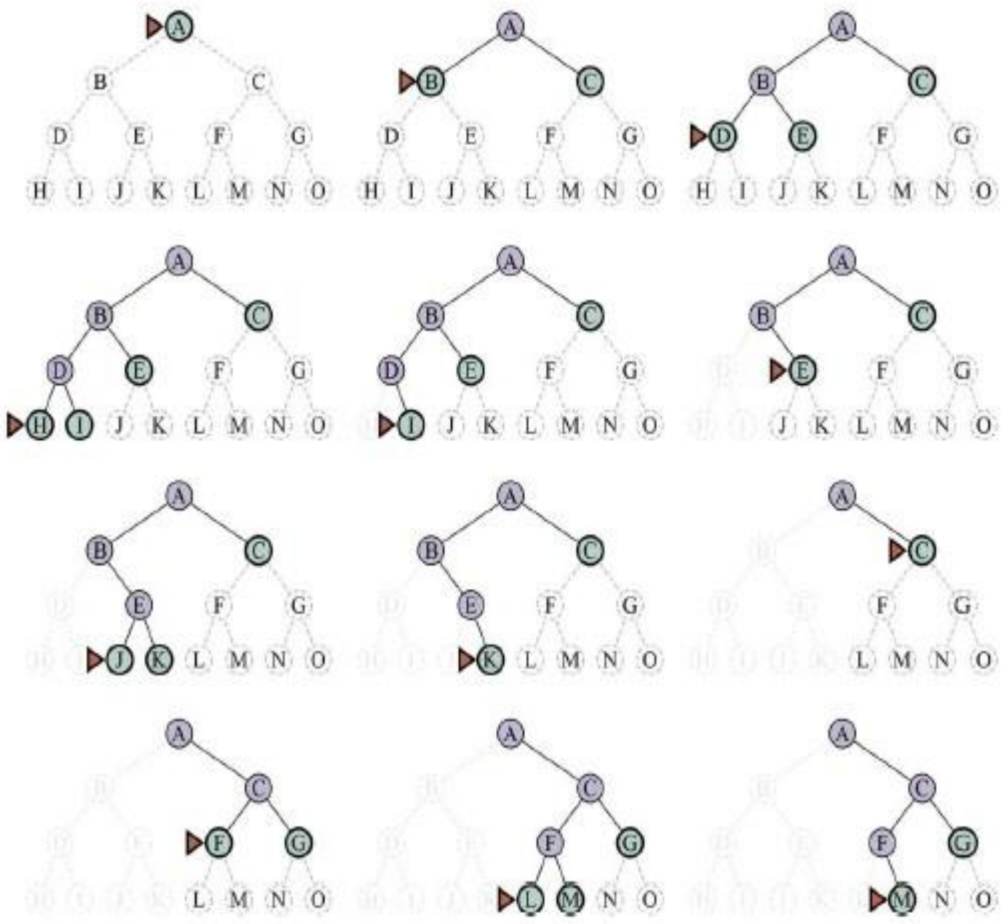


Figure 1.14: DFS Example

Advantage of Depth-first Search:

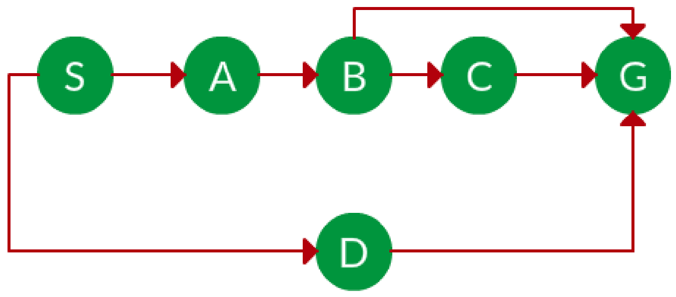
- DFS uses extremely little memory.
- It takes less time to reach the goal node than the BFS method.

Disadvantage of Depth-first Search:

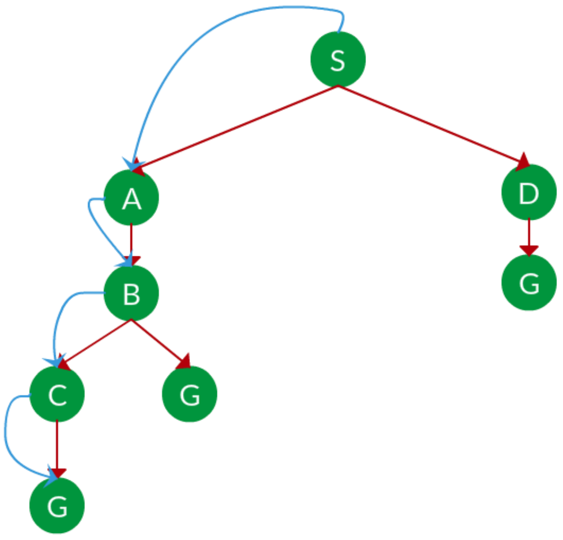
- There's a chance that many states will recur, and there's no certainty that a solution will be found.
- The DFS algorithm performs deep searching and may occasionally enter an infinite cycle.

Example:

Which solution would DFS find to move from node S to node G if run on the graph below?

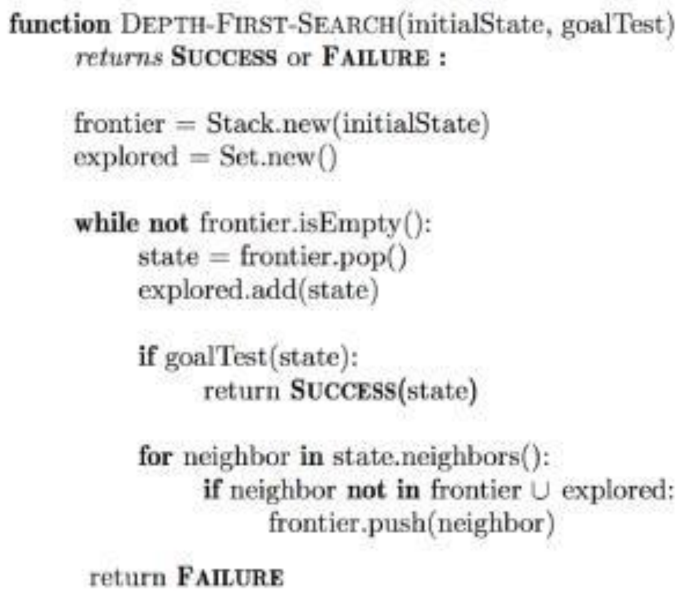


Solution.



Path: S -> A -> B -> C -> G

<u>Algorithm</u>



```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

7.2.1.2 DEPTH-LIMITED SEARCH ALGORITHM:

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit.
- Depth-limited search can solve the drawback of the infinite path in the Depth-first search.
- In this algorithm, the node at the depth limit will treat as it has no successor nodes further.
- Depth-limited search can be terminated with two Conditions of failure:
    - Standard failure value: It indicates that problem does not have any solution.
    - Cutoff failure value: It defines no solution for the problem within a given depth limit.

Depth-limited search Algorithm

- Set a variable **NODE** to the initial state, i.e., the *root node.*
- Set a variable **GOAL** which contains the value of the *goal state.*
- Set a variable **LIMIT** which carries a depth-limit value.
- Loop each node by traversing in **DFS** manner till the depth-limit value.
- While performing the looping, start removing the elements from the stack in ***LIFO*** order.
- If the goal state is found, **return goal state**. Else **terminate the search.**
- **Example – Refer Figure 1.15**

The performance measure of Depth-limited search

- **Completeness:** Depth-limited search does not guarantee to reach the goal node.
- **Optimality:** It does not give an optimal solution as it expands the nodes till the depth-limit.
- **Space Complexity:** The space complexity of the depth-limited search is **O(bl).**
- **Time Complexity:** The time complexity of the depth-limited search is **$O(b^l)$.**

Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:

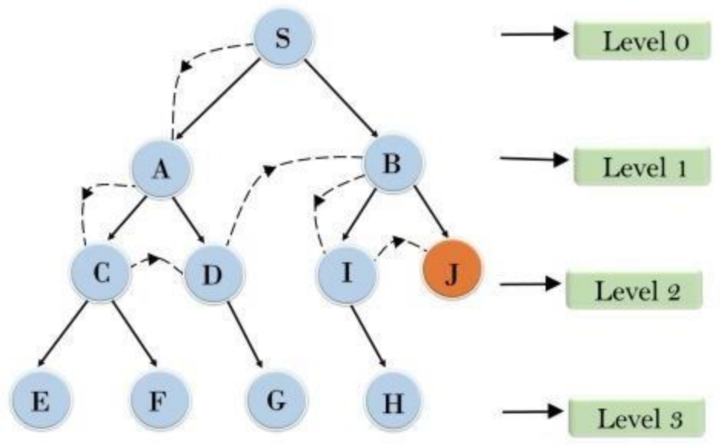


Figure 1.15 -Depth Limited Search

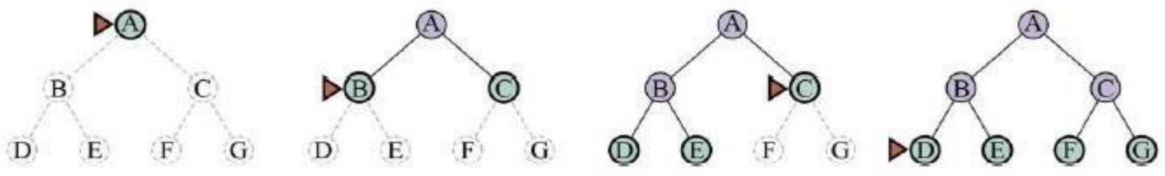
7.2.1.3 <u>BREADTH FIRST SEARCH:</u>

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level Refer Figure 1.16.
- It is implemented using a queue.

  S= the depth of the shallowest solution.

  n^i*= number of nodes in level* .

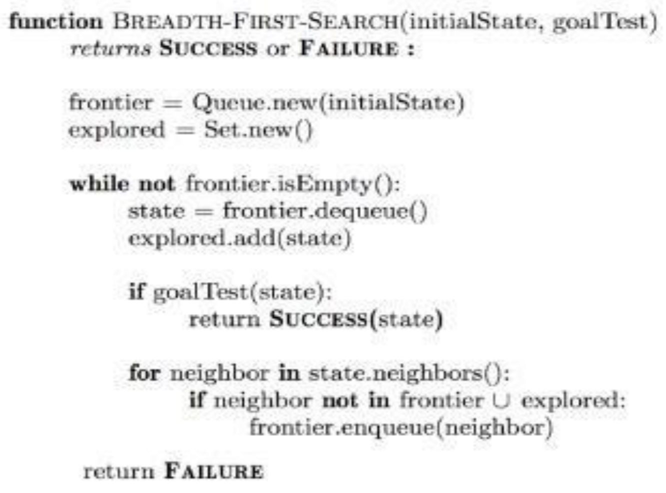
- **The performance measure of BFS is as follows:**
  - **Completeness:** It is a complete strategy as it definitely finds the goal state.
  - **Optimality:** It gives an optimal solution if the cost of each node is same.
  - **Space Complexity:** The space complexity of BFS is **$O(b^d)$**, i.e., it requires a huge amount of memory. Here, b is the **branching factor** and d denotes the **depth/level** of the tree
  - **Time Complexity:** BFS consumes much time to reach the goal node for large instances.



Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

Figure 1.16: Breadth First Search

<u>Breadth First Search Algorithm:</u>



