**Introduction**

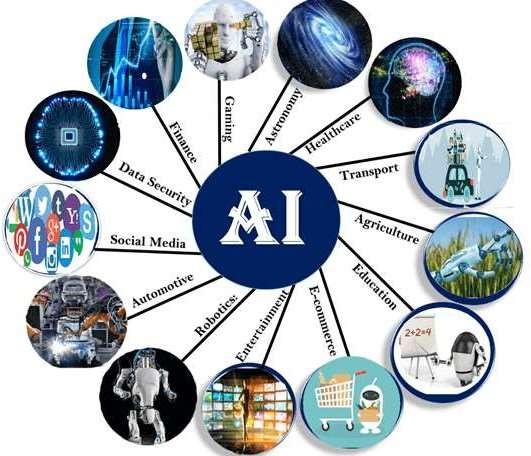
* Artificial Intelligence is composed of two words Artificial and Intelligence, where Artificial defines *"man-made, "thinking* defines intelligence and *power"*, hence AI means *"a man-made thinking power.“*
* "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions.“
* With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

## Goals of Artificial Intelligence

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as
   * Proving a theorem
   * Playing chess
   * Plan some surgical operation
   * Driving a car in traffic

## Why Artificial Intelligence?

* With the help of AI, **you can create such software or devices which can solve real-world problems** very easily and with accuracy such as health issues, marketing, traffic issues, etc.
* With the help of AI, **you can create your personal virtual Assistant**, such as Cortana, Google Assistant, Siri, etc.
* With the help of AI, **you can build such Robots** which can work in an environment where survival of humans can be at risk.AI **opens a path for other new technologies**, new devices, and new Opportunities.

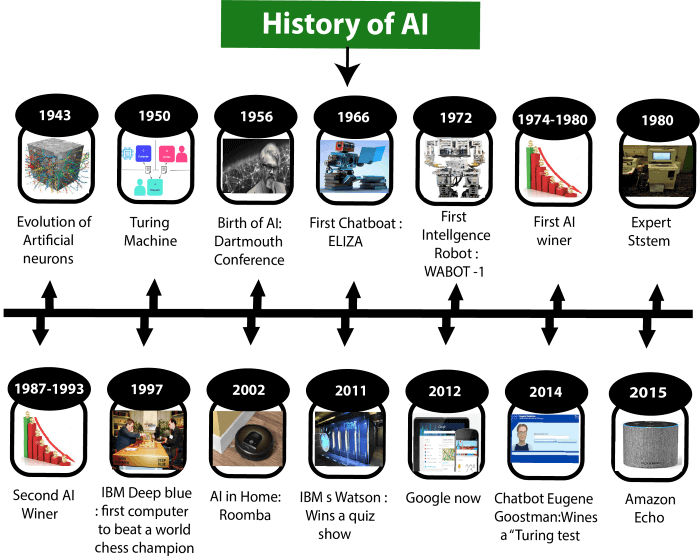
**Applications of AI**

**History of Artificial Intelligence**

Artificial Intelligence is not a new word and not a new technology for

researchers. This technology is much older than you would imagine.

Even there are the myths of Mechanical men in Ancient Greek and Egyptian

Myths. 

**Types of AI**

Artificial Intelligence can be broadly divided into two categories: AI based on capability and AI based on functionality. Let’s understand each type in detail.

****

1. Narrow AI

Narrow AI is a goal-oriented AI trained to perform a specific task. The machine intelligence that we witness all around us today is a form of narrow AI. Examples of narrow AI include Apple’s Siri and IBM’s Watson supercomputer.

Narrow AI is also referred to as weak AI as it operates within a limited and pre-defined set of parameters, constraints, and contexts.

2. General AI

General AI is an AI version that performs any intellectual task with a human-like efficiency. The objective of general AI is to design a system capable of thinking for itself just like humans do. Currently, general AI is still under research, and efforts are being made to develop machines that have enhanced cognitive capabilities.

3. Super AI

[Super AI](https://www.spiceworks.com/tech/artificial-intelligence/articles/super-artificial-intelligence/) is the AI version that surpasses human intelligence and can perform any task better than a human. Capabilities of a machine with super AI include thinking, reasoning, solving a puzzle, making judgments, learning, and communicating on its own. Today, super AI is a hypothetical concept but represents the future of AI.

Now, let’s understand the types of AI based on functionality.

4. Reactive machines

Reactive machines are basic AI types that do not store past experiences or memories for future actions. Such systems zero in on current scenarios and react to them based on the best possible action. Popular examples of reactive machines include IBM’s Deep Blue system and Google’s AlphaGo.

5. Limited memory machines

Limited memory machines can store and use past experiences or data for a short period of time. For example, a self-driving car can store the speeds of vehicles in its vicinity, their respective distances, speed limits, and other relevant information for it to navigate through the traffic.

6. Theory of mind

Theory of mind refers to the type of AI that can understand human emotions and beliefs and socially interact like humans. This AI type has not yet been developed but is in contention for the future.

7. Self-aware AI

Self-aware AI deals with super-intelligent machines with their consciousness, sentiments, emotions, and beliefs. Such systems are expected to be smarter than a human mind and may outperform us in assigned tasks. Self-aware AI is still a distant reality, but efforts are being made in this direction.

## Advantages of Artificial Intelligence

* **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
* **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
* **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
* **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
  + **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
  + **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

**AI Agents:**

What is an Agent?

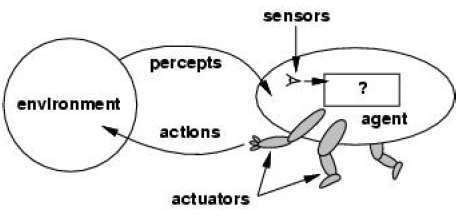
An agent can be anything that perceive its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of perceiving, thinking, and acting.

• Human-Agent: A human agent has eyes, ears, and other organs which work for

sensors and hand, legs, vocal tract work for actuators.

• Robotic Agent: A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.

• Software Agent: Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.



**Percept**

We use the term **percept** to refer to the agent's perceptual inputs at any given instant.

**Percept Sequence**

An agent's **percept sequence** is the complete history of everything the agent has ever perceived.

**Agent function**

Mathematically speaking, we say that an agent's behavior is described

by the **agent**

#### Specifying the Agent’s Task Environment

* + Performance measure
  + Environment
  + Actuators
  + Sensors

## PEAS

* All these are grouped together under the heading of the **task**

**environment.**

* We call this the **PEAS** (Performance, Environment, Actuators,

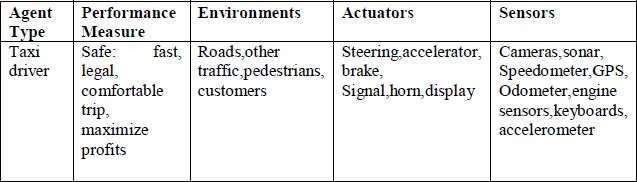
Sensors) description.

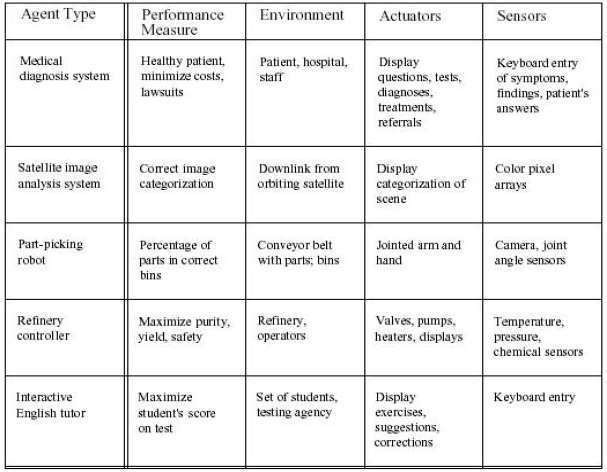
* In designing an agent, the first step must always be to specify

the task environment as fully as possible.

* The following table shows PEAS description of the task

environment for an automated taxi.





**The Level of the Model: -**

The term "level of the model" in AI can be interpreted in different contexts. Here are some specific ways to understand and categorize the levels of AI models:

1. Levels of Model Complexity

* Basic Models: These include simple algorithms and techniques like linear regression, logistic regression, k-nearest neighbors (KNN), and decision trees. These models are easy to implement and understand but may not capture complex patterns in data.
* Intermediate Models: More complex algorithms such as Random Forests, Gradient Boosting Machines (GBMs), Support Vector Machines (SVMs), and basic artificial neural networks (ANNs). These models can handle more complex patterns and interactions in data.
* Advanced Models: Deep learning models including Convolutional Neural Networks (CNNs) for image recognition, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks for sequential data, and advanced architectures like Transformers for natural language processing.
* State-of-the-Art Models: Cutting-edge models such as BERT, GPT-4, and other large pre-trained models that are used for a wide range of tasks and are fine-tuned for specific applications. These models often require significant computational resources and large datasets.

2. Levels of Model Deployment

* Experimental Level: Models that are in the research or experimental phase. They are typically developed to explore new algorithms, techniques, or applications.
* Prototype Level: Early versions of models that are being tested in a controlled environment. They are used to evaluate feasibility, performance, and potential issues.
* Production Level: Fully developed and tested models that are deployed in real-world applications. These models are optimized for performance, reliability, and scalability.
* Maintenance Level: Models that are actively monitored and maintained post-deployment. This includes updating models with new data, retraining, and ensuring they continue to perform well over time.

