UNIT–II: PROBLEM SOLVING BY SEARCHING (9 periods)

Problem solving agents, Search algorithms, Uninformed search strategies, Informed search strategies – Greedy best-first search, A* search; Heuristic functions.

2.1 PROBLEM SOLVING AGENTS

PROBLEM-SOLVING APPROACH IN ARTIFICIAL INTELLIGENCE PROBLEMS

"The **reflex agents** are known as the simplest agents because they directly map states into actions. Unfortunately, these agents fail to operate in an environment where the mapping is too large to store and learn. **Goal-based agent,** on the other hand, considers future actions and the desired outcomes.

Here, we will discuss one type of goal-based agent known as a **problem-solving agent**, which uses atomic representation with no internal states visible to the *problem-solving algorithms*.

Problem-solving agent

The problem-solving agent perfoms precisely by defining problems and its several solutions.

- According to psychology, "*a problem-solving refers to a state where we wish to reach toa definite goal from a present state or condition.*"
- According to computer science, *a problem-solving is a part of artificial intelligence whichencompasses a number of techniques such as algorithms, heuristics to solve a problem.*

Therefore, a problem-solving agent is a **goal-driven agent** and focuses on satisfying the goal.

PROBLEM DEFINITION

To build a system to solve a particular problem, we need to do four things:

- **Define** the problem precisely. This definition must include specification of the initial situations and also final situations which constitute (i.e) acceptable solution to the problem.
- **Analyze** the problem (i.e) important features have an immense (i.e) huge impact onthe appropriateness of various techniques for solving the problems.
- **Isolate and represent** the knowledge to solve the problem.
- **Choose the best** problem – solving techniques and apply it to the particular problem.

- **Steps performed by Problem-solving agent**

  - **Goal Formulation:** It is the first and simplest step in problem-solving. It organizesthe steps/sequence required to formulate one goal out of multiple goals as well as actions toachieve that goal. Goal formulation is based on the current situation and the agent'sperformance measure (discussed below).
  - **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
  - **Initial State:** It is the starting state or initial step of the agent towards its goal.
  - **Actions:** It is the description of the possible actions available to the agent.
  - **Transition Model:** It describes what each action does.
  - **Goal Test:** It determines if the given state is a goal state.
  - **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, **anoptimal solution has the lowest path cost among all the solutions.**

Note: Initial state, actions, and **transition model** together define the **state-space** of the problemimplicitly. State-space of a problem is a set of all states which can be reached from the initial statefollowed by any sequence of actions. The state-space forms a directed map or graph where nodesare the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.

- **Search:** It identifies all the best possible sequence of actions to reach the goal state fromthe current state. It takes a problem as an input and returns solution as its output.
- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as thebest optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach thegoal state from the current state.

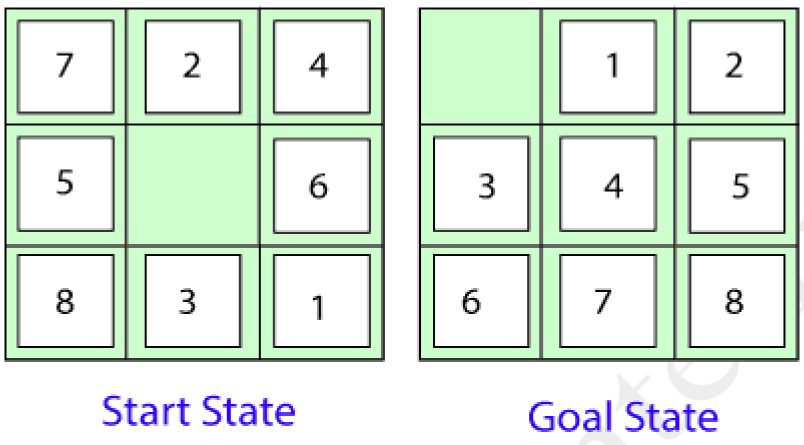
Example Problems

Basically, there are two types of problem approaches:

- **Toy Problem:** It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.
- **Real-world Problem:** It is real-world based problems which require solutions. Unlike a toyproblem, it does not depend on descriptions, but we can have a general formulation of the problem.

Some Toy Problems

- **8 Puzzle Problem:** Here, we have a 3×3 matrix with movable tiles numbered from 1 to 8 with a blank space. The tile adjacent to the blank space can slide into that space. The objective is to reach a specified goal state similar to the goal state, as shown in the below figure.
- In the figure, our task is to convert the current state into goal state by sliding digits into theblank space.



