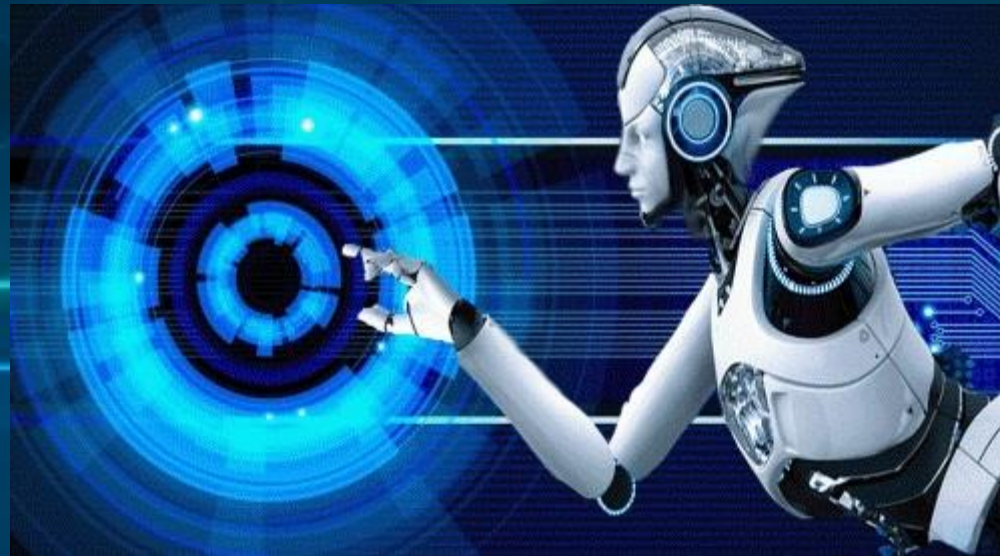


ARTIFICIAL INTELLIGENCE

BY
S. NITHIYA
ASSISTANT PROFESSOR
INFORMATION TECHNOLOGY

Definition

"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."



Why AI ?

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

What Comprises to Artificial Intelligence?



What is Intelligence?

The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.



Learning

- **Learning** – It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.
- The ability of learning is possessed by humans, some animals, and AI-enabled systems. Learning is categorized as –
 - **Auditory Learning** – It is learning by listening and hearing. **For example**, students listening to recorded audio lectures.
 - **Episodic Learning** – To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.
 - **Motor Learning** – It is learning by precise movement of muscles. **For example**, picking objects, Writing, etc.

Learning

- **Observational Learning** – To learn by watching and imitating others. **For example**, child tries to learn by mimicking her parent.
- **Perceptual Learning** – It is learning to recognize stimuli that one has seen before. **For example**, identifying and classifying objects and situations.
- **Relational Learning** – It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. **For Example**, Adding ‘little less’ salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.
- **Spatial Learning** – It is learning through visual stimuli such as images, colors, maps, etc. **For Example**, A person can create roadmap in mind before actually following the road.
- **Stimulus-Response Learning** – It is learning to perform a particular behavior when a certain stimulus is present. **For example**, a dog raises its ear on hearing doorbell.

Learning contd..

- **Problem Solving** – It is the process in which one perceives and tries to arrive at a desired solution from a present situation by taking some path, which is blocked by known or unknown hurdles.
- Problem solving also includes **decision making**, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.
- **Perception** – It is the process of acquiring, interpreting, selecting, and organizing sensory information.
- Perception presumes **sensing**. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data acquired by the sensors together in a meaningful manner.
- **Linguistic Intelligence** – It is one's ability to use, comprehend, speak, and write the verbal and written language. It is important in interpersonal communication.

Artificial Intelligence VS Human Intelligence

Artificial Intelligence	Human Intelligence
Created by human intelligence	Created by Divine intelligence
Process information faster	Process information slower
Highly objective	May be subjective
More accurate	May be less accurate
Uses 2 watts	Uses 25 watts
Cannot adapt to changes well	Can easily adapt to changes
Cannot multitask that well	Can easily multitask
Below average social skills	Excellent social skills
Still working towards self-awareness	Has self-awareness
Optimization	Innovation

Applications of AI



Applications of AI contd....

1. AI in Astronomy

To solve **complex universe problems**. AI can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

- ❖ Healthcare Industries are applying AI to make a better and **faster diagnosis** than humans.
- ❖ AI can **help doctors with diagnoses** and can **inform when patients are worsening** so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

The AI machines can **play strategic games** like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

The finance industry is implementing **automation, chatbot, adaptive intelligence, algorithm trading, and machine learning** into financial processes.

5. AI in Data Security

- ✓ cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure.
- ✓ Examples such as **AEG bot, AI2 Platform**, are used to determine software bug and cyber-attacks in a better way.

Applications of AI contd....

6. AI in Social Media

- ❖ Social Media sites such as Facebook, Twitter, and Snapchat contain **billions of user profiles**.
- ❖ AI can organize and manage massive amounts of data.
- ❖ AI can analyze lots of data to identify the latest trends, **hashtag**, and requirement of different users.

7. AI in Travel & Transport

- ✓ AI makes travel arrangement **to suggesting the hotels, flights, and best routes to the customers**.
- ✓ Travel industries are using **AI-powered chatbots** which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

- ❖ AI to provides virtual assistant to their user for better performance.
- ❖ Such as Tesla has introduced **TeslaBot**, an intelligent virtual assistant.
- ❖ Industries are currently working for developing **self-driven cars** which can make your journey more safe and secure.

9. AI in Robotics:

- ✓ creates intelligent robots which can perform tasks with their own experiences without pre-programmed.
- ✓ **Humanoid Robots** are best examples for AI in robotics,
- ✓ Intelligent Humanoid robot named as **Erica and Sophia** has been developed which can talk and behave like humans.

Applications of AI contd....

10. AI in Entertainment

- ❖ Entertainment services such as Netflix or Amazon.
- ❖ With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

- ✓ Now a day's agriculture is becoming digital.
- ✓ Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis.
- ✓ AI in agriculture can be very helpful for farmers.

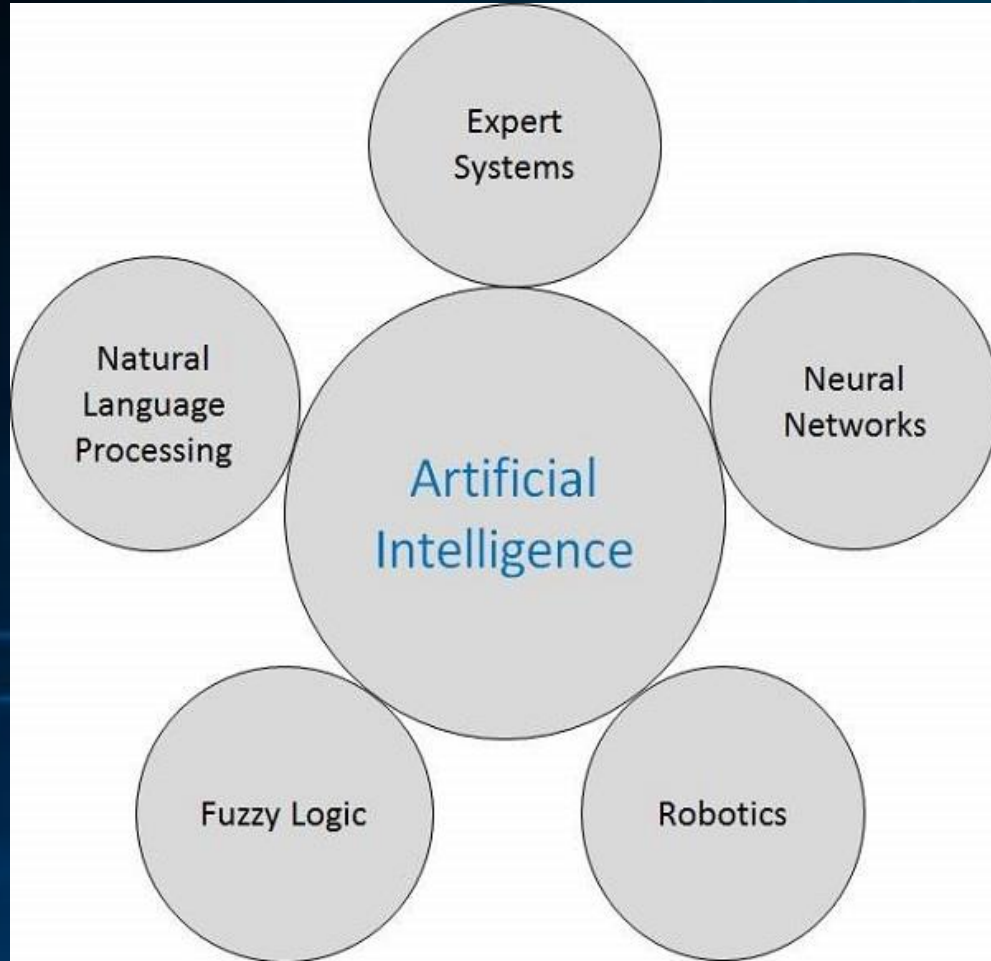
12. AI in E-commerce






AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in education:

