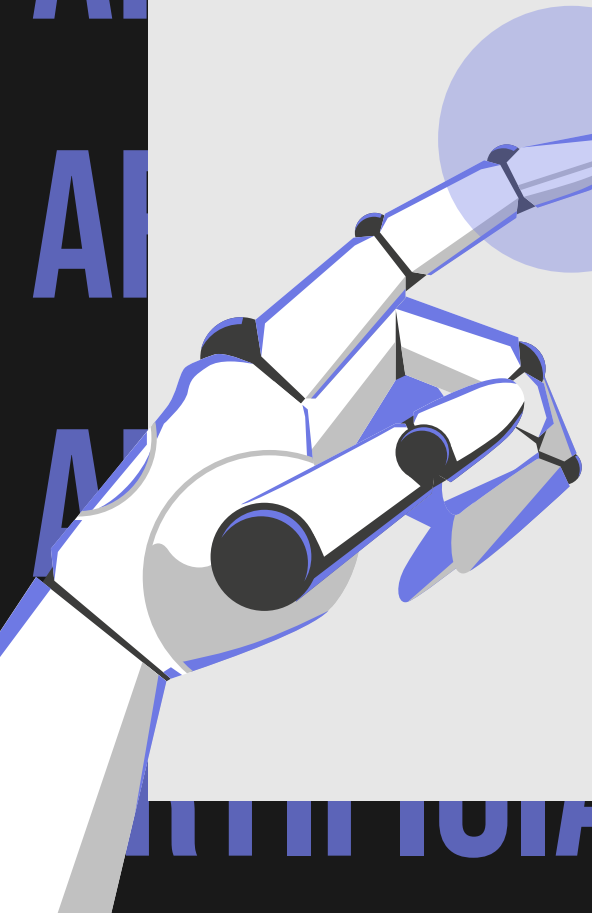


ARTIFICIAL INTELLIGENCE (AI)

PRAKTIKUM 3

Dosen Pengampu : Renovita Edelani S.ST., M.Tr.Kom

oleh : Dukhaan Kamimpangan
2 D4 IT A
3122600003

TUGAS PRAKTIKUM 1 BFS SOURCE CODE

ARTIFI
INTELI
(AI)

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

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

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

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, 'A')    # function calling
```

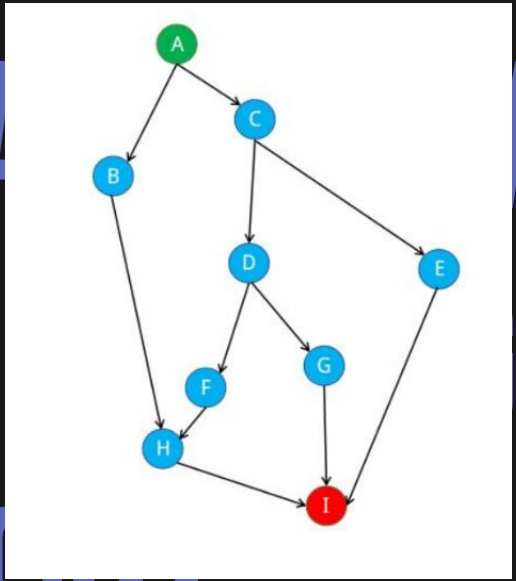
TUGAS PRAKTIKUM 1 DFS SOURCE CODE

ARTIFI
INTELI
(AI)

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

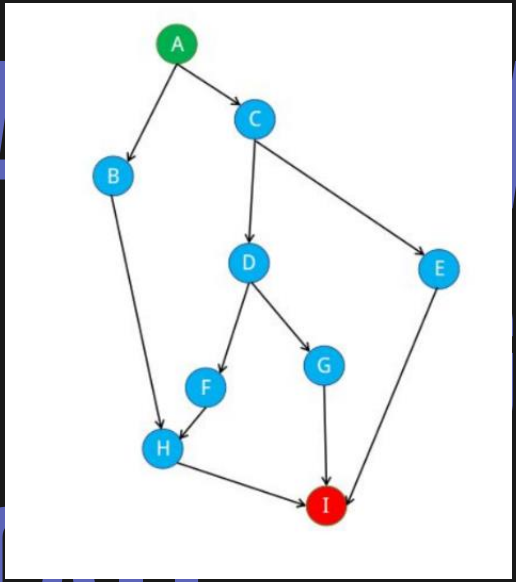
```
visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
          dfs(visited, graph, neighbour)


# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, 'A')
```

TUGAS PRAKTIKUM 1 BFS TRAVERSAL



Pop	Queue			
A	B	C		
B	C	H		
C	H	D	E	
H	D	E	I	
D	E	I	F	G
E	I	F	G	
I	F	G		
F	G			
G				

TUGAS PRAKTIKUM 1 DFS TRAVERSAL



POP	QUEUE		
A	B	C	
B	H	C	
H	I	C	
I	C		
C	D	E	
D	F	G	E
F	G	E	
G	E		
E			

TUGAS PRAKTIKUM 2 BFS SOURCE CODE

ARTIFI
INTELI
(AI)

```
graph = {
  '1' : ['2','3'],
  '2' : ['5','4'],
  '3' : ['4','6'],
  '4' : ['5','7','8','6'],
  '5' : ['7'],
  '6' : ['8','9'],
  '7' : ['9'],
  '8' : ['9'],
  '9' : []
}
```

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

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

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '1')    # function calling
```

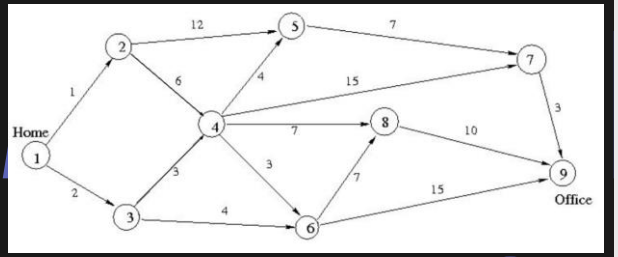
TUGAS PRAKTIKUM 2 DFS SOURCE CODE

ARTIFI
INTELI
(AI)

```
graph = {
  '1' : ['2','3'],
  '2' : ['5','4'],
  '3' : ['4','6'],
  '4' : ['5','7','8','6'],
  '5' : ['7'],
  '6' : ['8','9'],
  '7' : ['9'],
  '8' : ['9'],
  '9' : []
}
```

```
visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
          dfs(visited, graph, neighbour)


# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '1')
```

TUGAS PRAKTIKUM 2 BFS TRAVERSAL

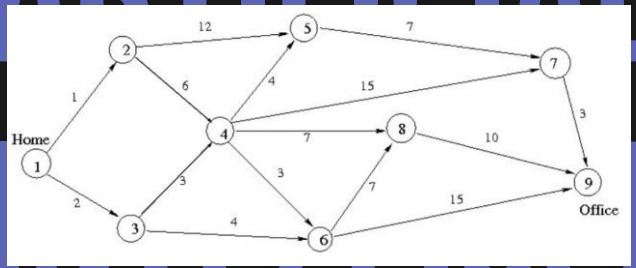


INTELLIGE
(AI)

Pop	Queue		
1	2	3	
2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8
6	7	8	9
7	8	9	
8	9		
9			

TUGAS PRAKTIKUM 2 DFS TRAVERSAL

ARTIFICIAL INTELLIGENCE (AI)



Pop	Queue		
1	2	3	
2	5	4	3
5	7	4	3
7	9	4	3
9	4	3	
4	8	6	3
8	6	3	
6	3		
3			

TUGAS PRAKTIKUM 3 BFS SOURCE CODE

ARTIFI
INTELI
(AI)

```
graph = {
  '0' : ['3','4'],
  '1' : ['0','6'],
  '2' : ['5','6'],
  '3' : ['1','7'],
  '4' : ['6'],
  '5' : ['6','7'],
  '6' : ['4','2'],
  '7' : ['5'],
}
```

