

# CHAPTER 2

## Search Algorithms

ARTIFICIAL INTELLIGENCE (BSD2513)  
DR. KU MUHAMMAD NA'IM KU KHALIF



MyMoheS



MyRA



5-STAR WORLD CLASS TECHNOLOGICAL UNIVERSITY

# Content

Chapter 2.1: Definitions of Search Algorithms

Chapter 2.2: Types of Search Algorithms

Chapter 2.3: Uninformed/ Blind Search

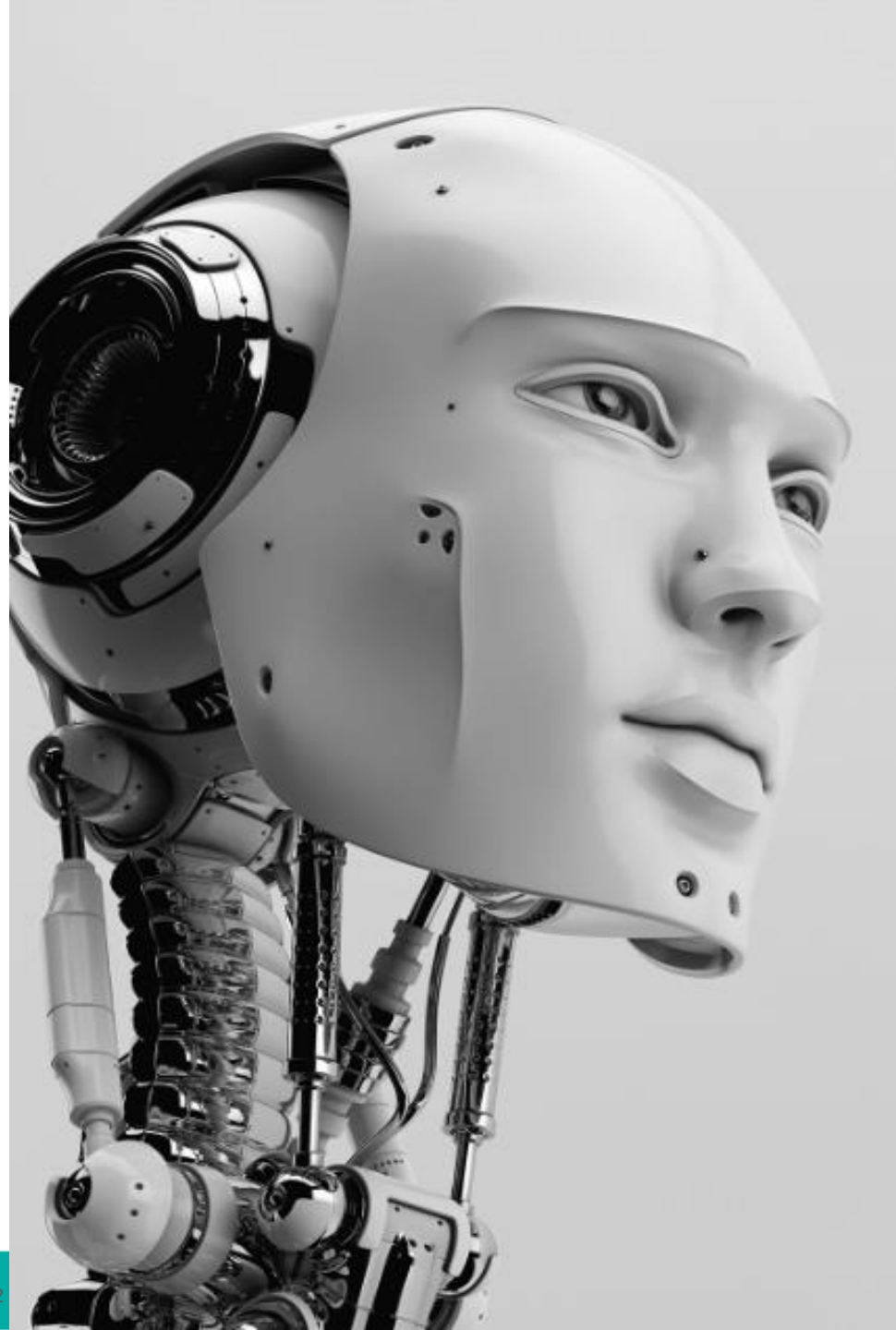
Chapter 2.4: Informed/ Heuristics Search

# Chapter 2.1:

## Definitions of Search Algorithms

By the end of this topic, you should be able to:

- understand the concept of search algorithms in AI and how it's applied in the real world.
- define the search algorithms in the useful in AI understanding.



# An Overview

- 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.
- Search algorithms are one of the most essential areas of artificial intelligence.

## **Problem-solving agents:**

- ✓ In AI, search techniques are universal problem-solving methods.
- ✓ 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.

# Definitions

*“A search algorithm is the step-by-step procedure used to locate specific data among a collection of data. It is considered a fundamental procedure in computing. In computer science, when searching for data, the difference between a fast application and a slower one often lies in the use of the proper search algorithm.”*

(techopedia, 2021)

*“A search algorithm is a unique formula that a search engine uses to retrieve specific information stored within a data structure and determine the significance of a web page and its content. Search algorithms are unique to their search engine and determine search engine result rankings of web pages.”*

(Jessica Teran, 2018)


*“Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:*

- *Sequential Search: The list or array is traversed sequentially and every element is checked. For example: Linear Search.*
- *Interval Search: These algorithms are specifically designed to search sorted data structures. These types of searching algorithms are much more efficient than Linear Search as they repeatedly target the centre of the search structure and divide the search space in half. For example: Binary Search.”*

(GeeksforGeeks, 2021)

## Linear Search

Find '20'