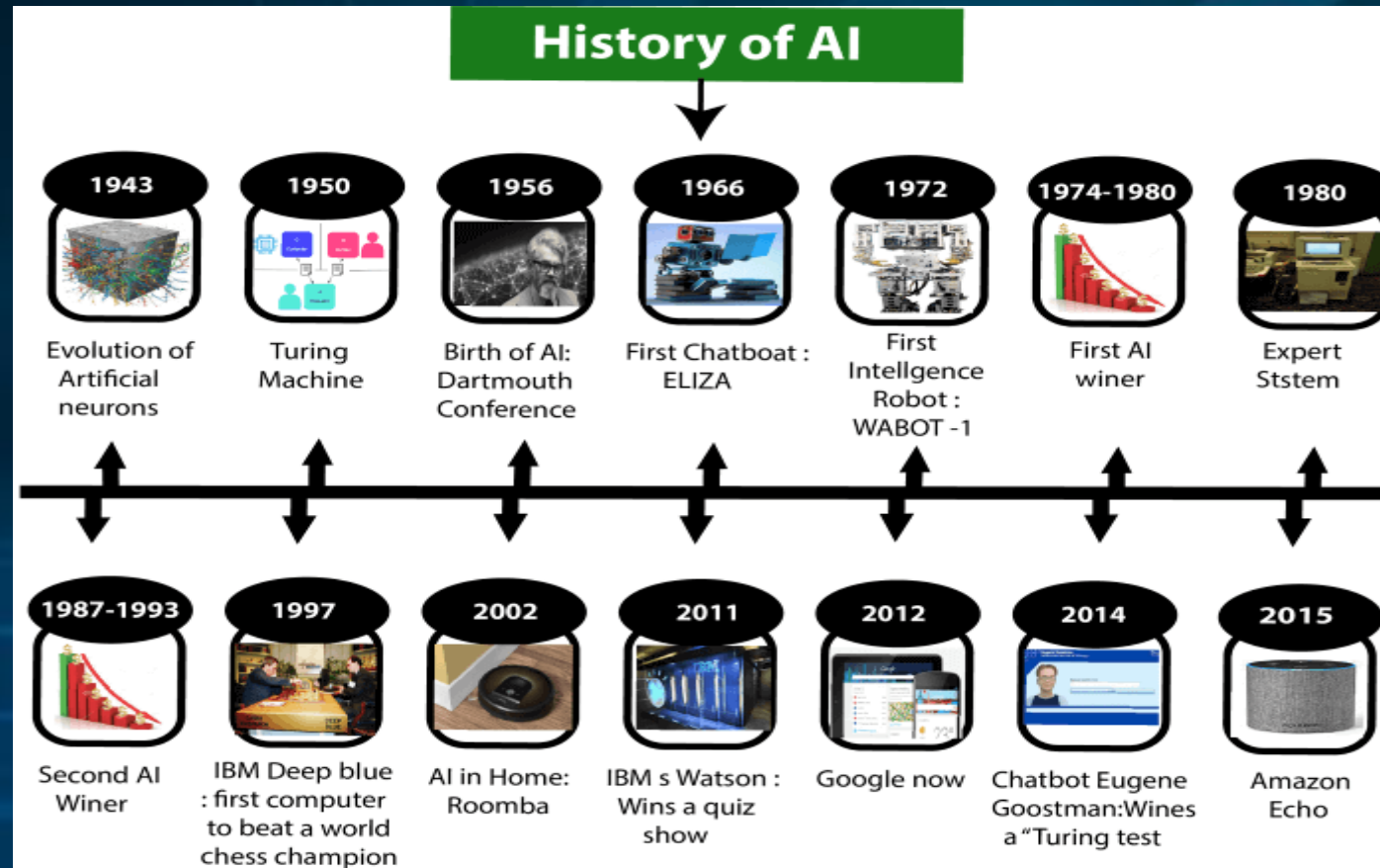
- ❖ AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- ❖ AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

Real Life Applications of Research Areas

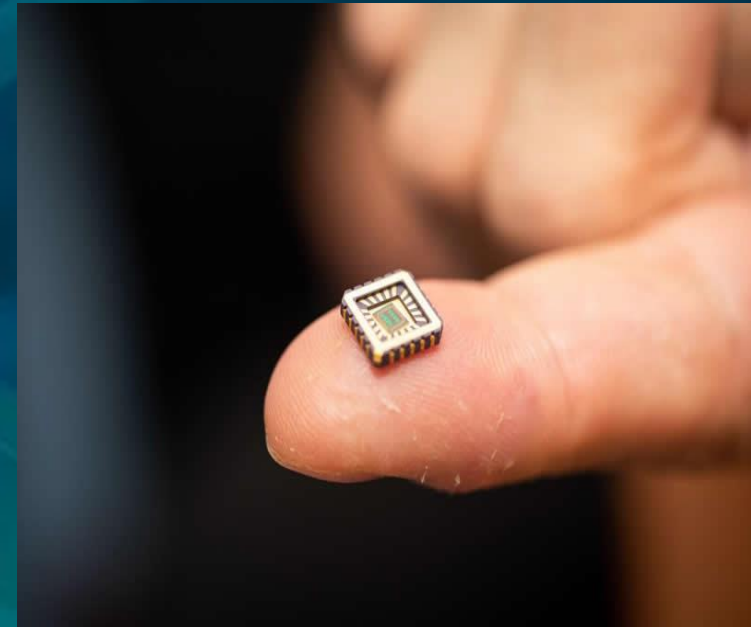
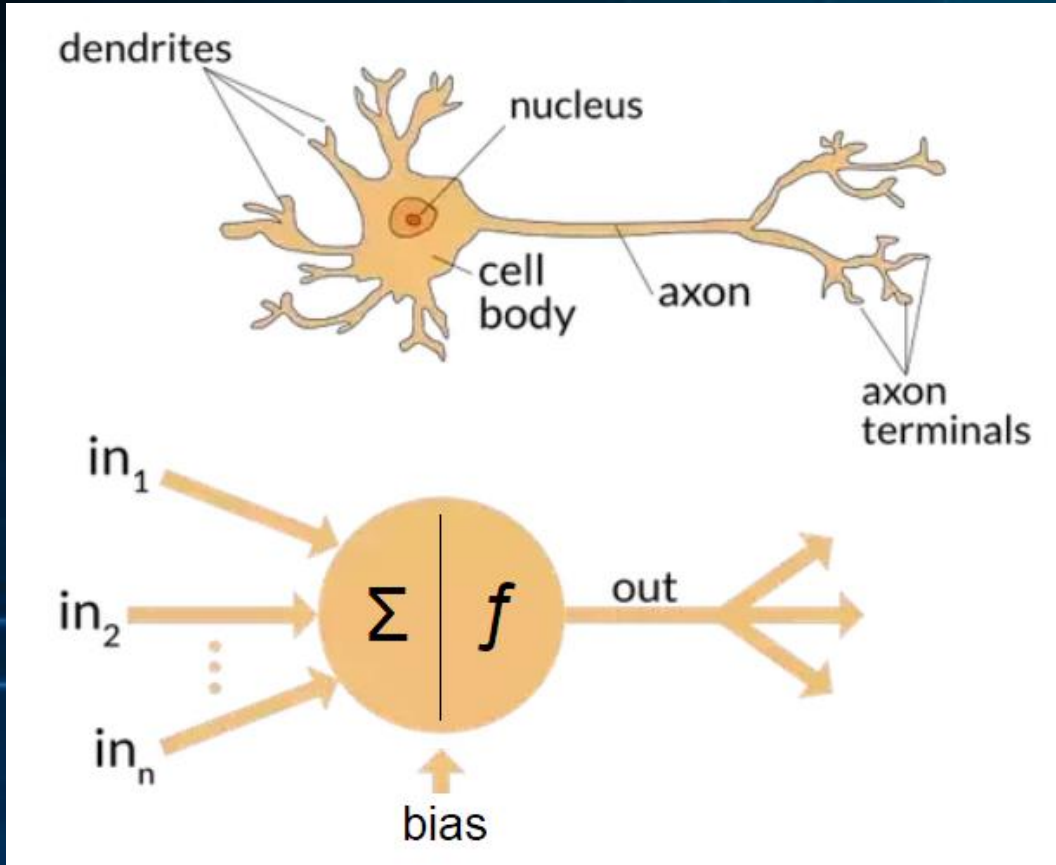


Sr.No.	Research Areas	Real Life Application
1	Expert Systems Examples – Flight-tracking systems, Clinical systems.	
2	Natural Language Processing Examples: Google Now feature, speech recognition, Automatic voice output.	
3	Neural Networks Examples – Pattern recognition systems such as face recognition, character recognition, handwriting recognition.	
4	Robotics Examples – Industrial robots for moving, spraying, painting, precision checking, drilling, cleaning, coating, carving, etc.	
5	Fuzzy Logic Systems Examples – Consumer electronics, automobiles, etc.	

History of Artificial Intelligence



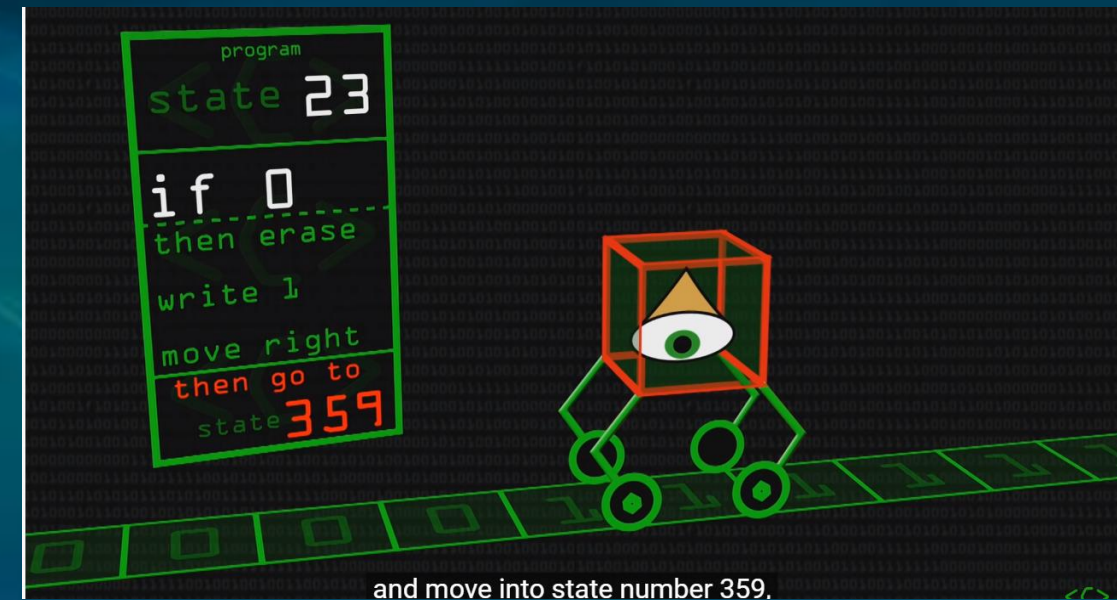
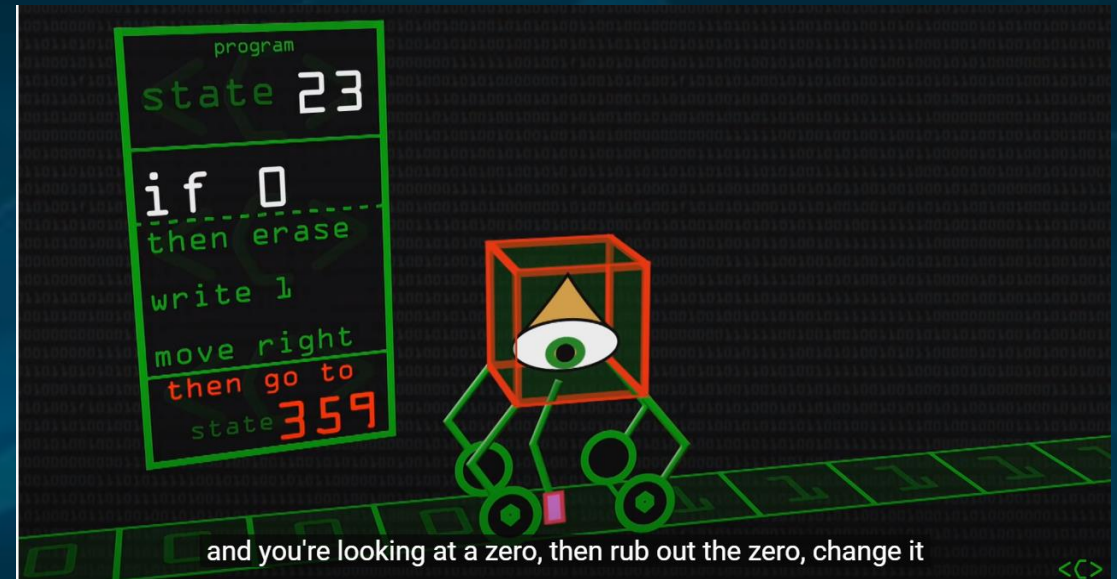
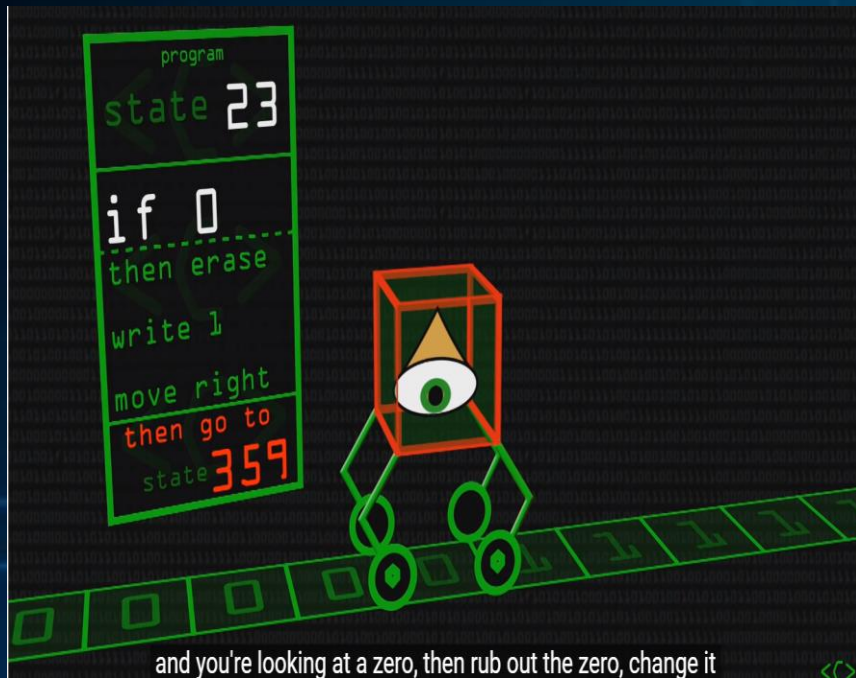
Artificial Neurons



