Name: Mahmud Hasan

ID: 0242220005343018

Assignment

on

Python implementation that aims to determine multiple linearly independent paths from a given flow graph

The code begins with three main functions:

1. **generate paths**: This function uses a depth-first search (DFS) approach to generate all possible paths in the flow graph from a given start node to an end node. It recursively explores the graph, appending each visited node to the current path until it reaches the end node. It returns a list of all generated paths.
2. **Is linearly independent**: This function checks if a path is linearly independent from a set of existing paths. It compares each node in the path with the nodes in the existing paths. If there is any intersection, it means the path is not linearly independent and returns False. Otherwise, it returns True.
3. **Determine linearly independent paths**: This function takes the flow graph, start node, end node, and the desired number of paths as input. It utilizes the generate paths function to obtain all possible paths in the graph. It then iterates over these paths, checking if each path is linearly independent using the is_linearly_independent function. It keeps track of the linearly independent paths until the desired number is reached or all paths have been processed.

Next, an example usage section is provided. It demonstrates how to use the functions with a specific flow graph.

The flow graph is represented as a dictionary, where each node is a key, and its corresponding value is a list of neighboring nodes. The graph provided in the example has several nodes connected by edges.

The code sets the start node, end node, and the desired number of paths to generate. It then calls the determine linearly independent paths function with the provided graph and parameters.

Finally, the code prints the linearly independent paths using a loop. Each path is displayed with a path number and the nodes separated by hyphens.

To summarize, this code implements a solution to determine multiple linearly independent paths in a given flow graph. It demonstrates the usage of DFS, path generation, and linear independence checking to achieve the desired outcome.

Python Code:

```python
# Function to generate all paths in the flow graph using DFS
def generate_paths(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return [path]
    paths = []
    for node in graph[start]:
        if node not in path:
            new_paths = generate_paths(graph, node, end, path)
            for new_path in new_paths:
                paths.append(new_path)
    return paths

# Function to check if a path is linearly independent from existing paths
def is_linearly_independent(path, existing_paths):
    for existing_path in existing_paths:
        if set(path).intersection(existing_path):
            return False
    return True

# Function to determine multiple linearly independent paths
def determine_linearly_independent_paths(graph, start, end, num_paths):
    paths = generate_paths(graph, start, end)
    linearly_independent_paths = []
    for path in paths:
        if is_linearly_independent(path, linearly_independent_paths):
            linearly_independent_paths.append(path)
            if len(linearly_independent_paths) == num_paths:
                break
    return linearly_independent_paths

# Example usage
graph = {
    '1': ['2'],
    '2': ['5', '3', '10'],
    '3': ['11', '7', '11'],
    '4': ['5'],
    '5': ['6', '8'],
    '6': ['7'],
    '7': ['8'],
    '8': ['9'],
    '9': ['2'],
    '10': ['11', '12'],
    '11': ['13'],
    '12': ['13'],
    '13': []
}

start_node = '1'
end_node = '13'
num_paths = 6  # Number of paths to be generated

linearly_independent_paths = determine_linearly_independent_paths(graph,
start_node, end_node, num_paths)

# Print the linearly independent paths
print("Linearly Independent Paths:")
for i, path in enumerate(linearly_independent_paths):
    print(f"Path {i+1}: {'-'.join(path)}")
```

Result:

```
Best Linearly Independent Paths:
Path 1: 1-2-3-11-13
```