0	1	2	3	4	5	6	7	8
10	50	30	70	80	60	20	90	40



## Binary Search

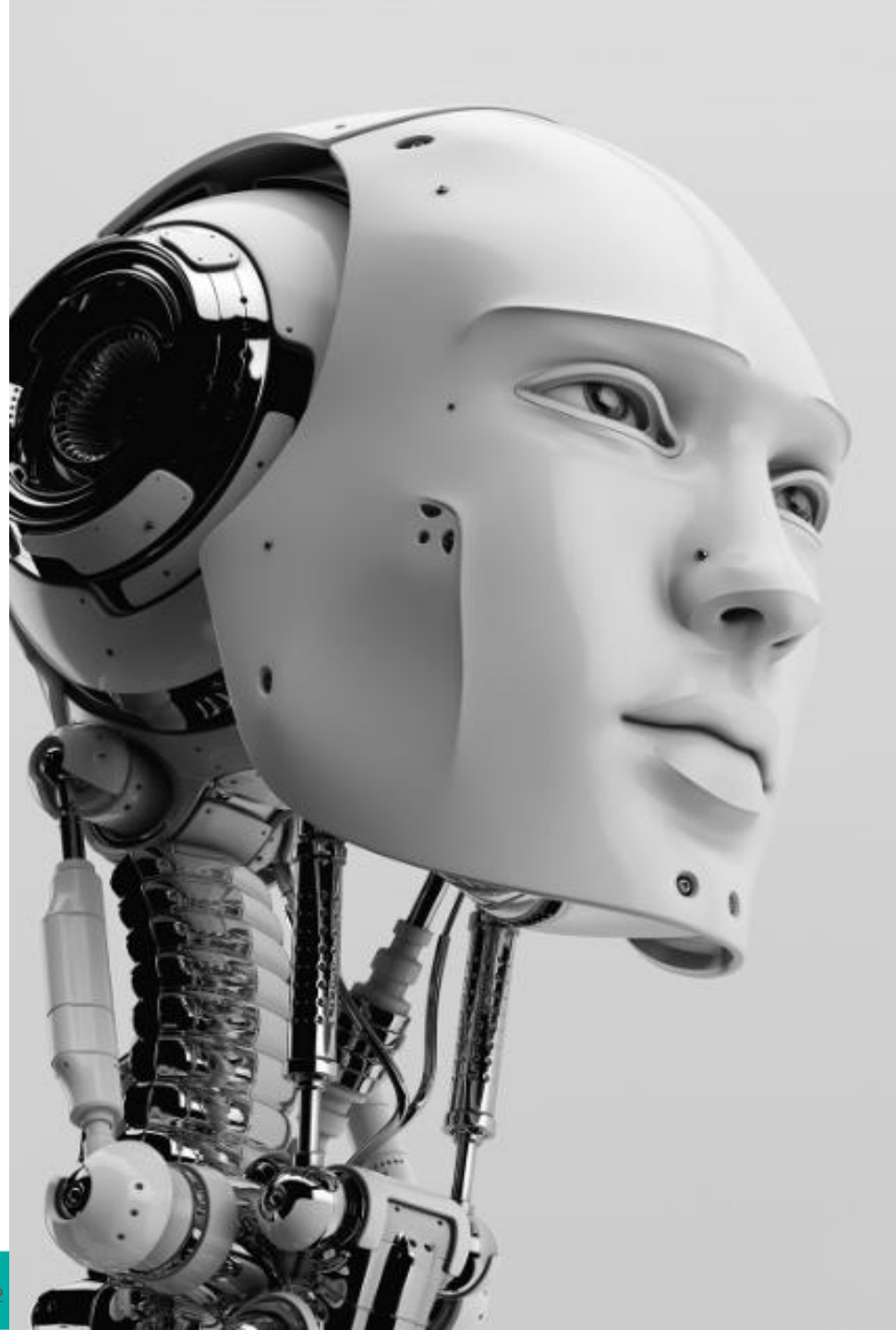
	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
23 > 16 take 2 <sup>nd</sup> half	L=0	1	2	3	M=4	5	6	7	8	H=9
	2	5	8	12	16	23	38	56	72	91
23 > 56 take 1 <sup>st</sup> half	0	1	2	3	4	L=5	6	M=7	8	H=9
	2	5	8	12	16	23	38	56	72	91
Found 23, Return 5	0	1	2	3	4	L=5, M=5	H=6	7	8	9
	2	5	8	12	16	23	38	56	72	91



## Chapter 2.2: Types of Search Algorithms

By the end of this topic, you should be able to:

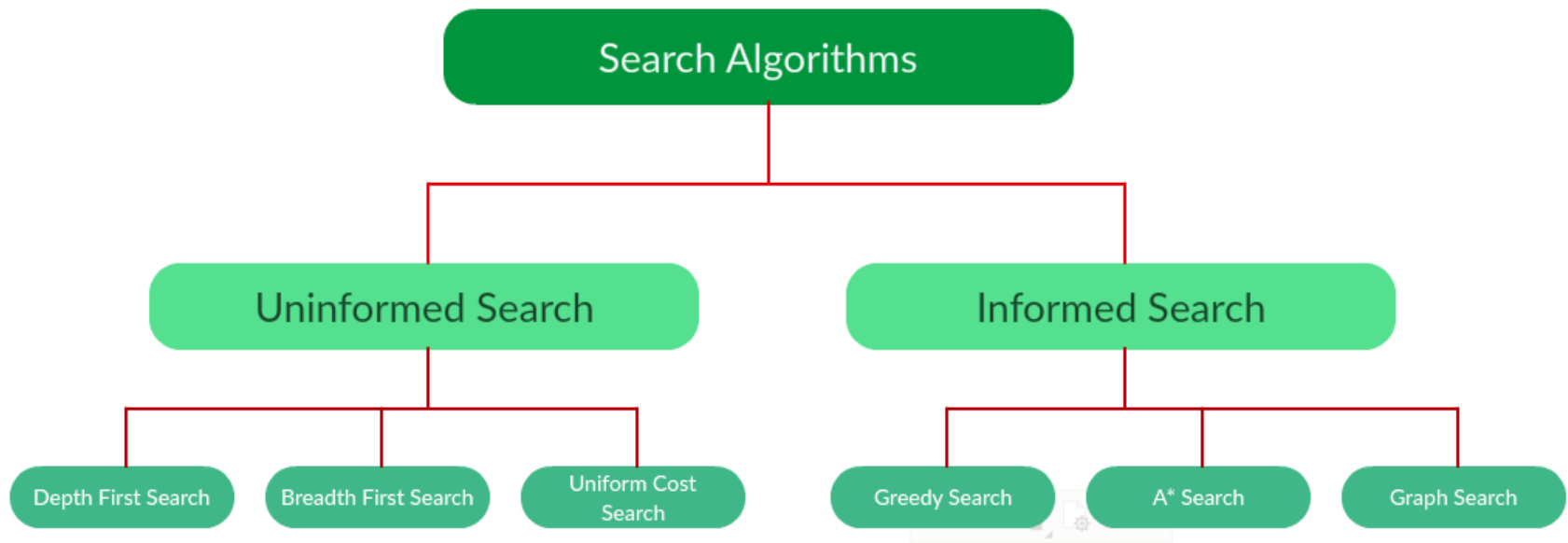
- understand the different types of search algorithms.
- acquire the knowledge to present of search algorithm that is applied on a directed rooted tree.





