Lab 8

30/07/2024

- ## Traveling Salesman Problem (TSP)

Summary

The Traveling Salesman Problem (TSP) aims to find the shortest possible route that visits each vertex exactly once and returns to the starting point. This solution uses backtracking to explore all possible routes and finds the minimum cost Hamiltonian Cycle.

Workflow

1. Initialization:

- Define the number of vertices (V).
- Initialize an empty list (answer) to store the costs of all Hamiltonian Cycles.

2. TSP Function:

- Base Case: If all nodes have been visited (count == n) and there is an edge back to the starting node, append the total cost to the answer list.
- Recursive Case: Traverse the adjacency list of the current position (currPos). For each unvisited node that is connected to currPos, mark it as visited, call the tsp function recursively with updated parameters, and then backtrack by marking the node as unvisited.

3. Driver Code:

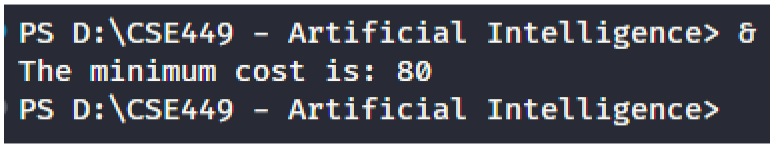
- Define the graph as a 2D list where graph[i][j] represents the cost of traveling from node i to node j.
- Create a boolean list (v) to keep track of visited nodes, initially marking the starting node (node 0) as visited.
- Call the tsp function with initial parameters.

- Print the minimum cost from the answer list, which contains all possible Hamiltonian Cycle costs.

Code:

```
# Number of vertices in the graph

V = 4

answer = []


# Function to find the minimum weight Hamiltonian Cycle

def tsp(graph, v, currPos, n, count, cost):

    # If last node is reached and it has a link to the starting node

    # i.e., the source, then keep the minimum value out of the total cost

    # of traversal and "answer"

    if count == n and graph[currPos][0]:

        answer.append(cost + graph[currPos][0])

        return


    # BACKTRACKING STEP

    # Loop to traverse the adjacency list of currPos node and increasing the count

    # by 1 and cost by graph[currPos][i] value

    for i in range(n):

        if not v[i] and graph[currPos][i]:

            v[i] = True
            tsp(graph, v, i, n, count + 1, cost + graph[currPos][i])

            v[i] = False


if __name__ == "__main__":

    n = 4

    graph = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]


    v = [False for i in range(n)]


    v[0] = True


    tsp(graph, v, 0, n, 1, 0)


    print("The minimum cost is:", min(answer))
```

Output:



```
PS D:\CSE449 - Artificial Intelligence> &
The minimum cost is: 80
PS D:\CSE449 - Artificial Intelligence>
```

- ## Monkey-Banana

Summary

The program uses functions to simulate each step and prints the sequence of actions taken by the monkey. The steps are incrementally numbered to show the order of actions. The user is prompted to input the initial locations of the monkey, banana, and box, which are then used to print the corresponding steps.

Explanation

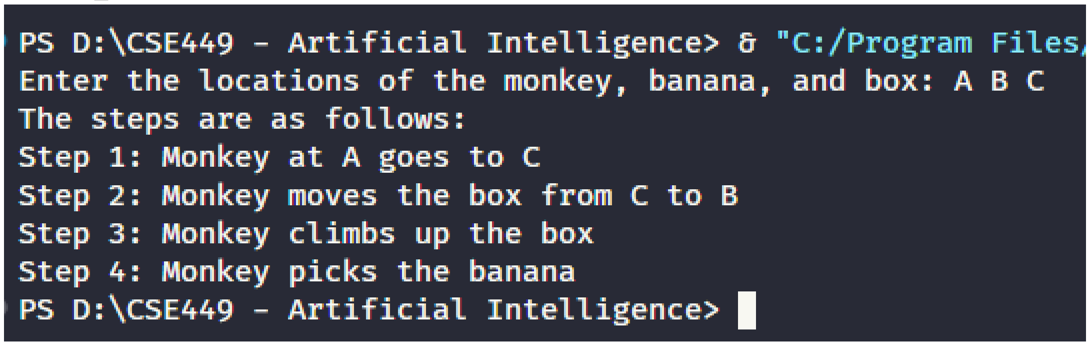
1. **Initialization:** The step counter i is initialized to 0.
2. **Functions:**
   - Monkey_go_box: Increments the step counter and prints the step where the monkey goes to the box.
   - Monkey_move_box: Increments the step counter and prints the step where the monkey moves the box to the banana location.
   - Monkey_on_box: Increments the step counter and prints the step where the monkey climbs up the box.
   - Monkey_get_banana: Increments the step counter and prints **the** step where the monkey picks the banana.
3. **Input:** The program prompts the user to enter the locations of the monkey, banana, and box, separated by spaces.
4. **Assign Input Values:** The input values are split into a list and assigned to the variables monkey_location, banana_location, and box_location.
5. **Print Steps:** The functions are called in sequence to simulate the steps the monkey takes to get the banana.

Code:

```
i = 0

def Monkey_go_box(monkey_location, box_location):
   global i
   i += 1
   print(f"Step {i}: Monkey at {monkey_location} goes to {box_location}")

def Monkey_move_box(box_location, banana_location):
   global i
   i += 1
      print(f"Step {i}: Monkey moves the box from {box_location} to
{banana_location}")

def Monkey_on_box():
   global i
   i += 1
```

```
    print(f"Step {i}: Monkey climbs up the box")

def Monkey_get_banana():
    global i
    i += 1
    print(f"Step {i}: Monkey picks the banana")

codeIn = input("Enter the locations of the monkey, banana, and box: ")
codeInList = codeIn.split()

monkey_location = codeInList[0]
banana_location = codeInList[1]
box_location = codeInList[2]

print("The steps are as follows:")

Monkey_go_box(monkey_location, box_location)
Monkey_move_box(box_location, banana_location)
Monkey_on_box()
Monkey_get_banana()
```

Output:



```
PS D:\CSE449 - Artificial Intelligence> & "C:/Program Files,
Enter the locations of the monkey, banana, and box: A B C
The steps are as follows:
Step 1: Monkey at A goes to C
Step 2: Monkey moves the box from C to B
Step 3: Monkey climbs up the box
Step 4: Monkey picks the banana
PS D:\CSE449 - Artificial Intelligence>
```