3. Levels of AI Capability

* Narrow AI (ANI): AI systems designed for specific tasks. These models are highly specialized and can perform specific tasks very well but lack generalization to other tasks.
* General AI (AGI): Hypothetical AI systems with the ability to understand, learn, and apply knowledge across a wide range of tasks at a human-like level. AGI is still theoretical and does not yet exist.
* Superintelligent AI (ASI): Theoretical AI that surpasses human intelligence in all aspects. ASI remains speculative and is a topic of significant debate regarding its implications and potential risks.

4. Application-Based Levels

* Rule-Based Models: AI systems based on a set of predefined rules. These are often used in expert systems and early AI applications.
* Supervised Learning Models: Models that are trained on labeled data. Examples include classification and regression models.
* Unsupervised Learning Models: Models that find patterns in unlabeled data. Examples include clustering and association models.
* Reinforcement Learning Models: Models that learn by interacting with an environment and receiving feedback in the form of rewards or penalties.
* Generative Models: Models that generate new data similar to the training data. Examples include Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

## Performance Measures (Criteria for Success)

The performance measure that defines the criterion of success of an Agent’s behavior.

–The agent's prior knowledge of the environment.

–The actions that the agent can perform.

–The agent's percept sequence.

## AI Problem Solving

* Before solving the problem, we must understand the problem clearly
* To solve the problem we need the formal description of the problem
* What are aspects to understand the problem?

1. What is the goal of the problem? (Goal Formulation)
2. What is the implicit criteria for success? (how success is defined?
3. What is the initial situation?
4. Ability to perform?

(What are the procedures?)

## Well Defined Problem

* The process of deciding what actions and states to be considered is called as “Problem Formulation”.
* The solution of many problems can be described by finding a sequence of actions that lead to a desirable goal.
* Components of Well Defined Problem:

1. **Initial state** (Where the agent starts in)
2. **Successor Function** (Description of the possible actions)
3. **State Space** (Set of all possible actions)
4. **Path Cost** (Cost to reach the Goal)
5. **Goal Test** (Determines whether a given state is final state or not? Eg: Checkmate in Chess is Goal)

## Steps in Problem Solving

**Step 1:** Goal Setting

**Step 2:** Goal Formulation

-> to observe the current state

-> to tabulate agent performance measures

**Step 3:** Problem Formulation

-> what will be the sequence of actions?

-> what will be the sequence of states?

**Step 4:** Search in unknown environment (Learning)

**Step 5:** Execution Phase

# State Space Search

A state space represents a problem in terms of states and operators

that change states.

**A state space consists of:**

1. A representation of the **states** the system can be in. For example, in a board game, the board represents the current state of the game.
2. A set of **operators** that can change one state into another state.

In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.

1. An **initial state**.
2. A set of **final states**; some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

## Problem Characteristics

1. Is the problem decomposable to smaller or easier problems?
2. Can problem solution steps be ignored or undone?
   1. Ignorable: Solution steps can be ignored
   2. Recoverable: Solution steps can be undone or

backtracking

* 1. Irrecoverable: Moves cannot be retracted

1. Is the problem universe predictable?
   1. Problem with certain outcome(8 Queens)
   2. Problem with uncertain outcome(Chess)
2. Is a good solution absolute or relative?
3. Is the solution a state or a path?
4. Is the problem using knowledge base?
5. What is the role of the knowledge?
6. Does the task require a periodic human interaction with computer?

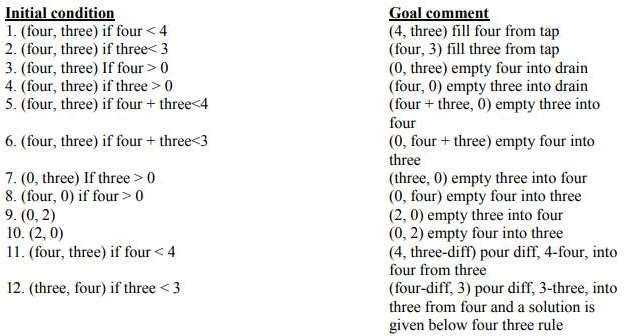
The Water Jug Problem

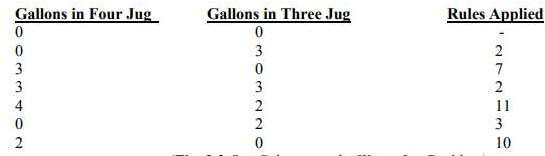
In this problem, we use two jugs called four and three; four holds a maximum of four gallons of water and three a maximum of three gallons of water. How can we get two gallons of water in the four jug?

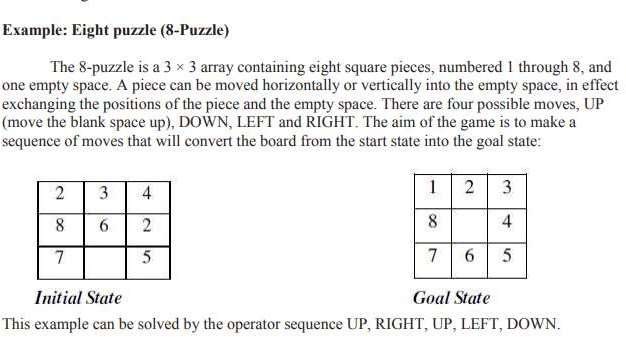
The state space is a set of prearranged pairs giving the number of gallons of water in the pair of jugs at any time, i.e., (four, three) where four = 0, 1, 2, 3 or 4 and three = 0, 1,

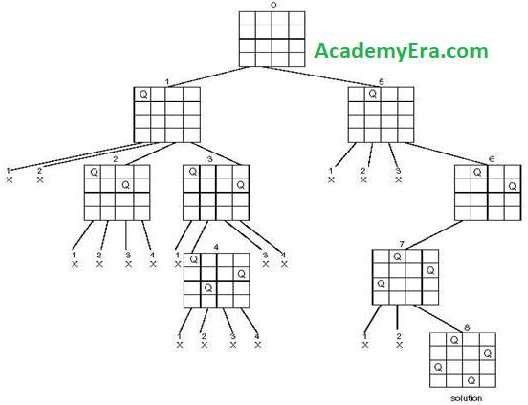
2 or 3.

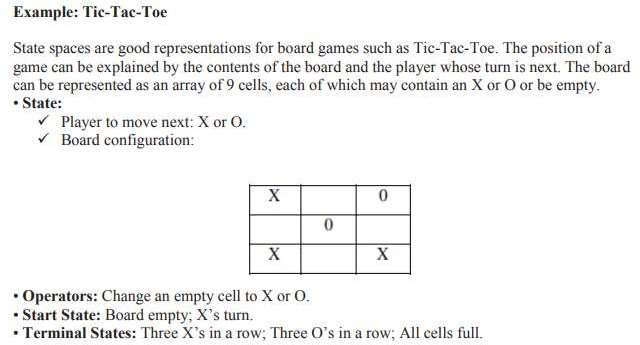
The start state is (0, 0) and the goal state is (2, n) where n may be any but it is limited to three holding from 0 to 3 gallons of water or empty. Three and four shows the name and numerical number shows the amount of water in jugs for solving the water jug problem. The major production rules for solving this problem are shown below:



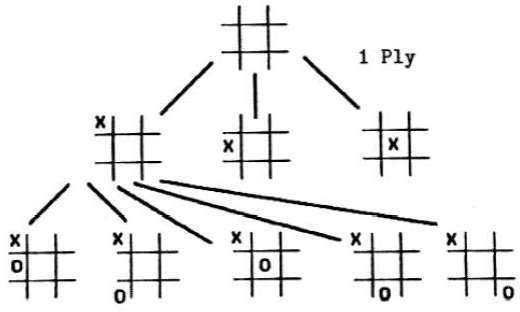




State Space Tree for 4 - Queens Problem



## Search Tree for Tic-Tac-Toe Game

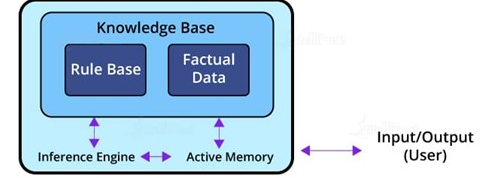
The sequence of states formed by possible moves is called a search tree. Each level of the tree is called a ply.

**Production Systems:**

**What is a Production System in AI?**

A production system in AI is a framework that assists in developing computer programs to automate a wide range of tasks.By establishing rules, a production system empowers machines to demonstrate particular behaviours and adapt to their surroundings.

In Artificial Intelligence, a production system serves as a cognitive architecture. It encompasses rules representing declarative knowledge, allowing machines to make decisions and act based on different conditions.



A production system’s architecture consists of rules structured as left-hand side (LHS) and right-hand side (RHS) equations. The LHS specifies the condition to be evaluated, while the RHS determines the output or action resulting from the estimated condition. This rule-based approach forms the foundation of production systems in AI, enabling machines to process information and respond accordingly.

**Components of a Production System in AI**



**Global Database**

A global database consists of the architecture used as a central data structure. A database contains all the necessary data and information required for the successful completion of a task. It can be divided into two parts as permanent and temporary. The permanent part of the database consists of fixed actions, whereas the temporary part alters according to circumstances.