```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

Advantages of Breadth-first Search:

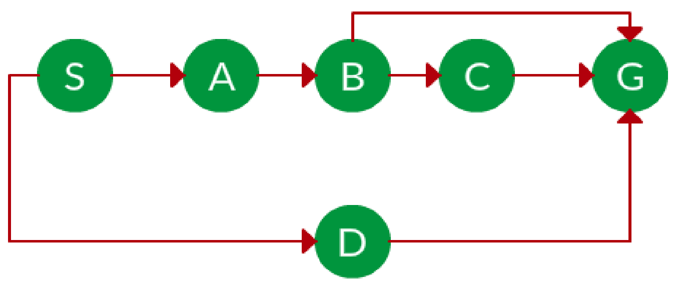
- If a solution is available, BFS will provide it.
- If there are multiple answers to a problem, BFS will present the simplest solution with the fewest steps.

Disadvantages of Breadth-first Search:

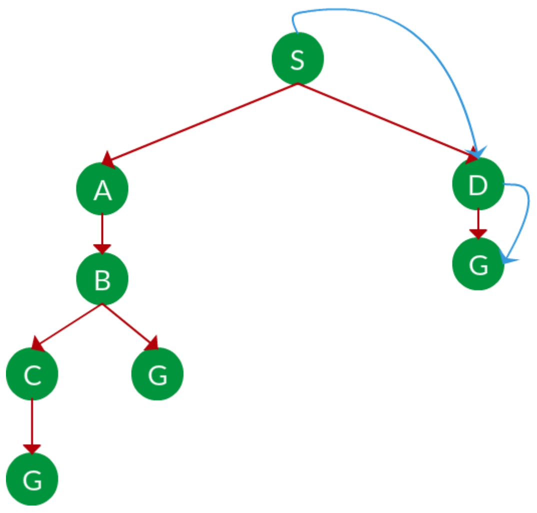
- It necessitates a large amount of memory.
- If the solution is located far from the root node, BFS will take a long time.

Example:

Which solution would BFS find to move from node S to node G if run on the graph below?



Solution.



Path: S -> D -> G

7.2.1.4 <u>ITERATIVE DEEPENING DEPTH-FIRST SEARCH/</u> ITERATIVE DEEPENING SEARCH

- This search is a combination of BFS and DFS, as BFS guarantees to reach the goal node and DFS occupies less memory space.
- Therefore, iterative deepening search combines these two advantages of BFS and DFS to reach the goal node.
- It gradually increases the depth-limit from 0,1,2 and so on and reach the goal node.

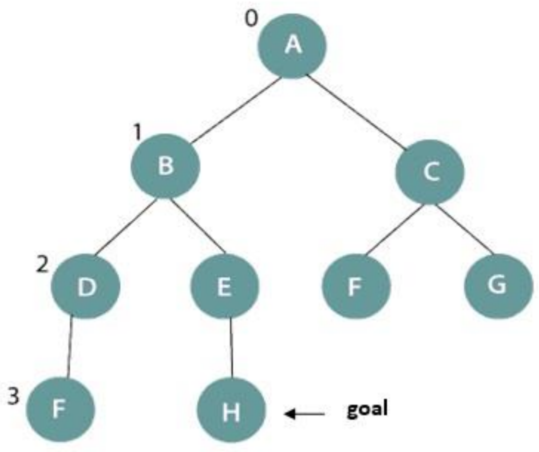


Figure 1.17 - Iterative Deepening Search

- In the above figure 1.17, the goal node is **H** and initial **depth-limit =[0-1]**.
- So, it will expand level 0 and 1 and will terminate with

A->B->C sequence.

- Further, change the **depth-limit =[0-3]**, it will again expand the nodes from level 0 till level 3 and the search terminate with **A->B->D->F->E->H** sequence where **H** is the desired goal node.

Iterative deepening search Algorithm

- Explore the nodes in DFS order.
- Set a LIMIT variable with a limit value.
- Loop each node up to the limit value and further increase the limit value accordingly.
- Terminate the search when the goal state is found.

The performance measure of Iterative deepening search

- **Completeness:** Iterative deepening search may or may not reach the goal state.
- **Optimality:** It does not give an optimal solution always.
- **Space Complexity:** It has the same space complexity as BFS, i.e., **$O(b^d)$.**
- **Time Complexity:** It has **$O(d)$** time complexity.

Disadvantages of Iterative deepening search

- The drawback of iterative deepening search is that it seems wasteful because it generates states multiple times.

7.2.1.5 UNIFORM COST SEARCH:

- A searching algorithm for traversing a weighted tree or graph is uniform-cost search.
- When a separate cost is provided for each edge, this algorithm is used.
- The uniform-cost search's main purpose is to discover the shortest path to the goal node with the lowest cumulative cost.
- **Cost of a node** is defined as:

  cost(node) = cumulative cost of all nodes from root

cost(root) = 0

- **Completeness:** The uniform-cost search is complete, so if a solution exists, UCS will discover it.
  - **Time Complexity:** Let C* **be the cost of the best solution**, and be the cost of each step toward the goal node. The number of steps is then equal to $C^*/\varepsilon+1$. We've chosen +1 because we're starting from state 0 and ending at C^*/ε.
  - As a result, the Uniform-cost search's worst-case time complexity is **$O(b^{1 + [C^*/\varepsilon]})$/.**

- **Space Complexity:** The same argument applies to space complexity, therefore Uniform-cost search's worst-case space complexity is $O(b^{1 + [C^*/\varepsilon]})$.
- **Optimal:** Uniform-cost search is always the best option because it only chooses the cheapest path.

Uniform Cost Search Algorithm

```
function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
    return BEST-FIRST-SEARCH(problem, PATH-COST)
```

The performance measure of Uniform-cost search

- **Completeness:** It guarantees to reach the goal state.
- **Optimality:** It gives optimal path cost solution for the search.
- **Space and time complexity:** The worst space and time complexity of

the uniform-cost search is $O(b^{1+LC^*/??})$.

Advantages of Uniform-cost Search:

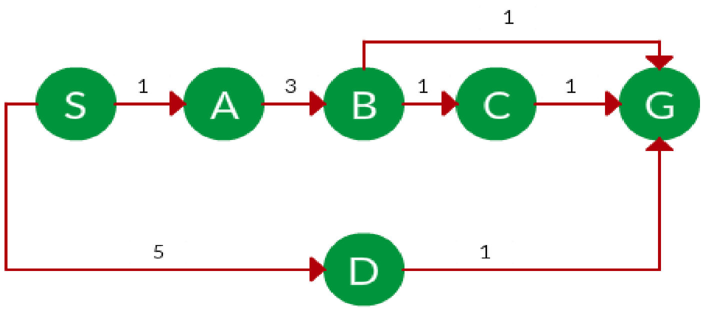
- The path with the lowest cost is chosen at each state.
- UCS is complete only if states are finite and there should be no loop with zero weight.
- UCS is optimal only if there is no negative cost.

