Prog 1>

```
class MonkeyBananaProblem:
    def __init__(self, grid_size):
        self.grid_size = grid_size
        self.monkey_pos = (0, 0)  # Initial position of the monkey
        self.has_banana = False

    def move(self, direction):
        if direction == "up":
            self.monkey_pos = (max(0, self.monkey_pos[0] - 1), self.monkey_pos[1])
        elif direction == "down":
            self.monkey_pos = (min(self.grid_size[0] - 1, self.monkey_pos[0] + 1),
self.monkey_pos[1])
        elif direction == "left":
            self.monkey_pos = (self.monkey_pos[0], max(0, self.monkey_pos[1] - 1))
        elif direction == "right":
            self.monkey_pos = (self.monkey_pos[0], min(self.grid_size[1] - 1, self.monkey_pos[1]
+ 1))
        else:
            print("Invalid direction. Use 'up', 'down', 'left', or 'right'.")

        # Check if the monkey reached the banana
        if self.monkey_pos == (self.grid_size[0] - 1, self.grid_size[1] - 1):
            self.has_banana = True

    def climb(self):
        # Check if the monkey is next to the chair
        if self.monkey_pos == (0, 0):
            self.monkey_pos = (1, 0)  # Climb the chair

    def jump(self):
        # Check if the monkey is on the chair and next to the bananas
        if self.monkey_pos == (1, self.grid_size[1] - 1) and self.has_banana:
            print("Monkey jumped and got the banana!")
        else:
            print("Cannot jump. Make sure the monkey is on the chair and has the banana.")

    def display_state(self):
        print("Monkey Position:", self.monkey_pos)
        print("Banana Status:", "Got it!" if self.has_banana else "Not yet")

if __name__ == "__main__":
    grid_size = (2, 4)  # Change the grid size as needed
    monkey_problem = MonkeyBananaProblem(grid_size)

    while not monkey_problem.has_banana:
        monkey_problem.display_state()
        action = input("Enter action ('move', 'climb', 'jump', or 'exit'): ")

        if action == "move":
            direction = input("Enter direction ('up', 'down', 'left', or 'right'): ")
            monkey_problem.move(direction)
        elif action == "climb":
            monkey_problem.climb()
        elif action == "jump":
            monkey_problem.jump()
        elif action == "exit":
            break
        else:
            print("Invalid action. Use 'move', 'climb', 'jump', or 'exit'.")
```

Prog 2>

```
class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, node, edge):
        if node not in self.graph:
            self.graph[node] = []
        self.graph[node].append(edge)

    def iddfs(self, start, goal, max_depth):
        for depth in range(max_depth + 1):
            visited = set()
            if self.dfs(start, goal, depth, visited):
                return True
        return False

    def dfs(self, node, goal, depth, visited):
        if depth == 0 and node == goal:
            return True
        if depth > 0:
            visited.add(node)
            if node in self.graph:
                for neighbor in self.graph[node]:
                    if neighbor not in visited:
                        if self.dfs(neighbor, goal, depth - 1, visited):
                            return True
        return False

if __name__ == "__main__":
    graph = Graph()
    graph.add_edge('A', 'B')
    graph.add_edge('A', 'C')
    graph.add_edge('B', 'D')
    graph.add_edge('B', 'E')
    graph.add_edge('C', 'F')
    graph.add_edge('C', 'G')

    start_node = 'A'
    goal_node = 'G'
    max_depth = 3  # Maximum depth to search

    if graph.iddfs(start_node, goal_node, max_depth):
        print(f"Goal node '{goal_node}' found within depth {max_depth}")
    else:
        print(f"Goal node '{goal_node}' not found within depth {max_depth}")
```