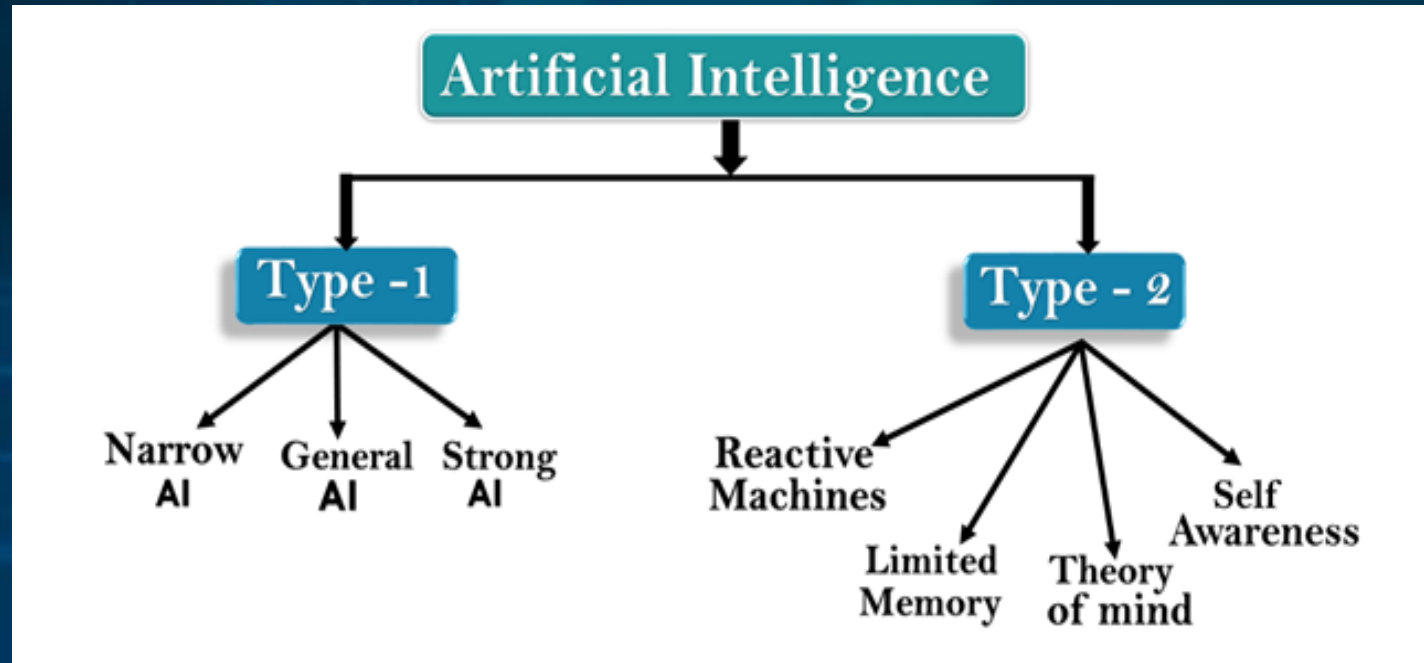
Artificial Neurons

- A neuron is a mathematical function that function as biological neuron.
- Neurons carry signal to and from brain and rest of the body.
- It receives one or more input then sums and perform activation function and produce output.
- Artificial nerve cells solves Alzheimer's disease.
- Cyborg brains
- Artificial Brain Booming.

Turing Machine



Types of AI



Weak AI or Narrow AI based on capability

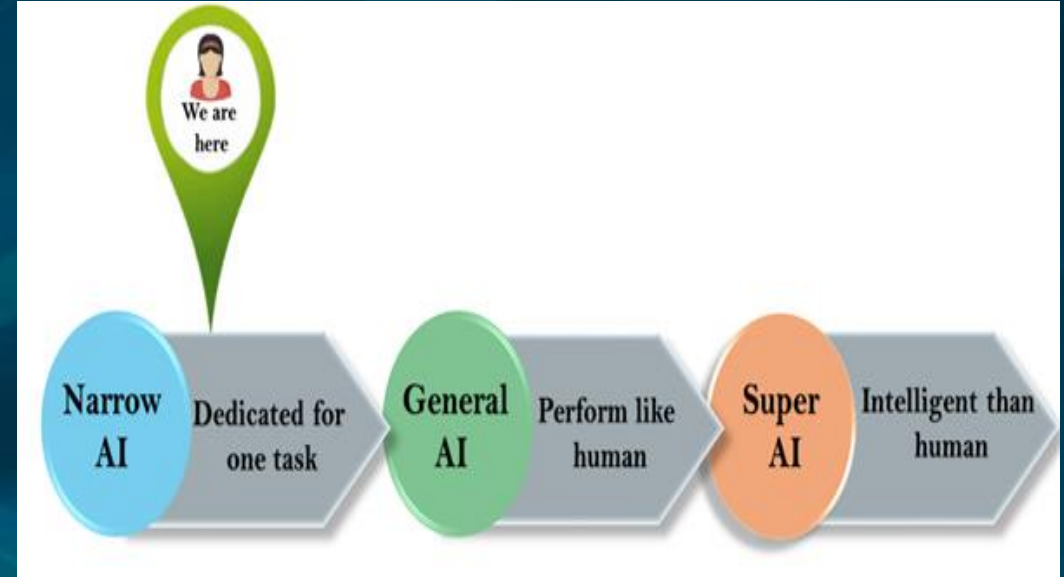
- Able to perform a dedicated task with intelligence
- Cannot perform beyond its limit
- example Apple Siri, IBM Watson
- Playing chess
- Purchase on ecommerce website
- Self driving cars
- Speech recognition
- Image recognition

General AI

- Perform any intelligent task like human
- Smarter than human
- Still in research

Super AI

- Machine surpass human intelligence
- Better cognitive property
- Ability to think, reason, solve puzzles, make judgements, plan, learn, communication by own.
- Solve hypothetical concept.



Cognitive property

Dimension	Example of cognitive process
L1: Remember	define, duplicate, list, memorize, recall, copy, repeat, reproduce state
L2: Understand	classify, describe, discuss, explain, identify, locate, recognize, report, select, translate, differentiate, plan
L3: Apply	choose, demonstrate, dramatize, employ, illustrate, interpret, operate, schedule, sketch, solve, calculate
L4: Analyse	appraise, compare, contrast, criticize, discriminate, distinguish, examine, experiment, question, test
L5: Evaluate	appraise, argue, defend, judge, select, support, value, evaluate
L6: Create	assemble, construct, create, design, develop, formulate

(Source: Anderson & Krathwohl, 2001)

Hypothetical Property

- A superintelligence is a **hypothetical** agent that possesses **intelligence** far surpassing that of the brightest and most gifted human minds.
- **CAPTCHA** to check whether human or robot



TYPE 2 AI Based on Functionality

1. Reactive Machines

- Purely reactive machines are the most basic types of Artificial Intelligence.
- Such AI systems do not store memories or past experiences for future actions.
- These machines only focus on current scenarios and react on it as per possible best action.
- IBM's Deep Blue system is an example of reactive machines.
- Google's AlphaGo is also an example of reactive machines.

2. Limited Memory

- Limited memory machines can store past experiences or some data for a short period of time.
- These machines can use stored data for a limited time period only.
- Self-driving cars are one of the best examples of Limited Memory systems. These cars can store recent speed of nearby cars, the distance of other cars, speed limit, and other information to navigate the road.

3. Theory of Mind

- Theory of Mind AI should understand the human emotions, people, beliefs, and be able to interact socially like humans.
- This type of AI machines are still not developed, but researchers are making lots of efforts and improvement for developing such AI machines.