Disadvantages of Uniform-cost Search:

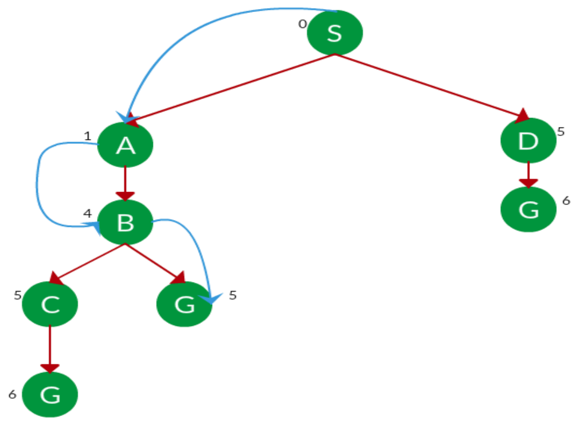
- It isjust concerned with the expense of the path.

Example:

Which solution would UCS find to move from node S to node G if run on the graph below?



Solution.



Path: *S -> A -> B -> G*

Cost: *5*

7.2.1.6 BIDIRECTIONAL SEARCH

- The strategy behind the bidirectional search is to run two searches simultaneously**--one forward search from the initial state and other from the backside of the goal**--hoping that both searches will meet in the middle.
- As soon as the two searches intersect one another, the bidirectional search terminates with the goal node.
- This search is implemented by replacing the goal test to check if the two searches intersect. Because if they do so, it means a solution is found.

The performance measure of Bidirectional search

- **Complete:** Bidirectional search is complete.
- **Optimal:** It gives an optimal solution.
- **Time and space complexity:** Bidirectional search has **$O(b^{d/2})$**

Disadvantage of Bidirectional Search

- It requires a lot of memory space.

7.2.2 INFORMED (HEURISTIC) SEARCH STRATEGIES:

- Here, the algorithms have information on the goal state, which helps in more efficient searching.
- This information is obtained by something called a *heuristic.*

Search Heuristics:

- In an informed search, a heuristic is a *functionh(n)* estimates how close a state is to the goal state.
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state.

Types of Informed search algorithms.

7.2.2.1 Greedy Search
7.2.2.2 A* Tree Search
7.2.2.3 A* Graph Search

7.2.2.1 GREEDY SEARCH:

- In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).
- **Heuristic:** A heuristic h is defined as-
  h(x) = Estimate of distance of node x from the goal node.
  Lower the value of h(x), closer is the node from the goal.
- **Strategy:** Expand the node closest to the goal state, *i.e.* expand the node with a lower h value.

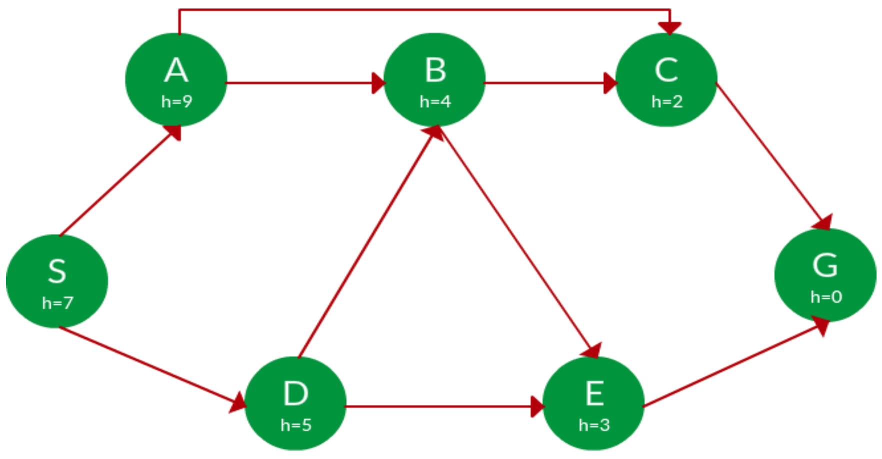
The performance measure of Best-first search Algorithm:

- **Completeness:** Best-first search is incomplete even in finite state space.
- **Optimality:** It does not provide an optimal solution.
- **Time and Space complexity:** It has **O(b^m)** worst time and space complexity, where **m is the maximum depth of the search tree**. If the quality of the heuristic function is good, the complexities could be reduced substantially.

- **Advantage:** Works well with informed search problems, with fewer steps to reach a goal.
- **Disadvantage:** Can turn into unguided DFS in the worst case.

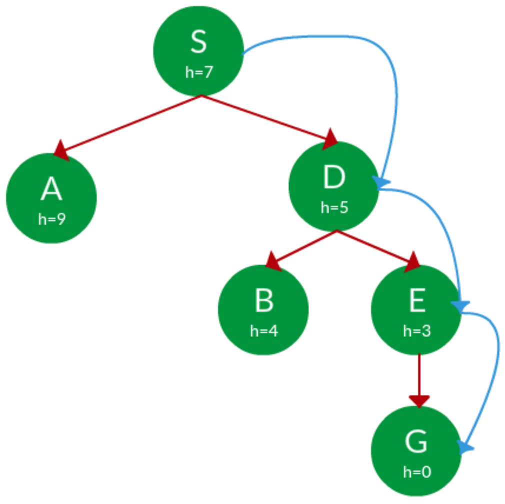
Example:

Question. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.



Solution.

Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.



Path: S -> D -> E -> G

7.2.2.2 <u>A* TREE SEARCH:</u>

- A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search.
- In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in the greedy search, denoted by h(x). The summed cost is denoted by f(x).

Heuristic:

The following points should be noted with heuristics in A* search.

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

- h(x) is called the **forward cost** and is an estimate of the distance of the current node from the goal node.
- g(x) is called the **backward cost** and is the cumulative cost of a node from the root node.
- A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal.
- This property of *A** heuristic is called **admissibility**.

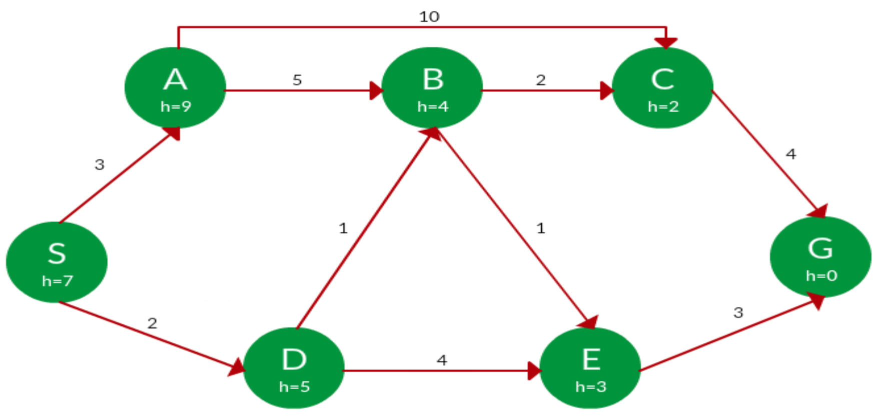
Admissibility: $0 <= h(x) <= h^*(x)$

Strategy: Choose the node with the lowest f(x) value.

The performance measure of A* search

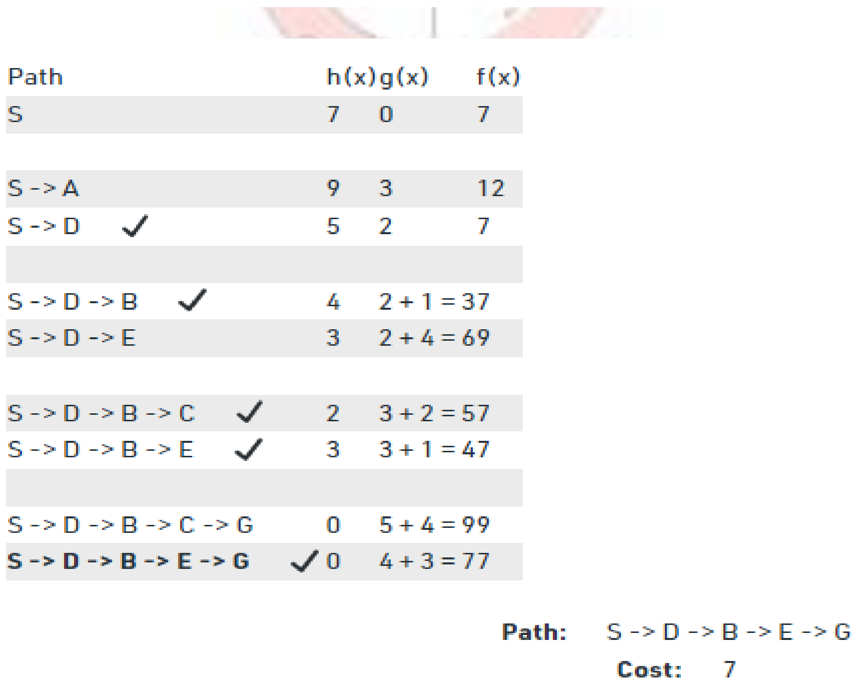
- **Completeness:** The star(*) in A* search guarantees to reach the goal node.
- **Optimality:** An underestimated cost will always give an optimal solution.
- **Space and time complexity:** A* search has **O(b^d) space and time** complexities.

