Lab 1: Implement Uninformed Searching Strategies

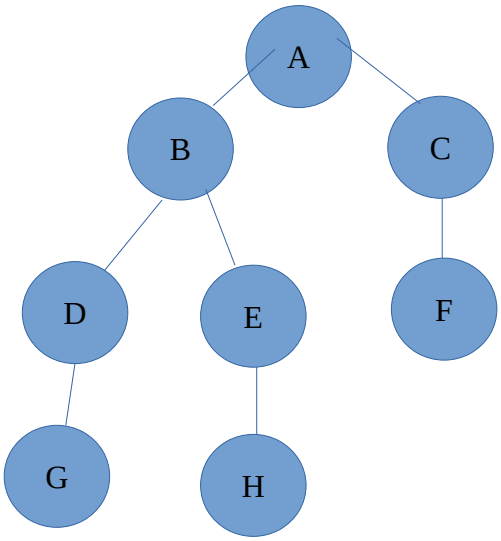
An uninformed (a.k.a. blind, brute-force) search algorithm generates the search tree without using any domain specific knowledge. Uninformed search algorithms are also called blind search algorithms. The search algorithm produces the search tree without using any domain knowledge, which is a brute force in nature. They don't have any background information on how to approach the goal or whatsoever. But these are the basics of search algorithms in AI. In this lab we will implement the following searching strategies in the graphs shown in figure below.

Depth First Search (DFS):

It is a search algorithm where the search tree will be traversed from the root node. It will be traversing, searching for a key at the leaf of a particular branch. If the key is not found the searching retraces its steps back to the point from where the other branch was left unexplored and the same procedure is repeated for that other branch.

Breadth-First Search(BFS):

It is another search algorithm in AI which traverses breadth-wise to search the goal in a tree. It begins searching from the root node and expands the successor node. It further expands along breadth-wise and traverses those nodes rather than searching depth-wise.



Source Code(DFS):

```
#we represent the graph in adjacent list representation
G  = {
     'A' : ['B', 'C'],
     'B' : ['A', 'D', 'E'],
     'C' : ['A', 'F'],
     'D' : ['B', 'G'],
     'E' : ['B', 'D', 'G', 'H'],
     'F' : ['C'],
     'G' : ['D'],
     'H' : ['E']
     }

start = 'A'
goal = 'H'

def DFS(G, start, goal):
     #we maintain a stack which stores a tuple
     #containig vertex and path to that vertex
     stack = [(start, [start])]
     #we maintain a set of visited vertices
     visited = set()
     #we repeat the following until the stack is empty
     while stack:
          (poppedVertex, path) = stack.pop()
          if poppedVertex not in visited:
               #if the popped vertex is the goal we return the path
               if poppedVertex == goal:
                    return path
               visited.add(poppedVertex)
               for vertex in G[poppedVertex]:
                    if vertex not in visited and vertex not in stack:
                         stack.append((vertex, path + [vertex]))

#Driver Code
path = DFS(G, start, goal)
print(path)
```

Output

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (A
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/Pukar Karki/Desktop/DFS.py ================
['A', 'B', 'E', 'H']
```

Source Code(BFS):

```
#we represent the graph in adjacent list representation
G  = {
     'A' : ['B', 'C'],
     'B' : ['A', 'D', 'E'],
     'C' : ['A', 'F'],
     'D' : ['B', 'G'],
     'E' : ['B', 'D', 'G', 'H'],
     'F' : ['C'],
     'G' : ['D'],
     'H' : ['E']
     }

start = 'A'
goal = 'H'

def BFS(G, start, goal):
     #we maintain a queue which stores a tuple
     #containig vertex and path to that vertex
     queue = [(start, [start])]
     #we maintain a set of visited vertices
     visited = set()
     #we repeat the following until the queue is empty
     while queue:
          (poppedVertex, path) = queue.pop(0)
          if poppedVertex not in visited:
               #if the popped vertex is the goal we return the path
               if poppedVertex == goal:
                    return path
               visited.add(poppedVertex)
               for vertex in G[poppedVertex]:
                    if vertex not in visited and vertex not in queue:
                         queue.append((vertex, path + [vertex]))
#Driver Code
path = BFS(G, start, goal)
print(path)
```

Output

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/Pukar Karki/Desktop/BFS.py ================
['A', 'B', 'E', 'H']
```