SD21063 TEAN JIN HE Lab Report 1

April 12, 2023

1 BSD2513 ARTIFICIAL INTELLIGENCE

1.1 LAB REPORT 1

NAME : TEAN JIN HE

MATRIC ID : SD21063

SECTION : 02G

Questions 1 : General Knowledge Discuss types of AI based on capabilities on your understanding.

[14]:
```
#Based on this classification, the following are the three types of AI:

#1. Artificial Narrow Intelligence (ANI)
#This type of AI can perform specific tasks that are usually done by humans,
 ↪such as image recognition, speech recognition, or recommendation systems.
 ↪However, it cannot perform tasks outside its domain or learn new skills.
 ↪Examples of narrow AI are Siri, Alexa and Google Assistant.

#2. Artificial General Intelligence (AGI)
#This type of AI can perform any intellectual task that a human can do, such as
 ↪reasoning, problem-solving, planning, learning, etc. It can also understand
 ↪and communicate in natural language. However, this type of AI does not exist
 ↪yet and is still a goal for many researchers.

#3. Artificial Super Intelligence (ASI)
#This type of AI can surpass human intelligence and capabilities in every
 ↪aspect, such as creativity, wisdom, emotion and others. It can also
 ↪self-improve and create new knowledge. This type of AI is also hypothetical
 ↪and may pose an existential threat to humanity if not aligned with human
 ↪values
```

Question 2 Python: Search Algorithms Consider the following graphs. Assume we always choose the letter closest to the beginning of the alphabet first. In what order will the nodes be visited using:

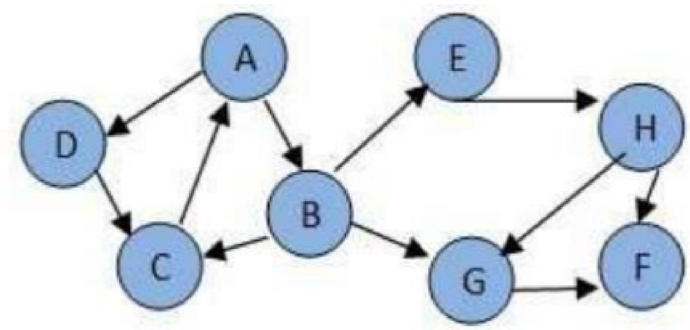
a. breadth first search (BFS) and

b. depth first search (DFS).

Discuss the level and the process path clearly.

[3]:
```
from IPython.display import Image
Image("C:/Users/user/OneDrive/Desktop/AI lecturer learning/lab 1 photo.png")
```

[3]:



a. breadth first search (BFS)

[5]:
```
# Breadth First Search (BFS)
graph = {
    'A' : ['B','D'],
    'B' : ['C','E','G'],
    'C' : ['A'],
    'D' : ['C'],
    'E' : ['H'],
    'F' : [],
    'G' : ['F'],
    'H' : ['F','G']
}

visited = [] #List to keep track of visited nodes.
queue = [] #Initialize a queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        b = queue.pop(0)
        print(b, end = " ")
```

```
        for neighbour in graph[b]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
print("Breadth First Search :")
bfs(visited, graph, 'A')
```

```
Breadth First Search :
A B D C E G H F
```

[ ]:
```
#BFS is an algorithm for traversing or searching tree or graph data structures.
 ↪The algorithm starts at a given node (in your case, 'A') and explores all
 ↪the neighboring nodes at the current depth level before moving on to the
 ↪nodes at the next depth level.

#The code defines a graph as a dictionary of nodes and their adjacent nodes.
 ↪Then it defines a function bfs that takes three parameters: visited, graph,
 ↪and node. The visited parameter is a list that keeps track of the nodes that
 ↪have been visited. The function does the following steps:

#- Append the node to the visited list and enqueue it to the queue.
#- While the queue is not empty, dequeue a node from the queue and print it.
#- For each neighbor of the node that is not in the visited list, append it to
 ↪the visited list and enqueue it to the queue.

#The code then calls the bfs function with 'A' as the node parameter and an
 ↪empty list as the visited parameter. The output of the code is:

#Breadth First Search : A B D C E G H F

#This means that the code has traversed the graph in this order: A -> B -> D ->
 ↪C -> E -> G -> H -> F. This is one possible BFS traversal of the graph.
```

b. depth first search (DFS)

[13]:
```
#Depth First Search (DFS)
graph = {
    'A' : ['B','D'],
    'B' : ['C','E','G'],
    'C' : ['A'],
    'D' : ['C'],
    'E' : ['H'],
    'F' : [],
    'G' : ['F'],
    'H' : ['F','G']
}
```

```python
def dfs(graph, node, visited): #function for dfs
    if node not in visited:
        visited.append(node)
        for neighbour in graph[node]:
            dfs(graph, neighbour, visited)
    return visited

#Driver Code
visited = dfs(graph, 'A', [])
print("Depth-First Search :")
print(visited)
```

```
Depth-First Search :
['A', 'B', 'C', 'E', 'H', 'F', 'G', 'D']
```

[ ]:
```python
#DFS is an algorithm for traversing or searching tree or graph data structures.
 ↪The algorithm starts at a given node (in your case, 'A') and explores as far
 ↪as possible along each branch before backtracking.

#The code defines a graph as a dictionary of nodes and their adjacent nodes.
 ↪Then it defines a function dfs that takes three parameters: graph, node, and
 ↪visited. The visited parameter is a set that keeps track of the nodes that
 ↪have been visited. The function does the following steps:

#- If the node is not in the visited set, add it to the visited set and print
 ↪it.
#- For each neighbor of the node that is not in the visited set, recursively
 ↪call the dfs function with that neighbor as the node parameter.
#- Return the visited set.

#Next,the code then calls the dfs function with 'A' as the node parameter and
 ↪an empty set as the visited parameter. The output of the code is:

#Depth-First Search : ['A', 'B', 'C', 'D', 'E', 'H', 'F', 'G']

#This means that the code has traversed the graph in this order: A -> B -> C ->
 ↪D -> E -> H -> F -> G. This is one possible DFS traversal of the graph.
```