Example: Find the path to reach from S to G using A* search.



Solution.

- Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum.
- The entire work is shown in the table below.



Path	h(x)	g(x)	f(x)
S	7	0	7
S -> A	9	3	12
S -> D ✓	5	2	7
S -> D -> B ✓	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C ✓	2	3 + 2 = 5	7
S -> D -> B -> E ✓	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G ✓	0	4 + 3 = 7	7

Path: S -> D -> B -> E -> G

Cost: 7

<u>A* algorithm</u>

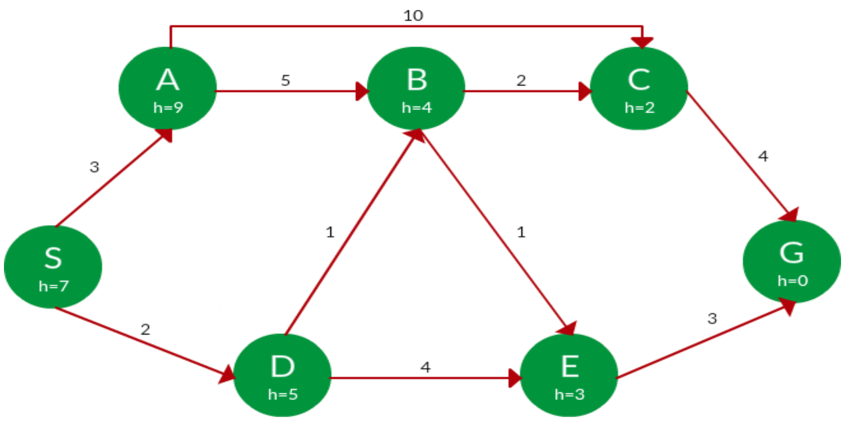
```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost $f(n) = g(n) + h(n)$ */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

7.2.2.3 A* Graph Search:

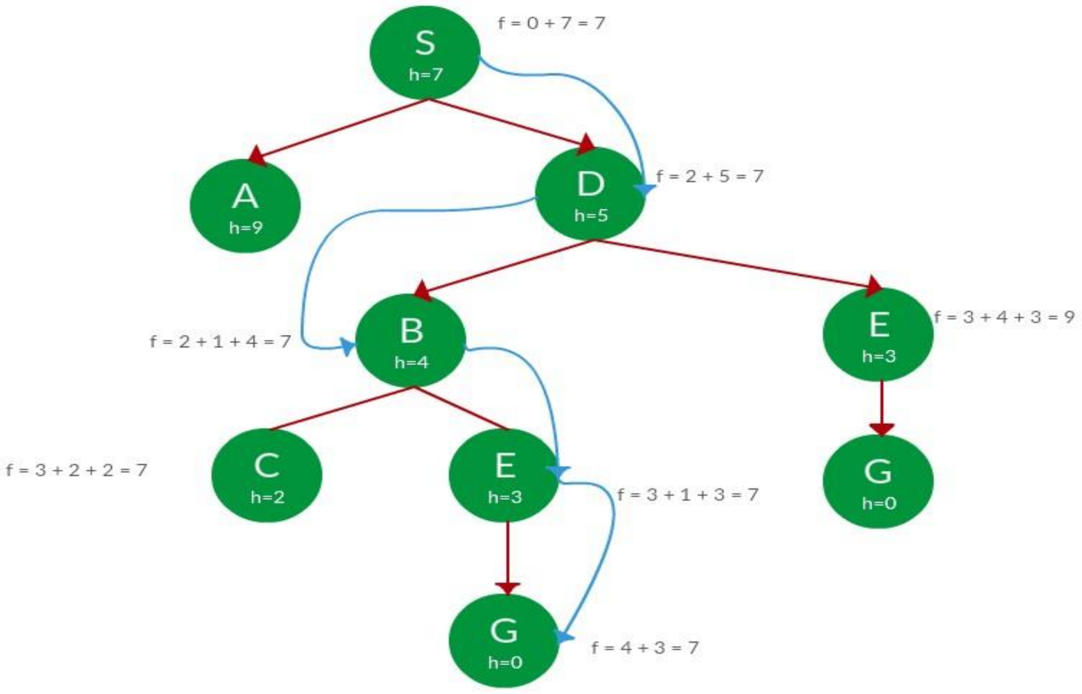
- In A* tree search, if the same node has expanded twice in different branches of the search tree, A* search might explore both of those branches, thus wasting time
- A* Graph Search, or simply Graph Search, removes this limitation by adding this rule: **do not expand the same node more than once.**
- **Heuristic.** Graph search is optimal only when the forward cost between two successive nodes A and B, given by h(A) – h (B), is less than or equal to the backward cost between those two nodes g(A -> B). This property of the graph search heuristic is called **consistency**.
- Consistency: ***h(A) - h(B) <= g (A->B)***

Example:

Question. Use graph searches to find paths from S to G in the following graph.



Solution



Path: S -> D -> B -> E -> G

Cost: 7

8. Explain in detail about Local Search Algorithms with an example.

LOCAL SEARCH ALGORITHMS

8.1. Local Search Algorithm - Definition

- **Local search** algorithms operate by searching from a start state to neighboring states,without keeping track of the paths, or the set of states that have been reached.
- **“Local search algorithms” where the path cost does not matters, and only focus on solution-state needed to reach the goal node.**

- It has two key advantages:
  - use very little memory;
  - Often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.
- Local search algorithms can also solve **optimization problems**, in which the aim is to find the best state according to an **objective function**.
- **Optimization Problems -** An optimization problem is one where all the nodes can give a solution. But the target is to find the best state out of all according to the **objective function**.
- **Objective Function -** An objective function is a function whose value is either minimized or maximized in different contexts of the optimization problems. In the case of search algorithms, an objective function can be the path cost for reaching the goal node, etc.

8.2. Working of a Local search algorithm

- Consider the below state-space landscape having both:
  - **Location:** It is defined by the state.
  - **Elevation:** It is defined by the value of the objective function or heuristic cost function.

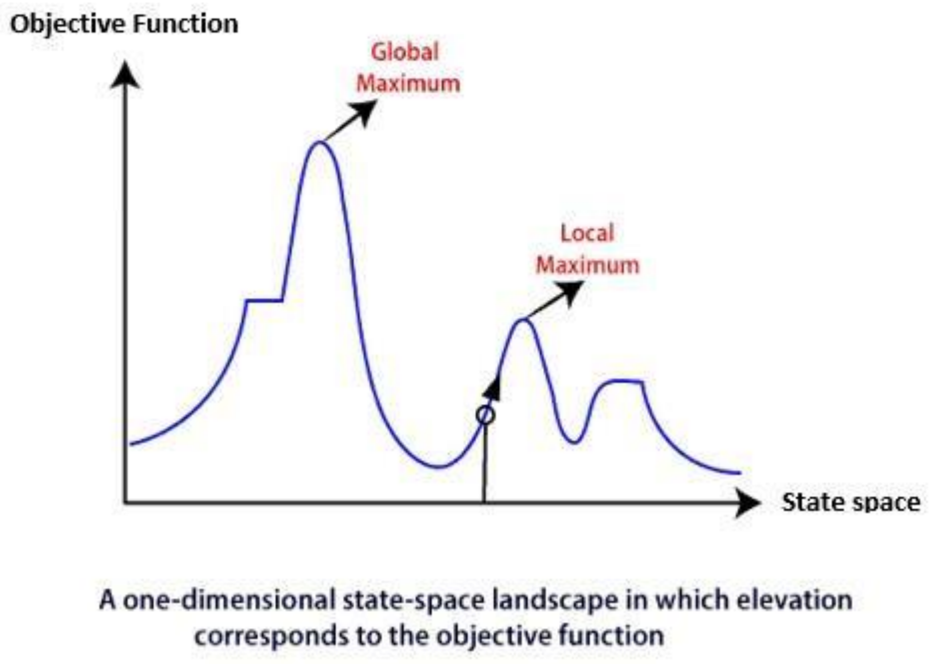

Objective Function
Global Maximum
Local Maximum
State space
A one-dimensional state-space landscape in which elevation corresponds to the objective function

Figure 1.18 - Local Search Algorithm

The local search algorithm explores the above landscape by finding the following two points:

- **Global Minimum:** If the elevation corresponds to the cost, then the task is to find the lowest valley, which is known as **Global Minimum.**

- **Global Maxima:** If the elevation corresponds to an objective function, then it finds the highest peak which is called as **Global Maxima**. It is the highest point in the valley. Refer fig 1.18

8.3 Different types of local searches:

8.3.1 Hill-climbing Search
8.3.2 Simulated Annealing
8.3.3 Local Beam Search

8.3.1 Hill-climbing Search

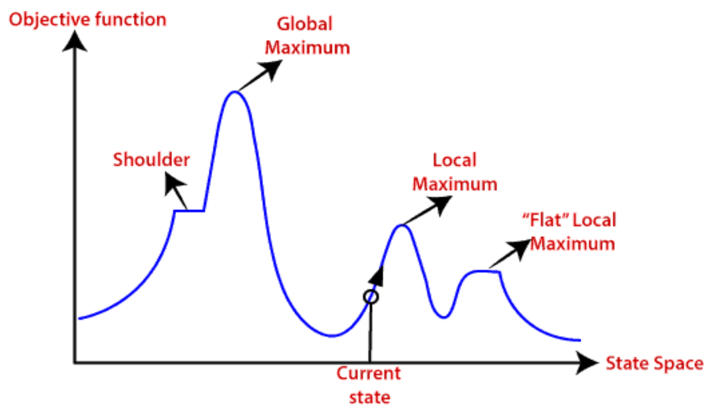
- **Hill Climbing Algorithm**: Hill climbing search is a local search problem.
- The hill-climbing search algorithm keeps track of one current state and on each iteration moves to the neighboring state with highest value.
- The purpose of the hill climbing search is to climb a hill and reach the topmost peak/ point of that hill.
