



Gardencity University

EMPHASIS IN LIFE

Name: Lalitha.T

Roll No:25MCAR149

AI Quantitative assignment Output Screenshot

01.DFS

```
[1] ✓ 0s # 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()

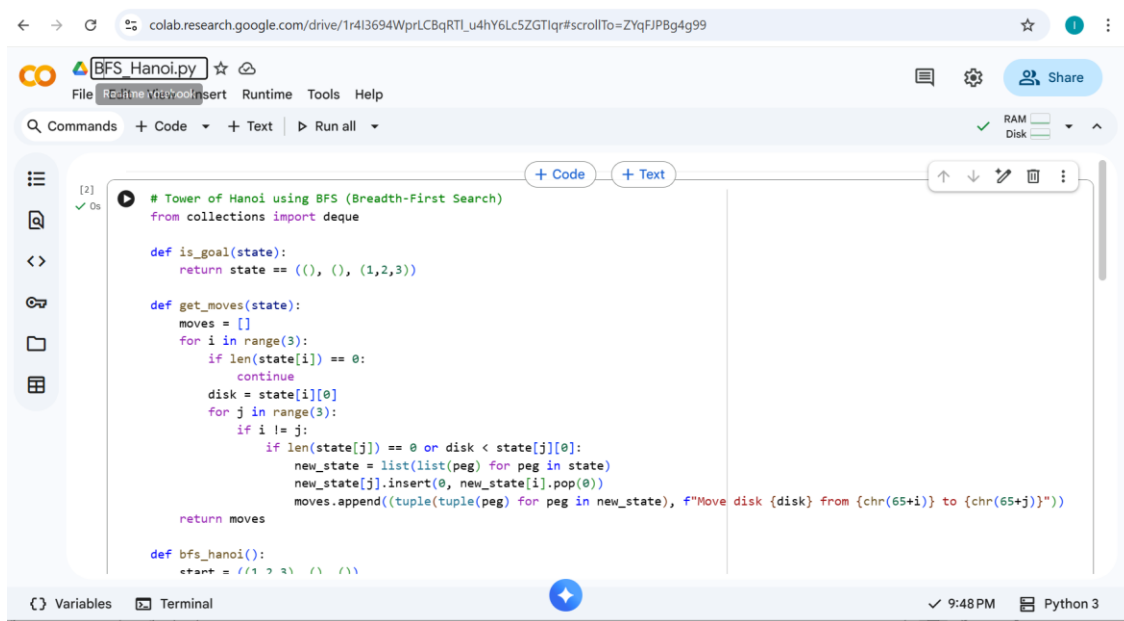
        if n == 1:
            print(f"Move disk 1 from {source} to {target}")
            continue

        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')
```

```
✓ 0s
...
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



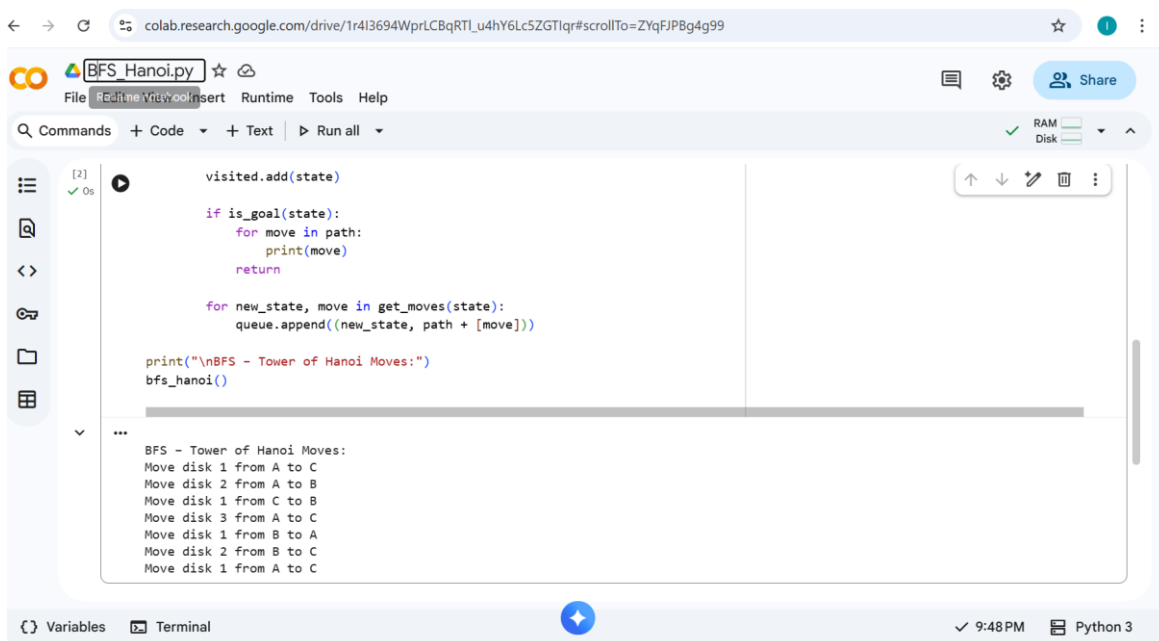
The image shows a Google Colab notebook titled "BFS_Hanoi.py". The code defines functions for checking the goal state, getting possible moves, and performing a BFS search. The goal state is defined as a tuple where the first two elements are empty and the third contains disks 1, 2, and 3 in order. Moves are generated by considering each disk and its possible target pegs, ensuring no larger disk is placed on top of a smaller one. The BFS search starts from the initial state and explores all possible paths until the goal state is reached.