**Production Rules**

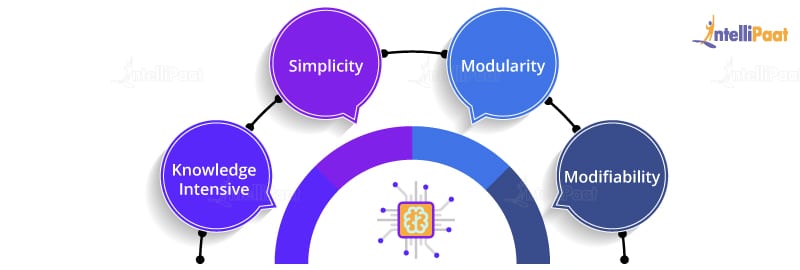
Production rules in AI are the set of rules that operate on the data fetched from the global database. Also, these production rules are bound with precondition and postcondition that gets checked by the database. If a condition is passed through a production rule and gets satisfied by the global database, then the rule is successfully applied. The rules are of the form A®B, where the right-hand side represents an outcome corresponding to the problem state represented by the left-hand side.

**Control System**

The control system checks the applicability of a rule. It helps decide which rule should be applied and terminates the process when the system gives the correct output. It also resolves the conflict of multiple conditions arriving at the same time. The strategy of the control system specifies the sequence of rules that compares the condition from the global database to reach the correct result.

**Characteristics of a Production System**

There are mainly four characteristics of the production system in AI that is simplicity, modifiability, modularity, and knowledge-intensive.



**Simplicity**

The production rule in AI is in the form of an ‘IF-THEN’ statement. Every rule in the production system has a unique structure. It helps represent knowledge and reasoning in the simplest way possible to solve real-world problems. Also, it helps improve the readability and understanding of the production rules.

**Modularity**

The modularity of a production rule helps in its incremental improvement as the production rule can be in discrete parts. The production rule is made from a collection of information and facts that may not have dependencies unless there is a rule connecting them together. The addition or deletion of single information will not have a major effect on the output. Modularity helps enhance the performance of the production system by adjusting the parameters of the rules.

**Modifiability**

The feature of modifiability helps alter the rules as per requirements. Initially, the skeletal form of the production system is created. We then gather the requirements and make changes in the raw structure of the production system. This helps in the iterative improvement of the production system.

**Knowledge-intensive**

Production systems contain knowledge in the form of a human spoken language, i.e., English. It is not built using any programming languages. The knowledge is represented in plain English sentences. Production rules help make productive conclusions from these sentences.

###### Search Algorithms

* The process of finding an element in the data structure is called

as “**Searching”**.

* **Search Graph** is an algorithm that visits vertices and edges in graph in an order based on the connectivity of the graph.
* **Components of Search Algorithm**
* **A State Space:** Set of all possible states where you can be.
* **A Start State:** The state from where the search begins.
* **A Goal Test:** A function that looks at the current state returns whether or not it is the goal state.
* The **Solution** to a search problem is a sequence of actions,

called the **plan** that transforms the start state to the goal state.

This plan is achieved through search algorithms

Search Terminology

* + **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
  + **Problem Instance** − It is Initial state + Goal state.
  + **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
  + **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
  + **Space Complexity** − The maximum number of nodes that are stored in memory.
  + **Time Complexity** − The maximum number of nodes that are created.
  + **Admissibility** − A property of an algorithm to always find an optimal solution.
  + **Branching Factor** − The average number of child nodes in the problem space graph.

**Depth** − Length of the shortest path from initial state to goal state.

**Properties of Search Algorithms:**

* + - **Completeness:** A search algorithm is said to be complete if it

guarantees to return a solution.

* + - **Optimality:** If a solution found for an algorithm is guaranteed to be

the best solution (lowest path cost) among all other solutions.

* + - **Time Complexity:** Time complexity is a measure of time for an

algorithm to complete its task.

* + - **Space Complexity:** It is the maximum storage space required at any

point during the search, as the complexity of the problem.

**Problem-solving agents:**

**Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem- solving agents are the goal-based agents and use atomic representation.

**Issues in the Design of Search Programs**

* Each search process can be considered to be a tree traversal. The object of the search is to find a path from the initial state to a goal state using a tree. The number of nodes generated might be huge; and in practice many of the nodes would not be needed. The secret of a good search routine is to generate only those nodes that are likely to be useful, rather than having a precise tree.

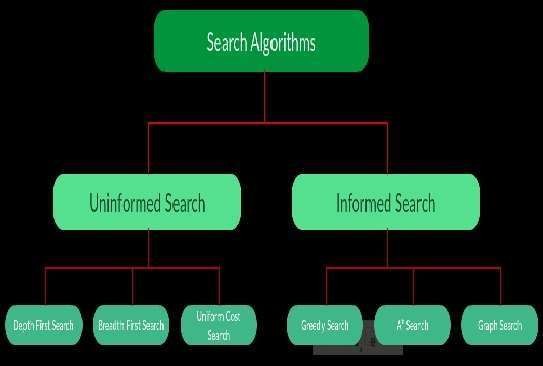
The following issues arise when searching:

* The tree can be searched forward from the initial node to the goal

state or backwards from the goal state to the initial state.

* To select applicable rules, it is critical to have an efficient procedure for matching rules against states.
* How to represent each node of the search process? This is the knowledge representation problem or the frame problem. In games, an array suffices; in other problems, more complex data structures are needed.
* nodes at the next level are expanded.
* Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out(FIFO) queue, assuring that the nodes that are visited first will be expanded first.
* In otherwards, calling TREE-SEARCH (problem, FIFO- QUEUE()) results in breadth-first-search.
* The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.

#### **TYPES OF SEARCH ALGORITHMS**

****

* 1. **Brute-Force Search Strategies (Uninformed / Blind)**
* Breadth-First Search
* Depth-First Search
* Bidirectional Search
* Uniform Cost Search
* Iterative Deepening Depth-First Search
  1. **Informed (Heuristic) Search Strategies**
* Pure Heuristic Search (Open and Closed List)
* Greedy Best First Search
* A \* Search
  1. **Local Search Algorithms**
* Hill-Climbing Search
* Local Beam Search
* Simulated Annealing
* Travelling Salesman Problem

UNINFORMED SEARCH ALGORITHMS

* The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.
* The following uninformed search algorithms are discussed in this section.

1. Depth First Search
2. Breath First Search
3. Uniform Cost 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) that you can go 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.

**UNINFORMED SEARCH STRATEGIES**

* **Uninformed Search Strategies** have no additional information

about states beyond that provided in the **problem definition**.

* **Strategies** that know whether one non goal state is "more promising" than another are called I**nformed search or heuristic search** strategies.
* There are six uninformed search strategies as given below.
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search
  6. Bidirectional Search

**BREADTH FIRST SEARCH (BFS)**

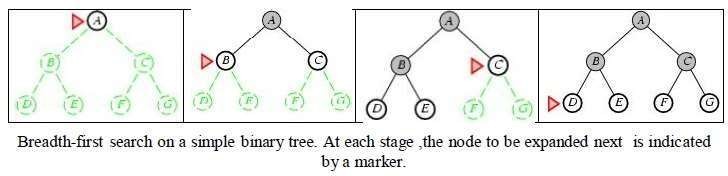
* Breadth-first search in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
* In general, all the nodes are expanded at a given depth in the search

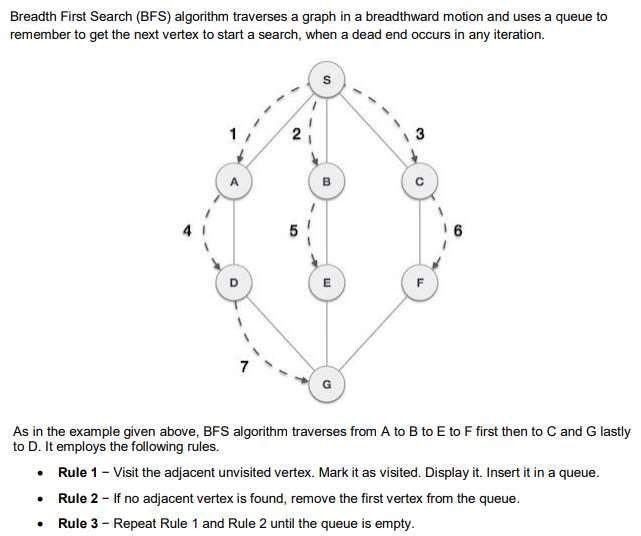
tree before any nodes at the next level are expanded.

