



Gardencity University

EMPHASIS IN LIFE

Name: Lalitha.T

Roll No:25MCAR149

AI Quantitative assignment Output Screenshot

01.DFS

The screenshot shows a Google Colab notebook titled "DFS_Hanoi.py". The code implements Depth-First Search (DFS) for the Tower of Hanoi problem. It defines a function `dfs_hanoi` that takes the number of disks `n`, the source rod, the auxiliary rod, and the target rod. The function uses a stack to keep track of moves, printing each move as it is made. The code runs successfully, producing the following output:

```
# Tower of Hanoi using DFS (Depth-First Search)
from collections import deque

def dfs_hanoi(n, source, auxiliary, target):
    stack = deque()
    stack.append((n, source, auxiliary, target))

    while stack:
        n, source, auxiliary, target = stack.pop()

        if n == 1:
            print(f"Move disk 1 from {source} to {target}")
            continue

        stack.append((n-1, auxiliary, source, target))
        stack.append((1, source, auxiliary, target))
        stack.append((n-1, source, target, auxiliary))

    print("\nDFS - Tower of Hanoi Moves:")
    dfs_hanoi(3, 'A', 'B', 'C')
```

Output:

```
Move disk 1 from A to C
Move disk 1 from A to B
Move disk 1 from C to B
Move disk 1 from A to C
Move disk 1 from B to A
Move disk 1 from B to C
Move disk 1 from A to C
```

This screenshot shows the same Google Colab notebook and code as the previous one. The output is identical, displaying the step-by-step moves for the 3-disk Tower of Hanoi problem using Depth-First Search.

```
... DFS - Tower of Hanoi Moves:
Move disk 1 from A to C
Move disk 1 from A to B
Move disk 1 from C to B
Move disk 1 from A to C
Move disk 1 from B to A
Move disk 1 from B to C
Move disk 1 from A to C
```

02.BFS

The screenshot shows a Google Colab notebook titled "BFS_Hanoi.py". The code implements BFS to solve the Tower of Hanoi problem. It defines functions for checking if a state is a goal, getting moves from a state, and performing the search. The search starts from a state where all disks are on peg A. The code uses a deque for the queue and prints each move as it is performed.

```
# Tower of Hanoi using BFS (Breadth-First Search)
from collections import deque

def is_goal(state):
    return state == (((), (), (1,2,3)))

def get_moves(state):
    moves = []
    for i in range(3):
        if len(state[i]) == 0:
            continue
        disk = state[i][0]
        for j in range(3):
            if i != j:
                if len(state[j]) == 0 or disk < state[j][0]:
                    new_state = list(list(peg) for peg in state)
                    new_state[j].insert(0, new_state[i].pop(0))
                    moves.append((tuple(tuple(peg) for peg in new_state), f"Move disk {disk} from {chr(65+i)} to {chr(65+j)}"))
    return moves

def bfs_hanoi():
    start = (((), (), (1,2,3)))
    queue = deque([start])
    visited = set()

    while queue:
        state = queue.popleft()
        if is_goal(state):
            for move in path:
                print(move)
            return

        for new_state, move in get_moves(state):
            if new_state not in visited:
                queue.append(new_state)
                visited.add(state)

    print("\nBFS - Tower of Hanoi Moves:")
    bfs_hanoi()
```

The screenshot shows the execution results of the BFS algorithm. The terminal output displays the sequence of moves required to solve the Tower of Hanoi problem with 3 disks. The moves are printed in a specific order, showing the movement of each disk from one peg to another.

```
BFS - Tower of Hanoi Moves:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

03.A* Algorithm

colab.research.google.com/drive/1r4l3694WprLCBqRTI_u4hY6Lc5ZGTlqr#scrollTo=ZYqFJPBg4g99

A* Hanoi.py

```
# Tower of Hanoi using A* Search
import heapq

def heuristic(state):
    return 3 - len(state[2])

def is_goal(state):
    return state == (((), (), (1,2,3)))

def get_moves(state):
    moves = []
    for i in range(3):
        if not state[i]:
            continue
        disk = state[i][0]
        for j in range(3):
            if i != j:
                if not state[j] or disk < state[j][0]:
                    new_state = list(list(peg) for peg in state)
                    new_state[j].insert(0, new_state[i].pop(0))
                    moves.append((tuple(tuple(peg) for peg in new_state),
                                  f"Move disk {disk} from {chr(65+i)} to {chr(65+j)}"))
    return moves
```

Variables Terminal 9:51PM Python 3

colab.research.google.com/drive/1r4l3694WprLCBqRTI_u4hY6Lc5ZGTlqr#scrollTo=ZYqFJPBg4g99

A* Hanoi.py

```
visited.add(state)

if is_goal(state):
    for move in path:
        print(move)
    return

for new_state, move in get_moves(state):
    new_g = g + 1
    new_f = new_g + heuristic(new_state)
    heapq.heappush(heap, (new_f, new_g, new_state, path + [move]))

print("\nA* Search - Tower of Hanoi Moves:")
astar_hanoi()
```

```
...
A* Search - Tower of Hanoi Moves:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Variables Terminal 9:51PM Python 3