4. Self-Awareness

- Self-awareness AI is the future of Artificial Intelligence. These machines will be super intelligent, and will have their own consciousness, sentiments, and self-awareness.
- These machines will be smarter than human mind.
- Self-Awareness AI does not exist in reality still and it is a hypothetical concept.

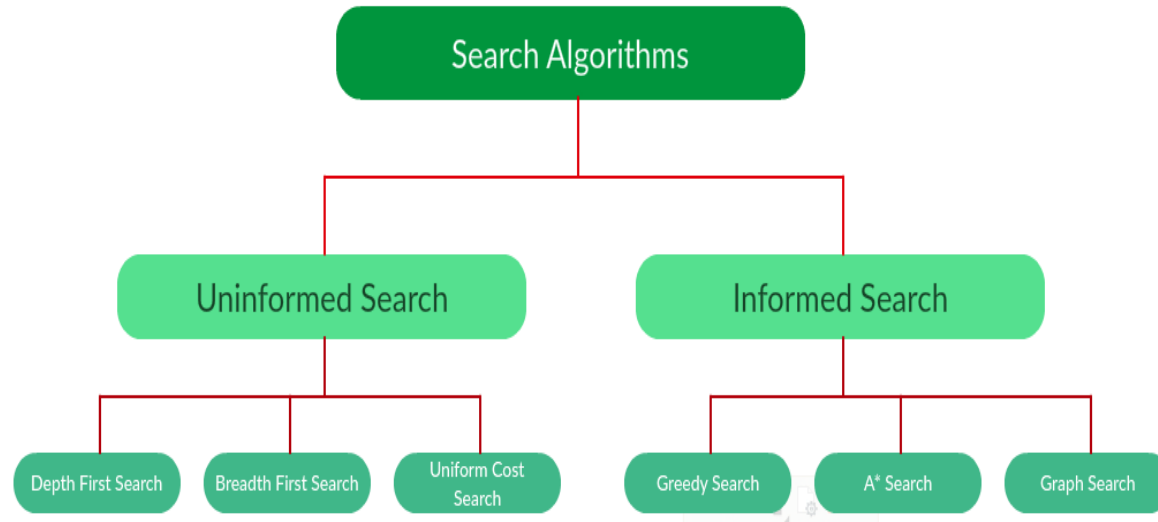
SEARCHING ALGORITHMS

PRESENTED BY

S. NITHIYA
ASSISTANT PROFESSOR
INFORMATION TECHNOLOGY
SARADHA GANGADHARAN COLLEGE

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

Types Of Search algorithm



Properties of Search Algorithm

- ❖ **Completeness:** A search algorithm is said to be complete if it guarantees to **return a solution** if at least any solution exists for any random input.
- ❖ **Optimality:** If a solution found for an algorithm is guaranteed to be the **best solution** (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- ❖ **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.

Search Algorithm Terminologies:

- ✓ **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
 - **Search Space:** Search space represents a set of possible solutions, which a system may have.
 - **Start State:** It is a state from where agent begins **the search**.
 - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- ✓ **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- ✓ **Actions:** It gives the description of all the available actions to the agent.
- ✓ **Transition model:** A description of what each action do, can be represented as a transition model.
- ✓ **Path Cost:** It is a function which assigns a numeric cost to each path.
- ✓ **Solution:** It is an action sequence which leads from the start node to the goal node.
- ✓ **Optimal Solution:** If a solution has the lowest cost among all solutions.

Uninformed/Blind Search:

- ✓ The uninformed search **does not contain any domain knowledge** such as closeness, the location of the goal.
- ✓ It operates in a **brute-force way** as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.
- ✓ Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called **blind search**.
- ✓ It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Search

- ✓ Informed search algorithms use **domain knowledge**.
- ✓ In an informed search, problem information is available which can guide the search.
- ✓ Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- ✓ Informed search is also called a **Heuristic search**.
- ✓ A heuristic is a way which might **not always be guaranteed for best solutions** but **guaranteed to find a good solution in reasonable time**.
- ✓ Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

- Greedy Search
- A* Search

Breadth-first Search:

- ❖ Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- ❖ BFS algorithm **starts** searching from the **root node** of the tree and **expands all successor node at the current level** before moving to nodes of next level.
- ❖ The breadth-first search algorithm is an example of a general-graph search algorithm.
- ❖ Breadth-first search implemented using **FIFO queue** data structure.

Advantages

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example for Breadth First Search

In the given tree structure,

- The traversing of the tree starts from the root node S to goal node K.
- BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S----> A---->B---->C---->D---->G---->H---->E---->F---->I---->K

- **Time Complexity:**

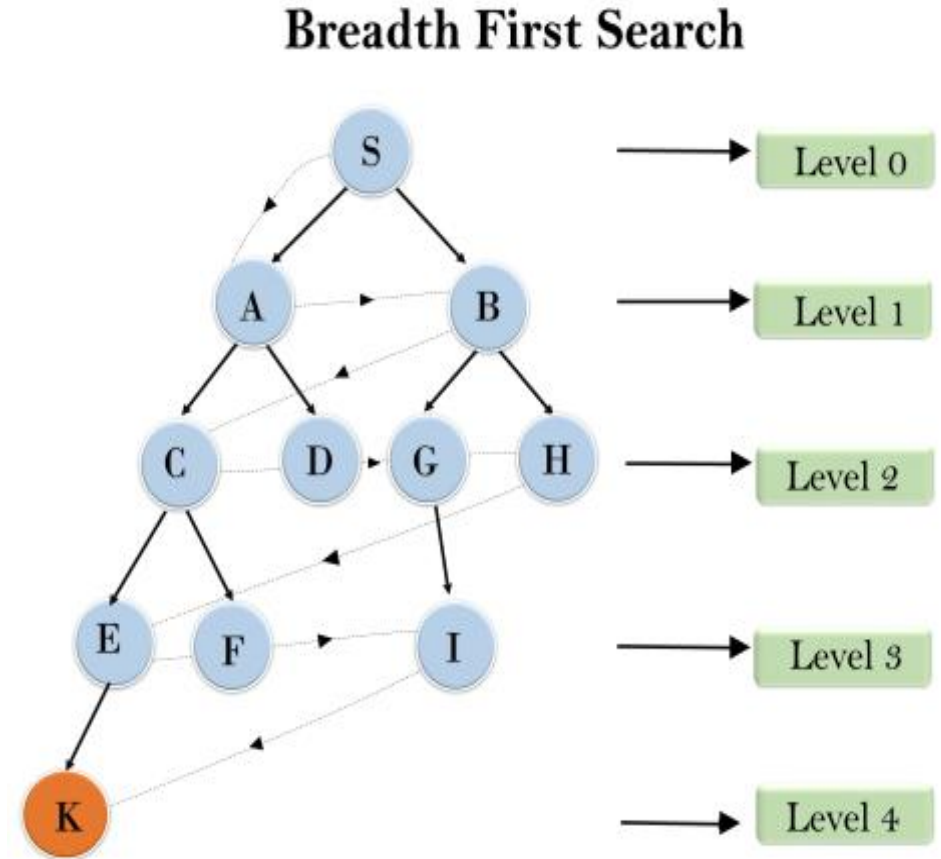
Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the

d= depth of shallowest solution and

b is a node at every state.

$$T(b) = 1 + b^2 + b^3 + \dots + b^d = O(b^d)$$

- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.



Uniform-cost Search Algorithm:

- ❖ Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge.
- ❖ The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.
- ❖ Uniform-cost search expands nodes according to their path costs from the root node.
- ❖ It can be used to solve any graph/tree where the optimal cost is in demand.
- ❖ A uniform-cost search algorithm is implemented by the priority queue.
- ❖ It gives maximum priority to the lowest cumulative cost.
- ❖ Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

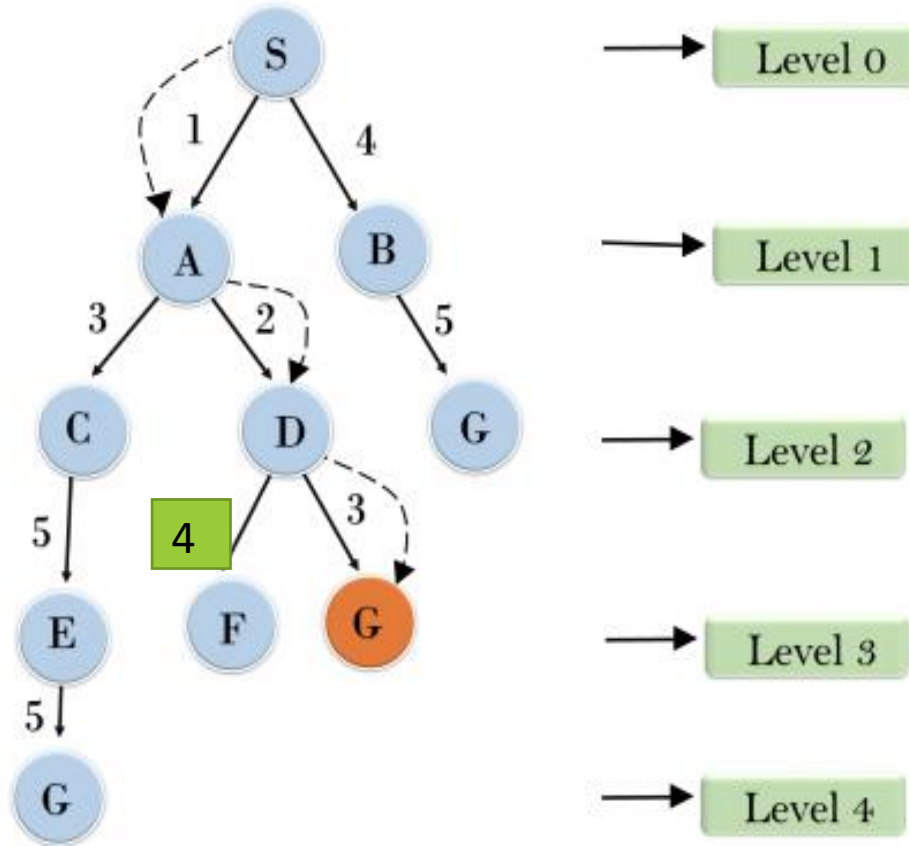
Advantages:

Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

It does not care about the number of steps involved in searching and is only concerned about path cost. Due to which this algorithm may be stuck in an [infinite loop](#).