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

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

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '0')    # function calling
```

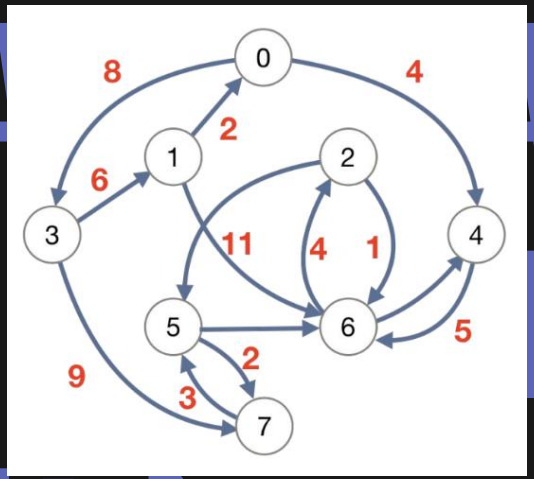
TUGAS PRAKTIKUM 3 DFS SOURCE CODE

ARTIFI
INTELI
(AI)

```
graph = {
  '0' : ['3','4'],
  '1' : ['0','6'],
  '2' : ['5','6'],
  '3' : ['1','7'],
  '4' : ['6'],
  '5' : ['6','7'],
  '6' : ['4','2'],
  '7' : ['5'],
}
```

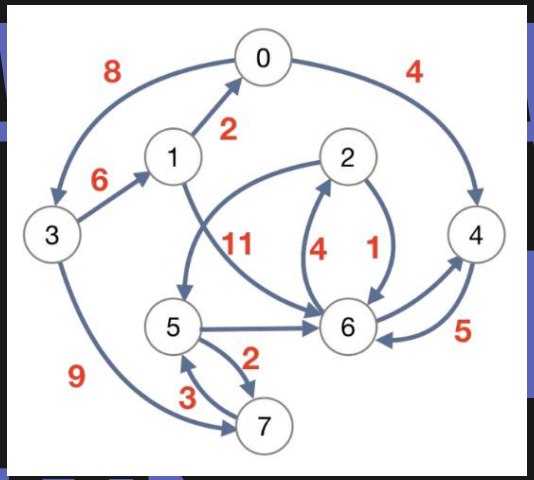
```
visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
          dfs(visited, graph, neighbour)


# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '0')
```

TUGAS PRAKTIKUM 3 BFS TRAVERSAL



Pop	Queue		
0	3	4	
3	4	1	7
4	1	7	6
1	7	6	
7	6	5	
6	5	2	
5	2		
2			

TUGAS PRAKTIKUM 3 DFS TRAVERSAL



Pop	Queue		
0	3	4	
3	1	7	4
1	6	0	4
6	4	2	
4	2	6	
2	5	6	
5	7		
7			

KESIMPULAN

1. Depth-First Search (DFS):

Prinsip: DFS mengunjungi node sejauh mungkin sebelum mundur dan melanjutkan ke node berikutnya.
Implementasi: Fungsi dfs menggunakan rekursi untuk mengunjungi node secara rekursif dan menambahkannya ke dalam set visited untuk menghindari pengulangan.
Urutan output: Output DFS adalah urutan node yang diunjungi, dimulai dari node awal 'A'.

2. Breadth-First Search (BFS):

Prinsip: BFS mengunjungi semua tetangga dari suatu node sebelum melanjutkan ke node tetangga selanjutnya.
Implementasi: Fungsi bfs menggunakan antrian (queue) untuk mengelola urutan node yang akan dikunjungi. Node yang dikunjungi ditambahkan ke dalam list visited.
Urutan output: Output BFS adalah urutan node yang diunjungi, dimulai dari node awal 'A'.

DFS dan BFS adalah dua metode pencarian pada graf. DFS menggunakan pendekatan rekursif dan mengeksplorasi sejauh mungkin sebelum kembali, sementara BFS mengeksplorasi secara lebih "lebar" dengan melihat semua tetangga sebelum memeriksa tetangga yang lebih jauh.

Keduanya digunakan dalam berbagai konteks seperti traversal graf, mencari jalur terpendek, dan sebagainya, tergantung pada kebutuhan aplikasi.

Output DFS dan BFS dapat bervariasi tergantung pada struktur graf.

TERIMA KASIH

ARTI
CIAL
INTE
IGEN
(AI)

/(AI)/(AI)/