```
# 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 = ((), (), ())
```



The image shows the same Google Colab notebook after execution. The code continues from the previous cell, adding a queue and a visited set to the BFS function. The queue is initialized with the start state. The BFS loop processes each state, checking for the goal and adding new states to the queue. The output shows the sequence of moves required to solve the Tower of Hanoi puzzle.

```
visited.add(state)

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

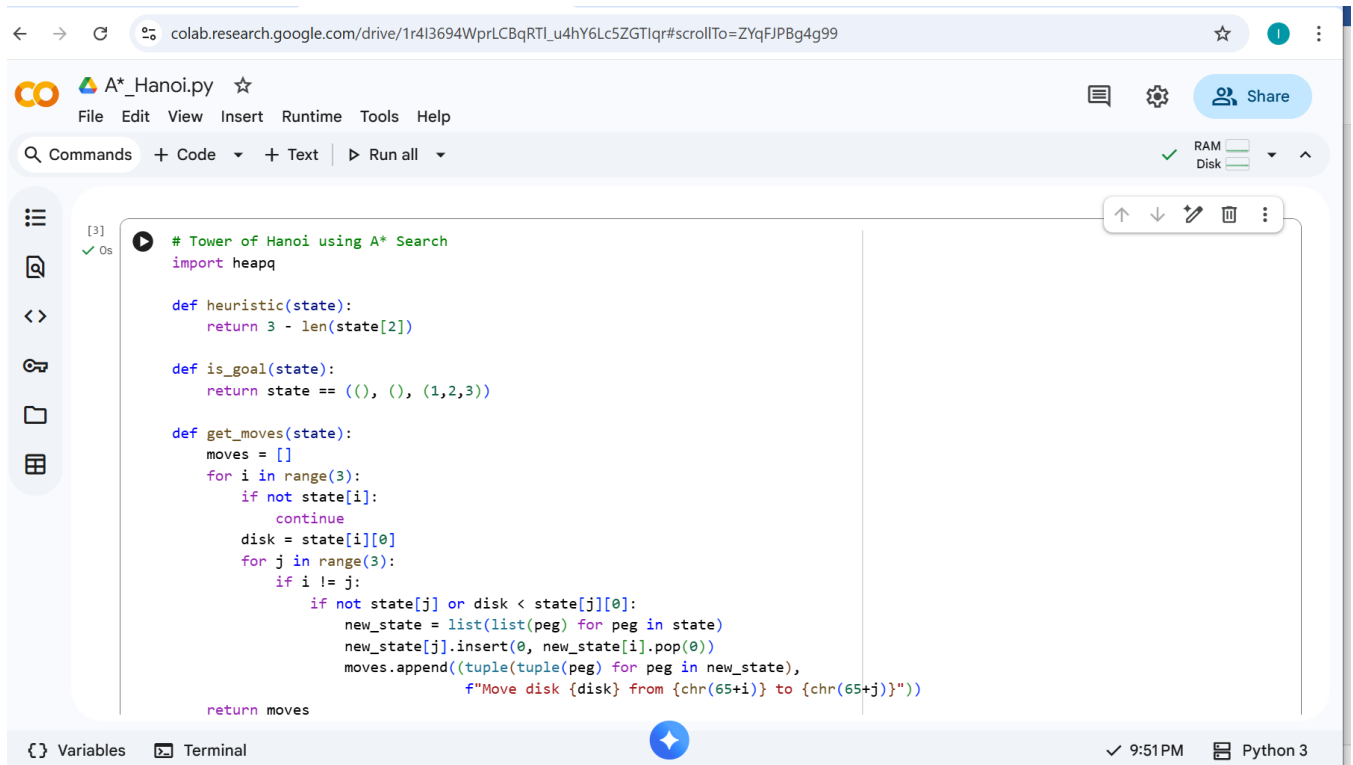
for new_state, move in get_moves(state):
    queue.append((new_state, path + [move]))

