main.py

Share Run

```python
# 8-puzzle A* solver (Manhattan heuristic)
# State: tuple of 9 ints, 0 = blank, row-major order
# Goal: (1,2,3,4,5,6,7,8,0)

import heapq

GOAL = (1, 2, 3, 4, 5, 6, 7, 8, 0)
GOAL_POS = {val: (i // 3, i % 3) for i, val in enumerate(GOAL)}

def manhattan(state):
    total = 0
    for idx, val in enumerate(state):
        if val == 0:
            continue
        r, c = divmod(idx, 3)
        gr, gc = GOAL_POS[val]
        total += abs(r - gr) + abs(c - gc)
    return total

def neighbors(state):
    z = state.index(0)
    zr, zc = divmod(z, 3)
```

Output

Clear

```
Start state:
1 6 5
7 3 8
_ 4 2

Solved in 24 moves; expanded 3248 nodes; solution cost 24

Step 0:
1 6 5
7 3 8
_ 4 2

Move: Up
Step 1:
1 6 5
_ 3 8
7 4 2

Move: Up
Step 2:
_ 6 5
```

Programiz
Python Online Compiler

main.py

Share Run

```
        return

    # Move n-1 disks from source → auxiliary
    hanoi(n-1, source, destination, auxiliary)

    # Move last disk from source → destination
    print(f"Move disk {n} from {source} → {destination}")

    # Move n-1 disks from auxiliary → destination
    hanoi(n-1, auxiliary, source, destination)

# ------------------------------
# MAIN PROGRAM
# ------------------------------

if __name__ == "__main__":
    n = int(input("Enter number of disks: "))

    print("\nSequence of moves:\n")
    hanoi(n, 'A', 'B', 'C')
```

Output Clear

```
Enter number of disks: 3

Sequence of moves:

Move disk 1 from A → C
Move disk 2 from A → B
Move disk 1 from C → B
Move disk 3 from A → C
Move disk 1 from B → A
Move disk 2 from B → C
Move disk 1 from A → C

=== Code Execution Successful ===
```

main.py

Share | Run | Output | Clear

```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    print("BFS Traversal:", end=" ")

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node, end=" ")
            visited.add(node)

            # Add all neighbors
            for neigh in graph[node]:
                if neigh not in visited:
                    queue.append(neigh)

# ------------------------------
```

```
Graph: {0: [2, 1, 3], 1: [], 2: [4], 3: [], 4: []}
BFS Traversal: 0 2 1 3 4
=== Code Execution Successful ===
```

main.py

Share | Run | Output | Clear

```python
# Monkey and Banana Problem

class Monkey:
    def __init__(self):
        self.monkey_pos = "door"
        self.box_pos = "window"
        self.banana_pos = "middle"
        self.on_box = False

    def move(self, place):
        print(f"Monkey moves from {self.monkey_pos} to {place}.")
        self.monkey_pos = place

    def push_box(self, place):
        print(f"Monkey pushes box from {self.box_pos} to {place}.")
        self.box_pos = place
        self.monkey_pos = place

    def climb_box(self):
        if self.monkey_pos == self.box_pos:
            self.on_box = True
```

```
Monkey moves from door to window.
Monkey pushes box from window to middle.
Monkey climbs onto the box.
Monkey takes the banana! 👍

=== Code Execution Successful ===
```

main.py

Share Run

```python
# Eight Queens Problem using Backtracking

board_size = 8
solution = []

def is_safe(row, col, solution):
    for r, c in solution:
        if c == col or r == row or abs(r - row) == abs(c - col):
            return False
    return True

def solve(row=0):
    if row == board_size:
        return True

    for col in range(board_size):
        if is_safe(row, col, solution):
            solution.append((row, col))
            if solve(row + 1):
                return True
            solution.pop()
```

Output Clear

```
Solution for 8 Queens:
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .

=== Code Execution Successful ===
```

Programiz
Python Online Compiler

main.py | Share | Run | Output | Clear

```python
# Depth First Search (DFS) on a Graph

graph = {
    1: [2, 3],
    2: [4, 8],
    3: [5, 7],
    4: [],
    5: [],
    7: [],
    8: []
}

visited = set()

def dfs(node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbor in graph[node]:
            dfs(neighbor)

print("DFS Traversal starting from node 1:")
```

```
DFS Traversal starting from node 1:
1 2 4 8 3 5 7
=== Code Execution Successful ===
```

main.py

Share Run

```python
# Water Jug Problem (4-gallon and 3-gallon)

from collections import deque

def water_jug():
    # state = (jug4, jug3)
    start = (0, 0)
    goal = (2, 0)
    visited = set([start])
    queue = deque([(start, [])])

    while queue:
        (a, b), path = queue.popleft()

        if (a, b) == goal:
            print("Steps to reach 2 gallons in 4-gallon jug:\n")
            for step in path:
                print(step)
            print(f"\nFinal state: {a} gallons in 4-gallon jug, {b}
                in 3-gallon jug.")
            return
```

Output Clear

```
Steps to reach 2 gallons in 4-gallon jug:

Fill 3-gallon jug => (0, 3)
Pour from 3-gallon to 4-gallon => (3, 0)
Fill 3-gallon jug => (3, 3)
Pour from 3-gallon to 4-gallon => (4, 2)
Empty 4-gallon jug => (0, 2)
Pour from 3-gallon to 4-gallon => (2, 0)

Final state: 2 gallons in 4-gallon jug, 0 in 3-gallon jug.

=== Code Execution Successful ===
```