Programming Assignment 2 Report

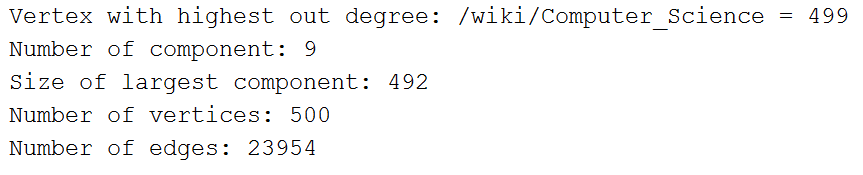
Team: Cheng Song, Yangxiao Wang

1. Data structures used for Q and visited. Your rationale behind the choice of data structures.  
   For Q we used queue as the data structure because queue can allow us easily to remove the first element.  
   For visited, we used HashSet because HashSet has the better performance for adding and accessing data
2. Number of edges and vertices in the graph WikiCS.txt  
   Number of vertices: 500

Number of edges: 23954

1. Vertex with largest out degree in the graph WikiCS.txt  
   Vertex with highest out degree: /wiki/Computer\_Science = 499
2. Number of strongly connected components in WikiCS.txt  
   Number of component: 9
3. Size of the largest component in WikiCS.txt  
   Size of largest component: 492
4. The data structures that you built/used in GraphProcessor  
   **LinkedHashMap**<**String**, **LinkedList**<**String**>> to store graph data  
   **ArrayList**<**ArrayList**<**String**>> to store all the components  
   **HashMap**<**String, Integer**> to store visited list  
   By using hashmap and LinkedHashMap we would be able to access our graph data much more fast and easier. Although, it requires more memory space. In the constructor, we traversal the input file and add all vertice as **KEY** in the **LinkedHashMap**<**String**, **LinkedList**<**String**>> , and all the adjacent vertices of the **key** are added into the **LinkedList**<**String**>.   
   In addition, we called a helper method to find all Strongly Connected Components and stored them into **<comp>.** So we only need to run the algorithm at the beginning,
5. Analyze and report the asymptotic run time for each of the public methods from the class GraphProcessor. Note that the run times depend on the choice of your data structures.  
     
   **public int outDegree(String v)**The runtime for outDegree would be equal to the runtime of HashMap.get which is **O(1)**  
     
   **public boolean sameComponent(String u, String v)**runtime equals to number of component so runtime is **O(n)**n = number of components  
     
   **public ArrayList<String> componentVertices(String v)**runtime equals to number of component so runtime is **O(n)**n = number of components  
    **public int largestComponent()**runtime equals to number of component so runtime is **O(n)**n = number of components  
     
   **public int numComponents()**The run time = **O(1)** because the method **return comp**.**size**()  
   Which comp is ArrayList of ArrayList that stores all components

**public ArrayList<String> bfsPath(String u, String v)**General runtime of BFS is **O(V+E)  
  
The run time to find all the Strongly connected components:**We used Kosaraju's algorithm which runtime is **O(V+E)** equal to the runtime of Depth-first search



*Output of report*