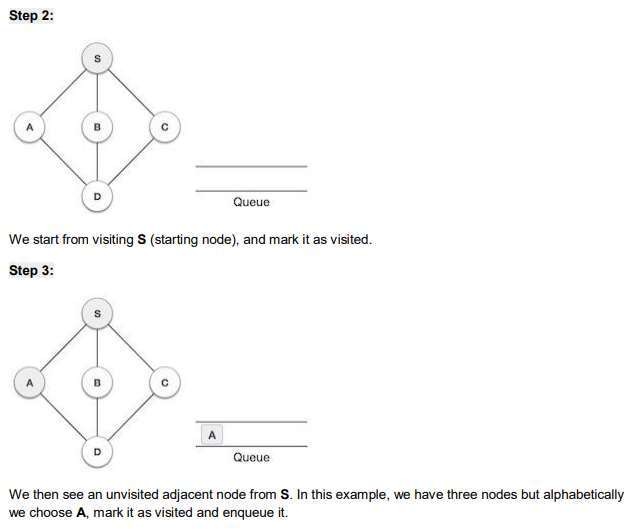
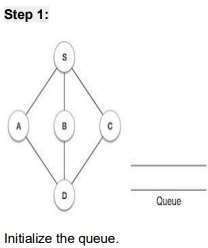
* Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out(FIFO) queue, assuring that the nodes that are visited first will be expanded first.
* In otherwards, calling TREE-SEARCH (problem, FIFO-QUEUE())

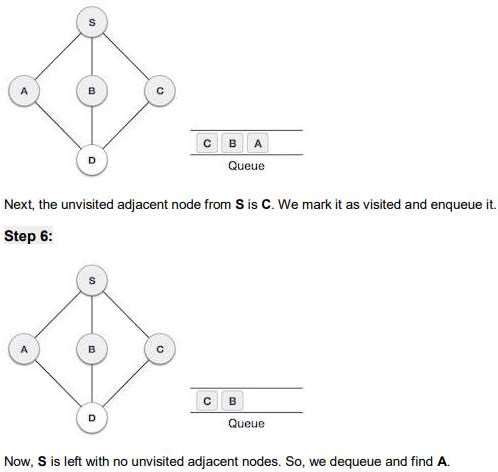
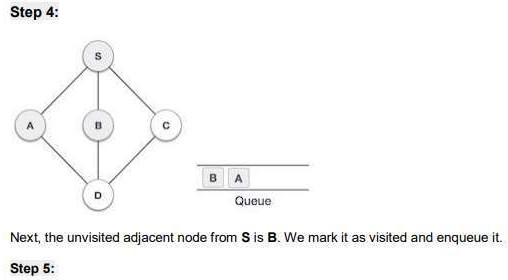
results in breadth-first-search.

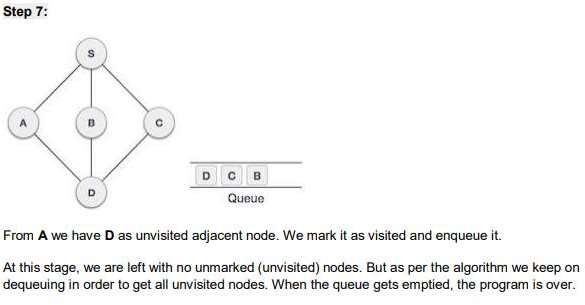
* The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.

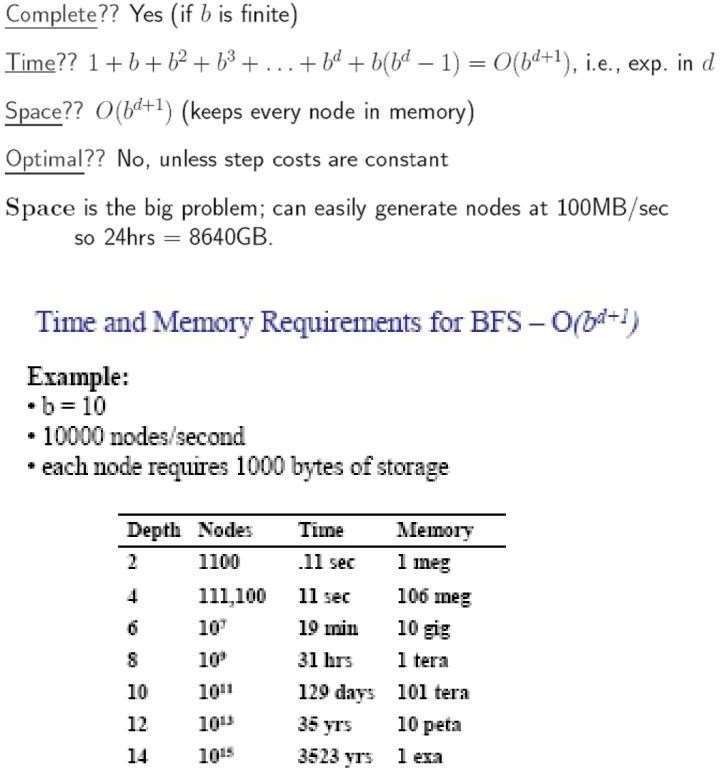


**BFS - Example**



**Properties of BFS**

#### **DEPTH FIRST SEARCH (DFS)**

#### Depth-first-search always expands the deepest node in the current fringe of the search.The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.

* + As those nodes are expanded, they are dropped from the fringe,
  + so then the search "backs up" to the next shallowest node that still has unexplored successors.
  + This strategy can be implemented by TREE-SEARCH with a last-in-

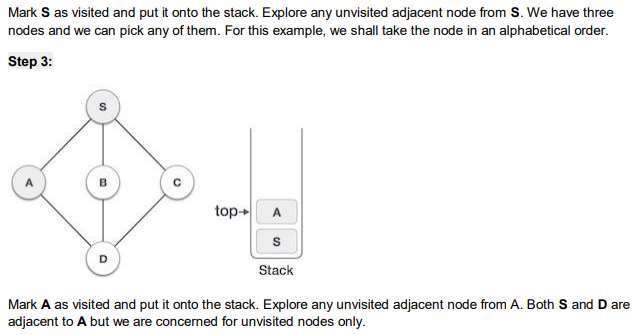
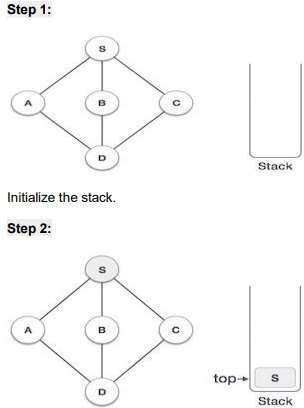
first- out (LIFO) queue, also known as a stack.

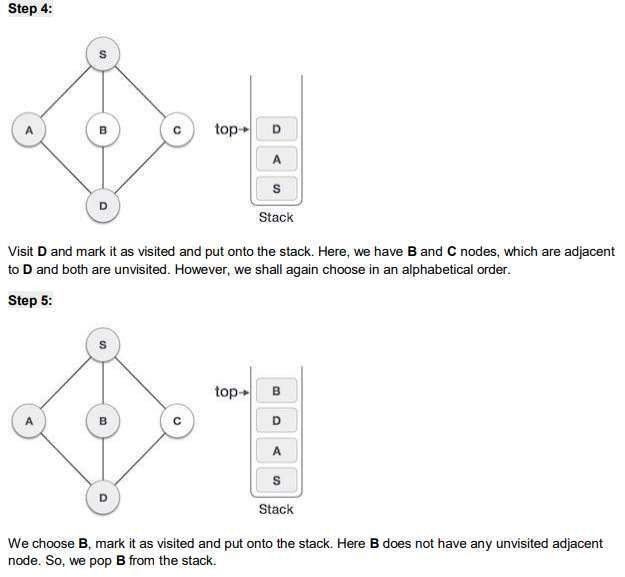
* + Depth-first-search has very modest memory requirements.
  + It needs to store only a single path from the root to a leaf node along with the remaining unexpanded sibling nodes for each node on the path.
  + Once the node has been expanded, it can be removed from the

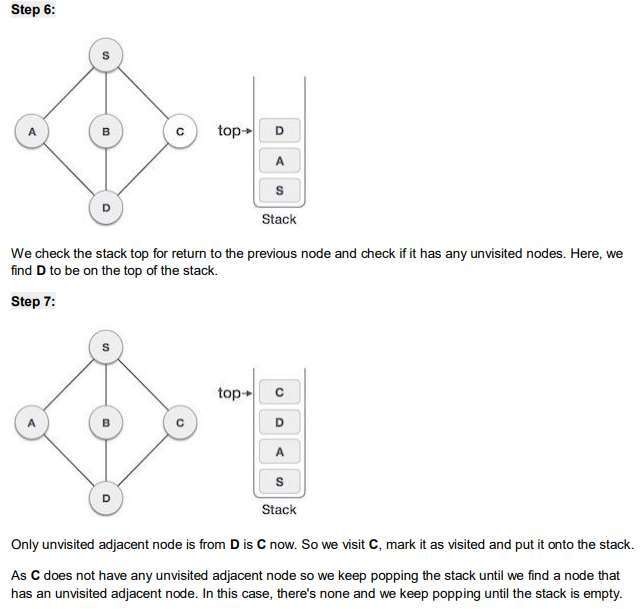
memory, as soon as its descendants have been fully explored.

* + For a state space with a branching factor b and maximum depth m, depth- first-search requires storage of only bm + 1 nodes

# DFS - Example





|  |  |  |
| --- | --- | --- |
| **Parameters** | **BFS** | **DFS** |
| **Data structure** | Queue (FIFO) | Stack (LIFO) |
| **Source** | It is better when target is closer to source | It is better when target is far from source |
| **Speed** | Slower | Faster |
| **Time Complexity** | O(V+E) | O(V+E) |
| **Suitability for Decision Trees** | BFS considers all neighbor, so it is not suitable for decision tree | Suitable for decision tree |

**INFORMED SEARCH ALGORITHMS**

* Here, the algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a *heuristic.*
* In this section, we will discuss the following search algorithms.

1. Greedy Search
2. A\* Tree Search
3. AO\* Graph Search

* **Search Heuristics:** In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state. For examples – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.)

**INFORMED SEARCH AND EXPLORATION**

**Informed (Heuristic) Search Strategies**

* **Informed search strategy** is one that uses problem-specific knowledge beyond the definition of the problem itself.
* It can find solutions more efficiently than uninformed strategy.

**Best-first search**

* **Best-first search** is an instance of general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function** f(n).
* The node with lowest evaluation is selected for expansion, because the evaluation measures the distance to the goal.
* This can be implemented using a priority-queue, a data structure that will maintain the fringe in ascending order of f- values.

