Brett Wilhelm
Harrison Zey

Edges: 2868

Vertices: 200

Centrality: 9382 (Computer_Science)

OutDegree: 199 (Computer_Science)

Diameter: 6

Data Structure (Queue):

We used a LinkedList for our queue as insertion and deletion with a queue needs to be quick; a LinkedList allows for a worst-case runtime (insertion and deletion) of O(1).

Data Structure (visited):

Hashmap used for efficiency. The average case for runtime is O(1) for search, insertion and deletion. This meant that our speeds would be better on average than had we used other data structures.

Number of Edges and Vertices:

Having 200 vertices makes perfect sense as the "max" parameter (max vertices) was 200 and there was no way that our WikiCrawler was going to find less than 199 unique wikipages from something as generic as "Computer_Science." We found 2868 edges to our graph and conceptually this number seems to be accurate.

Vertex with the Largest Out Degree

Wiki page "Computer_Science" had the largest Out Degree of any page from our test at 199. This number is correct as there were at least 199 unique wiki page links on the Computer Science wiki page. This number would only be smaller than the total number of vertices-1 if the Computer Science page didn't have a large enough selection of unique links, meaning more would have to be searched for on crawled pages.

Diameter:

The diameter of 6 was surprising, however it makes sense when reflecting. When running this test using "Iowa_State_University"'s wiki page in a query of 20 max vertices we found the diameter to be 5, but after testing multiple sizes between 20 and 2 we realized the

growth was exponential to the point where it had slowed so much after hitting 5 at around 20 that 6 at 200 became much more believable.

Vertex with the Highest Centrality:

Page "Computer_Science" holds the record of largest centrality in our test at 9382. Because this page is both A) our initial page/vertex and b) a very generic page about a topic it makes sense that it would be involved in most of our vertices' shortest paths to their destination vertices.

Method Runtimes:

outDegree():

Worst case our hashmap that we use to store our nodes in allows us to retrieve nodes with a runtime of O(n), however on average the runtime will be O(1).

bfsPath():

Time complexity of the BFS search is O(|V|+|E|) as is typical of a BFS search.

diameter():

Runtime will be O(|V^2| * (|V|+|E|)), with the V^2 coming from the nested loops and the V+E coming from the aforementioned bfsPath()'s runtime.

centrality():

Runtime will be O(|V^2| * (|V|+|E|)) for the same reasons as diameter();