**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**REPORT**

Laboratory Work nr.4

*at Algorithms Analysis*

Elaborated by:

st. gr. FAF-211 Hariton Dan

Verified by:

asist. univ. Fiștic Cristofor

Chişinău - 2023

**ALGORITHM ANALYSIS**

**Subject:** Empirical analysis of algorithms: Depth First Search (DFS), Breadth First Search(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.

**Establish Comaparation:**

We will compare these 2 algorithms we used on a balanced graph, and an unbalanced graph.

**Comparison Metric:**

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n))

**IMPLEMENTATION**

**DFS:**

Depth-first search (DFS) is a graph traversal algorithm that traverses as far as possible along each branch before backtracking. It starts at a designated source vertex and visits all vertices that are reachable from it. DFS uses a stack data structure to keep track of the vertices that have been visited and to determine the next vertex to visit. The algorithm continues until all vertices have been visited, or until a specific goal has been reached. DFS is commonly used to solve problems involving graphs, such as finding paths or cycles, determining connectivity, or solving puzzles.

def dfs(graph, start, visited=None):  
 if visited is None:  
 visited = set()  
 visited.add(start)  
 print(start)  
 for neighbor in graph[start]:  
 if neighbor not in visited:  
 dfs(graph, neighbor, visited)  
i += 1

## BFS:

## Breadth-first search (BFS) is a graph traversal algorithm that explores all the vertices of a graph in a breadth ward motion. Starting from a specified source vertex, BFS visits all vertices at distance 1 from the source, then all vertices at distance 2, and so on until all vertices have been visited. BFS uses a queue to keep track of the vertices to be explored, and a set to keep track of the vertices that have already been visited. BFS is guaranteed to find the shortest path between the source and any other reachable vertex in an unweighted graph. The time complexity of BFS is O(V + E), where V is the number of vertices and E is the number of edges in the graph.

def bfs(graph, start):  
 # initialize the queue and visited set  
 queue = deque([start])  
 visited = set([start])  
  
 while queue:  
 # dequeue a vertex from the queue  
 vertex = queue.popleft()  
 print(vertex)  
  
 # enqueue all unvisited neighbors  
 for neighbor in graph[vertex]:  
 if neighbor not in visited:  
 visited.add(neighbor)  
 queue.append(neighbor)

## Here are the trees used:

## Balanced:

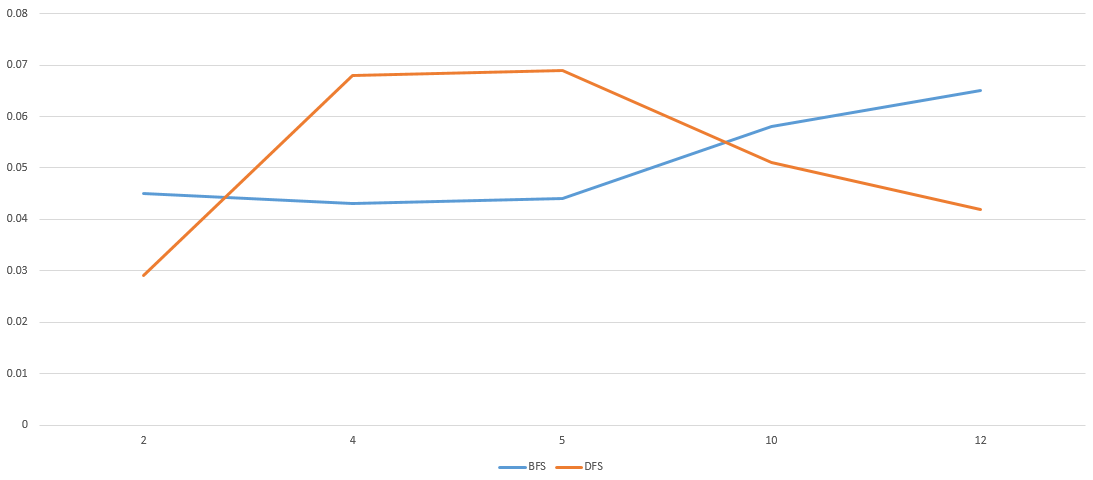
## Unbalanced:

## 

**Results:**

**The Balanced Tree Table:**

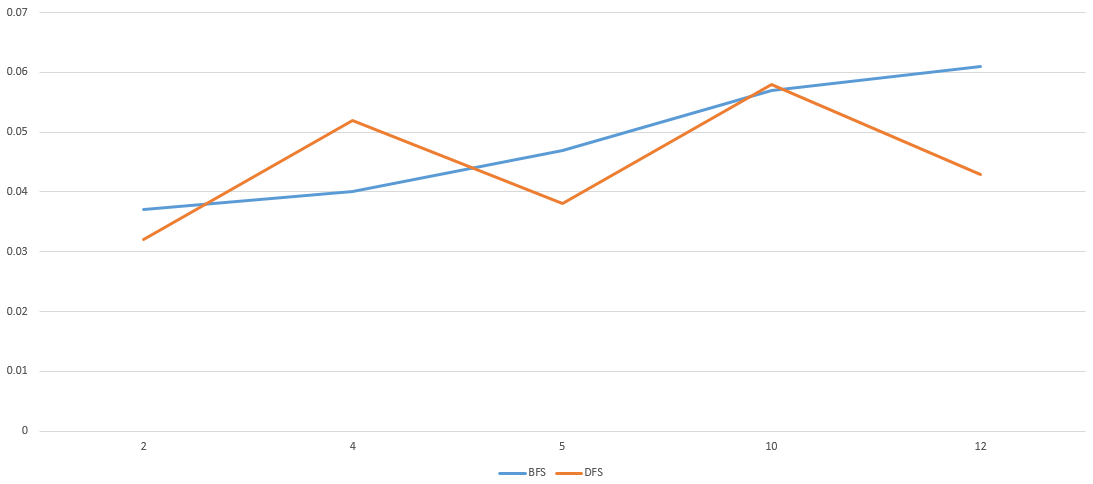
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Balanced** | **2** | **4** | **5** | **10** | **12** |
| **BFS** | **0.045** | **0.043** | **0.044** | **0.058** | **0.065** |
| **DFS** | **0.029** | **0.068** | **0.069** | **0.051** | **0.042** |

**The Balanced Tree Graph:  
**

**The Unbalanced Tree Table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Unbalanced** | **4** | **5** | **10** | **2** | **12** |
| **BFS** | **0.04** | **0.047** | **0.057** | **0.037** | **0.061** |
| **DFS** | **0.052** | **0.038** | **0.058** | **0.032** | **0.043** |

**The Unbalanced Tree Graph:**

****

**Conclusion:**

The proposed work's aim, which consists of the time-based and theoretical analysis of DFS adn BFS algorithms, was successfully accomplished. Two different algorithms were used to determine their temporal complexity and to highlight the most efficient algorithm that will display the desired result using the least time that passed by. The comparison of the algorithms was performed using a table where we collected time based data on the iterations received during the work and the graphs constructed in Excel, which show the noticeable difference between the complexities of the algorithms. From this we observe that in the balanced tree case the BFS method is better for nodes closer to the root, and then DFS takes over as the best for the nodes further. On the other side the BFS and DFS are pretty even when it comes to for searches on the unbalanced tree. For a more accurate answer we should test the algorithms on a bigger number of graphs but that exeedes the scope of our laboratory work. In conclusion, both algorithms are very efficient and the use case of the depends on the problems or graph you have to traverse.

**Link to GitHub**: <https://github.com/haritondan/AA-Labs>