**Heuristic functions**

* A **heuristic function** or simply a **heuristic** is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search.
* The key component of Best-first search algorithm is a

**heuristic function**, denoted by h(n).

h(n) = estimated cost of the **cheapest path** from node n to a **goal node**.

* Heuristic function are the most common form in which additional knowledge is imparted to the search algorithm.

**Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

**Admissibility of the heuristic function is given as:**

h(n) <= h\*(n)

Here h(n) is heuristic cost, and h\*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

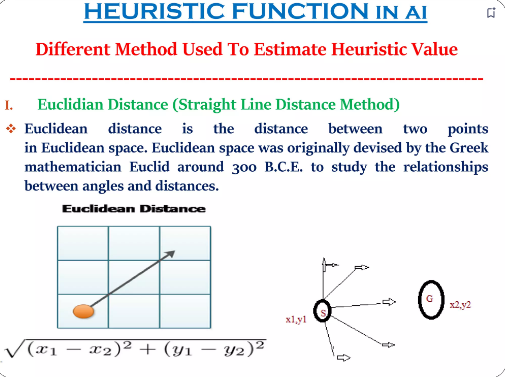
* 1. **Search with closed and open list**

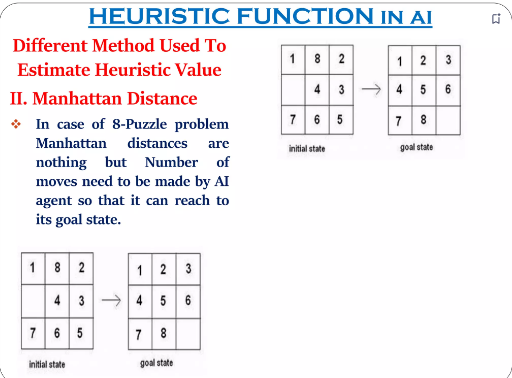
Pure Heuristic Search

* It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.
* In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

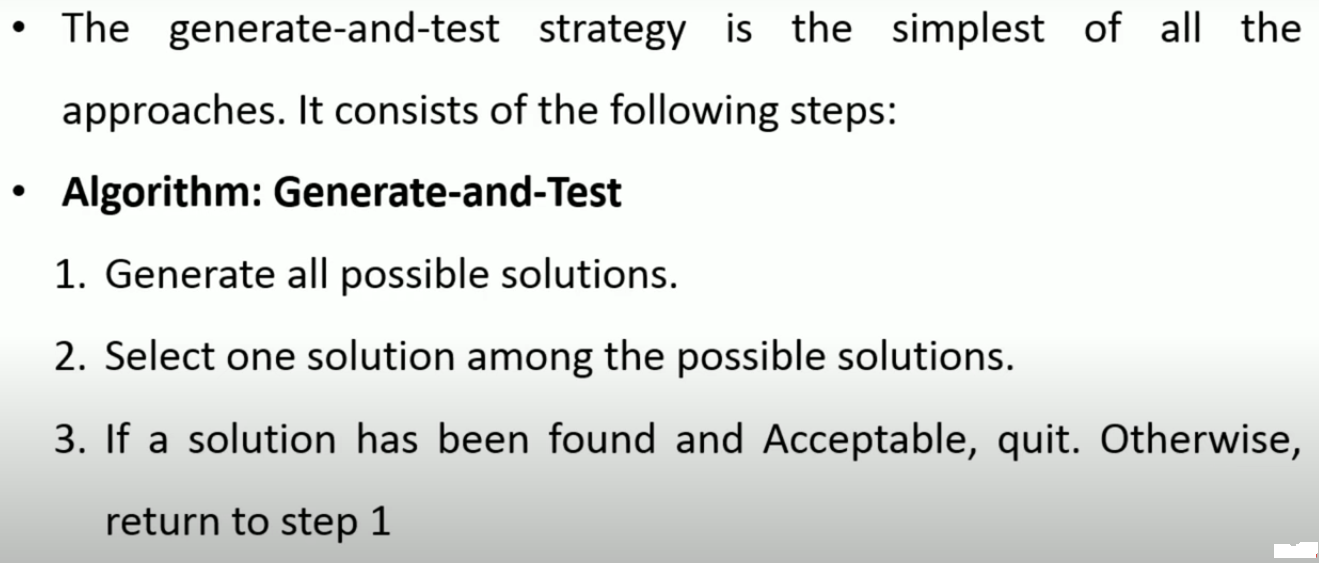
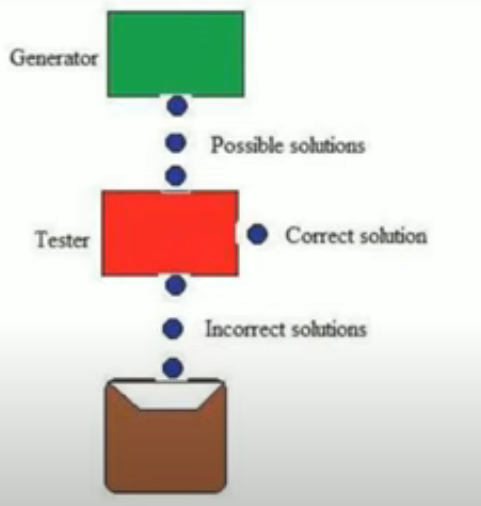
**Pure Heuristic Search:**

* Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
* On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.









* It’s a informed search technique.
* It Follows Heuristic approach.
* It is based on DFS with backtracking.

Properties of Good generators:

1. Complete: It should be giving an optimal solution from the given possible solutions.

2) Non- Redundancy: It should not be taking the same paths again and again. If we visit

the same states again and again the number of solutions will be in

exponential number thereby increasing the time complexity.

1. Informed: The generator should have enough knowledge of the domain it is working

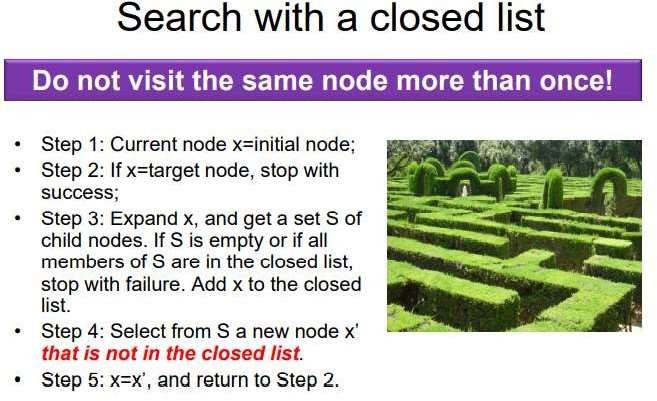
on, which then can make it work faster and reduce the time complexity.

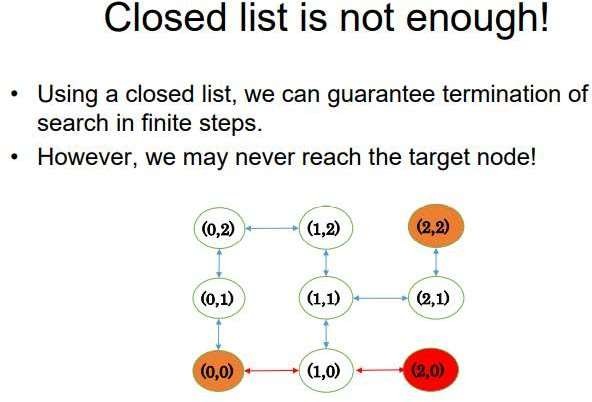
(example: we need to generate a pin of 3 numbers, each number having two digits

taking 0-99 i.e. 100 numbers which make the time complexity sum up to (100)3= 1Million

But if we have knowledge of domain , lets say it is just taking prime numbers

from 0-100 so we get time complexity as (25)3= 15625.





**Hill Climbing Algorithm**

 Hill climbing algorithm is a local search algorithm which continuously moves in the

direction of increasing elevation/value to find the peak of the mountain or best solution to the

problem. It terminates when it reaches a peak value where no neighbor has a higher value.

 It is also called greedy local search as it only looks to its good immediate neighbor state

and not beyond that.

 Hill Climbing is mostly used when a good heuristic is available.

 In this algorithm, we don't need to maintain and handle the search tree or graph as it only

keeps a single current state.

The idea behind hill climbing is as follows.

1. Pick a random point in the search space.

2. Consider all the neighbors of the current state.

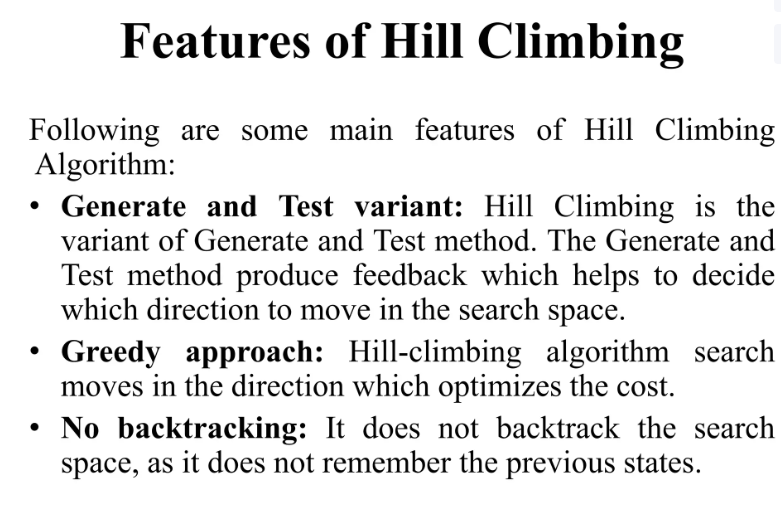
3. Choose the neighbor with the best quality and move to that state.

