# AIR UNIVERSITY

## Department of Electrical and Computer Engineering

## Lab #2: Uniformed Vs Informed Search

### Lab Instructor: Muhammad Awais

## Objective

The objective of this lab is to explore and compare different search strategies in artificial intelligence, particularly focusing on the differences between informed (heuristic-based) and uninformed (blind) search algorithms, as well as the distinction between finding any path versus finding the optimal path.

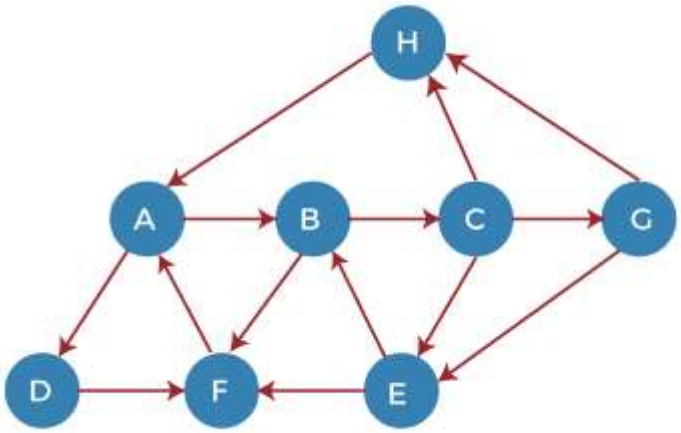
1. Implementing and analyzing uninformed search algorithms such as breadth-first search (BFS) and depth-first search (DFS).
2. Implementing and analyzing informed search algorithms such as A* search.

## Breath-First-Search

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

```
graph = {
    'A' : ['B', 'D'],
    'B' : ['C', 'F'],
    'C' : ['H', 'G', 'E'],
    'D' : ['F'],
    'E' : ['B', 'F'],
    'F' : ['A'],
    'G' : ['E', 'H'],
    'H' : ['A']
}
```

```
from IPython import display
display.Image("BFS_DFS.PNG")
```



```
def BFS(graph, Start, Goal):
  queue = [Start]
  visited = []
  if Goal == Start:
    return Start
  else:
    while True:
      temp = queue.pop(0)
      if temp not in visited:
        visited.append(temp)
      if temp == Goal:
        return visited
      else:
        for siblings in graph[temp]:
          if siblings not in visited:
            queue.append(siblings)


def get_key(tree, val):
  for key, value in tree.items():
    if val in value:
      return key


def short(graph, Start, Goal):
  shortest = [Goal]
  temp = Goal
  while Start != temp:
    temp = get_key(graph, temp)
    shortest.append(temp)
  shortest.reverse()
  return shortest


Visited = search(graph, 'A', 'E')
print(f"\nVisited : {Visited}")
shortest = short(graph, 'A', 'E')
print(f"\nShortest: {shortest[::1]}")
```

```
Visited : ['A', 'B', 'D', 'C', 'F', 'H', 'G', 'E']

Shortest: ['A', 'B', 'C', 'E']
```

## Depth-First-Search

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

```
def DFS(graphs, start, goal, visited=None):
  stack = [start]
  visited = [] if visited is None else visited

  if goal == start:
      return [start]
  else:
      while stack:
          current = stack.pop()
          if current not in visited:
              visited.append(current)
          if current == goal:
              return visited
          else:
              for neighbor in reversed(graphs[current]):
                  if neighbor not in visited:
                      stack.append(neighbor)


Visited_DFS = DFS(graph, 'A', 'D')
print(f"\nVisited with DFS: {Visited_DFS}")
shortest = short(graph, 'A', 'E')
print(f"\nShortest: {shortest}")
```

```
Visited with DFS: ['A', 'B', 'C', 'H', 'G', 'E', 'F', 'D']
```

```
Shortest: ['A', 'B', 'C', 'E']
```

## Conclusion

In this lab, we explored and compared two fundamental search strategies: breadth-first search (BFS) and depth-first search (DFS). BFS systematically explores nodes level by level, guaranteeing the shortest path. DFS explores as deeply as possible before backtracking. We implemented both algorithms in Python and modified BFS to find the shortest path. BFS is memory-intensive but optimal for shortest paths, while DFS is memory-efficient but may not find the shortest path. This lab provides insights into the trade-offs between memory usage and optimality in search algorithms.