- It is based on the heuristic search technique where the person who is climbing up on the hill estimates the direction which will lead him to the highest peak.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.

State-space Landscape of Hill climbing algorithm

- Consider the below landscape representing the **goal state/peak** and the **current state** of the climber.



A one-dimensional state-space landscape in which elevation corresponds to the objective function

Figure 1.19 - Local Search Algorithm

- The topographical regions shown in the Figure 1.19 can be defined as:
  - **Global Maximum:** It is the highest point on the hill, which is the goal state.
  - **Local Maximum:** It is the peak higher than all other peaks but lower than the global maximum.
  - **Flat local maximum:** It is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.
  - **Shoulder:** It is also a flat area where the summit is possible.
  - **Current state:** It is the current position of the person.

Types of Hill climbing search algorithm

8.3.1.1 Simple hill climbing
8.3.1.2 Steepest-ascent hill climbing
8.3.1.3 Stochastic hill climbing
8.3.1.4 Random-restart hill climbing

8.3.1.1 Simple hill climbing search

- Simple hill climbing is the simplest technique to climb a hill.
- The task is to reach the highest peak of the mountain.
- Here, the movement of the climber depends on his move/steps.
- If he finds his next step better than the previous one, he continues to move else remain in the same state.
- This search focus only on his previous and next step.

Simple hill climbing Algorithm

1. Create a **CURRENT** node, **NEIGHBOUR** node, and a **GOAL** node.
2. If the **CURRENT node=GOAL node**, return **GOAL** and terminate the search.
3. Else **CURRENT node<= NEIGHBOUR node,** move ahead.
4. Loop until the goal is not reached or a point is not found.

8.3.1.2 Steepest-ascent hill climbing

- Steepest-ascent hill climbing is different from simple hill climbing search. Unlike simple hill climbing search,
- It considers all the successive nodes, compares them, and choose the node which is closest to the solution.
- Steepest hill climbing search is similar to **best-first search** because it focuses on each node instead of one.

Steepest-ascent hill climbing algorithm

1. Create a **CURRENT** node and a **GOAL** node.
2. If the **CURRENT node=GOAL** node, return **GOAL** and terminate the search.
3. Loop until a better node is not found to reach the solution.

4. If there is any better successor node present, expand it.
5. When the **GOAL** is attained, return **GOAL** and terminate.

8.3.1.3 Stochastic hill climbing

- Stochastic hill climbing does not focus on all the nodes.
- It selects one node at random and decides whether it should be expanded or search for a better one.

8.3.1.4 Random-restart hill climbing

- Random-restart algorithm is based on **try and try strategy**.
- It iteratively searches the node and selects the best one at each step until the goal is not found.
- The success depends most commonly on the shape of the hill.
- If there are few plateaus, local maxima, and ridges, it becomes easy to reach the destination.

Limitations of Hill climbing algorithm

- **Local Maxima:** It is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it. Refer Figure 1.20.

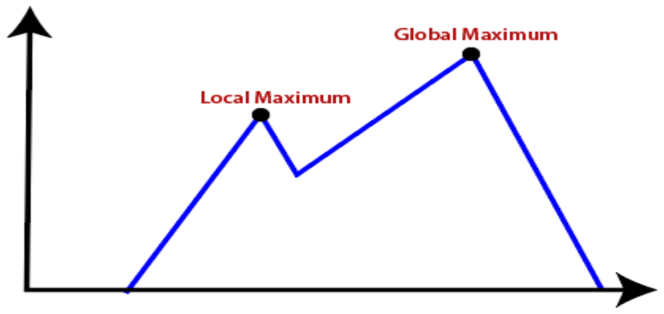


Figure 1.20 - Local Maxima

- **Plateau:** It is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point.Refer Figure 1.21.

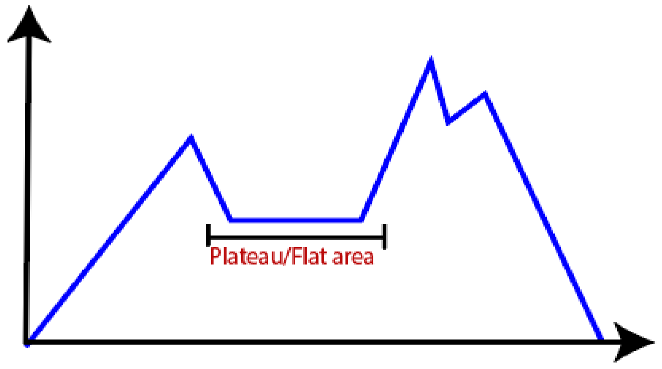


Figure 1.21 - Plateau

- **Ridges:** It is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself. Refer Figure 1.22

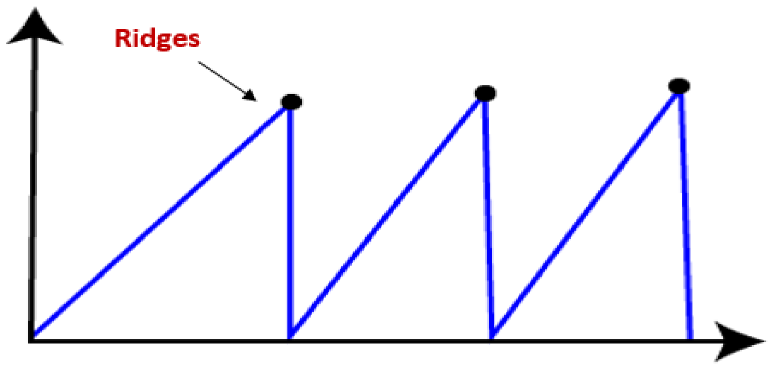


Figure 1.22 - Ridges

8.3.2 Simulated Annealing

- Simulated annealing is similar to the hill climbing algorithm.
- It works on the current situation.
- It picks **a random move** instead of picking **the best move**.
- If the move leads to the improvement of the current situation, it is always accepted as a step towards the solution state, else it accepts the move having **a probability less than 1**.
- This search technique was first used in **1980** to solve **VLSI layout** problems.
- It is also applied for factory scheduling and other large optimization tasks.

