Data Structures used for Q and $visited$.

We used a combination of HashMaps, Stacks, Queues, and lists. We used a Queue first to hold the order of links that are pulled from a wiki page. They were then popped from the Queue, checked to see if the pages contained the key words, then pushed to a stack. We stored whether the pages contained the key words in a HashMap¡String,boolean¿ and we also cached the wiki page in a HashMap so we did not have to scrape the page from the web again. Once we had emptied the first Queue and filled the stack (this reversed the order so we had the first link found at the top). Wiki links were only added to the stack if they contained the key words. We also added the edge to the textfile for each wiki link that did contain the key words. Now that the stack was full, we could pop each one, and then find the cached web page it was associated with in the HashMap and start the process over again with the Queue and adding each link of that page. This continued until all links were found up to the number specified.

Number of edges and vertices.

There are 200 vertices.

Vertex with largest out degree.

Diameter of the Graph.

Vertex/page with highest centrality.

Run-time analysis of GraphProcessor.

V is the number of vertices, and E is the number of edges in graph G.

outDegree(String v)
Runs in $O(1)$ time because you just return the size of the Linkedlist of edges for the vertex v.

bfsPath(String u, String v)
Runs in $O(V + E)$ time because you look at each vertex and each edge once.

diameter()
Runs in $O((V^2)V + E)$ time because you do BFS for each pair of vertices.

centrality(String v)
Runs in $O((V^2)V + E)$ time because you do BFS for each pair of vertices.