Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 4:

Empirical analysis of algorithms: Depth first search (DFS), Breadth First Search (BFS)

Elaborated: Spătaru Dionisie
st. gr. FAF-211

Verified:

asist. univ. Fiștic Cristofor

Chişinău - 2023

ALGORITHM ANALYSIS

Objective

Study and empirical analysis of algorithms: DFS and BFS .

Tasks

1. Implement the algorithms listed below in a programming language
2. Establish the properties of the input data against which the analysis is performed
3. Choose metrics for comparing algorithms
4. Perform empirical analysis of the proposed algorithms
5. Make a graphical presentation of the data obtained
6. Make a conclusion on the work done.

Theoretical Notes:

Depth-First Search (DFS) and Breadth-First Search (BFS) are algorithms used to traverse or search a graph or tree data structure.DFS starts at a root node and explores as far as possible along each branch before backtracking. This means that it goes deep into the graph or tree structure before backtracking to explore other branches. DFS uses a stack data structure to keep track of the nodes that it has visited and still needs to explore.

BFS, on the other hand, explores the graph or tree level by level, visiting all nodes at the same distance from the starting node before moving on to the next level. BFS uses a queue data structure to keep track of the nodes that it has visited and still needs to explore.

DFS and BFS can be used to solve a variety of problems such as finding the shortest path between two nodes, determining if a graph is connected or not, and detecting cycles in a graph.In summary, DFS and BFS are two common graph traversal algorithms that are used to explore and search graphs or trees. DFS goes deep into the structure before backtracking, while BFS explores the graph level by level.

Comparison Metric:

Efficiency and speed of sorting

IMPLEMENTATION

DFS

```
# Define DFS algorithm
def DFS(graph, start_node):
    visited = []
    stack = [start_node]
    while stack:
        node = stack.pop()
```

```
        if node not in visited:
            visited.append(node)
            stack.extend([n for n in graph.neighbors(node) if n not in visited])
    return visited
```

visited = [] creates an empty list to keep track of the nodes that have been visited so far.
stack = [start_node] creates a stack data structure and adds the starting node to it. The stack will be used to keep track of the nodes that have been visited but still need to be explored.
while stack: creates a loop that continues until the stack is empty. This means that all nodes have been visited and explored.
node = stack.pop() removes the last node that was added to the stack, which is the most recently visited node.
if node not in visited: checks if the node has already been visited. If it has not been visited, then it is marked as visited and added to the list of visited nodes.
visited.append(node) adds the node to the list of visited nodes.
stack.extend([n for n in graph.neighbors(node) if n not in visited]) adds all unvisited neighboring nodes of the current node to the stack. This means that the algorithm will explore all the nodes connected to the current node before backtracking.
return visited returns the list of visited nodes once all nodes have been explored.

So, in summary, this portion of the code implements a Depth-First Search algorithm that explores all the nodes in the graph by starting at the start_node, visiting each node connected to it before backtracking and visiting any unexplored neighboring nodes. The algorithm keeps track of visited nodes using a list and a stack, and returns a list of all visited nodes once all nodes have been explored.

BFS

```
# Define BFS algorithm
def BFS(graph, start_node):
    visited = []
    queue = [start_node]
    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            queue.extend([n for n in graph.neighbors(node) if n not in visited])
    return visited
```

visited = [] creates an empty list to keep track of the nodes that have been visited so far.
queue = [start_node] creates a queue data structure and adds the starting node to it. The queue will be used to keep track of the nodes that have been visited but still need to be explored.
while queue: creates a loop that continues until the queue is empty. This means that all nodes have been visited and explored.
node = queue.pop(0) removes the first node that was added to the queue, which is the oldest visited node.
if node not in visited: checks if the node has already been visited. If it has not been visited, then it is marked as visited and added to the list of visited nodes.
visited.append(node) adds the node to the list of visited nodes.
queue.extend([n for n in graph.neighbors(node) if n not in visited]) adds all unvisited neighboring nodes of the current node to the end of the queue. This means that the algorithm will explore all the nodes at the same level of the graph before moving on to the next level.

return visited returns the list of visited nodes once all nodes have been explored.

So, in summary, this portion of the code implements a Breadth-First Search algorithm that explores all the nodes in the graph by starting at the start_node, visiting all neighboring nodes at the same level before moving on to the next level. The algorithm keeps track of visited nodes using a list and a queue, and returns a list of all visited nodes once all nodes have been explored.