```
**function** SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE(current) – VALUE(next)
      if ΔE > 0 then current ← next
      else current ← next only with probability $e^{-\Delta E/T}$
```

8.3.3. Local Beam Search

- The local beam search algorithm keeps track of states rather than just one.
- It begins with randomly **k** generated states.

- It selects **k** randomly generated states, and expands them at each step.
- If any state is a goal state, the search stops with success.
- Else it selects the best **k** successors from the complete list and repeats the same process.
- In local beam search, the necessary information is shared between the parallel search processes.

Disadvantages of Local Beam search

- This search can suffer from a lack of diversity among the **k** states.
- It is an expensive version of hill climbing search.

9. Explain in detail about Adversarial Search and its types in Artificial Intelligence.

ADVERSARIAL SEARCH

9.1 Adversarial Search in Artificial Intelligence

AI 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 (multiagent).
- 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.

9.2 Elements of Game Playing search

- To play a game, we use a game tree to know all the possible choices and to pick the best one out.
- There are following elements of a game-playing:
  - **S_0:** It is the initial state from where a game begins.
  - **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
  - **ACTIONS (s):** It defines the set of legal moves to be used in a state.

- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**.
- The price which the winner will get i.e.
    - **(-1):** If the PLAYER loses.
    - **(+1):** If the PLAYER wins.
    - **(0):** If there is a draw between the PLAYERS.

9.3 Example:

- **Game tic-tac-toe,** has two or three possible outcomes.
- Either to win, to lose, or to draw the match with values **+1,-1 or 0.**
- Game tree designed for **tic-tac-toe refer Figure 1.23**.
- Here, the node represents the game state and edges represent the moves taken by the players.

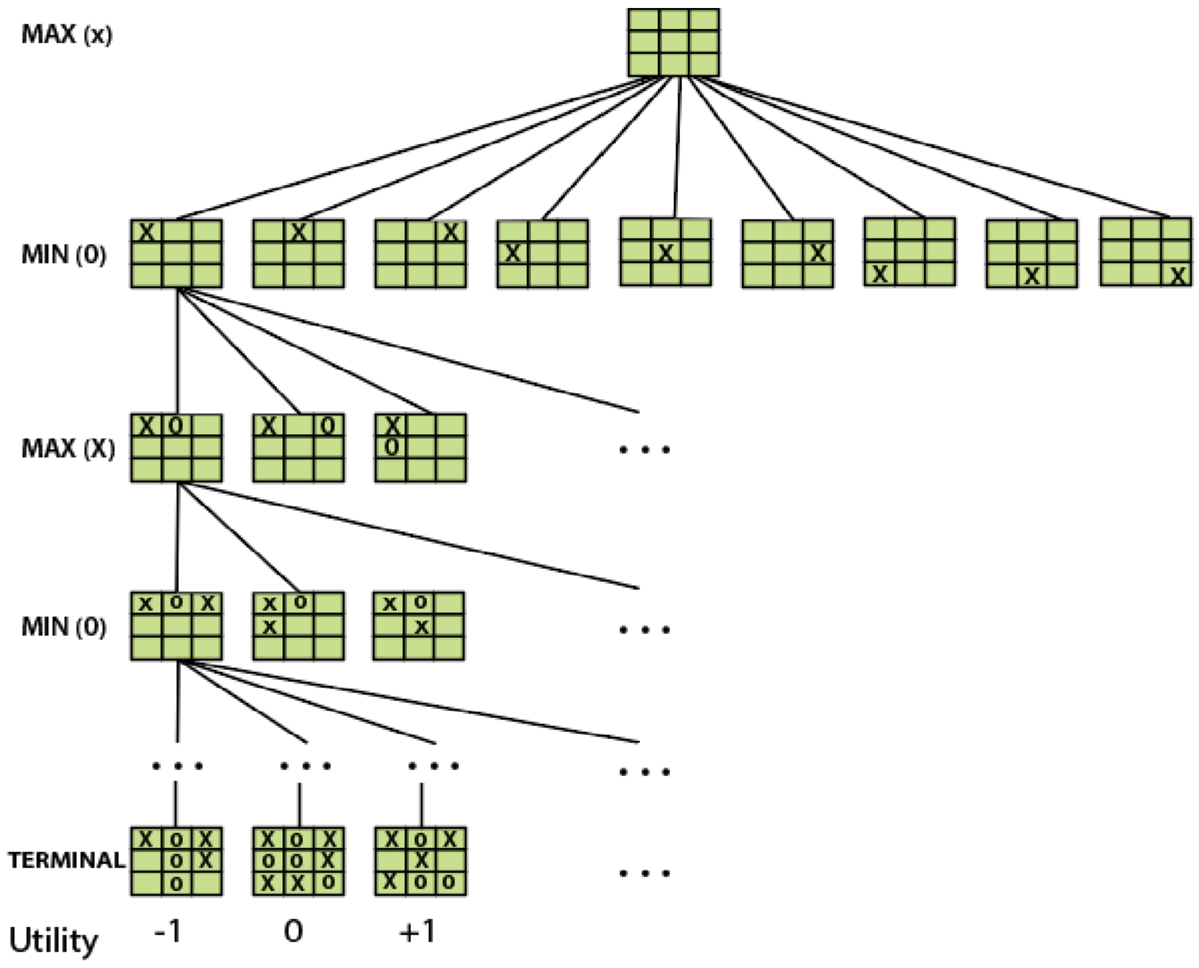


Figure 1.23 - A Game Tree for Tic Tac Toe

9.3.1 GAME TREE FOR TIC TAC TOE

- **INITIAL STATE (S_0):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN,** and accordingly, the price will be given to them**.**

9.4 TYPES OF ALGORITHMS IN ADVERSARIAL SEARCH

9.4.1 Minimax Algorithm
9.4.2 Alpha-beta Pruning

9.4.1 MINIMAX STRATEGY / MINIMAX ALGORITHM

- In artificial intelligence, minimax is a **decision-making** strategy under **game theory,** which is used to minimize the losing chances in a game and to maximize the winning chances.
- This strategy is also known as '**Minmax,' 'MM,' or 'Saddle point.'**
- Basically, it is a two-player game strategy where if one wins, the other loose the game.
- Example: playing chess
- MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices.
- MINIMAX strategy follows the **DFS (Depth-first search)** concept.
- Here, we have two players **MIN and MAX,** and the game is played alternatively between them, i.e., when **MAX** made a move, then the next turn is of **MIN.**

Game Tree

- **A game tree** is where the nodes represent the states of the game and edges represent the moves made by the players in the game.
- Players will be two namely:
    - **MIN:** Decrease the chances to win the game.
    - **MAX:** Increases his chances of winning the game.

- MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

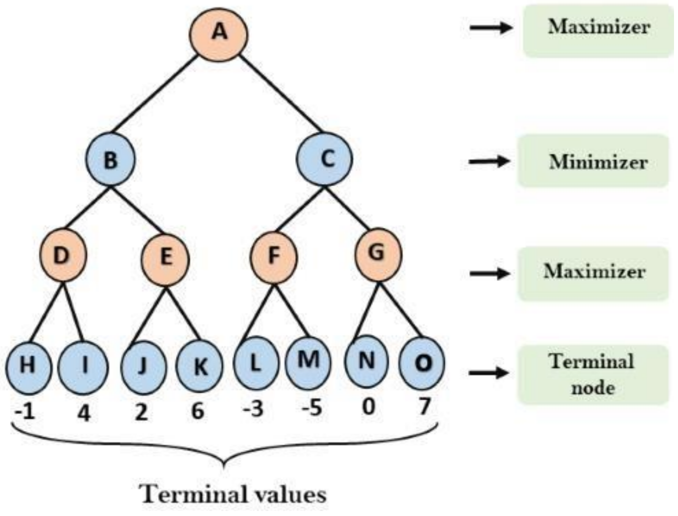
Working of Min-Max Algorithm:

- Consider an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.

Steps involved in solving the two-player game tree:

Step-1:

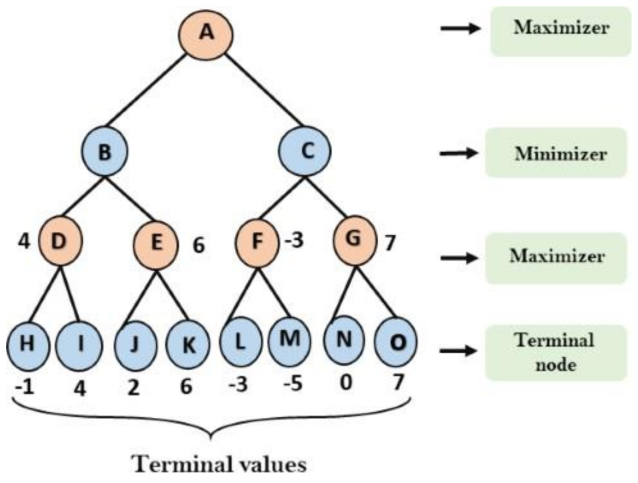
Let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Step 2:

Now, first we find the utilities value for the Maximizer, its initial value is -∞,

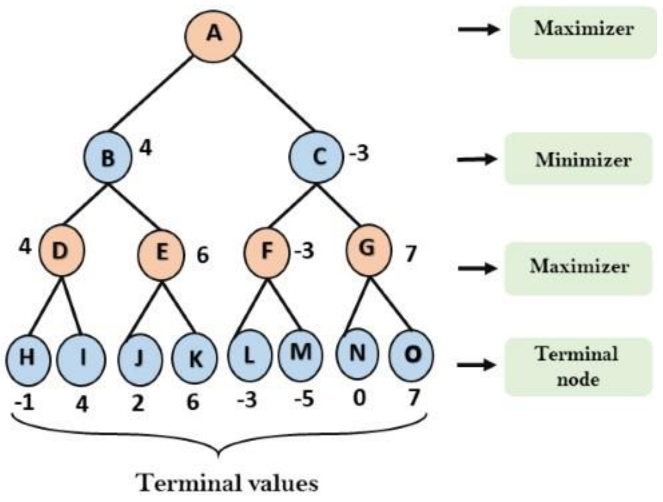
- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



Step 3:

In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3

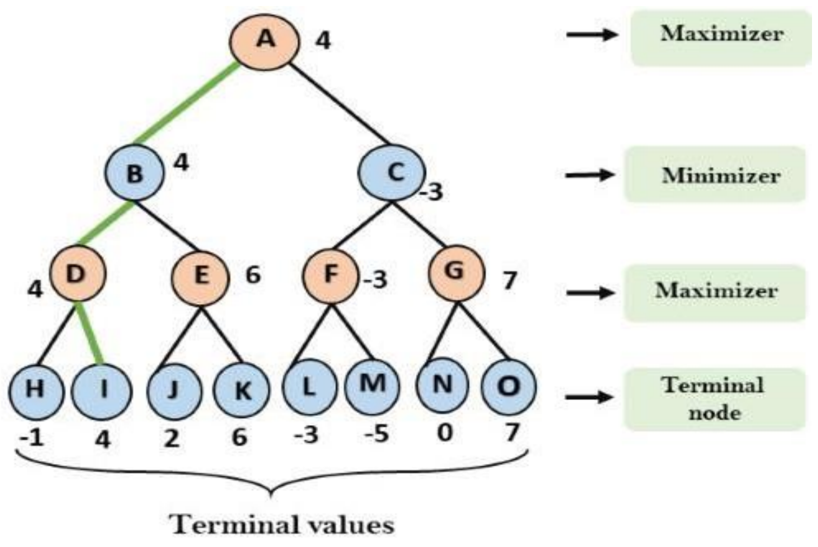


Step 4:

Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.

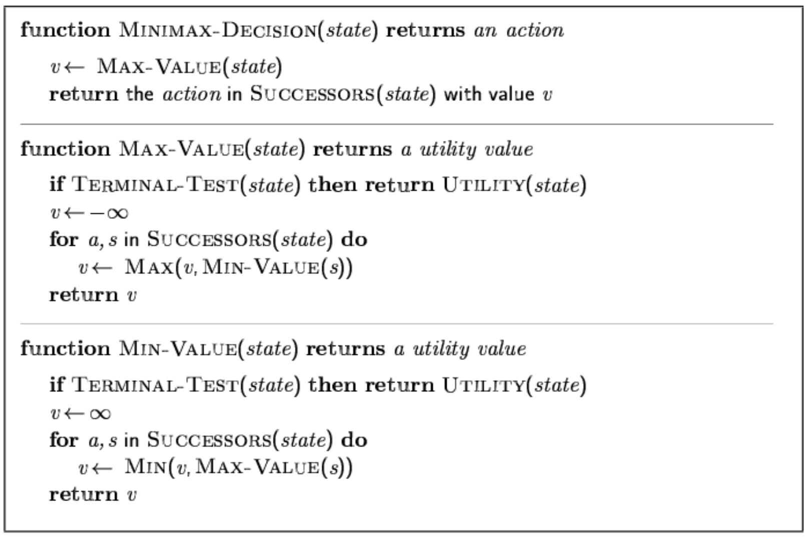
In this game tree, there are only 4 layers, hence reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.

Minimax algorithm



