**CSC8103 Distributed Algorithms Coursework Assignment**

**Written Report**

**Description of Trees**

My program models the tree wave and tree election algorithms on 3 different trees.

**Arbitrary**

An arbitrary/general tree is where each node has an arbitrary number of children. My arbitrary tree is the simplest form of this concept, since there are no branches and all the leaf nodes are children of the root node.

**Balanced Binary**

The balanced binary tree seeks to fill every level with nodes completely. The nodes are connected from left to right. Therefore, this is a complete binary tree, and depending on the number of nodes may be a full binary tree.

**Unbalanced Binary**

An unbalanced binary tree has a higher depth than is needed to store all the nodes. My unbalanced tree doesn’t allow any of the right nodes to have any children, whilst giving all left nodes 2 children.

**Results**

I ran each algorithm 10 times and took an average reading of the number of iterations each took to have the nodes decide.

**Wave Tree Algorithm**

As per the specification, the wave algorithm does not contain any diffusion. This has the result of only 2 nodes deciding each time the program is run.

**Arbitrary Tree**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes (N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Second Node Decides** |
| 10 | 5 | 5 |
| 50 | 5 | 7 |
| 100 | 6 | 6 |

**Balanced Binary Tree**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes (N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Second Node Decides** |
| 10 | 3 | 4 |
| 50 | 14 | 14 |
| 100 | 19 | 21 |

|  |  |  |
| --- | --- | --- |
| **Number of Nodes (N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Second Node Decides** |
| 10 | 5 | 5 |
| 50 | 22 | 24 |
| 100 | 45 | 50 |

**Unbalanced Binary Tree**

**Election Tree Algorithm**

In each execution of the program an initiator node is randomly chosen to send a wakeup call to its neighbours before the iteration of the algorithm begins.

**Arbitrary Tree**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes(N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Last Node Decides** |
| 10 | 7 | 12 |
| 50 | 6 | 9 |
| 100 | 7 | 11 |

**Balanced Binary Tree**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes(N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Last Node Decides** |
| 10 | 7 | 12 |
| 50 | 34 | 46 |
| 100 | 32 | 45 |

**Unbalanced Binary Tree**

|  |  |  |
| --- | --- | --- |
| **Number of Nodes(N)** | **Number of Iterations before First Node Decides** | **Number of Iterations before Last Node Decides** |
| 10 | 15 | 21 |
| 50 | 42 | 77 |
| 100 | 81 | 165 |

**Analysis/Observation**

An initial observation of both algorithms performance is that performing 100N iterations each execution is far too much. Since all the processes that are going to decide (2 in the wave algorithm, N in the election algorithm) decide fairly quickly, having hundreds or thousands of iterations where all the nodes have finished the algorithm (in the case of the election algorithm) or are waiting for a response from a silent neighbour that will never come (wave algorithm, due to the lack of diffusion) represent an inefficient program. Therefore, were I to repeat this test I would reduce the number of iterations to 50N or even 25N- the same results would be achieved and the program would run much quicker.

In an arbitrary tree executing the wave algorithm, when the number of nodes is increased the number of iterations stays similar, indeed when the program is run with 50 nodes it takes 1 iteration more for the second node to decide than when it is run with 100. With 100 nodes the algorithm takes 1 more iteration to decide on the first node, but this is to be expected. Since the arbitrary tree only has depth 1 the results are likely to remain in this area no matter how many more nodes are added to the tree.

Similarly, when the arbitrary tree is executing the election algorithm the results remain similar for the three sizes of tree I have tested for, and therefore can be inferred that larger trees with the same depth would perform similarly. All the executions managed to decide within 13 iterations and had no more than 5 iterations between the first and last node deciding.

When the balanced binary tree executes the wave algorithm with 10 nodes, the results for the number of iterations are the lowest overall. By taking only 4 iterations for the second node to decide and thus conclude the algorithm, use of a binary tree is even more effective than the use of an arbitrary tree in this case. As the number of nodes increase, the number of iterations of the algorithm required increase as one would expect, however the amount of time between the first and second nodes deciding remains very low.

With the election algorithm on a balanced binary tree, the increase in iterations required before the algorithm terminates rises dramatically- the program run with 50 nodes requires almost as many iterations for all the nodes to decide. However, these numbers stay the same when the number of nodes double to 100, suggesting that this will be an efficient configuration to use with more nodes and deeper trees.

The unbalanced binary tree executing the wave algorithm has an inefficient performance, since the number of iterations until termination increases exponentially with the number of nodes added, and usually equates to approximately half of the number of nodes.

When executing an election algorithm, the unbalanced binary tree performs even more poorly. In all cases, this is the only tree-algorithm pairing that takes more iterations to complete than there are nodes in the tree- when there are 10 nodes the algorithm requires over twice as many iterations.