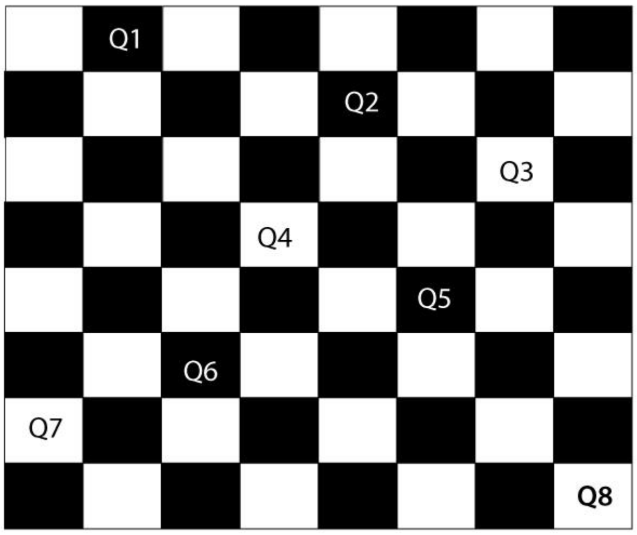
In the above figure, our task is to convert the current(Start) state into goal state by sliding digitsinto the blank space.

The problem formulation is as follows:

- **States:** It describes the location of each numbered tiles and the blank tile.
- **Initial State:** We can start from any state as the initial state.
- **Actions:** Here, actions of the blank space is defined, i.e., either **left, right, up or down**
- **Transition Model:** It returns the resulting state as per the given state and actions.
- **Goal test:** It identifies whether we have reached the correct goal-state.
- **Path cost:** The path cost is the number of steps in the path where the cost of each step is 1. **Note:** The 8-puzzle problem is a type of **sliding-block problem** which is used for testingnew search algorithms in artificial intelligence.

- **8-queens problem:** The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either **diagonallyor in same row and column.**

From the following figure, we can understand the problem as well as its correct solution.



It is noticed from the above figure that each queen is set into the chessboard in a position where no other queen is placed diagonally, in same row or column. Therefore, it is one right approach tothe 8-queens problem.

For this problem, there are two main kinds of formulation:

1. **Incremental formulation:** It starts from an empty state where the operator augments a queenat each step.

Following steps are involved in this formulation:

- **States:** Arrangement of any 0 to 8 queens on the chessboard.
- **Initial State:** An empty chessboard
- **Actions:** Add a queen to any empty box.
- **Transition model:** Returns the chessboard with the queen added in a box.
- **Goal test:** Checks whether 8-queens are placed on the chessboard without any attack.
- **Path cost:** There is no need for path cost because only final states are

counted. In this formulation, there is approximately **1.8×10^{14}** possible sequence to investigate.

2. **Complete-state formulation:** It starts with all the 8-queens on the chessboard and moves themaround, saving from the attacks.

Following steps are involved in this formulation

- **States:** Arrangement of all the 8 queens one per column with no queen attacking the otherqueen.

- **Actions:** Move the queen at the location where it is safe from the attacks.

This formulation is better than the incremental formulation as it reduces the state space from **1.8 x10^{14}** to **2057**, and it is easy to find the solutions.

Some Real-world problems

- **Traveling salesperson problem (TSP):** It is a **touring problem** where thesalesman can visit each city only once. The objective is to find the shortest tour and sell-out the stuff in each city.
- **VLSI Layout problem:** In this problem, millions of components and connectionsare positioned on a chip in order to minimize the area, circuit-delays, stray-capacitances, and maximizing the manufacturing yield.

The layout problem is split into two parts:

- **Cell layout:** Here, the primitive components of the circuit are grouped into cells, each performing its specific function. Each cell has a fixed shape and size. The task is to place the cells on the chip without overlapping each other.
- **Channel routing:** It finds a specific route for each wire through the gaps betweenthe cells.

- **Protein Design:** The objective is to find a sequence of amino acids which will foldinto 3D protein having a property to cure some disease.