Uniform Cost Search



Expand the nodes of S and put in the CLOSED list

Initialization: Open [S(0)], Closed []

Iteration 1: Open [A(1), B(4)], Closed [S(0)]

Iteration 2: Open [B(4), C(4), D(3)], Closed [S(0), A(1)]

Iteration 3: Open [B(4), C(4), F(7), G(6)],
Closed [S(0), A(1), D(3)]

Iteration 4: Open [B(4), C(4), F(7)],
Closed [S(0), A(1), D(3), G(6)]

Hence the final solution path will be: **S-----> A----->D-----> G**
Path cost: 6

Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

- Let C^* is **Cost of the optimal solution**, and ϵ is each step to get closer to the goal node.
- Then the number of steps is $= C^*/\epsilon + 1$. Here we have taken $+1$, as we start from state 0 and end to C^*/ϵ .
- Hence, the worst-case time complexity of Uniform-cost search is $O(b^{1 + \lceil C^*/\epsilon \rceil})$.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $O(b^{1 + \lceil C^*/\epsilon \rceil})$.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Depth-first Search

- Depth-first search is a **recursive** algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it **starts from the root node** and **follows each path to its greatest depth node** before moving to the next path.
- DFS uses a **stack** data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.
- **Note:** Backtracking is an algorithm technique for finding all possible solutions using recursion.

Advantage:

- DFS requires very **less memory** as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes **less time** to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is **no guarantee of finding the solution**.
- DFS algorithm goes for deep down searching and sometime it may go to the **infinite loop**.

Example For Depth first Search

- In the search tree, it is shown the flow of depth-first search, and it will follow the order as:

Root node---->Left node ----> right node.

- It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found.
- After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Completeness:

DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity:

Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$$

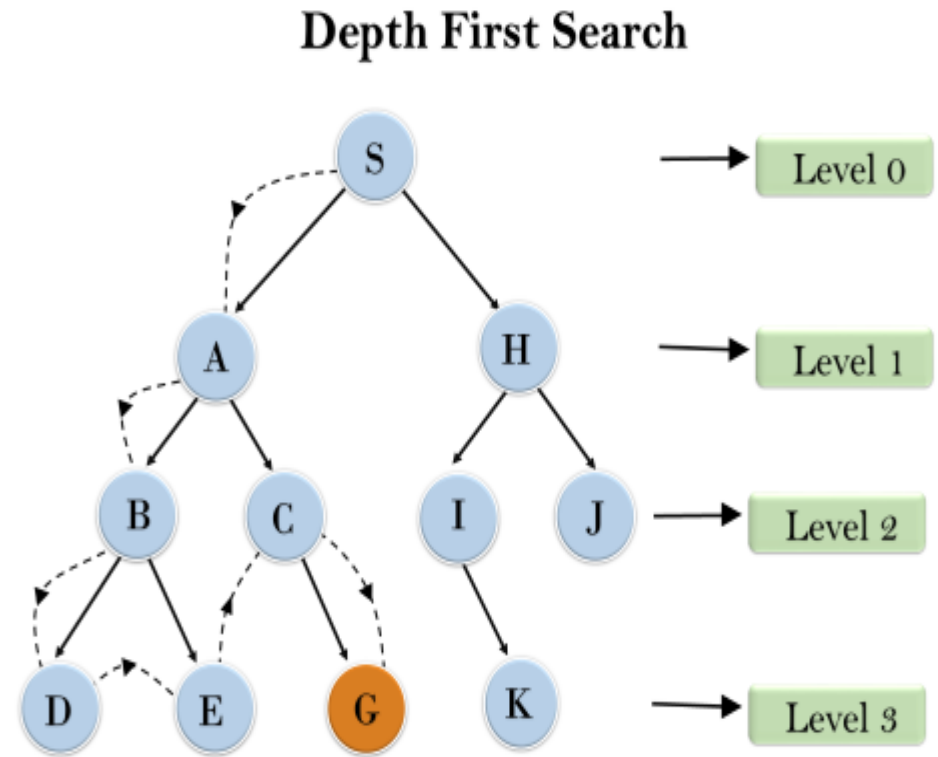
Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity:

DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is $O(bm)$.

Optimal:

DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.



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.

Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

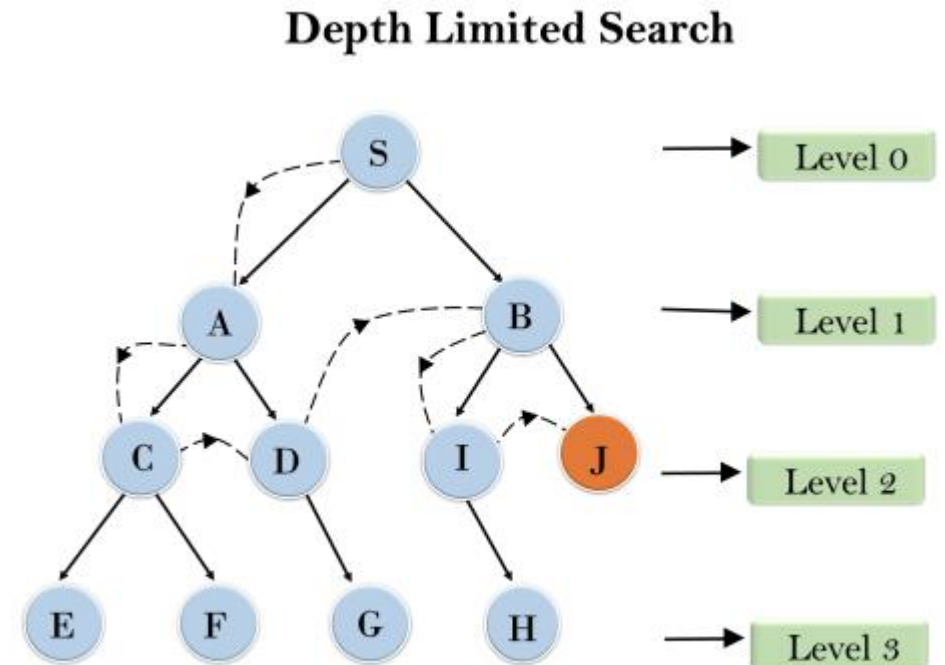
- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is $O(b^l)$.

Space Complexity: Space complexity of DLS algorithm is $O(b \times l)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $l > d$.



Iterative deepening depth-first Search:

- The iterative deepening algorithm is a combination of DFS and BFS algorithms.
- This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example

1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration----->A, B, D, E, C, F, G

4'th Iteration----->A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is $O(b^d)$.

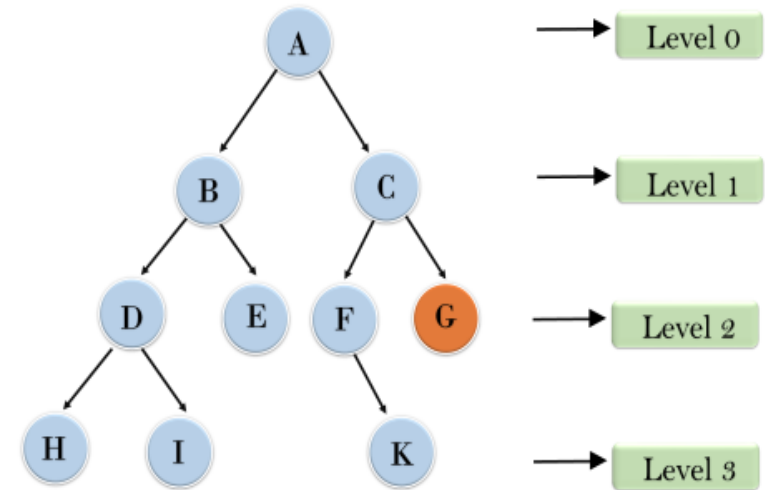
Space Complexity:

The space complexity of IDDFS will be $O(bd)$.

Optimal:

IDDFS algorithm is optimal if path cost is a non-decreasing function of the depth of the node.

Iterative deepening depth first search



Bidirectional Search Algorithm:

- Bidirectional search algorithm runs two simultaneous searches, one from initial state called as forward-search and other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

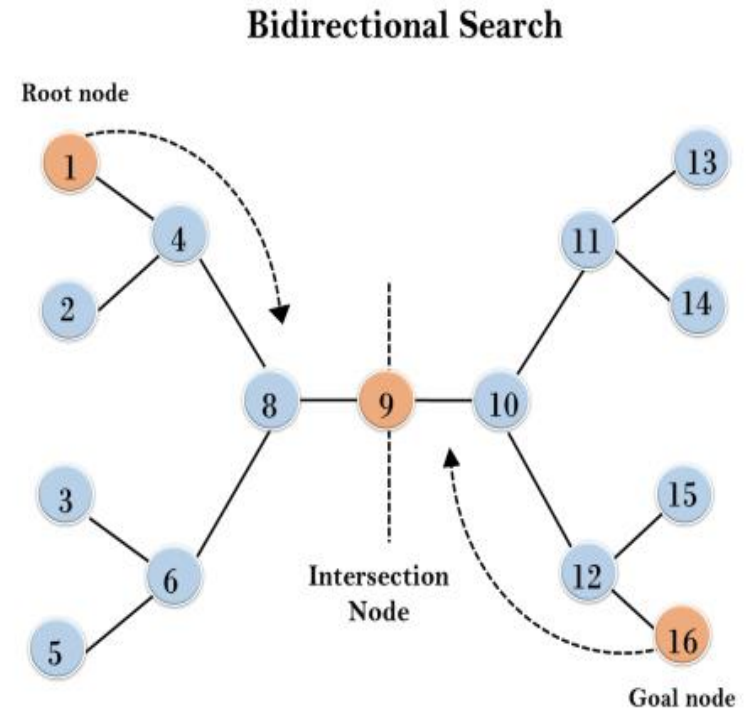
- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Example:

- In the given search tree, bidirectional search algorithm is applied.
- This algorithm divides one graph/tree into two sub-graphs.
- It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.
- The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is $O(b^d)$.

Space Complexity: Space complexity of bidirectional search is $O(b^d)$.

Optimal: Bidirectional search is Optimal.

INFORMED SEARCH ALGORITHMS

- Informed search algorithm contains **an array of knowledge** such as
 - ✓ how far we are from the goal,
 - ✓ path cost,
 - ✓ how to reach to goal node, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for **large search space**.
- Informed search algorithm uses the idea of heuristic, so it is also called **Heuristic search**.

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:

$$h(n) \leq h^*(n)$$

- ✓ 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**.

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 until a goal state is found.
- ❖ In the informed search we will discuss two main algorithms which are given below:
 - ✓ **Best First Search Algorithm(Greedy search)**
 - ✓ **A* Search Algorithm**

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, i.e.
 - $f(n) = h(n).$
 - 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.