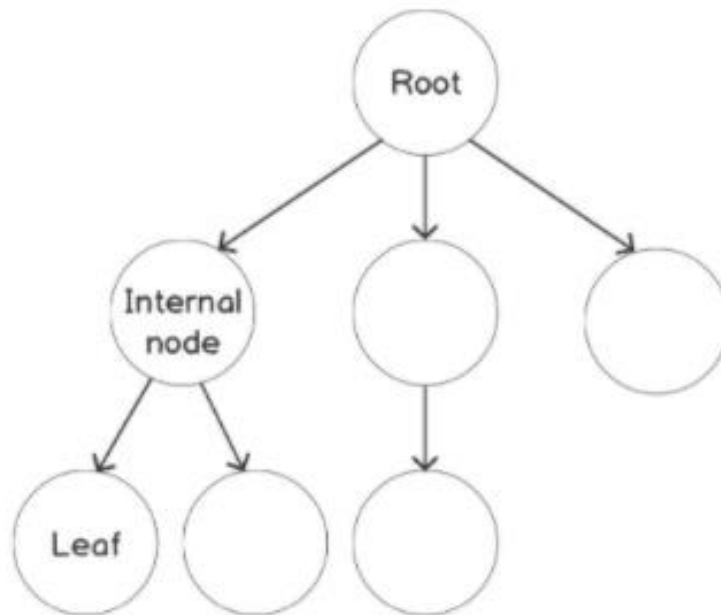
# Types of Search Algorithms

Search algorithms can be classified into uninformed (Blind) search and informed (Heuristic) search algorithms.



**Figure 2.1:** Types of search algorithms.

- These search techniques are applied on a directed rooted tree.
- A tree is a data structure that has nodes, and edges connecting these nodes in such a way that any two nodes of the tree are connected by exactly one path. Have a look at the following figure:



**Figure 2.2:** A directed rooted tree.

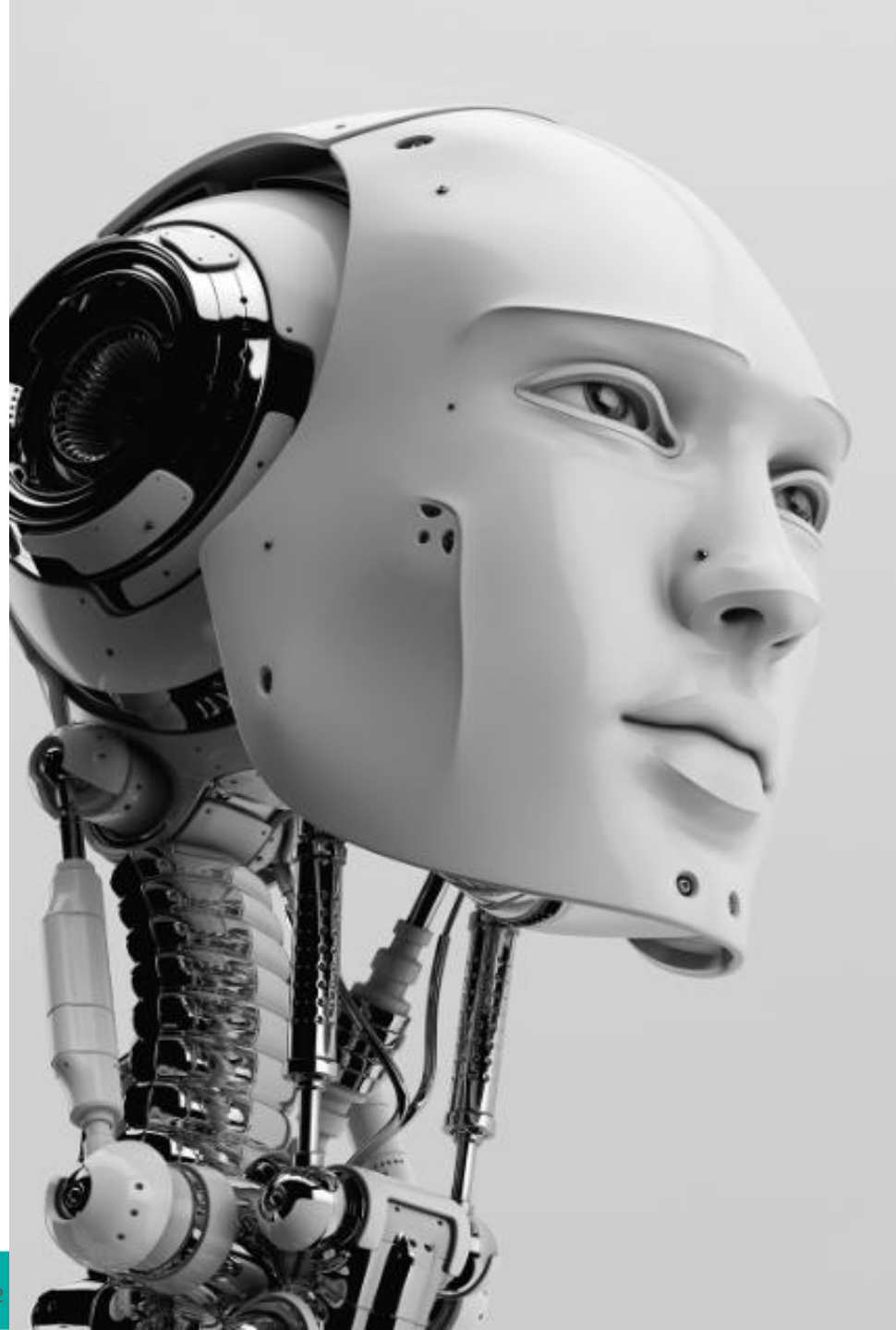
- When the tree is rooted, there is a special node in the tree called the root, which is where we begin our traversal.
- A directed tree is a tree where the edges may only be traversed in one direction.
- Nodes may be internal nodes or leaves. Internal nodes have at least one edge, through which we can leave the node. A leaf has no edges pointing out from the node.
- In AI search, the root of the tree is the starting state. We traverse from this state by generating the successor nodes of the search tree.
- Search techniques differ, depending on the order in which we visit these successor nodes.

