CS590 Assignment 5

1. **Abstract**

In this assignment, depth first search (DFS) and breadth first search (BFS) are explored by hand, and then subsequently implemented in C++. The node ordering between BFS and DFS is shown to be different. It is also observed that BFS does not always exhaust all the nodes in a directed graph, unlike DFS. Finally, an algorithm for discovering strongly connected components (SCC) using DFS, known as “Tarjan’s algorithm”, is explored. Finally, it is shown by example that the starting node used for running DFS does not affect the output of Tarjan’s algorithm.

1. **DFS and Tarjan’s algorithm**

Please see the attached PDF.

1. **C++ DFS and BFS**

A rudimentary DFS and BFS algorithm is implemented in C++; they are rudimentary, as we are not keeping track of discovery and finish times. These algorithms are implemented simply to report the order in which each vertex is discovered. The example graph provided is shown below in figure 1.

Diagram

Description automatically generated

**Figure 1.** Graph used for BFS/DFS c++ development.

Before running this through c++, we can trace out the BFS and DFS runs by hand, to generate our expected output. The hand trace results are shown below, with both traces beginning at node 0 (in green, above).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Current Vertex** | **Queue** | | 0 | 1, 2, 3 | | 1 | 2, 3, 5 | | 2 | 3, 5 | | 3 | 5, 4, 7 | | 5 | 4, 7 | | 4 | 7 | | 7 | 6 | | 6 | <empty> | | |  |  |  | | --- | --- | --- | | **Vertex** | **Discovery** | **Finish** | | 0 | 0 | 15 | | 1 | 1 | 8 | | 2 | 2 | 3 | | 3 | 9 | 14 | | 4 | 10 | 13 | | 5 | 4 | 7 | | 6 | 11 | 12 | | 7 | 5 | 6 | | 8 | 16 | 19 | | 9 | 17 | 18 | |

**Table 1.** BFS (left) and DFS (right) manual trace results for example graph.

Given the results from the manual trace, we can see our two expected outputs. For the BFS, we can trivially see the expected output. To determine the expected output of the DFS algorithm, we simply report the nodes in ascending order of the discovery time.

BFS: 0, 1, 2, 3, 5, 4, 7, 6

DFS: 0, 1, 2, 5, 7, 3, 4, 6, 8, 9

**Figure 2.** Expected BFS and DFS outputs.

Now, we can run our algorithm to verify the results match.

> ./graph

BFS output:

0,1,2,3,5,4,7,6,

DFS output:

0,1,2,5,7,3,4,6,8,9,

**Figure 3.** C++ BFS and DFS outputs.

Here, we can plainly see that our algorithm matches the expected output.