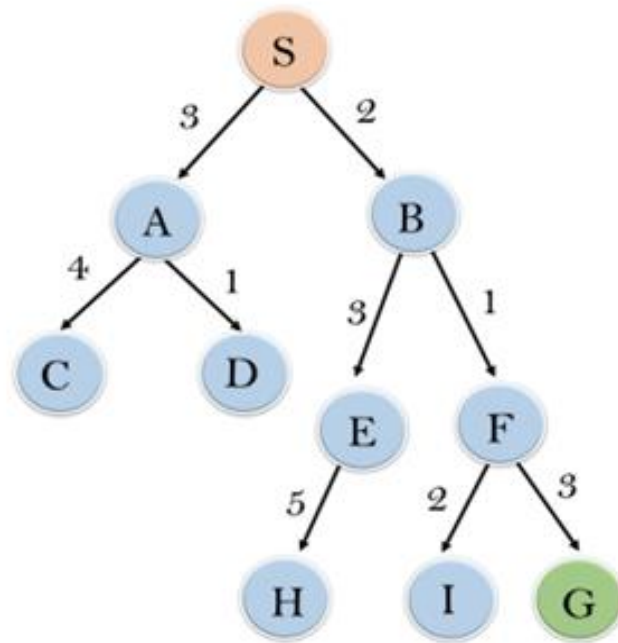
Disadvantages:

- 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.

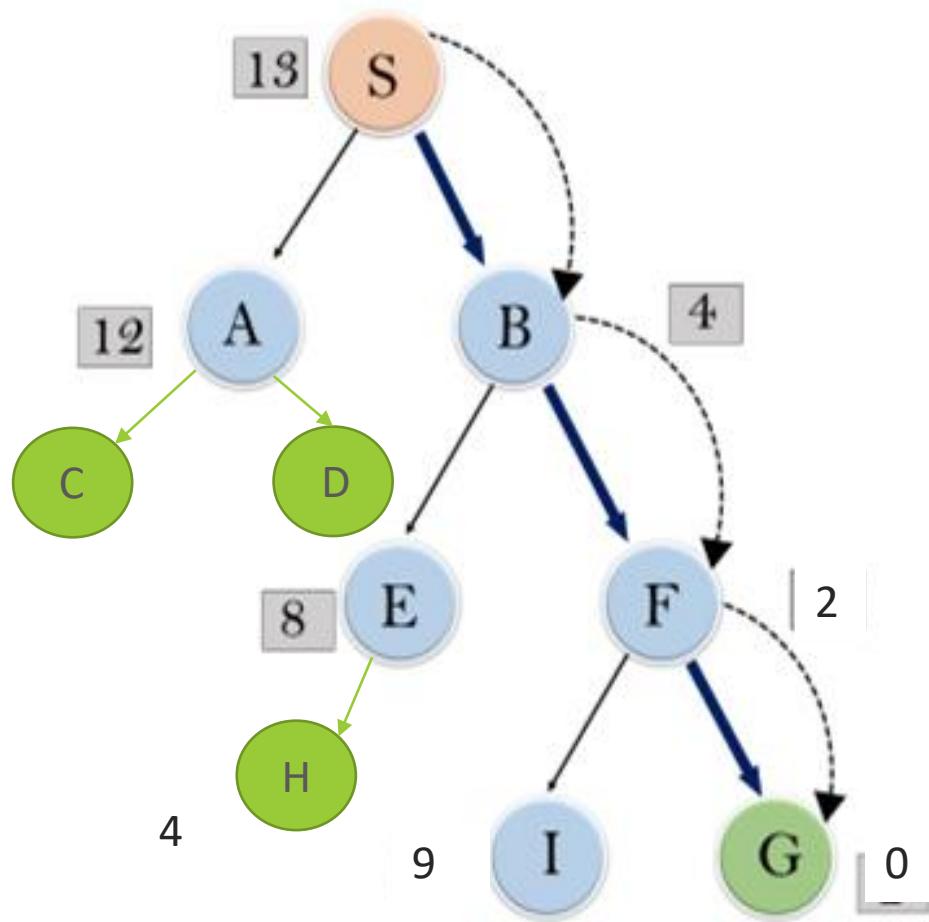
Example:

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.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0



In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the given 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**
Path cost: 6

✓ **Time Complexity:**

The worst case time complexity of Greedy best first search is $O(b^m)$.

✓ **Space Complexity:**

The worst case space complexity of Greedy best first search is $O(b^m)$.
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

Motivation

To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

What is A* Search Algorithm?

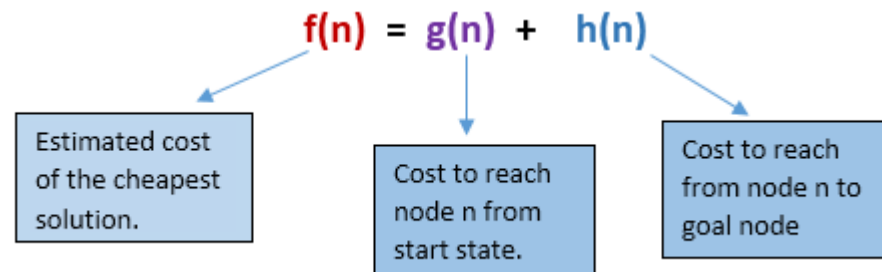
A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Why A* Search Algorithm ?

- Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”.
- It is really a smart algorithm which separates it from the other conventional algorithms.
- **Note:** It is used in games & web based maps.

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 solve 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 it combines both costs as following, and this sum is called as a **fitness number**.
- At each point in the search space, only those node is expanded which have the lowest value of $f(n)$, and the algorithm terminates when the goal node is found.



Algorithm of A* search:

Step 1: 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:

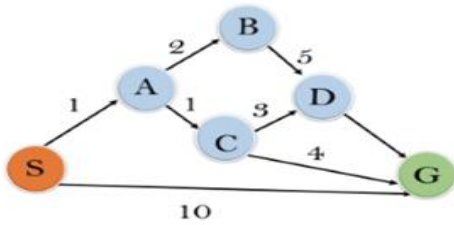
- ✓ It does not always produce the shortest path as it mostly based on heuristics and approximation.
- ✓ A* search algorithm has some complexity issues.
- ✓ The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

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.



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

Initialization: $\{(S, 5)\}$

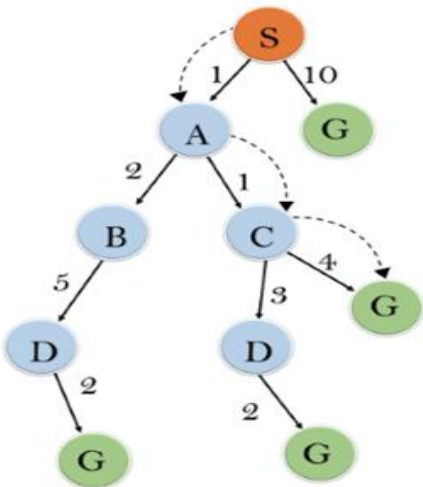
Iteration1: $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration2: $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration3: $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

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

Solution:



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:

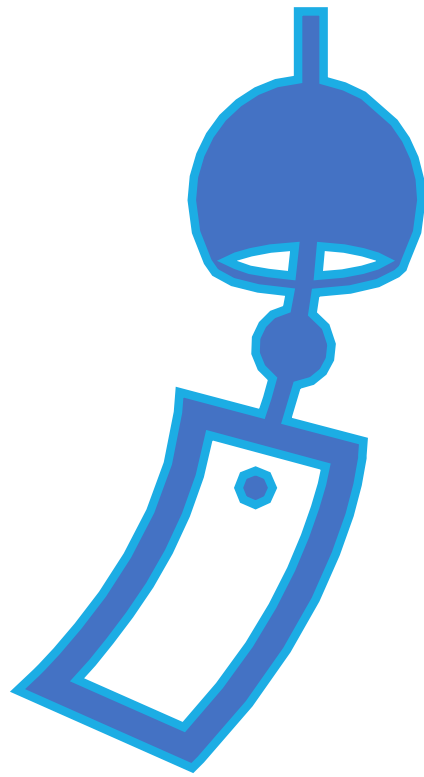
The space complexity of A* search algorithm is $O(b^d)$

Any Queries





THANK YOU



ALPHA - BETA PRUNING

S. Nithiya
Assistant Professor
Information Technology



Alpha-beta pruning is a modified version of the minimax algorithm.



It is an optimization technique for the minimax algorithm.



It cuts off branches in the game tree which need not be searched because there already exists a better move available.

It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.



Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.



The two-parameter can be defined as:

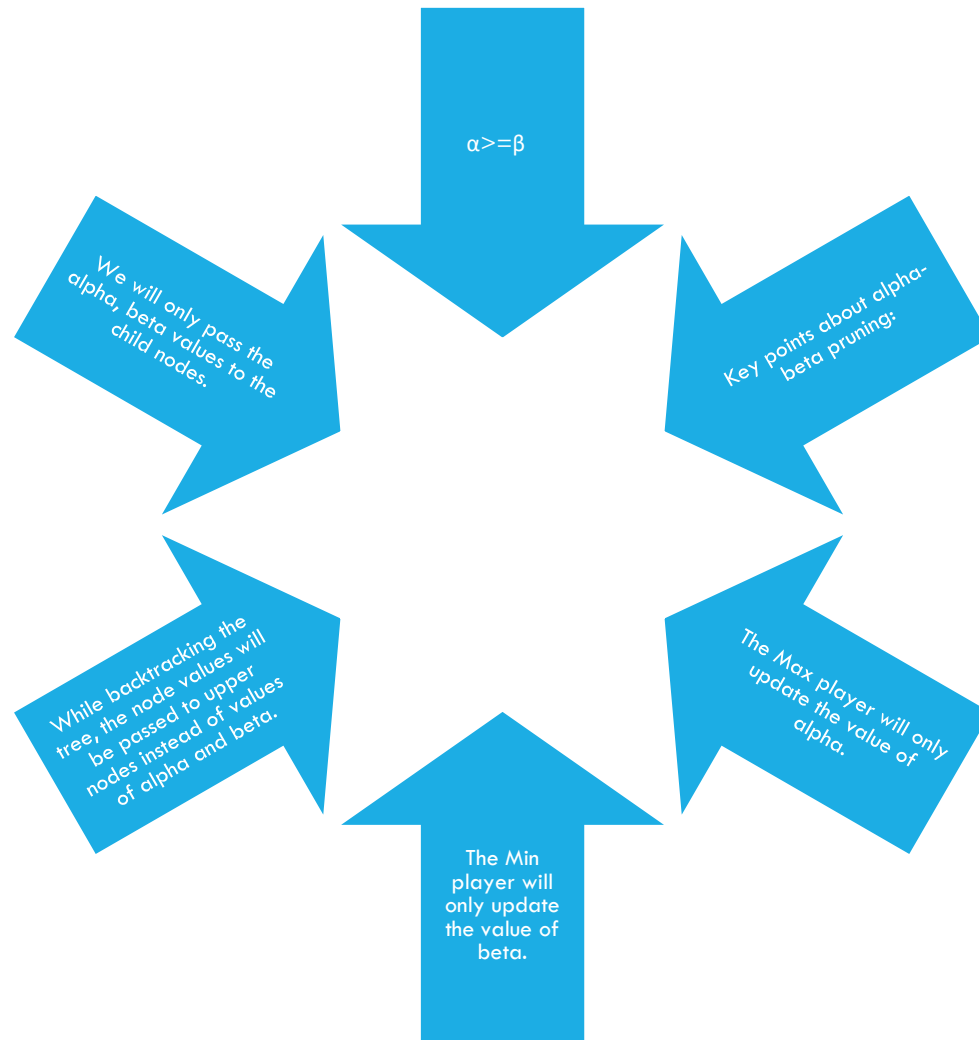
Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.

Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.



By pruning these nodes, it makes the algorithm fast.

ALPHA - BETA PRUNING

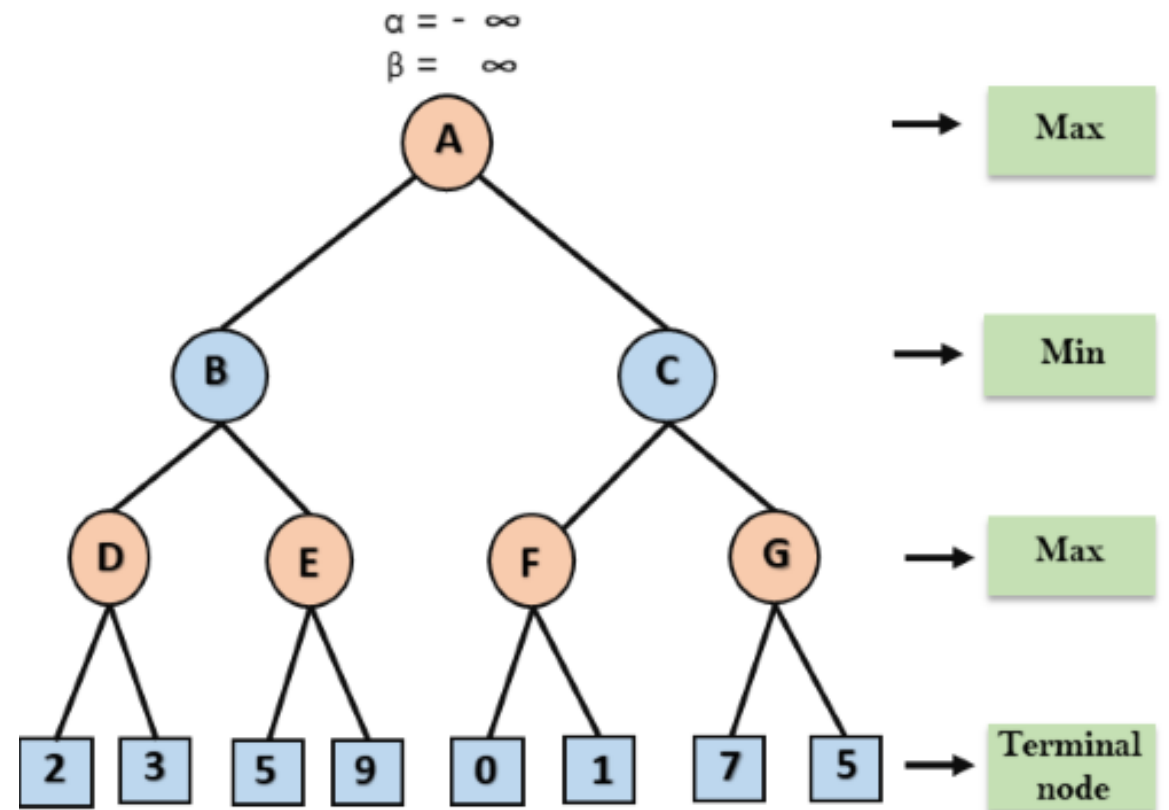


CONDITION FOR ALPHA-BETA PRUNING

WORKING OF ALPHA-BETA PRUNING:

Step 1:

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

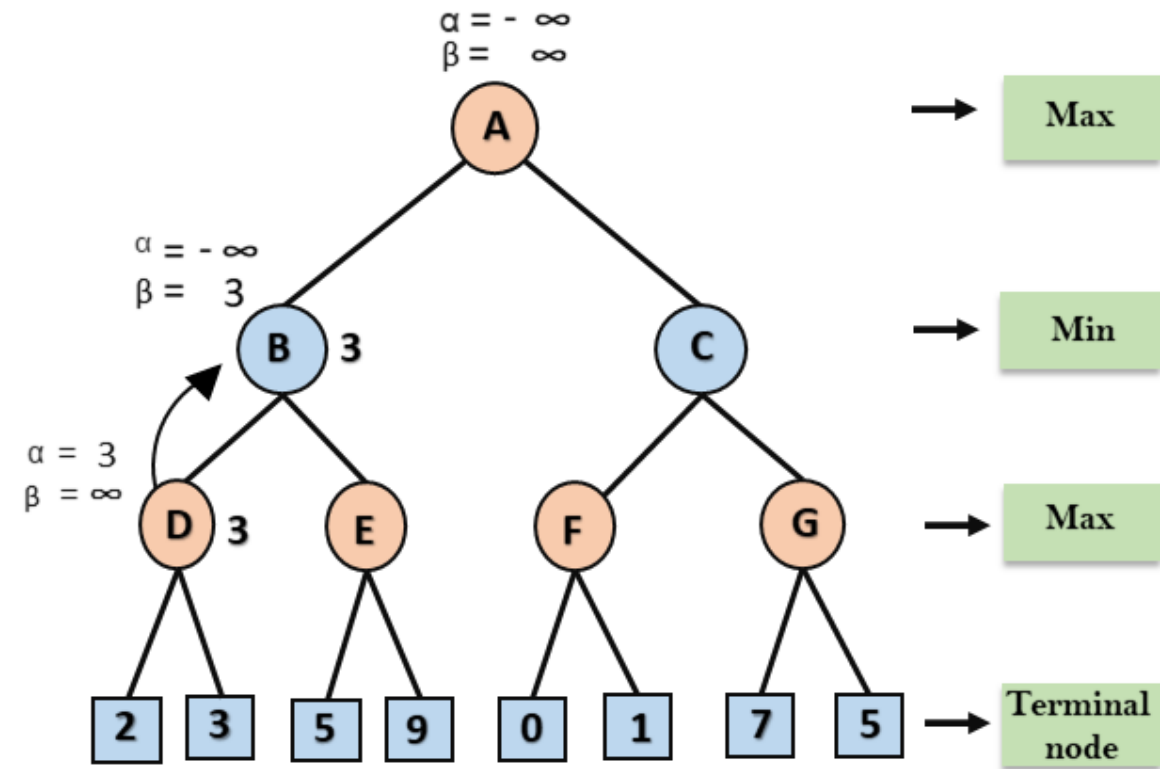


Step 2:

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:

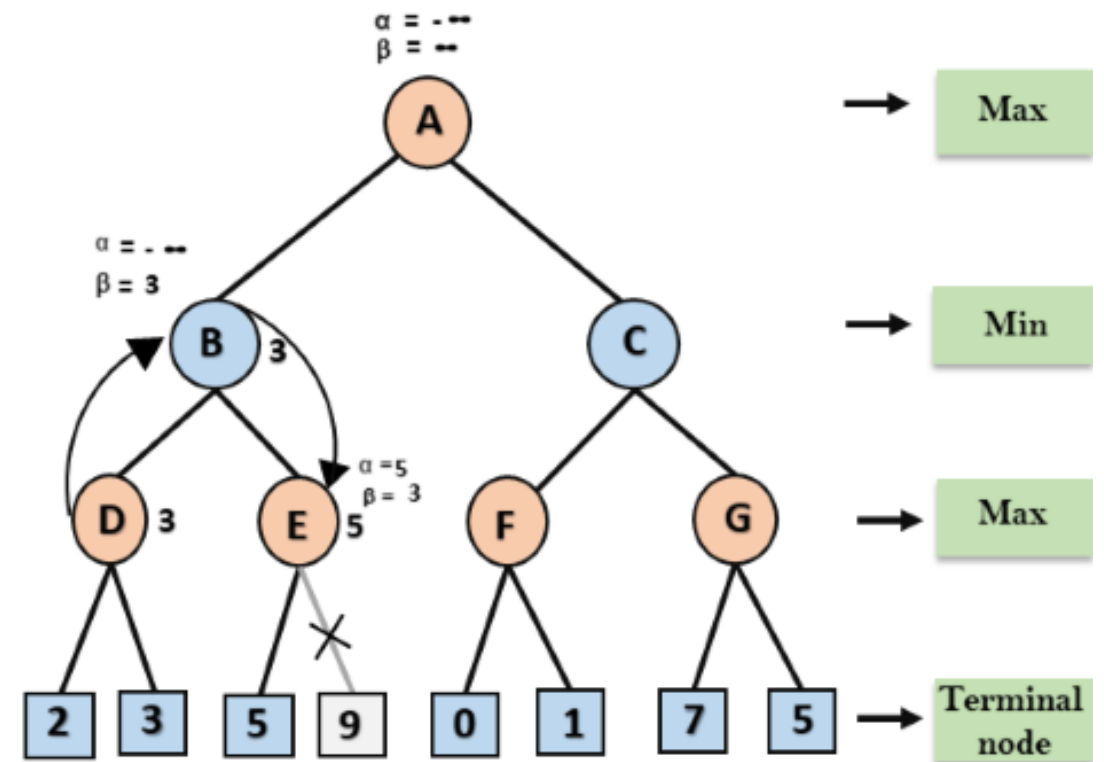
Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now $\beta = +\infty$, will compare with the available subsequent nodes value, i.e. $\min(\infty, 3) = 3$, hence at node B now $\alpha = -\infty$, and $\beta = 3$.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of $\alpha = -\infty$, and $\beta = 3$ 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 $\max(-\infty, 5) = 5$, hence at node E $\alpha = 5$ and $\beta = 3$, where $\alpha \geq \beta$, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