```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

9.4.2 ALPHA – BETA PRUNING

- Alpha-beta pruning is an advance version of MINIMAX algorithm.

Drawbacks of MINIMAX

- The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths.
- This increases its time complexity.
- Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.

- The method used in alpha-beta pruning is that it **cutoff the search** by exploring less number of nodes. It prunes the unwanted branches using the pruning technique (discussed in adversarial search).
- Alpha-beta pruning works on two threshold values, i.e., **? (alpha)** and **? (beta).**
  - **?: (alpha) -** It is the best highest value, a **MAX** player can have. It is the lower bound, which represents negative infinity value.
  - **?:(beta) -** It is the best lowest value, a **MIN** player can have. It is the upper bound which represents positive infinity.
- The main condition which required for alpha-beta pruning is:

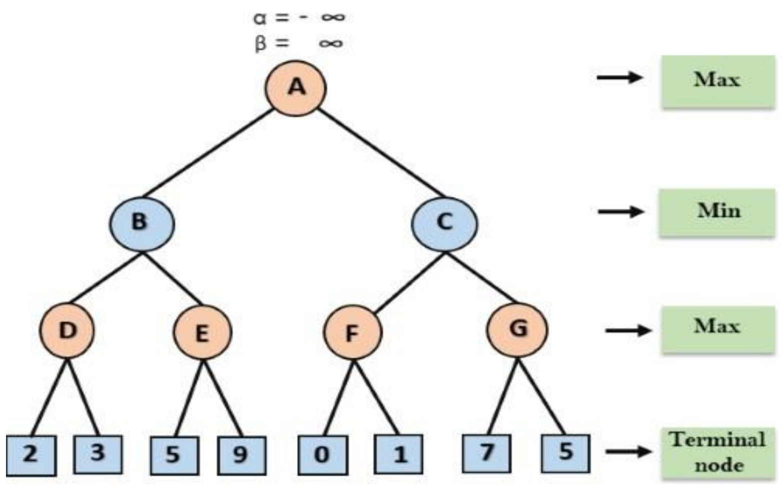
$$\boldsymbol{\alpha >= \beta}$$

Working of Alpha-beta Pruning

- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning
- here, **? (alpha)** will represent the maximum value of the nodes, and **?(beta)** will represent the minimum value of the nodes.

Step 1:

At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.

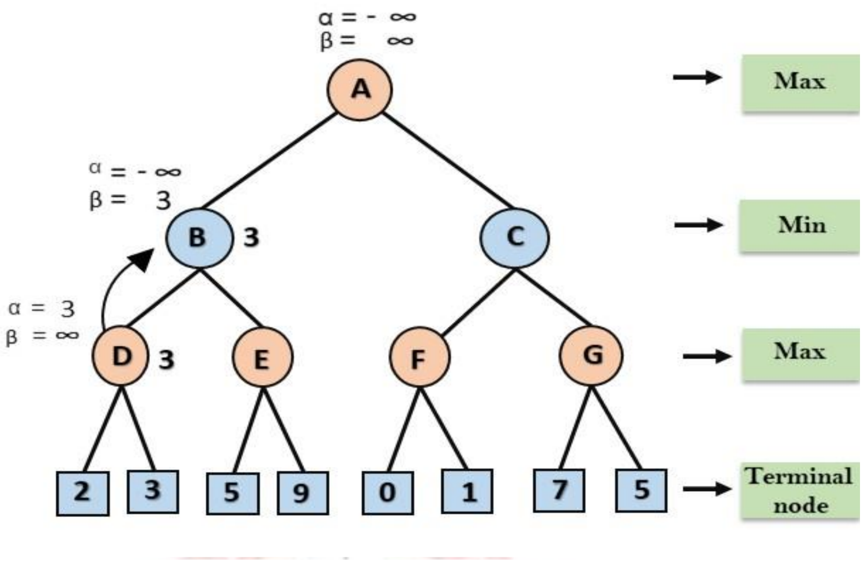


Step 2:

At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3:

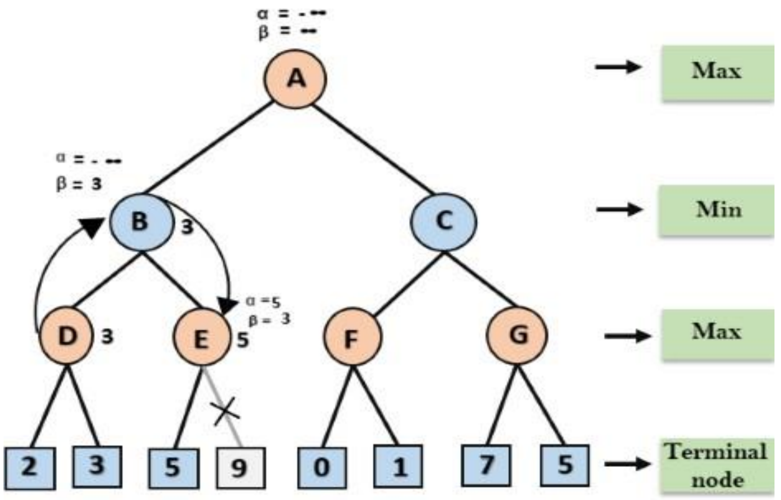
Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

Step 4:

At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



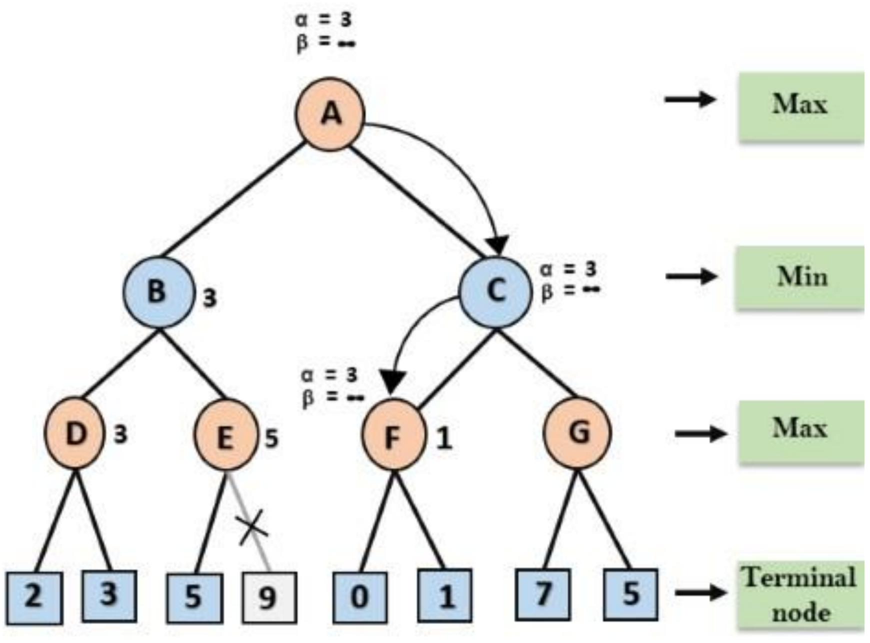
Step 5:

At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

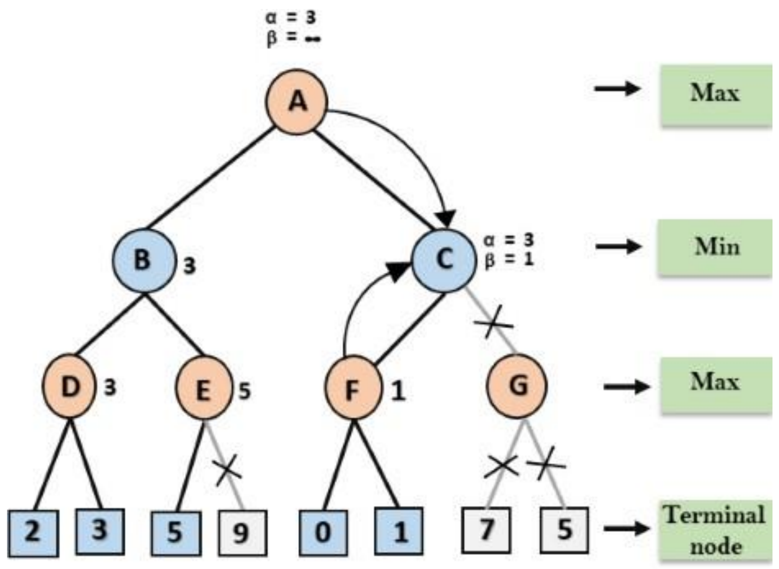
Step 6:

At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7:

Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.