# Chapter 2.3:

## Uninformed/ Blind Search

By the end of this topic, you should be able to:

- understand the concept of uninformed search.
- understand some uninformed techniques involved:
  1. Breadth – First search
  2. Depth – First search
  3. Uniform Cost Search



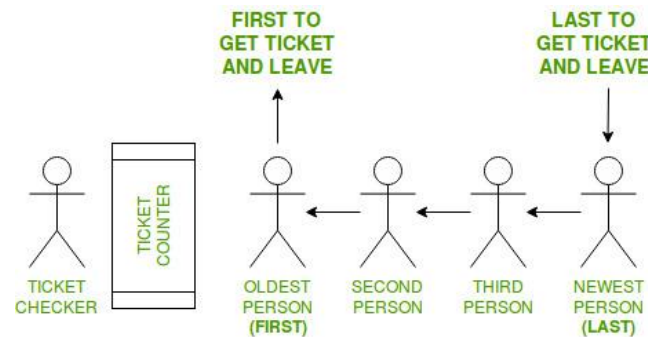
# Uninformed/ Blind Search

- 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.
- In this course, we only cover for these three types of uninformed search algorithms:
  - a. Breadth – First Search
  - b. Depth – First Search
  - c. Uniform Cost Search

# Breadth – First Search

- Breadth-first search (BFS) 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 BFS is an example of a general-graph search algorithm.
- Breadth-first search implemented using **FIFO queue data structure**.

- FIFO is an abbreviation for first in, first out.
- It is a method for handling data structures where the first element is processed first and the newest element is processed last. Real life example:



- From this example, following things are to be considered:
  - ✓ There is a ticket counter where people come, take tickets and go.
  - ✓ People enter a line/ queue to get to the Ticket Counter in an organized manner.
  - ✓ The person to enter the queue first, will get the ticket first and leave the queue.
  - ✓ The person entering the queue next will get the ticket after the person in front of him.
  - ✓ In this way, the person entering the queue last will the tickets last.
  - ✓ Therefore, the First person to enter the queue gets the ticket first and the Last person to enter the queue gets the ticket last.
- This is known as First-In-First-Out approach or FIFO.

## 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.

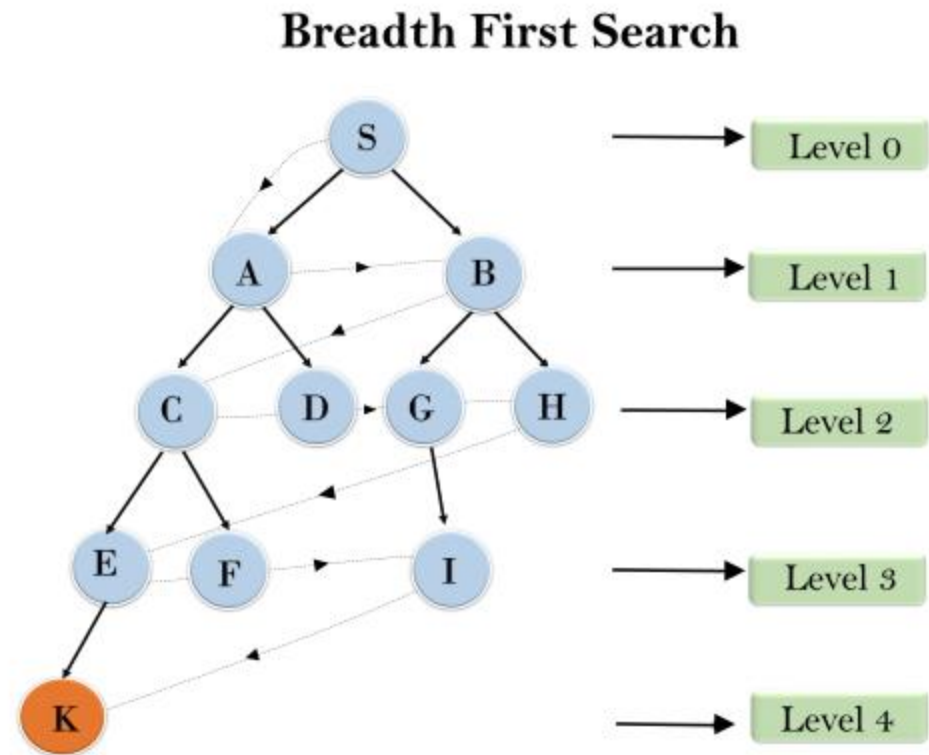


## Example:

In the tree structure, we have shown the traversing of the tree using BFS algorithm 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



## How do we evaluate a search algorithm?

- Primary criteria to evaluate search strategies
  - i. Completeness
    - Is it guaranteed to find a solution (if one exists)?
  - ii. Optimality
    - Does it find the “best” solution (if there are more than one)?
  - iii. Time complexity
    - Number of nodes generated/ expanded
    - How long does it takes to find a solution?
  - iv. Space complexity
    - How much memory does it require?
- Some performance measures
  - Best case
  - Worst case
  - Average case
  - Real-world case

## 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.

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

## 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.

# 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.

## Advantages:

- 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:

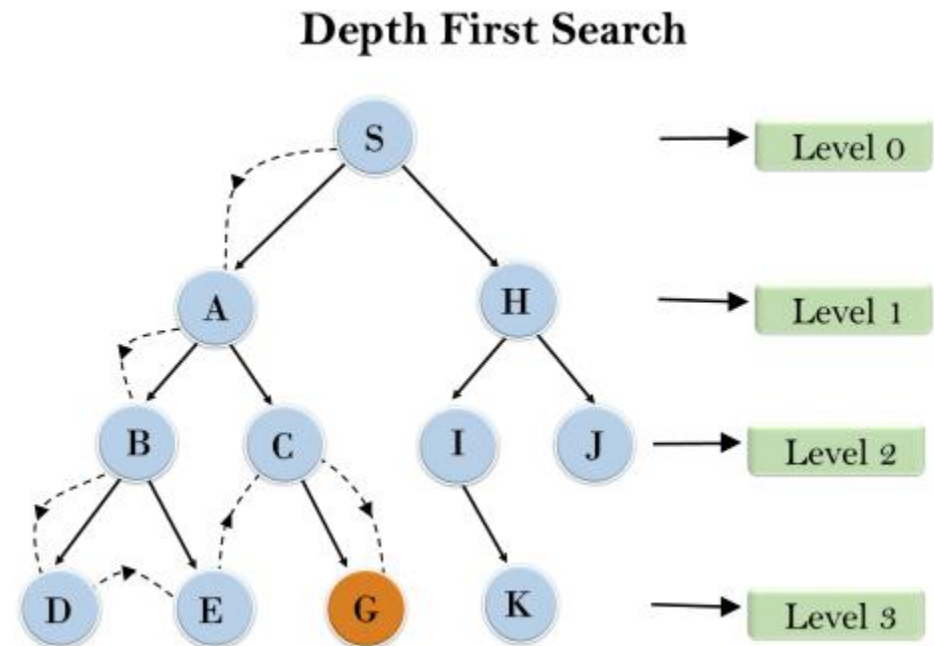
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

## Example:

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

Root node--->node ----> 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.

# Uniform Cost Search

- 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.