Searching for solutions

- We have seen many problems. Now, there is a need to search for solutions to solve them.
- In this section, we will understand how searching can be used by the agent to solve a problem.
- For solving different kinds of problem, an agent makes use of different strategies to reach the goal by searching the best possible algorithms. This process of searching is known as **search strategy.**

2.2 Search Algorithms

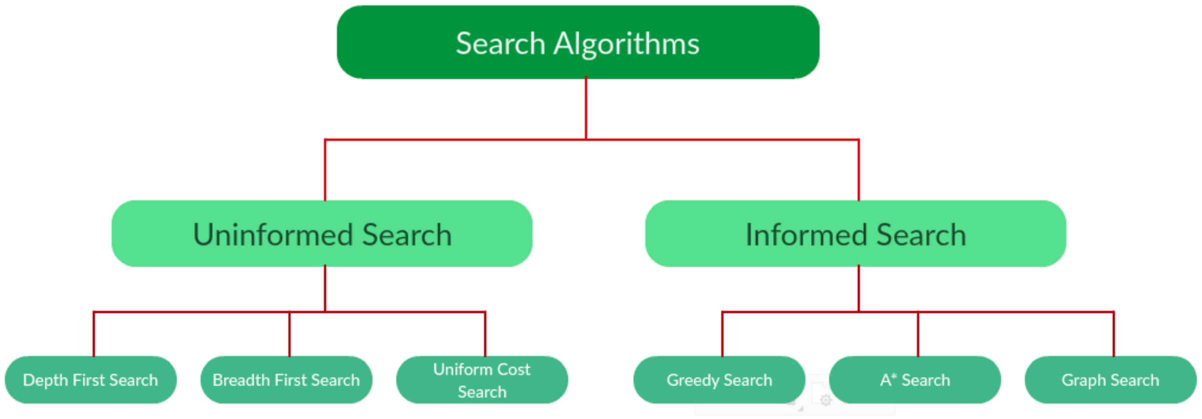
Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
    - **A State Space.** Set of all possible states where you can be.
    - **A Start State.** The state from where the search begins.
    - **A Goal State.** 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.

Types of search algorithms:

There are far too many powerful search algorithms out there to fit in a single article. Instead, this article will discuss six of the fundamental search algorithms, divided into two categories, as shown below.



Note that there is much more to search algorithms than the chart I have provided above. However, this article will mostly stick to the above chart, exploring the algorithms given there.

2.3 <u>Uninformed Search Algorithms:</u>

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**. These algorithms can only generate the successors and differentiate between the goal state and non goal state.

The following uninformed search algorithms are discussed in this section.

- Depth First Search
- Breadth First Search
- Uniform Cost Search

Each of these algorithms will have:

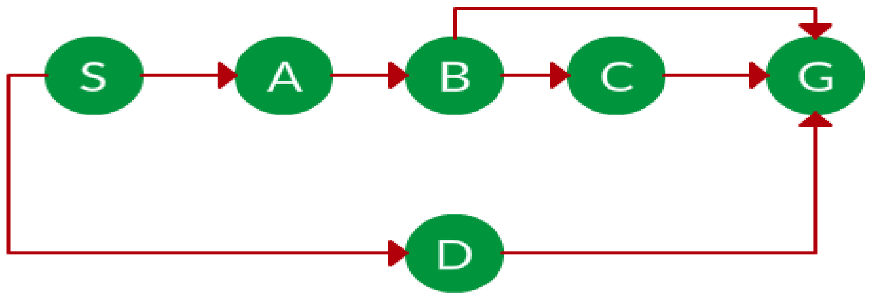
- A problem **graph,** containing the start node S and the goal node G.
- A **strategy,** describing the manner in which the graph will be traversed to get to G.
- A **fringe,** which is a data structure used to store all the possible states (nodes) that you can go from the current states.
- A **tree,**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

Depth First Search:

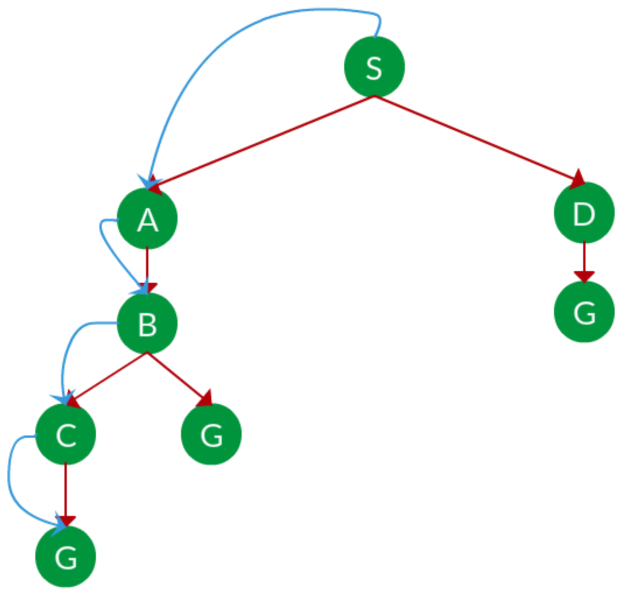
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. It uses last in-first-out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

Example:

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



Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



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

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

n^i= number of nodes in level .