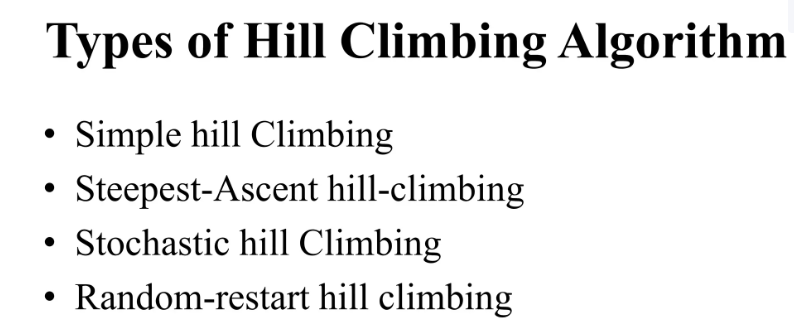
4. Repeat 2 thru 4 until all the neighboring states are of lower quality.

5. Return the current state as the solution state.

**Terminology in Hill Climbing**

****

****

****

**CHANLLEGES OF HILL CLIMBLING**

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

* Local maximum: At a local maximum all neighboring states have a value that is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.   
  To overcome the local maximum problem: Utilize the [backtracking technique](https://www.geeksforgeeks.org/backtracking-set-1-the-knights-tour-problem/). Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
* Plateau: On the plateau, all neighbors have the same value. Hence, it is not possible to select the best direction.   
  To overcome plateaus: Make a big jump. Randomly select a state far away from the current state. Chances are that we will land in a non-plateau region.
* Ridge: Any point on a ridge can look like a peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.   
  To overcome Ridge: In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth- first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best- first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function,

f(n)= g(n) + h(n), where h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

**Best first search algorithm:**

* **Step 1:** Place the starting node into the OPEN list.
* **Step 2:** If the OPEN list is empty, Stop and return failure.
* **Step 3:** Remove the node n, from the OPEN list which has the

lowest value of h(n), and places it in the CLOSED list.

* **Step 4:** Expand the node n, and generate the successors of node n.
* **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
* **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
* **Step 7:** Return to Step 2.

**Advantages:**

* Best first search can switch between BFS and DFS by gaining the

advantages of both the algorithms.

* This algorithm is more efficient than BFS and DFS algorithms.

**Disadvantages:**

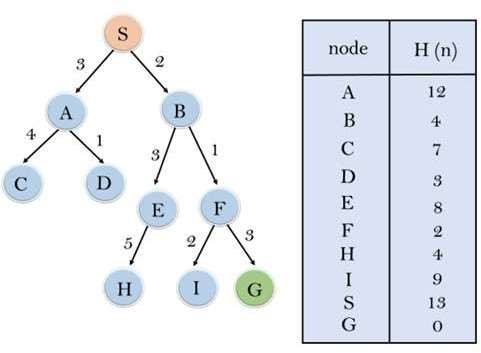
* It can behave as an unguided depth-first search in the worst case

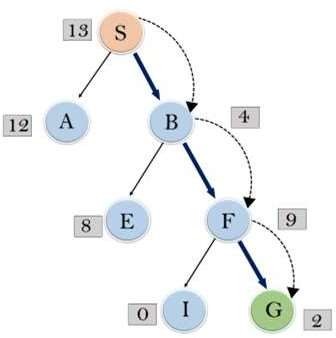
scenario.

* It can get stuck in a loop as DFS.
* This algorithm is not optimal.

Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.

* **Expand the nodes of S and put in the CLOSED list**

**Initialization:** Open [A, B], Closed [S] **Iteration 1:** Open [A], Closed [S, B] **Iteration 2:** Open [E, F, A], Closed [S, B]

: Open [E, A], Closed [S, B, F]

**Iteration 3:** Open [I, G, E, A], Closed [S, B, F]

: Open [I, E, A], Closed [S, B, F, G]

* Hence the final solution path will be: **S----> B----->F----> G**
* **Time Complexity:** The worst case time complexity of Greedy best

first search is O(bm).

* **Space Complexity:** The worst case space complexity of Greedy best first search is O(bm). Where, m is the maximum depth of the search space.
* **Complete:** Greedy best-first search is also incomplete, even if the

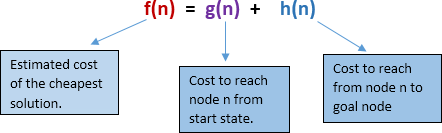
given state space is finite.

* **Optimal:** Greedy best first search algorithm is not optimal.

**A\* SEARCH ALGORITHM:**

A\* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A\* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A\* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A\* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



* **Heuristic:** The following points should be noted wrt heuristics in A\* search. **f(x)=g(x)+h(x)**
* Here, h(x) is called the **forward cost**, and is an estimate of the distance of the current node from the goal node. And, 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 lowest f(x) value.

**Algorithm of A\* search:**

**Step1:** Place the starting node in the OPEN list.

**Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

**Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

**Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

**Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

**Step 6:** Return to **Step 2**.

**Advantages:**

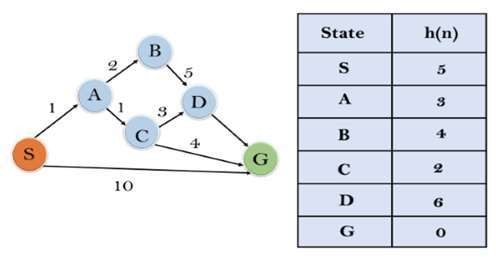
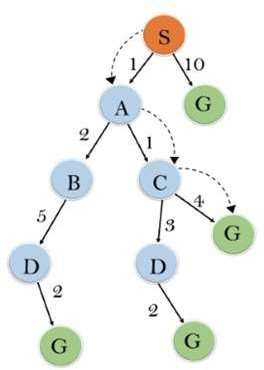
* A\* search algorithm is the best algorithm than other search algorithms.
* A\* search algorithm is optimal and complete.
* This algorithm can solve very complex problems.

**Disadvantages:**

* It does not always produce the shortest path as it mostly based on heuristics and approximation.
* A\* search algorithm has some complexity issues.
* The main drawback of A\* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

**Example:**

* In this example, we will traverse the given graph using the A\* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
* Here we will use OPEN and CLOSED list.



**Initialization:** {(S, 5)}

**Iteration1:** {(S--> A, 4), (S-->G, 10)}

**Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

**Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S-

-> A-->B, 7), (S-->G, 10)}

**Iteration 4** will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

**Points to remember:**

* A\* algorithm returns the path which occurred first, and it does not search for all remaining paths.
* The efficiency of A\* algorithm depends on the quality of heuristic.
* A\* algorithm expands all nodes which satisfy the condition f(n)

**Complete:** A\* algorithm is complete as long as:

* Branching factor is finite.
* Cost at every action is fixed.

**Optimal:** A\* search algorithm is optimal if it follows below two conditions:

**Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A\* tree search. An admissible heuristic is optimistic in nature.

**Consistency:** Second required condition is consistency for only A\* graph-search.

* If the heuristic function is admissible, then A\* tree search will always

find the least cost path.

**Time Complexity:** The time complexity of A\* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

**Space Complexity:** The space complexity of A\* search algorithm

**Why AO-Star Algorithm?**



Imagine you’re in a maze, looking for the exit. A normal search algorithm would try every path until it finds the way out. But what if the maze keeps changing? Walls appear and disappear, The map you have might not be entirely accurate, there could be hidden dangers or shortcuts you don’t know about. This is where the AO\* algorithm comes in, like a trusty guide who can help you find the best path even when the conditions changes.

**What is AO\* Algorithm?**

The [AO\* algorithm](https://www.iosrjournals.org/iosr-jce/papers/Vol17-issue4/Version-1/R01741124127.pdf), short for “Anytime Repairing A\*” or “Anytime Optimistic” is a type of search algorithm used in artificial intelligence and robotics for solving pathfinding problems in a graph or network. It is an extension of the A\* algorithm, which is widelyused for finding the shortest path between nodes in a graph.

The primary goal of AO-Star Algorithm is to find an optimal path between a start node and a goal node while minimizing the computational resources required. It achieves this by performing a best-first search, similar to A\*, but with the added capability of being able to incrementally improve the solution as more computational resources become available.

How it is different from A\* Algorithm?

Like[A\* Algorithm](https://aismartclass.net/blog/what-is-a-star-algorithm/), AO-Star Algorithm utilizes heuristics to guide the search towards the goal efficiently. However, unlike A\*, AO\* is designed to work with graphs or search spaces where the heuristic function may not always be accurate or consistent.