## Disadvantage:

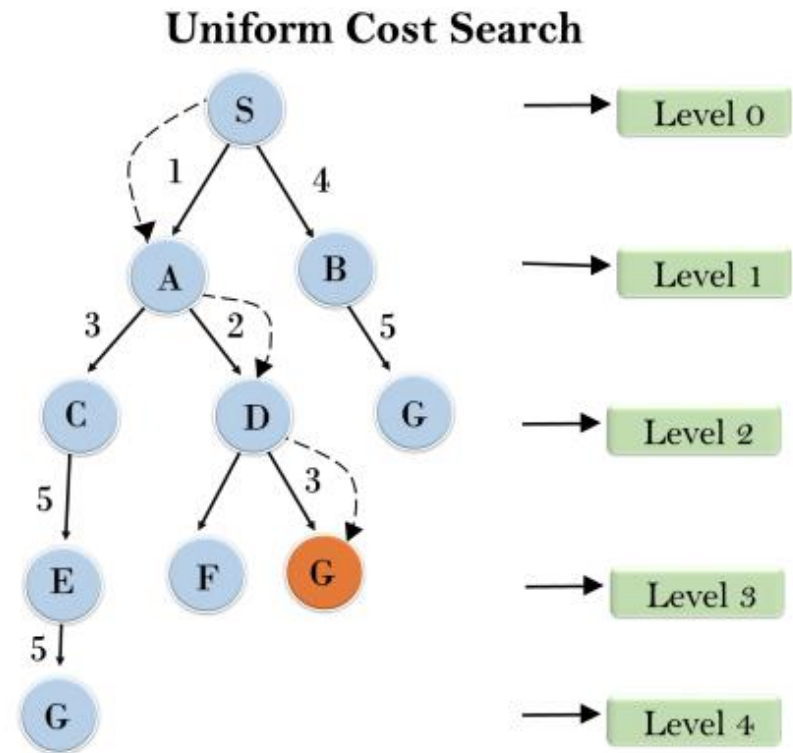
- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

## Example:

In the search tree, we have shown the flow of uniform – Cost search, and it will follow the order as:

Root node--->node (with the least cost) -  
---> node (with the least cost).

It will start searching from root node S, and traverse A, then D, then G.



## **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  $\epsilon$  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  $C^*/\epsilon$ .

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.

# Informed Search

- Informed search algorithms use domain knowledge. In an informed search, problem information can guide the search. Informed search strategies can find a solution more efficiently than uninformed ones.
- 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 suitable solution in a reasonable time.
- Informed search can solve many complex problems which could not be solved in another way. An example of informed search algorithms is a travelling salesman problem. Algorithms involved in this course:
  - a. Greedy Search
  - b. A\* Search
  - c. Genetic Algorithm

## Heuristics function:

The heuristic function takes the agent's current state as its input and estimates how close the agent is to the goal. The heuristic method, however, may only give the best solution, but it guarantees to find a suitable solution in a reasonable time. This function estimates how close a state is to the goal. It is represented by  $h(n)$  and 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.

## Greedy/ Best – First Search

- Greedy search always selects the path which appears best at that moment.
- It combines Depth – First search and Breadth – First search algorithms.
- It uses the heuristic function and search. Best-first search allows us to take advantage of both algorithms. With the help of the best-first search, we can choose the most promising node at each step.
- 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.

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

where  $g(n)$  = estimated cost from node  $n$  to the goal.

The **priority queue** implements the greedy best-first algorithm.

## Greedy 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 place 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 the goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, the 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 lists, then add it to the OPEN list.

Step 7: Return to Step 2.

## Advantages:

- Best first/ Greedy search can switch between BFS and DFS by gaining the advantages of both algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

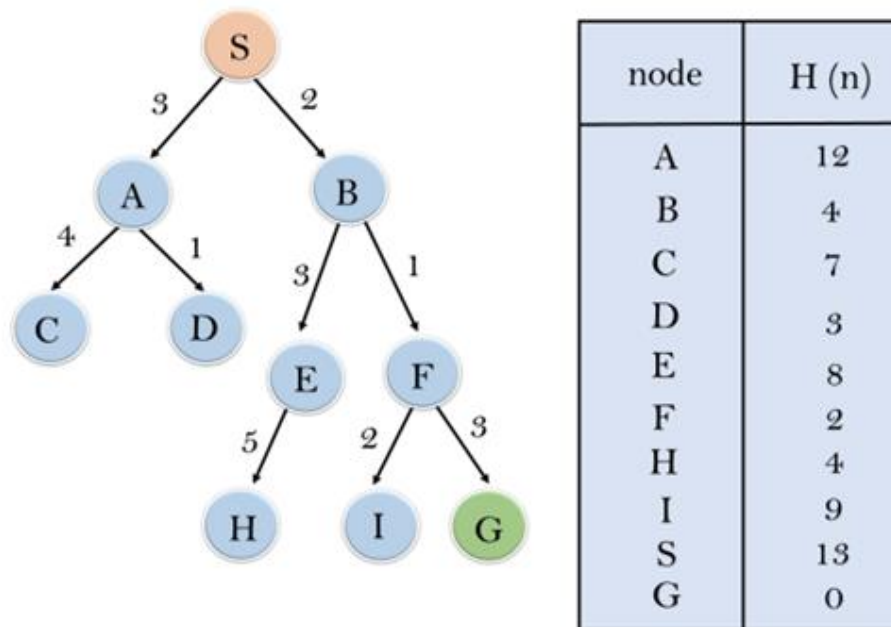
## Disadvantages:

- It can be an unguided depth-first search in the worst-case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.



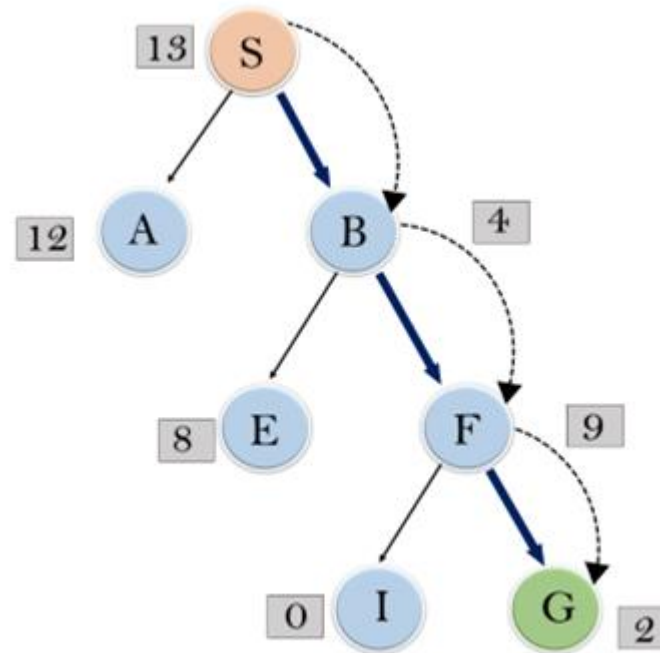
## Example:

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



In this search example, we use two lists which are OPEN and CLOSED Lists.