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

Output:

```
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



The image shows a Google Colab notebook titled "A*_Hanoi.py". The code implements an A* search algorithm for the Tower of Hanoi puzzle. It defines a heuristic function, a goal test, and a move generation function. The code is as follows:

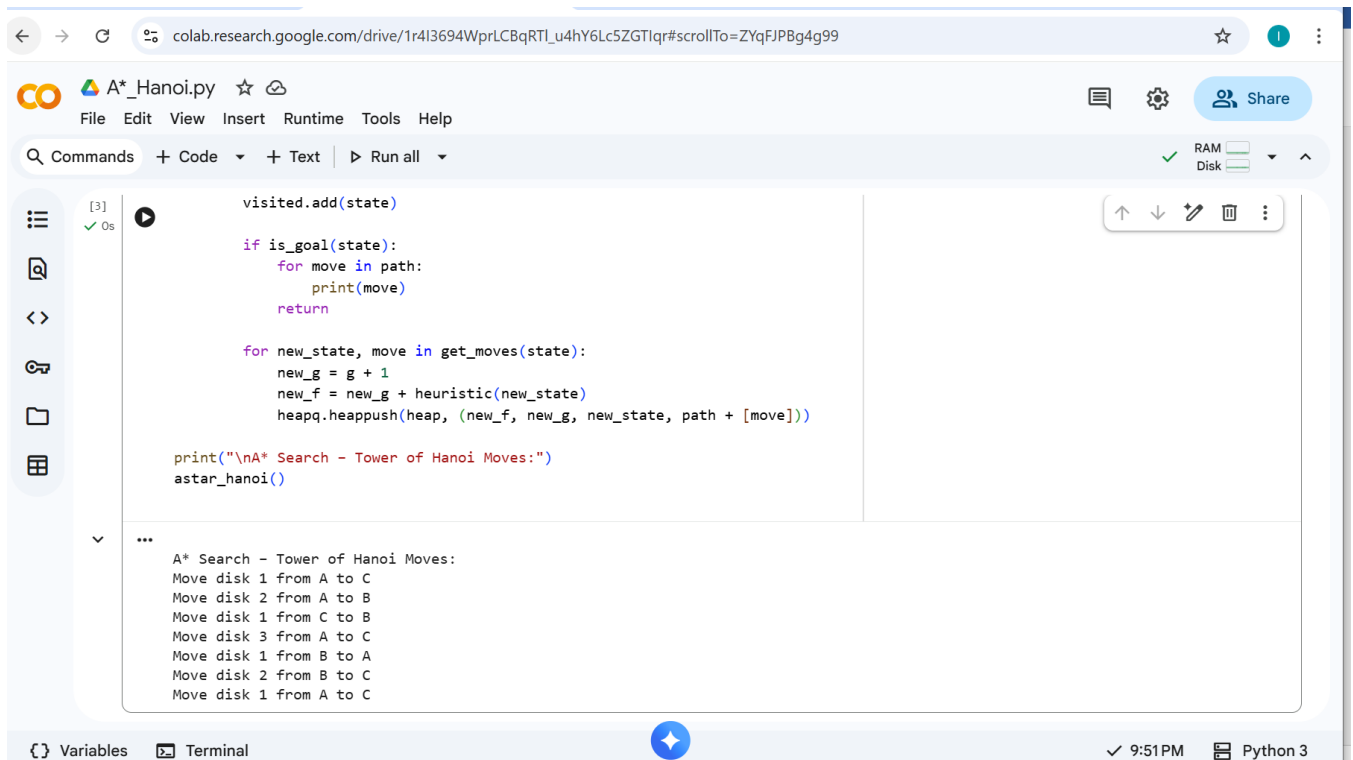
```
[3] ✓ 0s # Tower of Hanoi using A* Search
import heapq

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
```

The interface includes a left sidebar with navigation icons, a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", and a bottom status bar showing "9:51 PM" and "Python 3".



The image shows the same Google Colab notebook after execution. The code is now complete, including the main execution logic. The output of the program is displayed in a scrollable box at the bottom of the code editor.

```
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()
```

The output shows the sequence of moves required to solve the 3-disk Tower of Hanoi puzzle:

```
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
```

The interface elements are consistent with the first image, showing the same Colab environment and status bar.