Lab Four

Joseph McDonough

Joseph.McDonough1@marist.edu

November 13, 2020

1 Binary Tree Results

A file, titled magicitems, is a text file that has the name of 666 different items. The items appear in the text file in a random order, such that it is not presorted in any way. Those 666 unique items are ran through a binary tree. At random, 42 of those items are selected to be found using the binary tree.

After running a large sample size of finding 42 random items, the average search count varies between 10 and 13 searches.

Following an analysis of Big O, a perfect binary tree would result in $\log_2 n$ time, just like a binary sort on a sorted list. The worst possible binary tree would result in a search time of n, just like a linear search. As it is far more likely for a binary tree to be closer to a binary search than a linear search, an average search count is expected around the $\log_2 n$ range, which for a list of size 666 is 9. Although the average search time in this case was between 10 and 13, it is close enough to 9 and can be deemed an appropiate result.

2 Traversals

2.1 Results

1. Undirected 7-vertex 11-edge
   - Vertex Count - 7
   - Edge Count - 11
   - Expected Time Complexity - O$(V + E) \rightarrow 18$
   - Depth-First Search Comparison - 16
   - Breadth-First Search Comparison - 7
2. Undirected 8-Vertex Full
   - Vertex Count - 8
   - Edge Count - 28

- Expected Time Complexity - O($V + E$) $\rightarrow$ 36
- Depth-First Search Comparison - 15
- Breadth-First Search Comparison - 8

3. Undirected 63-Vertex Tree (branch factor = 2)
   - Vertex Count - 63
   - Edge Count - 62
   - Expected Time Complexity - O($V + E$) $\rightarrow$ 125
   - Depth-First Search Comparison - 125
   - Breadth-First Search Comparison - 63
4. Undirected 64-Vertex 72-Edge Erdos-Renyi Random
   - Vertex Count - 64
   - Edge Count - 72
   - Expected Time Complexity - O($V + E$) $\rightarrow$ 136
   - Depth-First Search Comparison - 127
   - Breadth-First Search Comparison - 72
5. Undirected Zork Map (ground level) w/ 3 disconnected components
   - Vertex Count - 21
   - Edge Count - 25
   - Expected Time Complexity - O($V + E$) $\rightarrow$ 46
   - Depth-First Search Comparison - 41
   - Breadth-First Search Comparison - 23

2.2 Asymptotic Analysis

Although they follow different algorithms, both depth-first search (DFS) and breadth-first search (BFS) have a time complexity of O($V + E$) where V is the amount of vertices and E is the total edge count. DFS utilizes the stack data structure. This traversal starts at a root node and traverse as long as it could along a branch and then backtracks. When it backtracks, it is utilizing the stack and popping of vertices until it reaches one where there is an unexplored branch. DFS explores that branch and backtracks and this process occurs until all nodes are visited.

In the instance of a disconnected graph where all nodes are not connected in some way, an additional function is required to reach out and make sure all vertices get accounted for.

```
109  for(Vertex checkProcess : this.getVertices())
110  {
111      if(!checkProcess.getProcessed())
112          depthFirstSearch(checkProcess);
113  }
```

BFS also takes has the same time complexity even though it utilizes a queue instead of a stack. Instead of exploring a branch til completion and going back, BFS explores all the neighbors of the root first. Once all

the neighbors are accounted for, it takes the neighbors of one of it neighbors, then takes the neighbors of the rest of the neighbors to the root node until they are all accounted for. Then it recursively goes out and gets the neighbors of all the nodes on the graph such that all get accounted for.

In the instance of a disconnected graph, a near identical helper function is called to ensure that all vertices are accounted for in the BFS.

For all the graphs that were experimented with on this assignment, none of the depth-first searches or breadth-first searches exceeded the sum of the edges and vertices. This makes sense as $O(V + E)$ is a worst case scenario. As the size of the graph grows, in both edge and/or vertex count, the amount of checks on the vertices increase. Throughout the algorithm, each vertex can be marked for processing (that is, has just been visited for the first time) only once. In all cases, the BFS was done with less comparisons than the DFS performs. This also makes sense as BFS slows works its way out from the root, exploring its neighbors before going farther out. DFS goes as far out as it could, before it makes it way back.

Regardless, the calculation of $O(V + E)$ makes sense for both graphs.

For breadth-first search, it will in posses all the vertices in its queue, which would result in $O(V)$. Likewise, it will traverse over all the edges in the graph, just once too. This results in $O(E)$. Both sets of operations work in linear time so there are no unaccounted for variables, meaning the total time complexity is $O(V + E)$.

Depth-first search is nearly identical as all vertices will enter the stack at one point in time, meaning $O(V)$ and each edge will be accounted for, resulting in $O(E)$ with a total of $O(V + E)$.