Following are the iteration for traversing the previous example.  
Solution:



Expand the node of S and put it in the CLOSED list

Initialisation: 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 the Greedy 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 search is also incomplete, even if the given state space is finite.

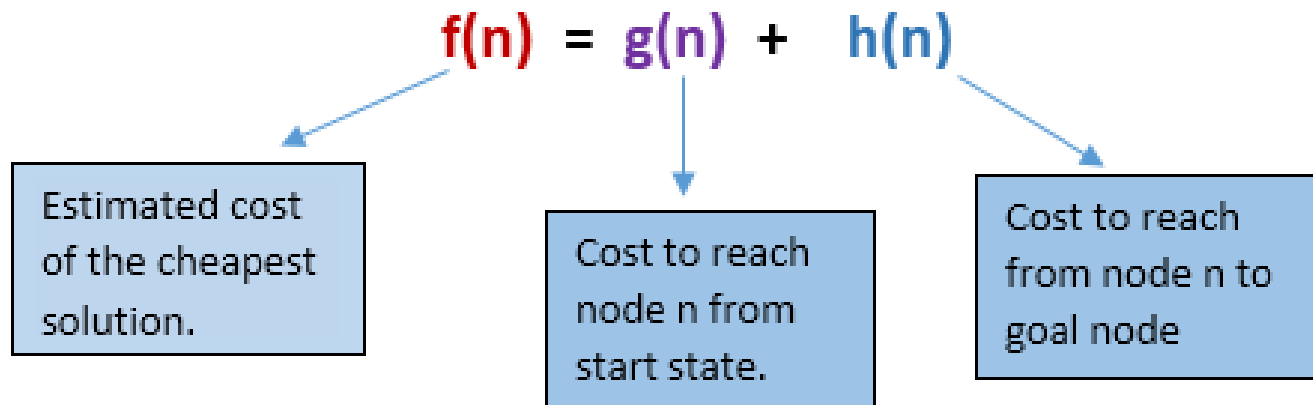
## **Optimal:**

Greedy best-first search algorithm could be more optimal.

# A\* Search

- A\* search algorithm is the most commonly known form of best-first search.
- It uses the 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.
- It finds the shortest path through the search space using the heuristic function.
- This search algorithm expands fewer search trees and provides optimal results faster.
- A\* algorithm is similar to UCS except that it uses  $g(n)+h(n)$  instead of  $g(n)$ .
- In the A\* algorithm, we use the search heuristic and the cost to reach the node.

- Hence we can combine both costs as following, and this sum is called as a **fitness number**.



**Figure 2.3:** Estimated cost of the cheapest solution for A\* search.

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 stop.

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

Step 4: Expand node  $n$ , generate all 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, compute the evaluation function for  $n'$  and place it into the 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:

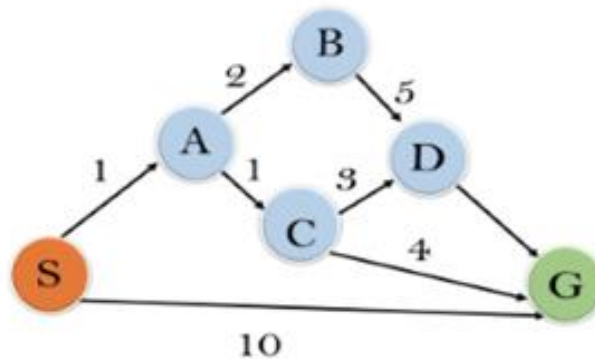
- It does not always produce the shortest path, as it is mainly based on heuristics and approximation.
- A\* search algorithm has some complexity issues.
- The main drawback of A\* is the memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale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



## Example:

We will traverse the graph using the A\* algorithm in this example. The heuristic value of all states is given in 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 the start state.

Here we will use the OPEN and CLOSED lists.



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

## Solution

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

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

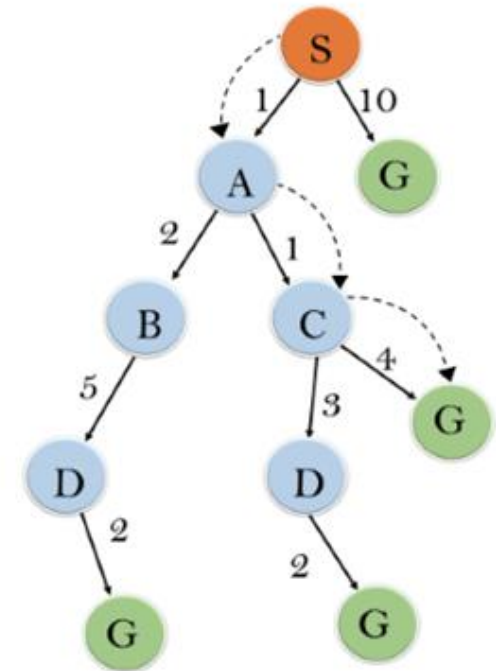
Iteration 4 will give the final result, as  $S \rightarrow A \rightarrow C \rightarrow G$  it provides the optimal path with cost 6.

Points to remember:

A\* algorithm returns the path that occurred first and does not search for all remaining paths.

The efficiency of the A\* algorithm depends on the quality of the heuristic.

A\* algorithm expands all nodes which satisfy the condition  $f(n)$



## **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  $O(b^d)$ .

## **Complete:**

A\* algorithm is complete as long as branching factor is finite and cost at every action is fixed.

## **Optimal:**

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

- a. 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.
- b. 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.

# Genetic Algorithms

- Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms.
- Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimisation and search problems.
- Genetic algorithms simulate the process of natural selection, which means species that adapt to environmental changes can survive, reproduce, and go to the next generation. Simply put, they simulate “survival of the fittest” among individuals of consecutive generations to solve a problem.

- Each generation consists of a population of individuals, and each individual represents a point in search space and a possible solution. Each individual is represented as a string of characters/ integers/ float/ bits. This string is analogous to the Chromosome.

## Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with the genetic structure and behaviour of chromosomes of the population.

