Artificial Intelligence Using Python [Lab Programs]

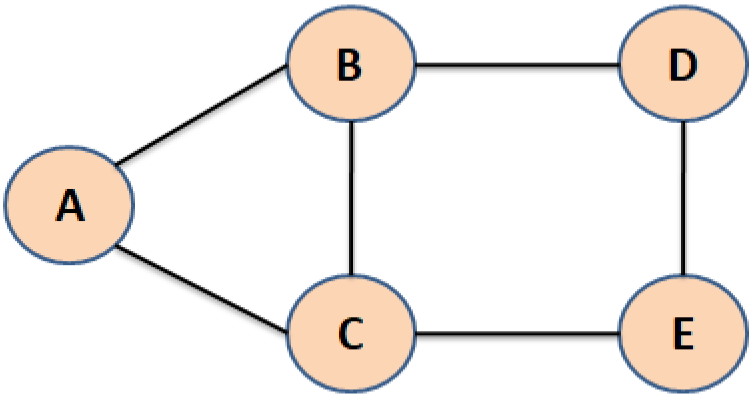
Exp1: Breadth First Search (BFS)

Aim: To write a Program to implement Breadth First Search (BFS) using Python.

Algorithm:

1. **Initialize a queue and enqueue the starting node.**
2. **Mark the starting node as visited.**
3. **While the queue is not empty:**
    - **Dequeue a node from the front.**
    - **Process the node (e.g., print it).**
    - **Enqueue all its adjacent unvisited nodes and mark them as visited.**
4. **Repeat until all reachable nodes have been visited.**

Input Graph



SOURCE CODE :

```
 # Input Graph
graph = {
'A' : ['B','C'],
'B' : ['A','C','D'],
'C' : ['A','B','E'],
'D' : ['B','E'],
'E' : ['C','D']
}
```

```
# To store visited nodes.
visitedNodes = []
# To store nodes in queue
queueNodes = []
# function
def bfs(visitedNodes, graph, snode):
    visitedNodes.append(snode)
    queueNodes.append(snode)
    print()
    print("RESULT :")
    while queueNodes:
        s = queueNodes.pop(0)
        print (s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visitedNodes:
                visitedNodes.append(neighbour)
                queueNodes.append(neighbour)

# Main Code
snode = input("Enter Starting Node(A, B, C, D, or E) :").upper()
# calling bfs function
bfs(visitedNodes, graph, snode)
```

OUTPUT :

```
Sample Output 1:
---------------------
Enter Starting Node(A, B, C, D, or E) :A

RESULT :
A B C D E
-------------------------------------------------------------------------
Sample Output 2:
---------------------
Enter Starting Node(A, B, C, D, or E) :B

RESULT :
B A C D E
```

Exp 8: Monkey Banana Problem

Aim: Write a Program to Implement the Monkey Banana Problem using Python.

Monkey Banana Problem

The **Monkey Banana Problem** is a classic AI problem where a monkey in a room wants to get a bunch of bananas hanging from the ceiling. The monkey needs to use a chair to reach the bananas by performing a sequence of actions.

Problem Statement

1. A monkey is in a room.
2. A chair is present in the room.
3. Bananas are hanging from the ceiling, out of the monkey's reach.
4. The monkey needs to:
    - Move to the chair.
    - Push the chair under the bananas.
    - Climb onto the chair.
    - Grab the bananas.

States and Actions

1. **Initial State** – Monkey is at a certain location, bananas are hanging, chair is at another location.
2. **Goal State** – Monkey is holding the bananas.
3. **Possible Actions**:
    - Walk to the chair.
    - Push the chair to the bananas.
    - Climb onto the chair.
    - Grab the bananas.

Algorithm (Using State Space Search)

1. Define the possible states:
    - Location of the monkey.
    - Location of the chair.
    - Whether the monkey is on the chair.
    - Whether the monkey is holding the bananas.
2. Define possible actions:
    - Move
    - Push
    - Climb