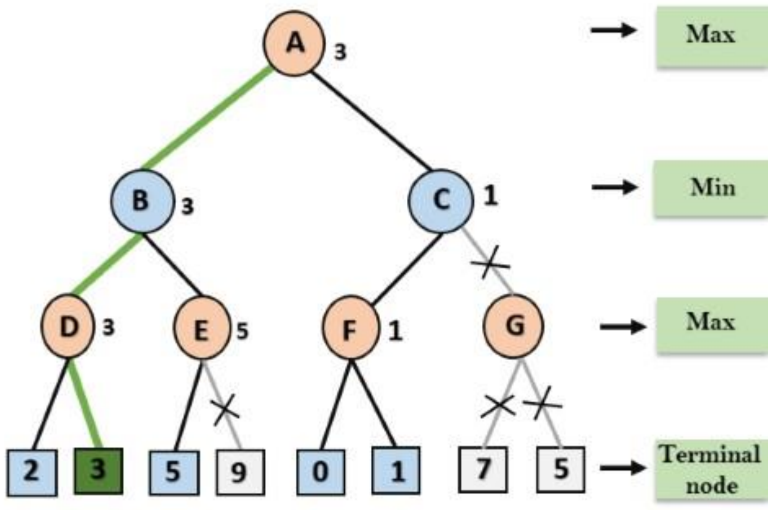


Step 8:

C now returns the value of 1 to A here the best value for A is max (3, 1) = 3.

Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed.

Hence the optimal value for the maximizer is 3 for this example.



- The game will be started from the last level of the game tree, and the value will be chosen accordingly.

The rule which will be followed is: **"Explore nodes if necessary otherwise prune the unnecessary nodes."**

10. Explain in detail about constraint satisfaction problems (CSP) with an Suitable example.

10.1 Constraint satisfaction problems (CSP) – Definition

- A **constraint satisfaction problem (CSP)** is a problem that requires its solution within some limitations or conditions also known as constraints.
- It consists of the following:
  - A finite set of **variables** which stores the solution (V = {V1, V2, V3, ……, Vn})
  - A set of **discrete** values known as **domain** from which the solution is picked (D = {D1, D2, D3, …… ,Dn})
  - A finite set of **constraints** (C = {C1, C2, C3,…….., Cn})

10.2 CSP Algorithms and Problems

1. CryptArithmetic (Coding alphabets to numbers.)
2. n-Queen (In an n-queen problem, n queens should be placed in an nXn matrix such that no queen shares the same row, column or diagonal.)
3. Map Coloring (coloring different regions of map, ensuring no adjacent regions have the same color)
4. Crossword (everyday puzzles appearing in newspapers)
5. Sudoku (a number grid)
6. Latin Square Problem

Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is:

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as **{X_1=v_1, X_2=v_2, and so on…}.**

An assignment of values to a variable can be done in three ways:

- **Consistent or Legal Assignment:** An assignment which does not violate any constraint or rule is called Consistent or legal assignment.

- **Complete Assignment:** An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
- **Partial Assignment:** An assignment which assigns values to some of the variables only. Such type of assignments are called Partial assignments.

Types of Domains in CSP

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example,** a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value $\mathbf{x_2}$ will contain a value which lies between **x1** and **x3**.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

10.3 Map Coloring / Graph Coloring Problem

Solution:

- It is a map of **Australia**; it consists of states and territories as in Figure 1.24.
- The task will be coloring each region with the colors red, green or blue.
- In such case, no neighboring regions will have the same color.
- The set of the domain in each variable is

$$D_i = \{red, green, blue\}$$

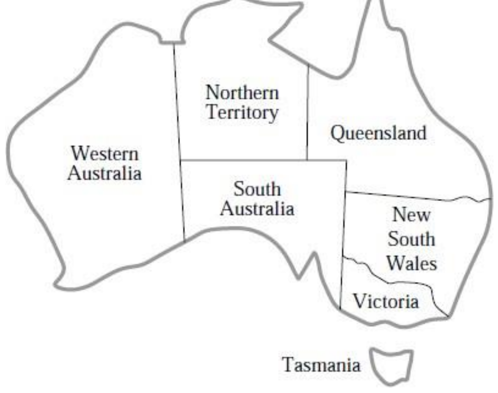


Figure 1.24- States and Territories of Australia

- Formulate the variables to the particular regions as shown in figure 1.25:

$$X = \{WA, NT, Q, NSW, V, SA, T\}.$$

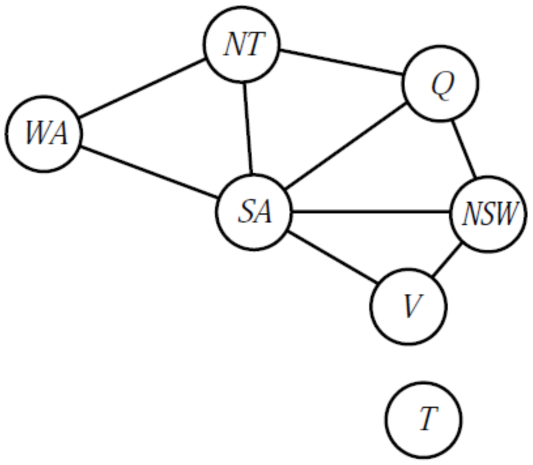


Figure 1.25- Map Coloring of Australia

- The constraints require neighboring regions to have distinct colors.
- Since there are nine places where regions border, there are nine constraints:

$$\begin{aligned} C = \{&SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ &WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}. \end{aligned}$$

Coloring a map with three colors:

- If it starts coloring with the state SA, then any of the 3 colors can be chosen.
- Means, here there are 3 possibilities.
- And when moving to the next node, it says WA and there are 2 colors that can be chosen. That means it has 2 possibilities.
- The remaining states can be colored with the colors carefully.
- And there will be possibility chance of choosing only one particular color that does not match the neighboring state's color.
- **So the number of possibilities is 3 x 2 = 6.**
- As the state T is not connected, it can be mapped with any of the 3 colors. There will be 3 possibilities to color this state.
- So, there can be 6 x 3 = 18 solutions that can be obtained by coloring each of the six possibilities with 3 times with 3 colors to the state TA.
- **So the number of solutions is 6 x 3 = 18.**

Therefore, there will be 18 solutions for coloring a map with three colors.

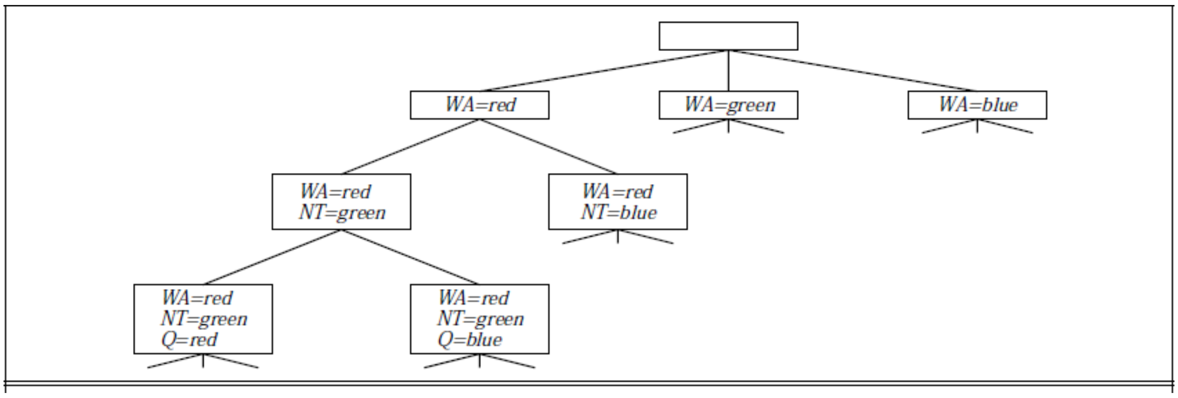


Figure 1.26 - Map Coloring Solution

There are many possible solutions to this problem, such as in figure 1.26

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red\}.$$