Time complexity: Equivalent to the number of nodes traversed in DFS.

$$T(n) = 1 + n^2 + n^3 + ... + n^d = O(n^d)$$

Space complexity: Equivalent to how large can the fringe get. $S(n) = O(n \times d)$

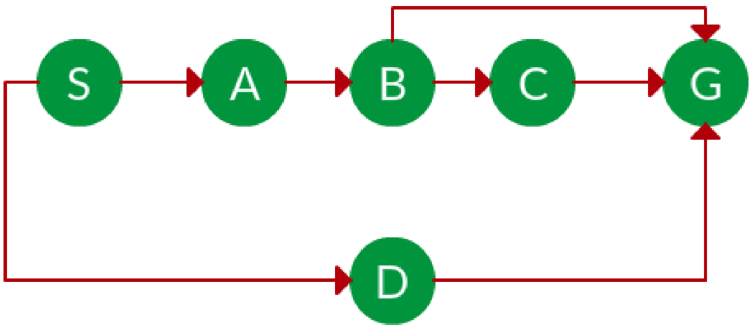
Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Optimality: DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

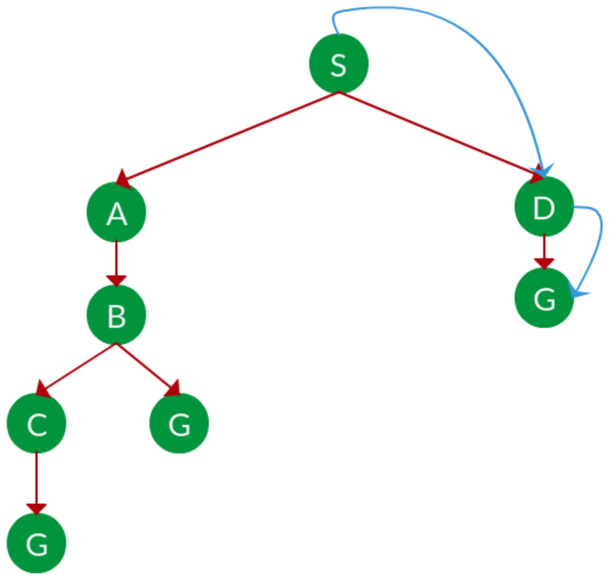
Breadth First Search:

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It is implemented using a queue.

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



Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> D -> G

s= the depth of the shallowest solution.

n^i= number of nodes in level .

Time complexity: Equivalent to the number of nodes traversed in BFS until the shallowest solution.

$$T(n) = 1 + n^2 + n^3 + ... + n^s = O(n^s)$$

Space complexity: Equivalent to how large can the fringe get. $S(n) = O(n^s)$

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality: BFS is optimal as long as the costs of all edges are equal.

Uniform Cost Search:

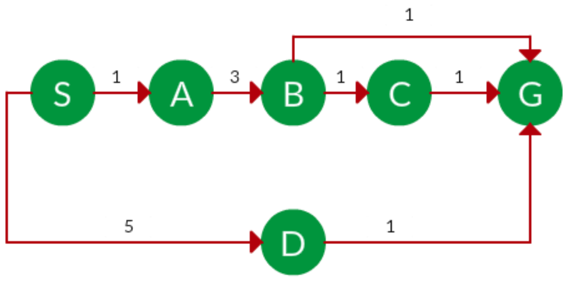
UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is the least.

Cost of a node is defined as:

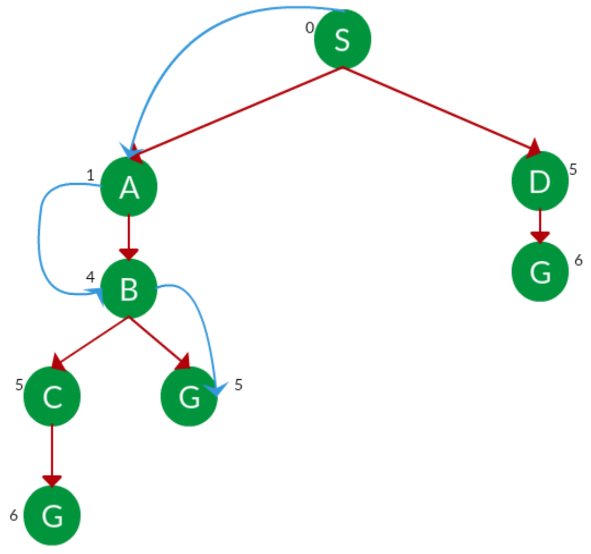
cost(node) = cumulative cost of all nodes from root

