CSCI 3104 Spring 2014 Problem Set 7
Tyler Behm (05/21)

Breadth-first search (BFS) on an adjacency list has $\theta(V+E)$ space complexity. This property will allow us to reach a solution with the space constraints.

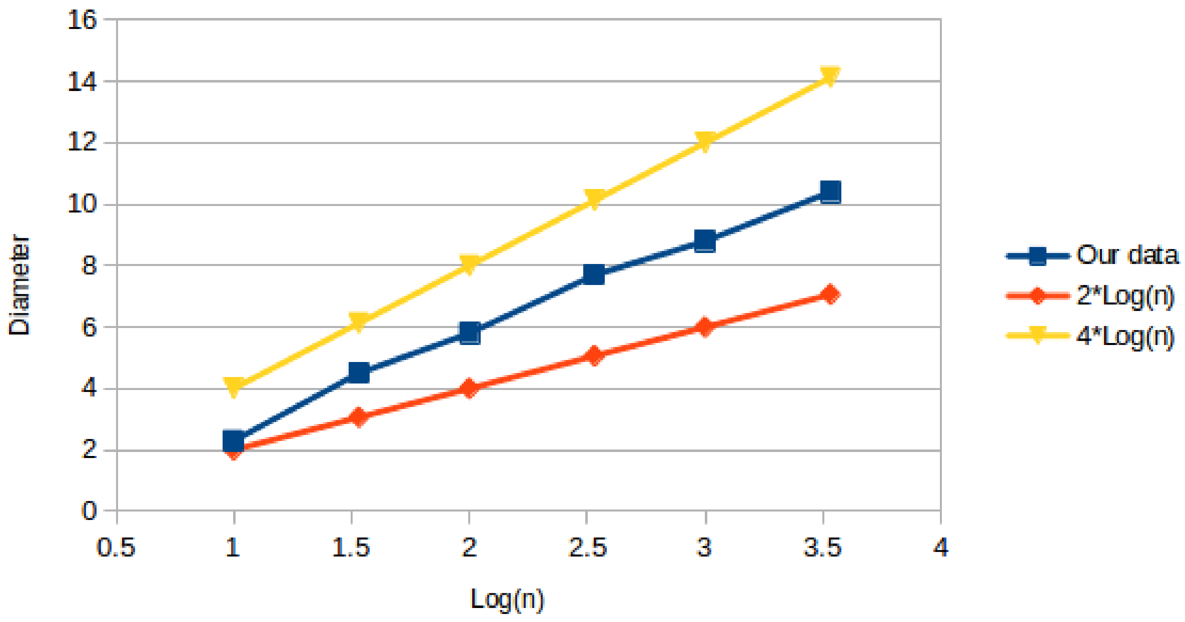
BFS finds the shortest path from a single source to all other vertices. We simply implemented the BFS pseudocode from the book (page 595). We modified the BFS code so that it returns the longest shortest path for that source. We then find the longest shortest path for all vertices and keep the maximum. The maximum is the diameter of the tree. A simple pseudocode of our solution is:

```
BFS(i){
returns longest shortest path for source i
}

diameter = 0
for i= 0 to |V| \\scan all vertices
    temp = BFS(i)
    diameter = max(diameter, temp)
return diameter
```

All calculations will use $c = 5$. To get a more accurate number, we averaged the diameter over 20 different Erdos-Renyi random graphs for each value of n. To illustrate the growth of the expected diameter, we plotted the expected diameter, an upper bound ($4 \log n$), and a lower bound ($2 \log n$) with respect to $\log n$. We varied n over a large range (10, 34, 100, 340, 1000, and 3400) to get a good sample.

We can see that our data stays well within the bounds. Thus we have shown that the expected diameter increases like $O(\log n)$.



