In this extra assignment, I used python 3.6 to implement the algorithm.

My algorithm will deal with multiple roots in parallel (Run BFS) by using map function and then aggregate the result from each map by using the reduceByKey function to get the sum of betweenness. Finally, I used the mapValues function to get the final result by dividing the sum by 2

Pros: This works very efficiently for small graphs and saves a lot of time

Cons: when the BFS tree is really large, the tree cannot be stored in the main memory

other thoughts about this question:

Besides using parallelize calculation in the root, there are many other parts can be calculated parallel. For instance, for each map, we can consider one shortest path and merge them later, or we could calculate each node parallel.

I also read some articles about how to apply spark in GN algorithm. The following is how pregal works in GN:

1. Messages are passed from one node to another, using a custom Message class. A vertex is activated for this iteration if it receives messages.

2. Every vertex aggregates the message and processes them updating its values (Spark has immutable objects, we need to create new instance of everything a node is holding). This is done using a VertexProgram. A VertexProgramis run every time a vertex receives messages.

3. Every vertex receives messages from its neighbors. The messages are aggregated into a single message using an aggregator program. So when a vertex receives messages, they are aggregated using the AggregatorProgram and then sent to the VertexProgram.

4. The vertex processes the message creates a new value and sends out a new Message to all its neighbors.

This is repeated till we have no messages sent from any vertex (no vertex is activated)

The vertexUpdate() receives ForwardMessages and creates a new ShortestPaths object and returns it. It retains the old values and updates only those that are passed in the ForwardMessages object.

The sendMessage() creates new set of messages for all update values (sources for which the number of shortest paths for updated).

The mergeMessages() adds up all Message objects and creates a new ForwardMessages.

This method will converge eventually since the graph is undirected and unweighted. The number of messages that needs to be passed is equal to the diameter of the graph. The beauty of this approach is that this works well for disconnected graphs.

At every step in the Girvan-Newman algorithm, we need to recompute betweenness centrality after removal of an edge. This can be done efficiently having disconnected components, without worrying if we need to find the connected components and run a BFS separately.