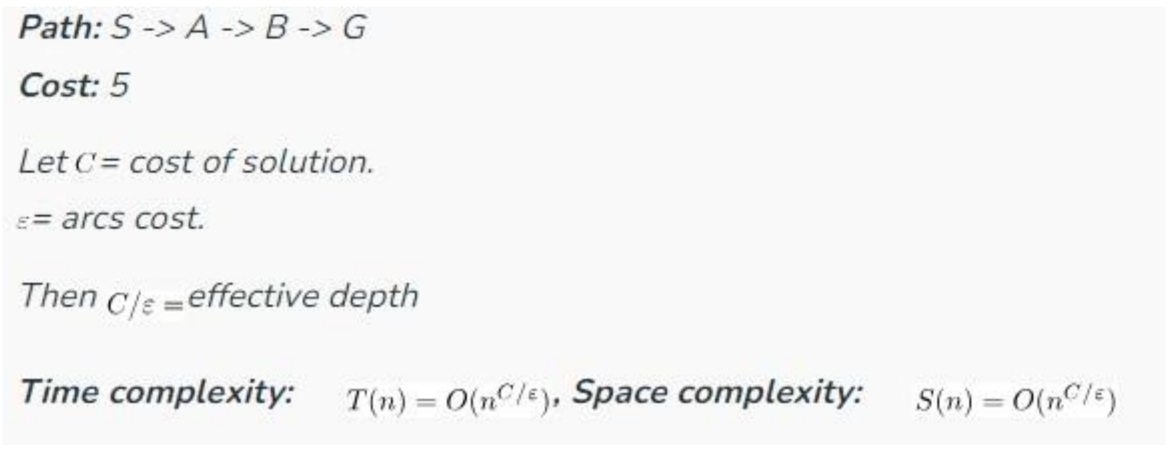
cost(root) = 0

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



Solution. The equivalent search tree for the above graph is as follows. The cost of each node is the cumulative cost of reaching that node from the root. Based on the UCS strategy, the path with the least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower-cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.





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

***Cost:** 5*

Let C = cost of solution.

ε = arcs cost.

Then C/ε = effective depth

Time complexity: $T(n) = O(n^{C/\varepsilon})$, ***Space complexity:*** $S(n) = O(n^{C/\varepsilon})$

Advantages:

- UCS is complete only if states are finite and there should be no loop with zero weight.
- UCS is optimal only if there is no negative cost.

Disadvantages:

- Explores options in every "direction".
- No information on goal location.

2.4 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. A* 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 example – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.) Different heuristics are used in different informed algorithms discussed below.

Greedy Search:

In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).

Heuristic: A heuristic h is defined as-

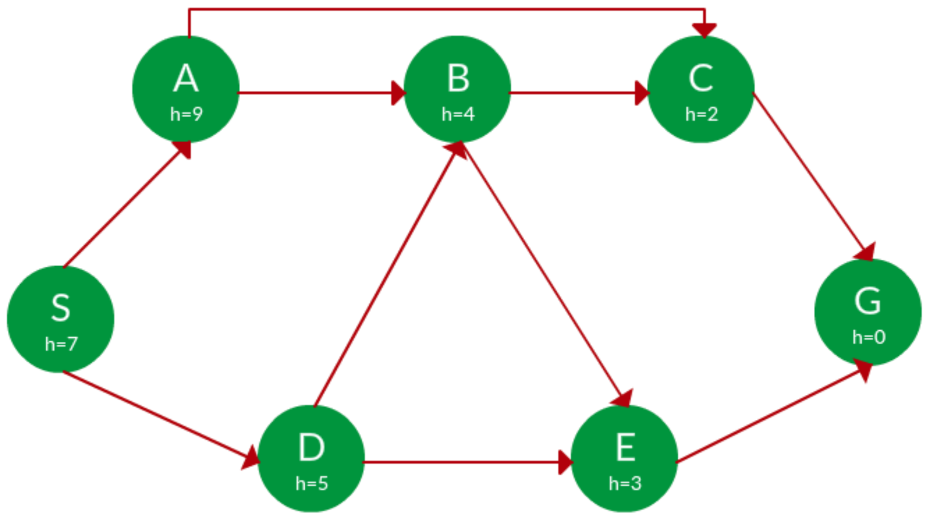
h(x) = Estimate of distance of node x from the goal node.

Lower the value of h(x), closer is the node from the goal.