COMPARISON AND RESULTS

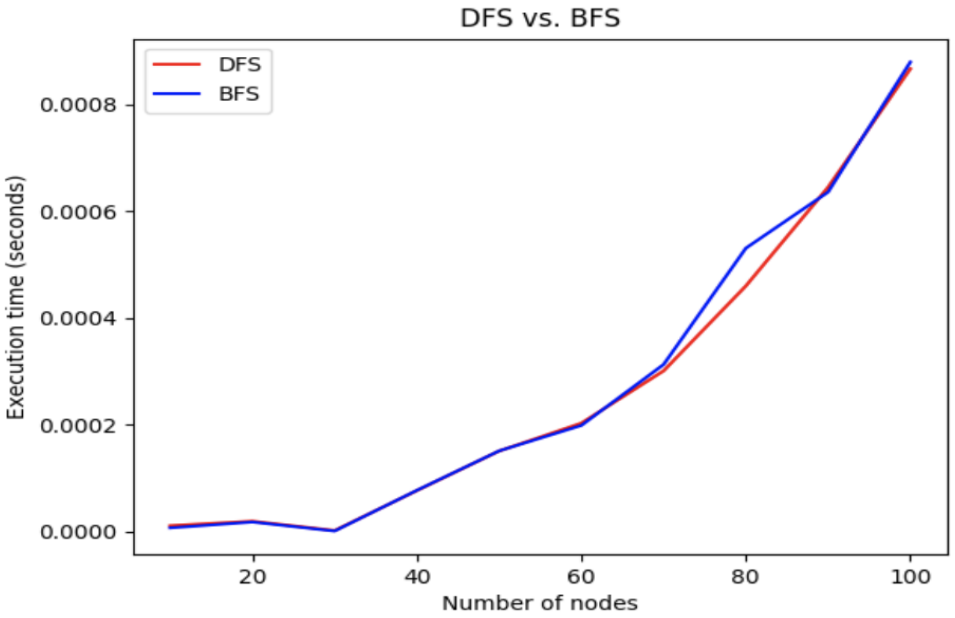

Fig1

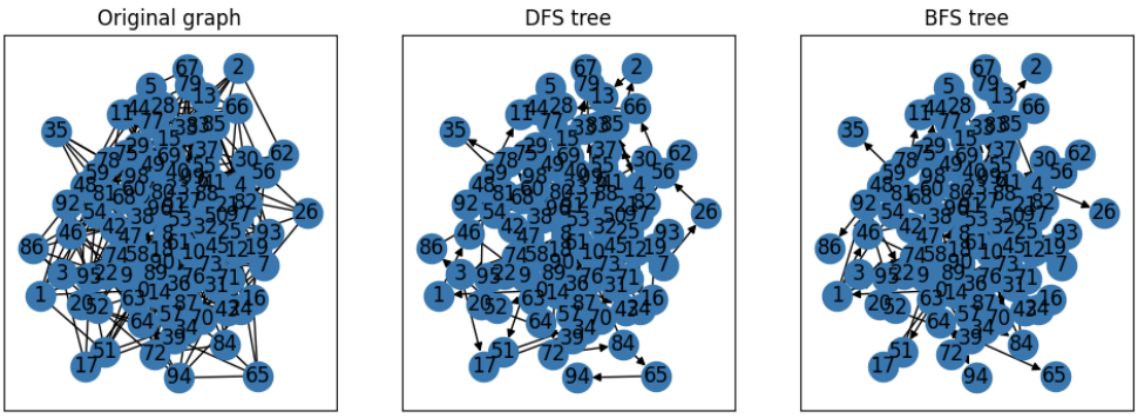

Fig2

CONCLUSION

In conclusion, the code you provided implements two popular graph traversal algorithms, Depth-First Search (DFS) and Breadth-First Search (BFS), and uses them to analyze graphs of different sizes. The purpose of the code is to compare the execution times of DFS and BFS as a function of the number of nodes in the graph and to visualize the trees generated by DFS and BFS for a specific graph.

DFS and BFS are both widely used algorithms in computer science for exploring and searching graphs. DFS is used to traverse a graph by going as deep as possible before backtracking, while BFS is used to explore a graph by visiting all neighboring nodes at the same level before moving on to the next level. Both algorithms have their strengths and weaknesses, and their choice depends on the specific problem and the characteristics of the graph being analyzed.

The code you provided effectively demonstrates the differences between DFS and BFS in terms of their execution times and the trees they generate for a specific graph. The plots generated by the code help visualize the performance of both algorithms for graphs of different sizes, which can be useful for selecting the appropriate algorithm for a given problem.

Overall, the code provides a clear and concise demonstration of DFS and BFS algorithms and their applications in graph analysis, making it a useful resource for anyone interested in learning more about these algorithms.

REPOSITORY

https://github.com/denyred/AA_LAb4