Following is the foundation of GAs based on this analogy –

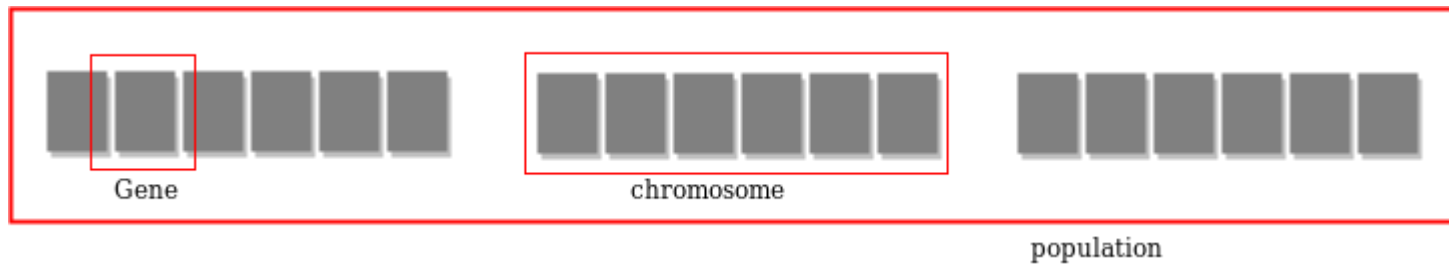
- i. Individual in the population compete for resources and mate.
- ii. Those individuals who are successful (fittest) then mate to create more offspring than others.
- iii. Genes from the “fittest” parent propagate throughout the generation. That is, sometimes parents create offspring which is better than either parent.
- iv. Thus each successive generation is more suited to its environment.

## Search Space

The population of individuals are maintained within the search space. Each individual represents a solution in search space for the given problem.

Each individual is coded as a finite-length vector (analogous to a chromosome) of components.

These variable components are analogous to Genes. Thus a chromosome (individual) comprises several genes (variable components).



## Fitness Score

- A Fitness Score is given to each individual, which shows the ability of an individual to “compete”. The individual having optimal fitness score (or near-optimal) is sought.
- The GAs maintains the population of  $n$  individuals (chromosome/solutions) and their fitness scores. Individuals with better fitness scores are given more chances to reproduce than others. The individuals with better fitness scores mate and produce better offspring by combining the parents’ chromosomes. The population size is static, so the room has to be created for new arrivals.
- So, some individuals die and get replaced by new arrivals, eventually creating a new generation when the old population’s mating opportunity is exhausted. It is hoped that better solutions will arrive over successive generations while most petite fit die.

- Each new generation has, on average more “better genes” than previous generations’ individuals (solution). Thus each new generations have better “partial solutions” than previous generations.
- Once the offspring produced has no significant difference from those produced by previous populations, the population converges.
- The algorithm is said to be converted to a set of solutions for the problem.

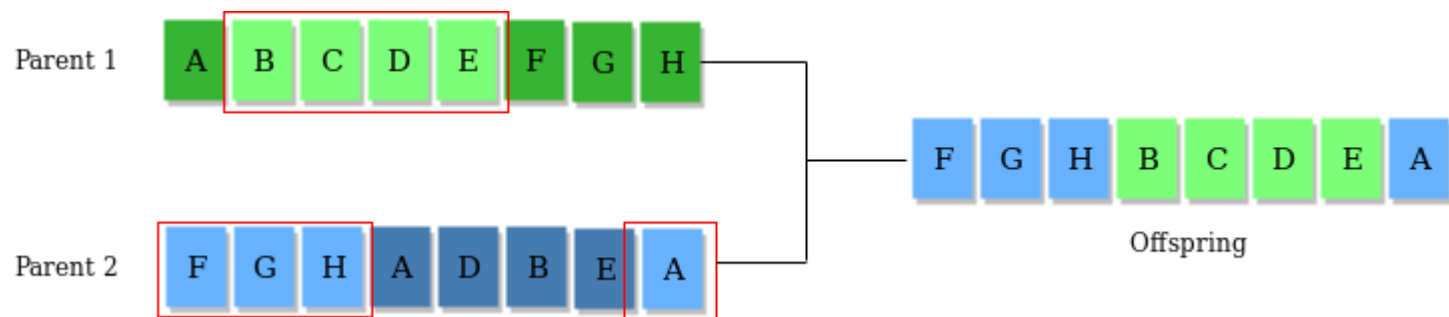


## Operators of Genetic Algorithms

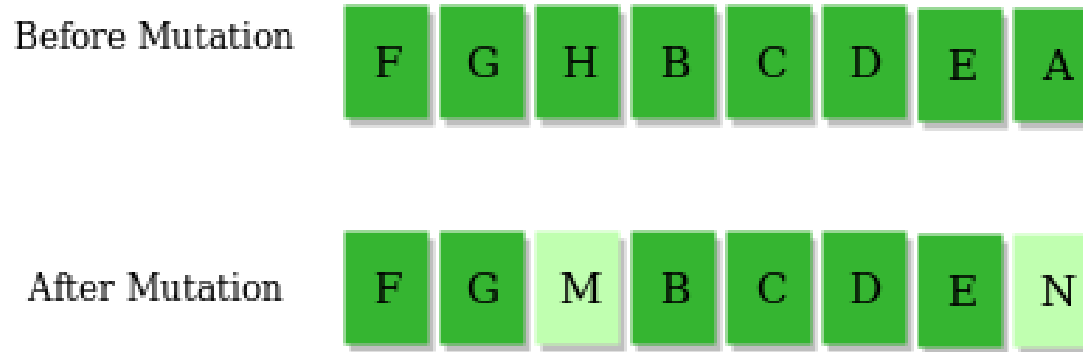
Once the initial generation is created, the algorithm evolves the generation using the following operators:

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using a selection operator, and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged, thus creating an entirely new individual (offspring). For example –



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For example –



The whole algorithm can be summarized as –

- 1) Randomly initialize populations p
- 2) Determine fitness of population
- 3) Until convergence repeat:
  - a) Select parents from population
  - b) Crossover and generate new population
  - c) Perform mutation on new population
  - d) Calculate fitness for new population

## How Genetic Algorithm Work?

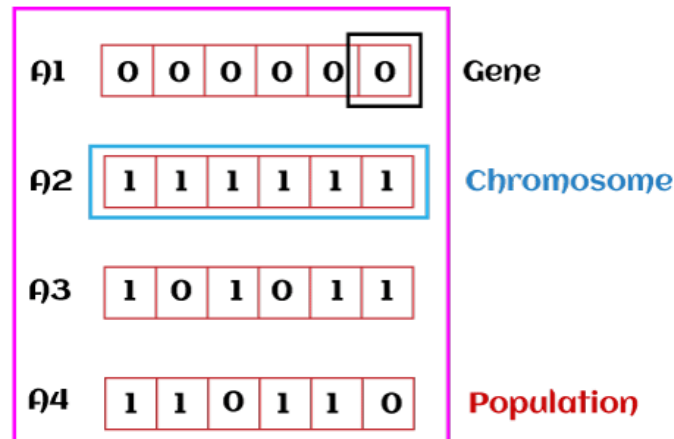
The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that enhance or replace the population to give an improved fit solution.