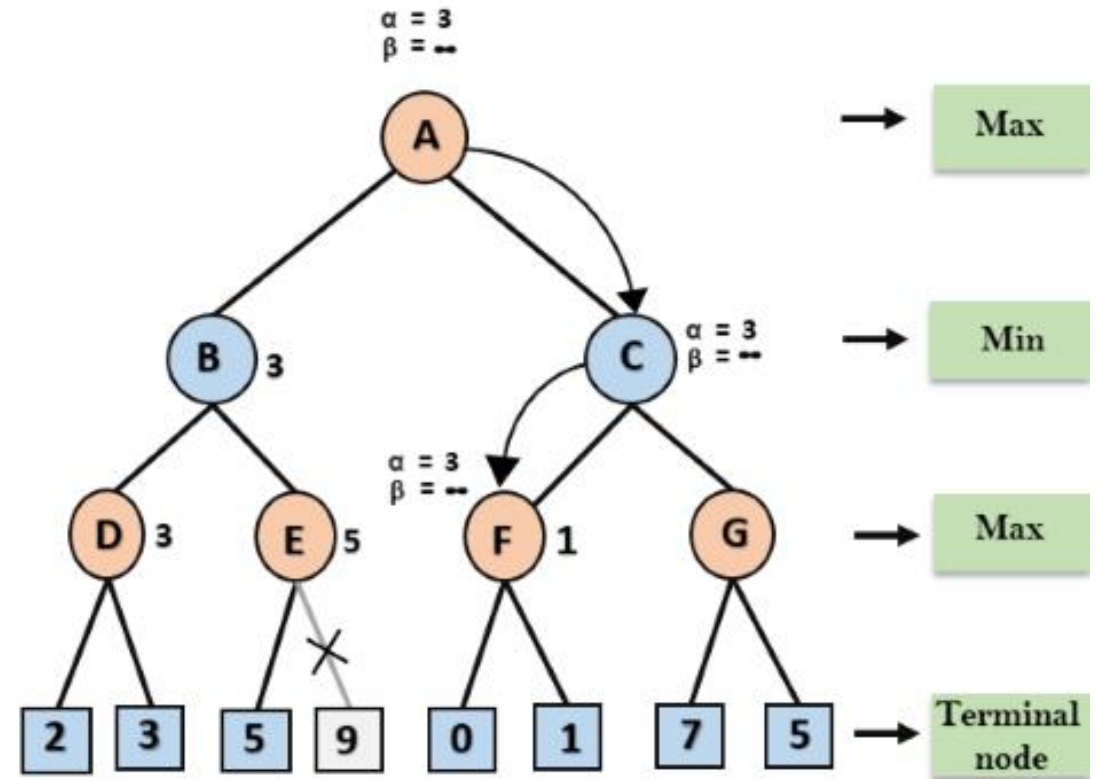
Step 5:

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 $\max(-\infty, 3) = 3$, and $\beta = +\infty$, these two values now passes to right successor of A which is Node C.

At node C, $\alpha = 3$ and $\beta = +\infty$, and the same values will be passed on to node F.

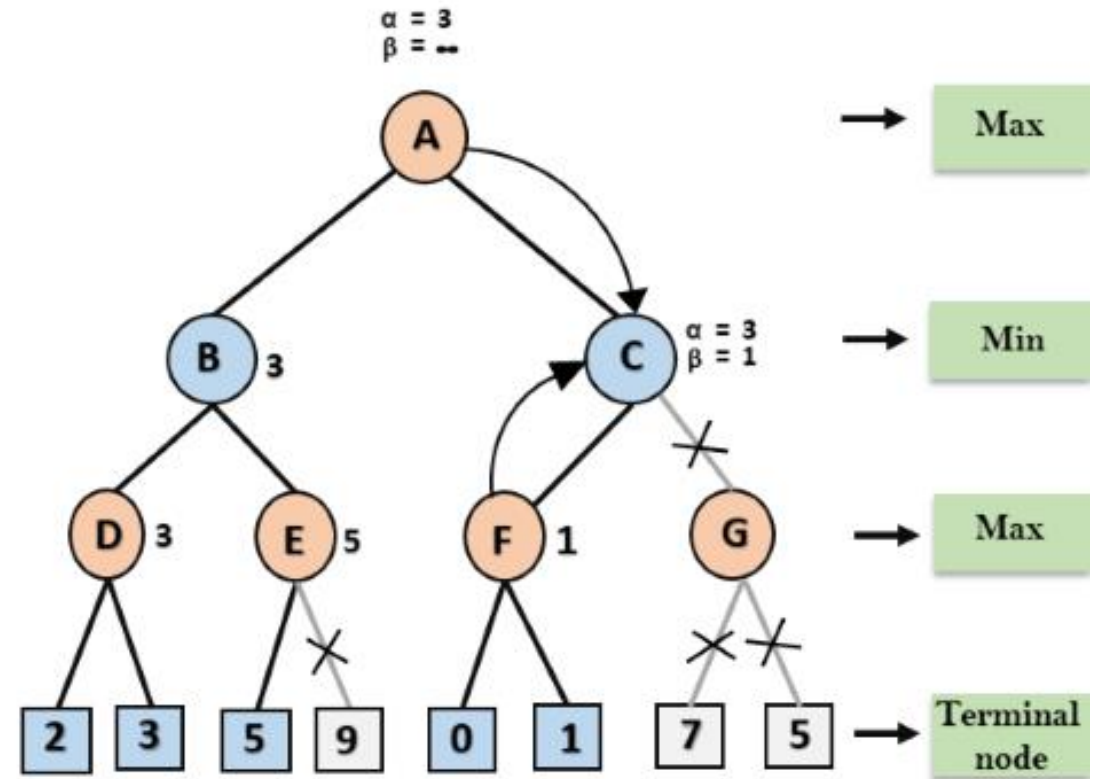
Step 6:

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.



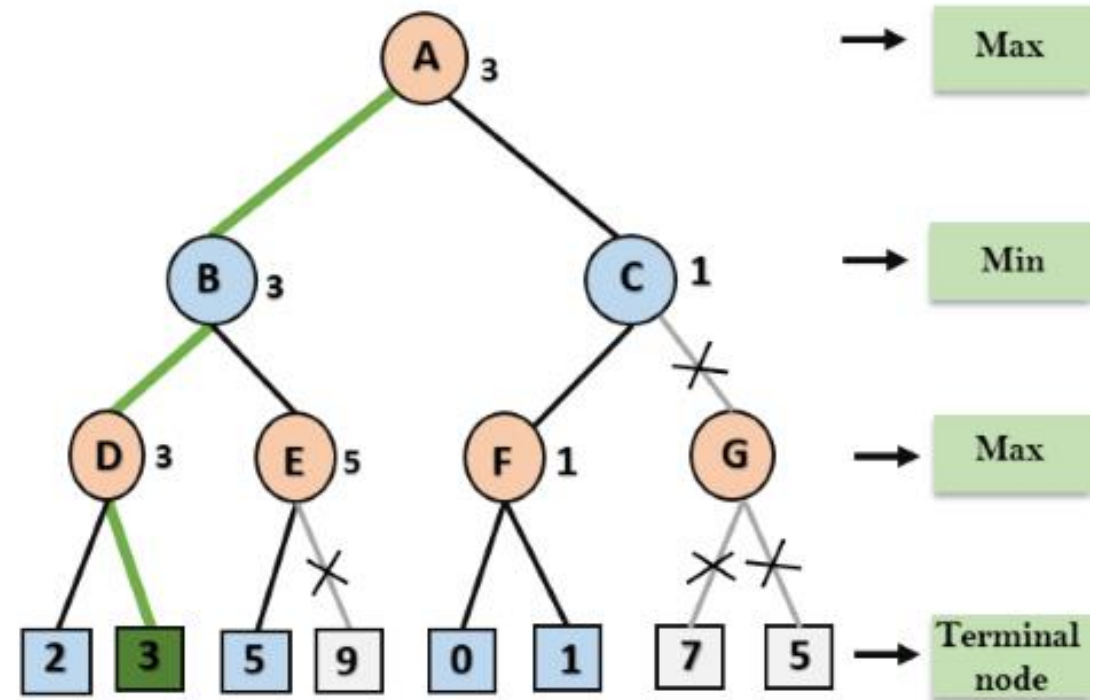
Step 7:

Node F returns the node value 1 to node C, at C $\alpha = 3$ and $\beta = +\infty$, here the value of beta will be changed, it will compare with 1 so $\min(\infty, 1) = 1$. Now at C, $\alpha = 3$ and $\beta = 1$, and again it satisfies the condition $\alpha \geq \beta$, 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 showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning

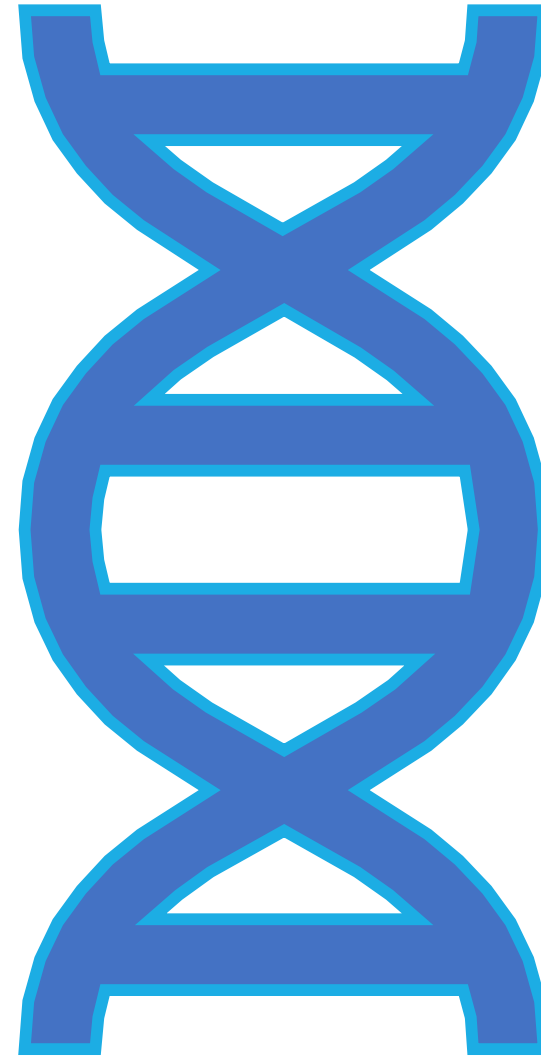
The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined.

Worst ordering:

- ✓ In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm.
- ✓ In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering.
- ✓ In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.

Ideal ordering:

- ✓ The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree.
- ✓ We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.



Any Queries