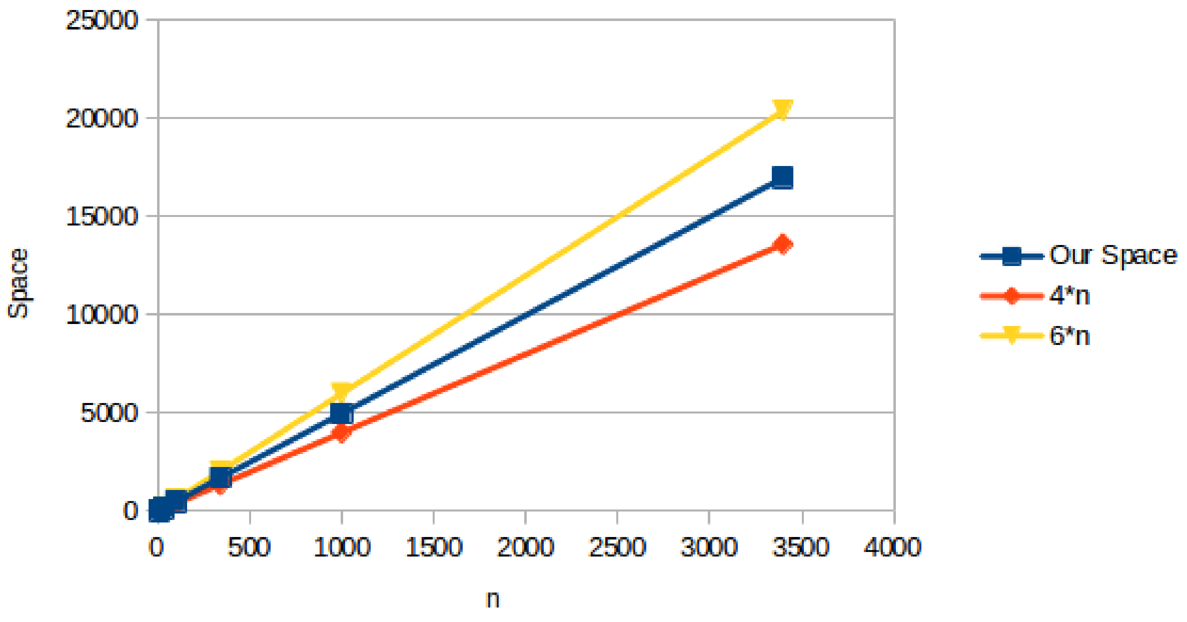
In problem set 5, we numerically measured the order of growth runtime of an algorithm. Because runtime is proportional to the number of atomic operations, we were able to correctly use atomic operations as a proxy measure of runtime. We will do the same process for our graph algorithm.

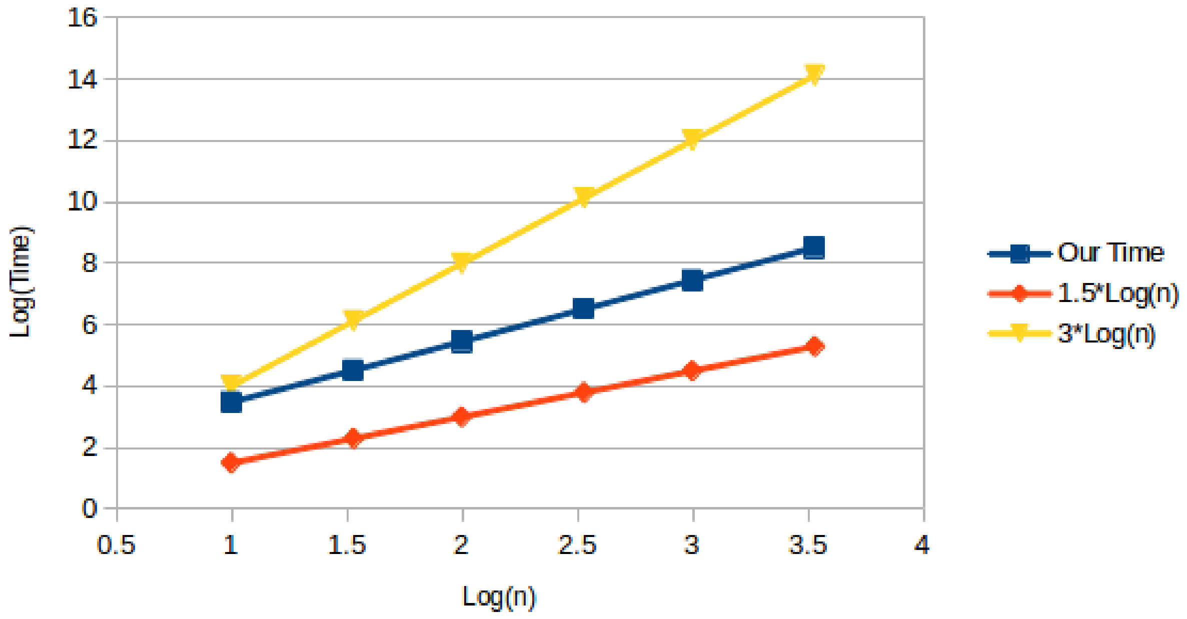
We also want to measure the space complexity of our algorithm. Analogous to our runtime measurement, we will count the number of elements in data structures (arrays, queues, etc.) and use that as a proxy measure of space complexity. For arrays, we can simply increment the space counter. But queues are a special case. We must add to the counter the maximum size that the queues grow to.

In both cases, we use global counters so that we can keep count of all atomic operations and elements. Also, we average over 20 trials to get a more accurate measurement.

It is important to note that the subroutine BFS in our code will produce zero net change in our space counter. This is because we can deallocate the memory used by BFS after its return.

We can see in our plots that the space counter is asymptotically bounded by 4n and 6n where n is our number of vertices. Thus it grows like $O(V + E)$ because its linear. Furthermore, the time counter is growing like $O(V^2)$ because it has a slope of 2 on a log-log plot.




16
14
12
10
8
6
4
2
0
Log(Time)
0.5
1
1.5
2
2.5
3
3.5
4
Log(n)
Our Time
1.5*Log(n)
3*Log(n)