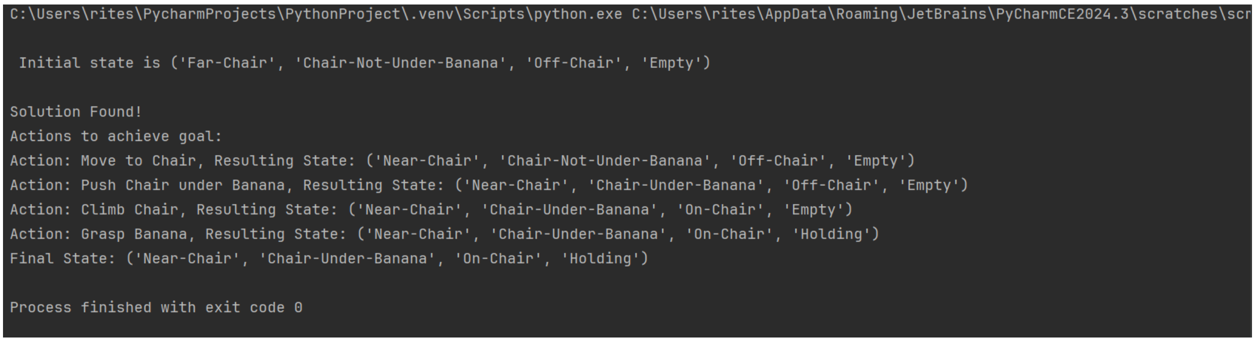
- Grab

3. Use **Breadth-First Search (BFS)** to explore all possible states.
4. If the state where the monkey holds the bananas is reached → Success!
5. If no state leads to success → Fail.

SOURCE CODE :

```python
def monkey_banana_problem():
    # Initial state
    initial_state = ('Far-Chair', 'Chair-Not-Under-Banana', 'Off-Chair', 'Empty')  # (Monkey's Location, Monkey's Position on Chair, Chair's Location, Monkey's Status)
    print(f"\n Initial state is {initial_state}")
    goal_state = ('Near-Chair', 'Chair-Under-Banana', 'On-Chair', 'Holding')     # The goal state when the monkey has the banana

    # Possible actions and their effects
    actions = {
        "Move to Chair": lambda state: ('Near-Chair', state[1], state[2], state[3]) if state[0] != 'Near-Chair' else None,
        "Push Chair under Banana": lambda state: ('Near-Chair', 'Chair-Under-Banana', state[2],  state[3]) if state[0] == 'Near-Chair' and state[1] != 'Chair-Under-Banana' else None,
        "Climb Chair": lambda state: ('Near-Chair', 'Chair-Under-Banana', 'On-Chair', state[3]) if state[0] == 'Near-Chair' and state[1] == 'Chair-Under-Banana' and state[2] != 'On-Chair' else None,
        "Grasp Banana": lambda state: ('Near-Chair', 'Chair-Under-Banana', 'On-Chair',  'Holding') if state[0] == 'Near-Chair' and state[1] == 'Chair-Under-Banana' and state[2] == 'On-Chair' and state[3] !='Holding' else None
    }

    # BFS to explore states
    from collections import deque
    dq = deque([(initial_state, [])])  # Each element is (current_state, actions_taken)
    visited = set()

    while dq:
        current_state, actions_taken = dq.popleft()

        # Check if we've reached the goal
        if current_state == goal_state:
            print("\nSolution Found!")
            print("Actions to achieve goal:")
            for action in actions_taken:
                print(action)
            print(f"Final State: {current_state}")
            return

        # Mark the current state as visited
        if current_state in visited:
            continue
        visited.add(current_state)

        # Try all possible actions
        for action_name, action_func in actions.items():
            next_state = action_func(current_state)
            if next_state and (next_state not in visited):
                dq.append((next_state, actions_taken + [f"Action: {action_name}, Resulting State: {next_state}"]))

    print("No solution found.")

# Run the program
monkey_banana_problem()
```

OUTPUT:



```
C:\Users\rites\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\rites\AppData\Roaming\JetBrains\PyCharmCE2024.3\scratches\scr

 Initial state is ('Far-Chair', 'Chair-Not-Under-Banana', 'Off-Chair', 'Empty')

Solution Found!
Actions to achieve goal:
Action: Move to Chair, Resulting State: ('Near-Chair', 'Chair-Not-Under-Banana', 'Off-Chair', 'Empty')
Action: Push Chair under Banana, Resulting State: ('Near-Chair', 'Chair-Under-Banana', 'Off-Chair', 'Empty')
Action: Climb Chair, Resulting State: ('Near-Chair', 'Chair-Under-Banana', 'On-Chair', 'Empty')
Action: Grasp Banana, Resulting State: ('Near-Chair', 'Chair-Under-Banana', 'On-Chair', 'Holding')
Final State: ('Near-Chair', 'Chair-Under-Banana', 'On-Chair', 'Holding')

Process finished with exit code 0
```