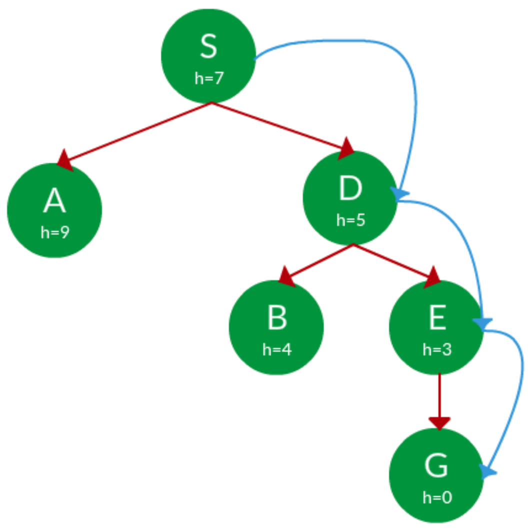
Strategy: Expand the node closest to the goal state, i.e. expand the node with a lower h value.

Example:

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



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



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

Advantage: Works well with informed search problems, with fewer steps to reach a goal.

Disadvantage: Can turn into unguided DFS in the worst case.

A* Tree Search:

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

Heuristic: The following points should be noted wrt heuristics in A* Search .

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

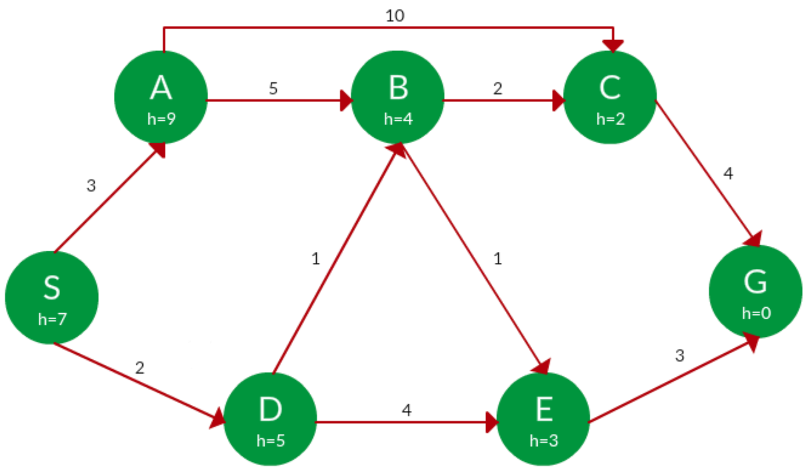
- Here, h(x) is called the **forward cost** and is an estimate of the distance of the current node from the goal node.
- And, g(x) is called the **backward cost** and is the cumulative cost of a node from the root node.
- A* search is optimal only when for all nodes, the forward cost for a node h(x)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

Admissibility: $0 \leqslant h(x) \leqslant h^*(x)$

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

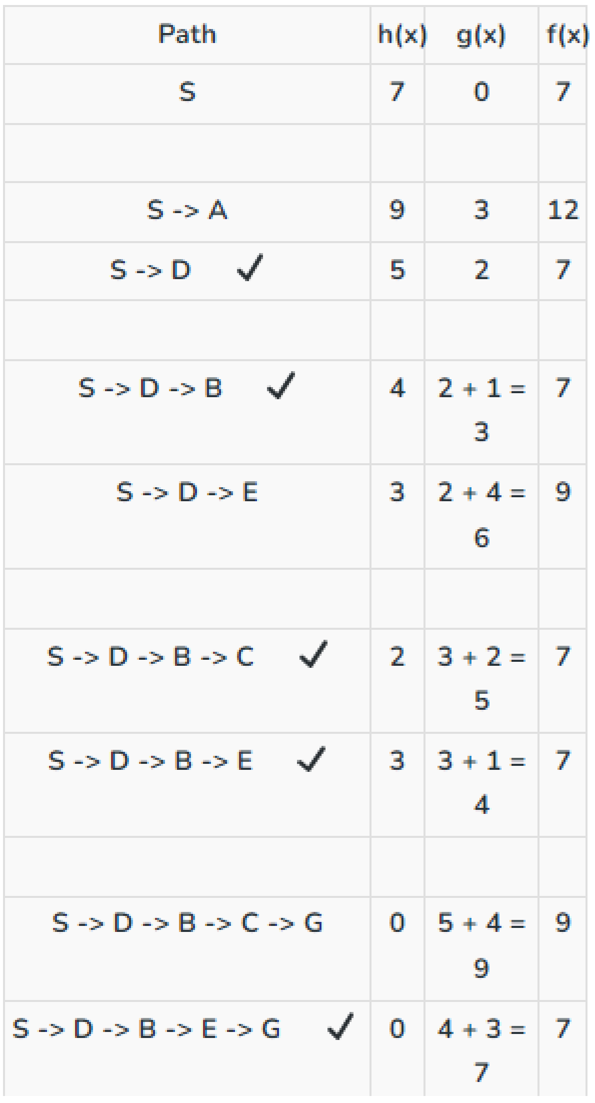
Example:

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



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

Note that in the fourth set of iterations, we get two paths with equal summed cost f(x), so we expand them both in the next set. The path with a lower cost on further expansion is the chosen path.