It involves five phases to solve the complex optimisation problems, which are given below:

1. Initialisation
2. Fitness Assignment
3. Selection
4. Reproduction
5. Termination

## Initialisation

- The process of a genetic algorithm starts by generating a set of individuals, called the population. Here each individual is the solution for the given problem.
- An individual contains or is characterised by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which solves the problem. One of the most popular initialisation techniques is using random binary strings.



## Fitness Assignment

- Fitness function is used to determine how to fit an individual is. It means the ability of an individual to compete with other individuals.
- In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual.
- This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction.

## Selection

- The selection phase involves the selection of individuals for the reproduction of offspring.
- All the selected individuals are then arranged in a pair of two to increase reproduction.
- Then these individuals transfer their genes to the next generation.

There are three types of Selection methods available, which are:

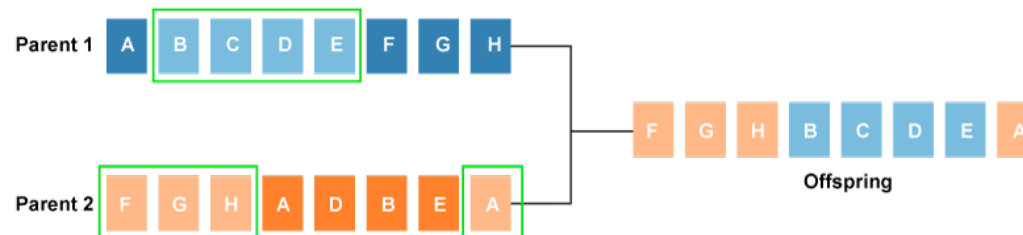
- ✓ Roulette wheel selection
- ✓ Tournament selection
- ✓ Rank-based selection

## Reproduction

- After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm applies two variation operators to the parent population. The two operators involved in the reproduction phase are given below:

### Crossover:

The crossover plays the most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected randomly within the genes. Then the crossover operator swaps the genetic information of two parents from the current generation to produce a new individual representing the offspring.



- Parents' genes are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called crossover.

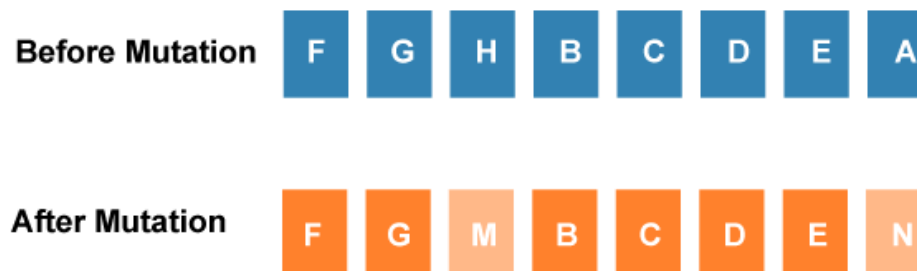
Types of crossover styles available:

- ✓ One point crossover
- ✓ Two-point crossover
- ✓ Livery crossover
- ✓ Inheritable Algorithms crossover



## Mutation

- The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.
- Mutation helps in solving the issue of premature convergence and enhances diversification. The below image shows the mutation process:
- Types of mutation styles available,
  - ✓ Flip bit mutation
  - ✓ Gaussian mutation
  - ✓ Exchange/ Swap mutation



## Termination

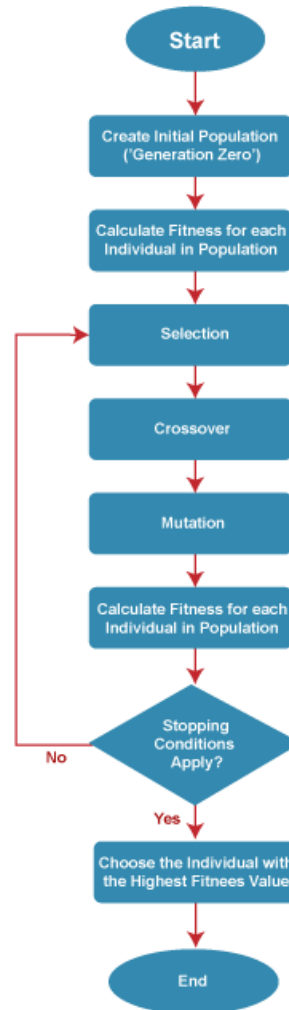
- After the reproduction phase, a stopping criterion is applied as a base for termination.
- The algorithm terminates after the threshold fitness solution is reached.
- It will identify the final solution as the best solution in the population.

- Parents' genes are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called crossover.

Types of crossover styles available:

- ✓ One point crossover
- ✓ Two-point crossover
- ✓ Livery crossover
- ✓ Inheritable Algorithms crossover

# General Workflow of Genetic Algorithm



## Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce a target string starting from a random string of the same length. In the following implementation, the following analogies are made –

- Characters A-Z, a-z, 0-9, and other special symbols are considered genes
- A string generated by these characters is considered a chromosome/ solution/ individual

The fitness score is the number of characters which differ from characters in the target string at a particular index. So individual having lower fitness value is given more preference.

## Why use Genetic Algorithms?

- Robust
- Provide optimisation over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

## Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

# Links to Read

<https://www.javatpoint.com/search-algorithms-in-ai>

<https://www.geeksforgeeks.org/search-algorithms-in-ai/>

<https://www.javatpoint.com/ai-uninformed-search-algorithms#:~:text=Uninformed%20search%20is%20a%20class,is%20also%20called%20blind%20search.>

<https://www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming/>

[https://www.javatpoint.com/ai-uninformed-search-algorithms#:~:text=Time%20Complexity%3A%20Time%20Complexity%20of,a%20node%20at%20every%20state.&text=Space%20Complexity%3A%20Space%20complexity%20of,is%20O\(bd\).](https://www.javatpoint.com/ai-uninformed-search-algorithms#:~:text=Time%20Complexity%3A%20Time%20Complexity%20of,a%20node%20at%20every%20state.&text=Space%20Complexity%3A%20Space%20complexity%20of,is%20O(bd).)

<https://www.javatpoint.com/ai-informed-search-algorithms>

<https://www.geeksforgeeks.org/genetic-algorithms/>

[https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_quick\\_guide.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_quick_guide.htm)

