**0EX ERCISE-10**

**AIM: Implementation of searching techniques in AI.**

**Problem**:

## What is a Search Algorithm in AI?

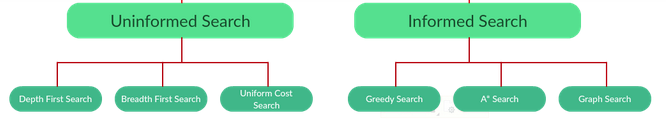
**Search** in AI is the process of navigating from a **starting state** to a **goal state** by **transitioning** through **intermediate states**.

Almost any AI problem can be defined in these terms.

* **State** — A potential outcome of a problem
* **Transition** — The act of moving between states.
* **Starting State —** Where to start searching from.
* **Intermediate State**- The states between the starting state and the goal state that we need to transition to.
* **Goal State —** The state to stop searching.
* **Search Space** — A collection of states.

## Types of Search Algorithms

We can divide search algorithms in artificial intelligence into **uninformed** (**Blind** search) and **informed** (**Heuristic** search) algorithms based on the search issues



**Uninformed Search Algorithms:**

The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**. These algorithms can only generate the successors and differentiate between the goal state and non goal state.

**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.***

**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

**a)** **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.

**Source Code:**

graph1={

'A':['B','S'],

'B':['A'],

'C':['D','E','F','S'],

'D':['C'],

'E':['C','H'],

'F':['C','G'],

'G':['F','S'],

'H':['E','G'],

'S':['A','C','G']

}

def dfs(graph,node,visited):

if node not in visited:

visited.append(node)

for k in graph[node]:

dfs(graph,k,visited)

return visited

visited=dfs(graph1,'A',[])

print(visited)

**Output:**

['A', 'B', 'S', 'C', 'D', 'E', 'H', 'G', 'F']

**b) 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.

**Source Code:**

graph={

'A':['B','C'],

'B':['D','E'],

'C':['F','G','H'],

'D':[],

'E':[],

'F':[],

'G':['J'],

'H':['I'],

'I':['J'],

'J':[]

}

visited=[]

queue=[]

def bfs(visited,graph,node):

visited.append(node)

queue.append(node)

while(queue):

s=queue.pop(0)

print(s,end=" ")

for neighbour in graph[s]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

bfs(visited,graph,'A')

**Output:**

A B C D E F G H J I