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

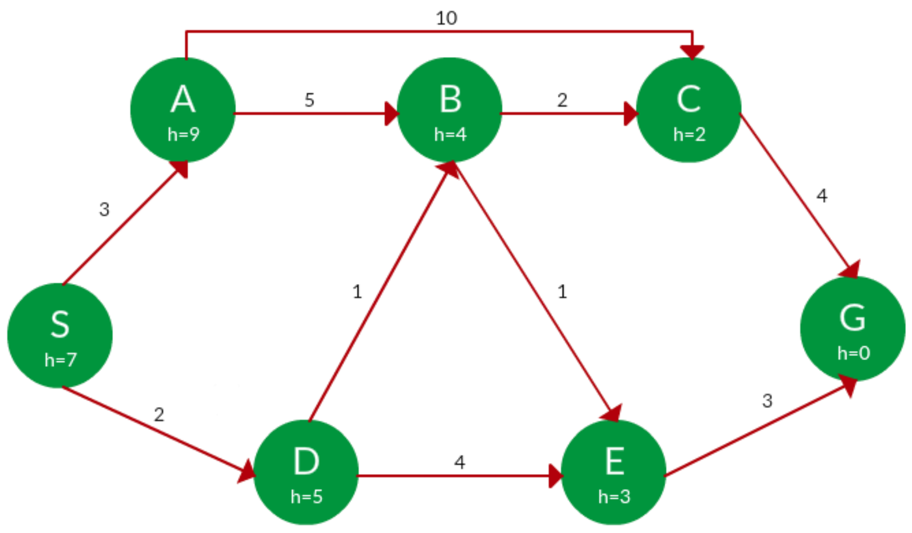
Path: S -> D -> B -> E -> G
Cost: 7

A* Graph Search:

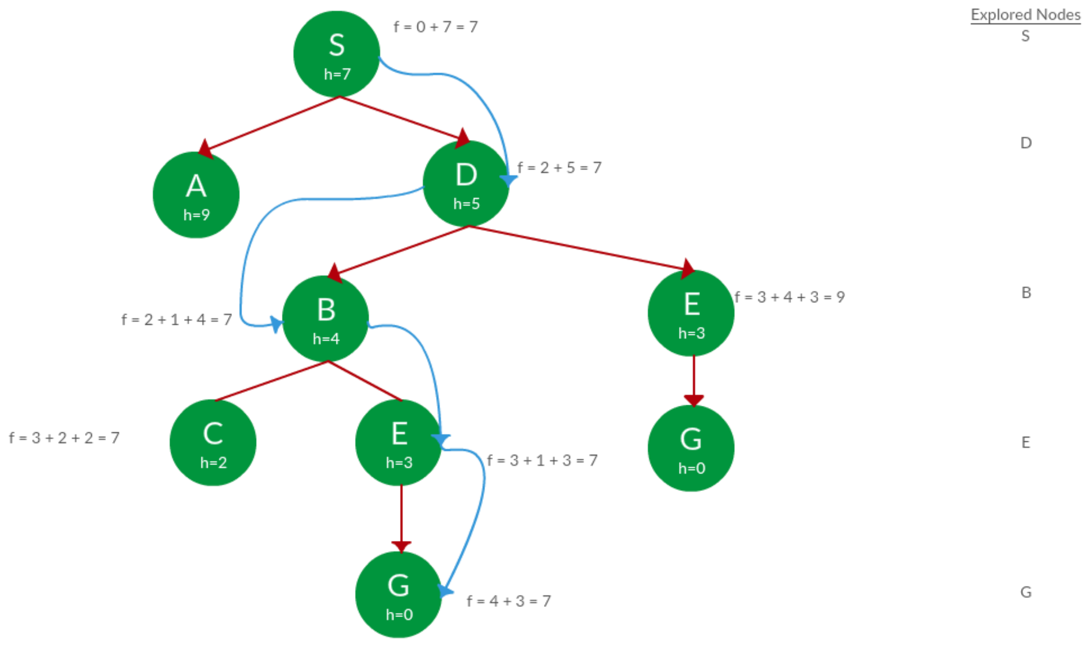
- A* tree search works well, except that it takes time re-exploring the branches it has already explored. In other words, if the same node has expanded twice in different branches of the search tree, A* search might explore both of those branches, thus wasting time
- A* Graph Search, or simply Graph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

Consistency: $h(A) - h(B) \leqslant g(A \to B)$

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



the **Solution.** We solve this question pretty much the same way we solved last question, but in this case, we keep a track of nodes explored so that we don't re-explore them.