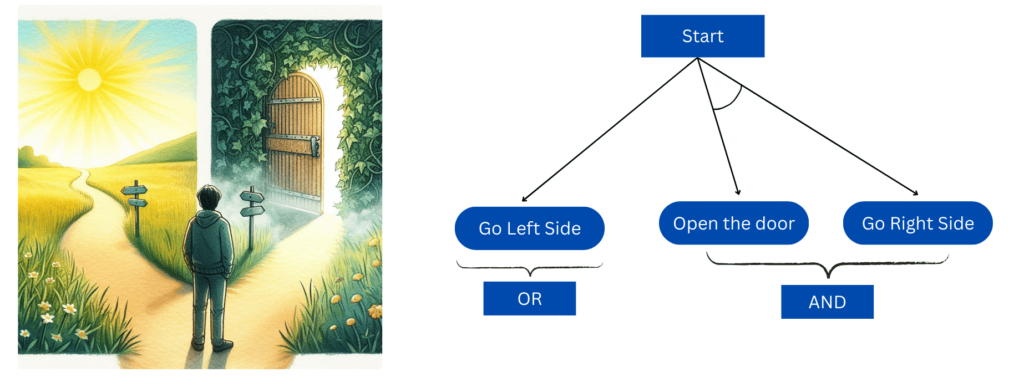
The key feature of AO\* is its ability to handle inadmissible or inconsistent heuristic functions, which means that the heuristic might overestimate or underestimate the actual cost to reach the goal. To accommodate this, AO\* employs a technique called “anytime optimization.” It continually updates its search based on new information to refine its estimate of the optimal solution.

Anytime Repairing A\* or Anytime Optimistic? Which terminology is correct?

The term “AO\*” or “AO-Star Algorithm” can indeed stand for “Anytime Repairing A\*” or “Anytime Optimistic”, depending on the context and the specific variant of the algorithm being discussed. Both interpretations are valid and are used in different contexts within the field of artificial intelligence and robotics.

1. Anytime Repairing A\*:
   * In this interpretation, “AO\*” refers to “Anytime Repairing A\*.” This variant of the algorithm focuses on continuously repairing or improving the current solution as more computational resources become available. It incrementally refines the solution while maintaining optimality guarantees.
2. Anytime Optimistic:
   * Alternatively, “AO\*” can also stand for “Anytime Optimistic.” In this context, the algorithm maintains an optimistic view of its current solution and continuously seeks to improve it within the given time or resource constraints. It aims to provide an optimal or near-optimal solution within the available resources.

Core Concept of AO\* Algorithm



##### Working of AO\* algorithm:

The evaluation function in AO\* looks like this:

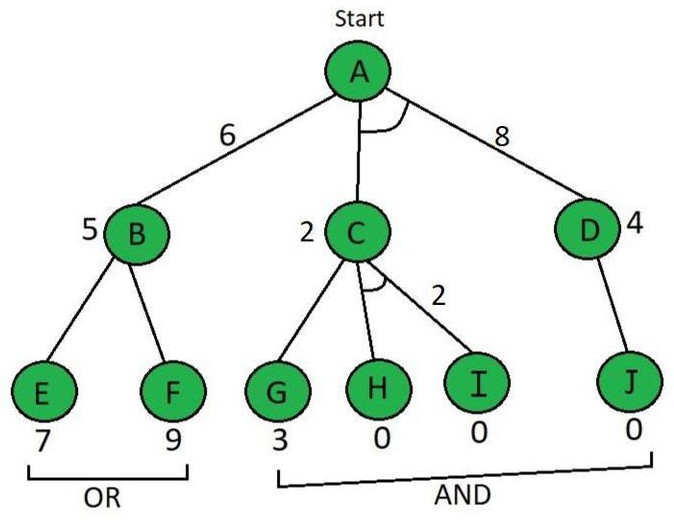
##### f(n) = g(n) + h(n)

**f(n) = Actual cost + Estimated cost**

here,

f(n) = The actual cost of traversal.

g(n) = the cost from the initial node to the current node.

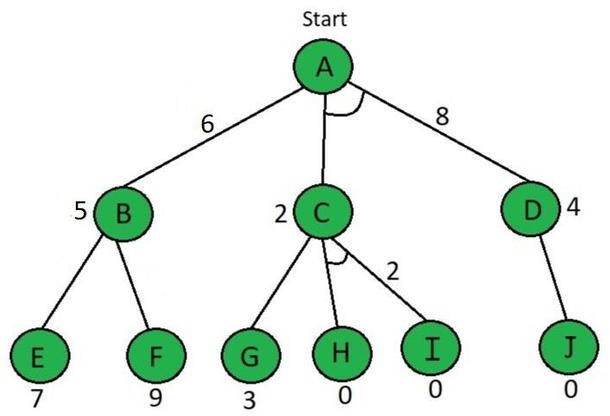
h(n) = estimated cost from the current node to the goal state.

*AO\* Algorithm – Question tree*

Here in the above example below the Node which is given is the heuristic value

i.e **h(n)**. Edge length is considered as **1**.

**Step 1**



Start from node A, f(A⇢B) = g(B) + h(B)

= 1 + 5 ……here **g(n)=1** is taken by default for path cost

= 6

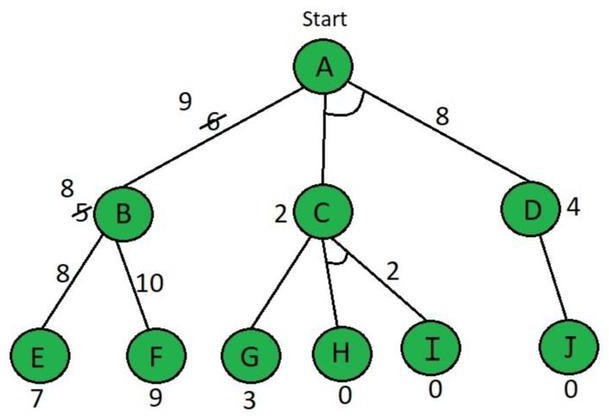
f(A⇢C+D) = g(c) + h(c) + g(d) + h(d)

= 1 + 2 + 1 + 4 ……here we have added **C & D** because they are in

##### AND

= 8

So, by calculation **A**⇢**B** path is chosen which is the minimum path, i.e **f(A**⇢**B) Step 2**



*AO\* Algorithm (Step-2)*

According to the answer of step 1, explore node B Here the value of E & F is calculated as follows,

f(B⇢E) = g(e) + h(e)

f(C⇢H+I) = g(h) + h(h) + g(i) + h(i)

f(C⇢H+I) = 1 + 0 + 1 + 0 ……here we have added **H & I** because they are in **AND**

= 2

**f(C**⇢**H+I)** is selected as the path with the lowest cost and the heuristic is also left unchanged

because it matches the actual cost. Paths H & I are solved because the heuristic for those paths is **0,**

but Path **A**⇢**D** needs to be calculated because it has an **AND**.

f(D⇢J) = g(j) + h(j) f(D⇢J) = 1 + 0

= 1

the heuristic of node D needs to be updated to 1.

f(A⇢C+D) = g(c) + h(c) + g(d) + h(d)

= 1 + 2 + 1 + 1

= 5

as we can see that path **f(A**⇢**C+D)** is get solved and this **tree has become a solved tree** now.

In simple words, the main flow of this algorithm is that we have to find **firstly level 1st** heuristic value and **then level 2nd** and after that **update the values** with going **upward** means towards the root node.

In the above tree diagram, we have updated all the values.

f(B⇢E) = 1 + 7

= 8

f(B⇢f) = g(f) + h(f) f(B⇢f) = 1 + 9

= 10

So, by above calculation B⇢E path is chosen which is minimum path, i.e **f(B**⇢**E)** because **B's** heuristic value is different from its actual value The heuristic is updated and the minimum cost path is selected. The minimum value in our situation

is **8**.

Therefore, the heuristic for **A** must be updated due to the change in **B's** heuristic. So we need to calculate it again.

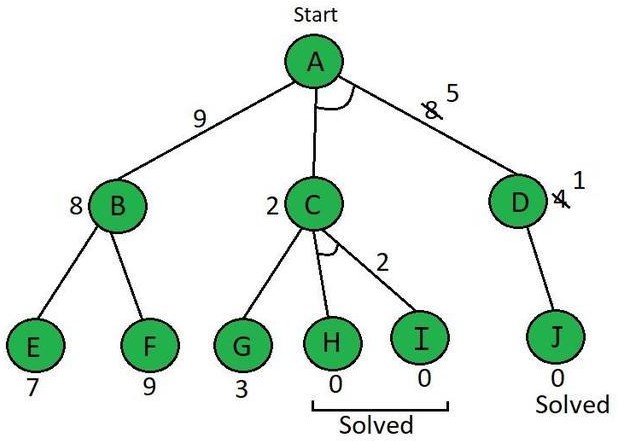
f(A⇢B) = g(B) + updated h(B)

= 1 + 8

= 9

We have Updated all values in the above tree.

##### Step 3



*AO\* Algorithm (Step-3)*

By comparing **f(A**⇢**B)** & **f(A**⇢**C+D)** f(A⇢C+D) is shown to be **smaller**. i.e 8 < 9 Now explore f(A⇢C+D)

So, the current node is **C**

f(C⇢G) = g(g) + h(g) f(C⇢G) = 1 + 3

= 4

f(C⇢H+I) = g(h) + h(h) + g(i) + h(i)

f(C⇢H+I) = 1 + 0 + 1 + 0 ……here we have added **H & I** because they are in **AND**

= 2

**CONSTRAINT SATISFACTION PROBLEMS (CSP) IN AI**

**What are CSPs?** Constraint Satisfaction Problems (CSPs) are a type of problem where you need to find a solution that meets a set of rules or restrictions. These restrictions tell you how different parts of the problem can or cannot be combined.

**Why Are CSPs Important in AI?** CSPs are important in AI because:

They help solve real-world problems that have specific rules.

They provide a structured way to represent and solve different types of problems.

Many AI tasks, like scheduling or planning, can be thought of as CSPs, making them easier to solve efficiently.

Key Parts of CSPs

**Variables**: These are the parts of the problem that need values. For example, in a scheduling problem, variables could be the different time slots or tasks.