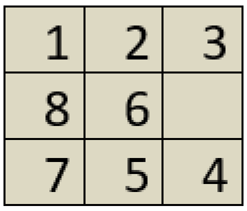
Path: S -> D -> B -> E -> G
Cost: 7

2.4 Heuristic Functions

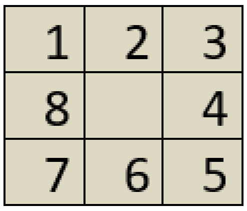
As we have already seen that an informed search make use of heuristic functions in order to reach the goal node in a more prominent way. Therefore, there are several pathways in a search tree to reach the goal node from the current node. The selection of a good heuristic function matters certainly. ***A good heuristic function is determined by its efficiency. More is the information about the problem, more is the processing time."***

Some toy problems, such as 8-puzzle, 8-queen, tic-tac-toe, etc., can be solved more efficiently with the help of a heuristic function. Let's see how:

Consider the following 8-puzzle problem where we have a start state and a goal state. Our task is to slide the tiles of the current/start state and place it in an order followed in the goal state. There can be four moves either **left, right, up, or down**. There can be several ways to convert the current/start state to the goal state, but, we can use a heuristic function h(n) to solve the problem more efficiently.



Start State



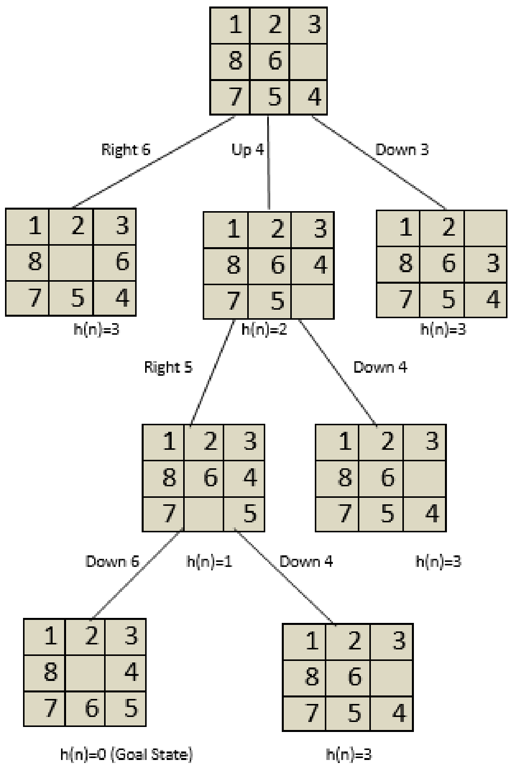
Goal State

A heuristic function for the 8-puzzle problem is defined below:

h(n)=Number of tiles out of position.

So, there is total of three tiles out of position i.e., 6,5 and 4. Do not count the empty tile present in the goal state). i.e. h(n)=3. Now, we require to minimize the value of **h(n) =0.**

We can construct a state-space tree to minimize the h(n) value to 0, as shown below:



It is seen from the above state space tree that the goal state is minimized from h(n)=3 to h(n)=0. However, we can create and use several heuristic functions as per the requirement. It is also clear from the above example that a heuristic function h(n) can be defined as the information required to solve a given problem more efficiently. The information can be related to the **nature of the state, cost of transforming from one state to another, goal node characteristics,** etc., which is expressed as a heuristic function.

Properties of a Heuristic search Algorithm

Use of heuristic function in a heuristic search algorithm leads to following properties of aheuristic search algorithm:

- **Admissible Condition:** An algorithm is said to be admissible, if it returns an optimal
  solution.
- **Completeness:** An algorithm is said to be complete, if it terminates with a solution (if the solution exists).
- **Dominance Property:** If there are two admissible heuristic algorithms **A1** and **A2** having **h1** and **h2** heuristic functions, then **A1** is said to dominate **A2** if **h1** is better than **h2** for all the values of node **n.**
- **Optimality Property:** If an algorithm is **complete, admissible**, and **dominating** other algorithms, it will be the best one and will definitely give an optimal solution.