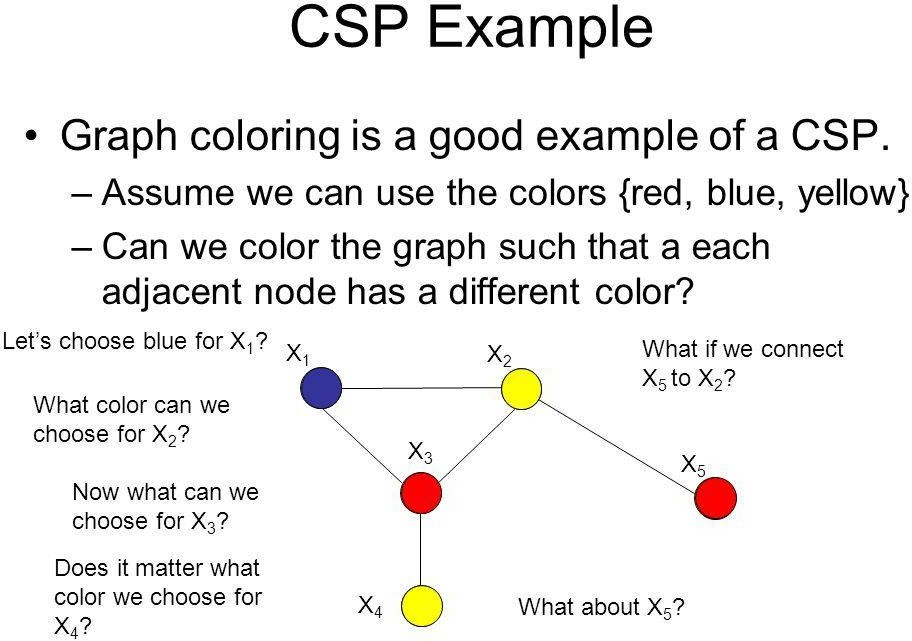
**Domains**: The domain of a variable is the set of possible values it can have. For example, the domain for a time slot variable could be a list of available times.

**Constraints**: These are the rules that limit which combinations of values the variables can take. Constraints can involve one variable (unary), two variables (binary), or more (n-ary).

Real-World Examples of CSPs

**Sudoku**: Each empty cell in the puzzle is a variable, the domain is numbers from 1 to 9, and the constraints ensure that no number is repeated in any row, column, or 3x3 subgrid.

**Scheduling**: Variables could be classes, domains are time slots, and constraints make sure that classes with the same students or teachers don’t overlap.

**Map Coloring**: Variables are regions, domains are colors, and constraints ensure that neighboring regions don’t have the same color.

##### REPRESENTATION OF CSPS:

The representation of Constraint Satisfaction Problems (CSPs) is crucial for effectively solving these problems. Let's explore how to represent CSPs using variables, domains, and constraints:

##### Variables as Placeholders:

Variables in CSPs act as placeholders for problem components that need to be assigned values. They represent the entities or attributes of the problem under consideration. For example:

* + In a Sudoku puzzle, variables represent the empty cells that need numbers.
  + In job scheduling, variables might represent tasks to be scheduled.
  + In map coloring, variables correspond to regions or countries that need to be colored.

The choice of variables depends on the specific problem being modeled.

##### Domains:

Each variable in a CSP is associated with a domain, which defines the set of values that the variable can take. Domains are a critical part of the CSP representation, as they restrict the possible assignments of values to variables. For instance:

* + In Sudoku, the domain for each empty cell is the numbers from 1 to 9.
  + In scheduling, the domain for a task might be the available time slots.
  + In map coloring, the domain could be a list of available colors.

Domains ensure that variable assignments remain within the specified range of values.

Constraints:

Constraints in CSPs specify the relationships or conditions that must be satisfied by the variables. Constraints restrict the combinations of values that variables can take.

Constraints can be unary (involving a single variable), binary (involving two variables), or n-ary (involving more than two variables). Constraints are typically represented in the form of logical expressions, equations, or functions. For example:

In Sudoku, constraints ensure that no two numbers are repeated in the same row, column, or subgrid.

In scheduling, constraints might involve ensuring that two tasks are not scheduled at the same time.

In map coloring, constraints require that adjacent regions have different colors.

Constraint specification is a crucial part of problem modeling, as it defines the rules that the variables must follow.

**Overall Representation:**

To represent a CSP, you need to define:

The set of variables: What entities or attributes need values?

The domains: What are the possible values that each variable can take?

The constraints: What conditions or limitations must be satisfied by the variables?

By defining these elements, you create a structured representation of the problem, which is essential for CSP solvers to find valid solutions efficiently.

**SOLVING CSPS IN AI**

**Backtracking**

Backtracking is like trial and error. You try assigning values to variables, and if a rule is broken, you go back and try a different value.

Example: In Sudoku, you pick a number for an empty cell, check if it breaks any rules, and if it does, you try a different number.

Constraint Propagation

This technique reduces the number of possible values (domains) for variables by applying the constraints.

Example: If a row in Sudoku already has numbers 1-8, the last empty cell can only be 9. This makes solving the puzzle easier because fewer options are left to check.

### MEANS-ENDS ANALYSIS IN ARTIFICIAL INTELLIGENCE

**Means-Ends Analysis (MEA)** is a problem-solving technique in AI that combines both forward and backward reasoning to tackle complex problems. Instead of solving a problem strictly from start to finish (forward) or from the end to the beginning (backward), MEA uses a mixed approach. This allows for solving major parts of the problem first and then addressing any smaller issues that arise during the process.

MEA is particularly useful in AI programs for limiting the search space, making problem-solving more efficient.

##### Background and Introduction

MEA was introduced in 1961 by Allen Newell and Herbert A. Simon in their computer program called the **General Problem Solver (GPS)**. The core idea of MEA is to evaluate the differences between the current state and the goal state, and then to apply actions (operators) that reduce these differences.

##### How Means-Ends Analysis Works

MEA works by recursively breaking down a problem into smaller subproblems, which are easier to solve. The process involves the following steps:

1. **Evaluate the Difference**: Compare the current state with the goal state and identify the differences between them.
2. **Select Operators**: Choose the appropriate operators (actions) that can reduce these differences.
3. **Apply Operators**: Apply the selected operators to transform the current state closer to the goal state.

##### Operator Subgoaling

In some cases, an operator might not be directly applicable to the current state. When this happens, a subgoal is created where the operator can be applied. This is known as **Operator Subgoaling**. Essentially, you solve a smaller problem (subgoal) that enables you to apply the operator, which then helps in solving the main problem.

##### Algorithm for Means-Ends Analysis

Let’s denote the current state as CURRENT and the goal state as GOAL. The MEA algorithm proceeds as follows:

1. **Compare** CURRENT to GOAL:
   * If there are no differences, return success and exit.

##### Identify Differences:

* + If differences exist, select the most significant one and attempt to reduce it by:
    - Selecting an operator O that can address this difference.
    - If no such operator exists, signal failure.
    - Attempt to apply O to CURRENT, creating:
      * **O-Start**: The state where O's preconditions are satisfied.
      * **O-Result**: The state that would result if O were applied in O-Start.

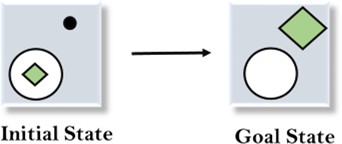
##### Recursive Application:

* + If the MEA applied to CURRENT and O-START is successful, and the MEA applied to O-RESULT and GOAL is also successful, then return success, combining the results.

This algorithm is more suited for relatively simple problems and might require additional modifications for more complex issues.

##### Example of Means-Ends Analysis

Consider a simple problem where you have an initial state and a goal state as shown below:



**Initial State:** A square is outside a circle, and there’s an extra dot inside the circle.

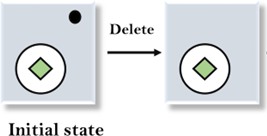
**Goal State:** The square should be inside the circle, and the dot should be removed.

##### Solution:

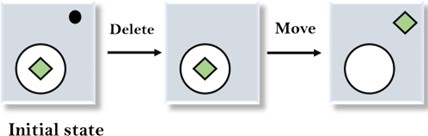
1. **Evaluate Initial State**: Compare the initial and goal states to identify differences. In this case, the differences are:
   * The dot in the initial state (which shouldn’t be in the goal state).
   * The square is outside the circle (it should be inside in the goal state).



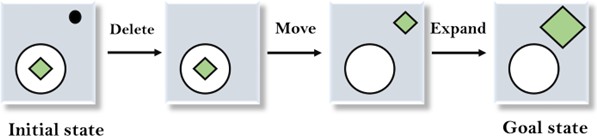
1. **Apply the Delete Operator**: The first difference is the dot, which needs to be removed. We apply the Delete operator to eliminate the dot.



1. **Apply the Move Operator**: Next, we notice the square is still outside the circle. We apply the Move operator to place the square inside the circle.



1. **Apply the Expand Operator**: Finally, if the square's size needs adjustment to match the goal state, we use the Expand operator to resize the square.



By following these steps, we systematically reduce the differences between the initial and goal states, ultimately solving the problem.

##### Conclusion

Means-Ends Analysis is an effective problem-solving method that combines both forward and backward reasoning. It’s particularly useful in AI for reducing the search space and efficiently reaching a solution. By breaking down large problems into smaller, manageable subproblems, MEA makes it easier to apply specific actions that bring you closer to the goal state.