Report: Careful runtime analysis of your isStronglyConnected method. Express answer as a function of the |V| and |E|. Typewrite your analysis. Include your isStronglyConnected method in the report.

The way I designed my isStronglyConnected method was by first analyzing what the main purpose of the method is. This method is supposed to return true if every Vertex has an Edge that connects it to every other Vertex. So, what must be done is to traverse through every Vertex you have and check the EdgeLists connected to them and so on.

My isStronglyConnected method below tackles this problem by creating an Object Array of SPP packets: taking in the start location, the array of distances/cost, and the parent array showing where every vertex has come from. I then traverse through my parent array while making sure I do not exceed the number of my vertices for the graph. The check I implement looks at the parent of the current index of my Object Array. If the parent is -1 or if the vertex looks to be a parent of itself, this check will return false. The -1 comes from when the parent array is initialized in bfsShortestPaths which means that the vertex has not yet been visited. The same goes for the distance array initialization which starts at infinity but will update once the “path to the vertex cost” has been calculated. If they do not return false, it will continue traversing through all the vertices until a path from the start vertex to every other vertex has been found to show the graph is strongly connected.

My runtime analysis is below:

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*isStronglyConnected\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*public boolean isStronglyConnected()  
{  
 SPPacket[] whereIsDaddy = new SPPacket[nVertices]; ............ c  
  
 for(int a = 0; a < nVertices; a++) { ............ V \* (V + E)  
 whereIsDaddy[a] = bfsShortestPaths(a); ............ V + E  
 for(int y = 0; y < whereIsDaddy[a].parent.length; y++) { ............ V  
 if(whereIsDaddy[a].parent[y] == -1 && a != y) { ............ c  
 return false; ............ c  
 }  
 }  
 }  
 return true; ............ c  
 }

O(V \* (V + E)) ultimately is the runtime analysis of isStronglyConnected but we have to evaluate bfsShortestPath below because isStronglyConnected relies on it.

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BFS Shortest paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* public SPPacket bfsShortestPaths (int start) {  
 int [] distance = new int[nVertices]; ............ c  
 int [] daddy = new int [nVertices]; ............ c  
 Queue<Integer> mainQueue = new PriorityQueue<>(nVertices); ............ c  
 boolean[] detective = new boolean[nVertices]; ............ c  
  
 for(int d = 0; d < nVertices; d++){ ............ V  
 distance[d] = Integer.*MAX\_VALUE*; ............ c  
 daddy[d] = -1; ............ c  
 }  
 int current = start; ............ c  
 mainQueue.add(start); ............ c  
 detective[start] = true; ............ c  
 distance[start] = 0; ............ c  
 while(!mainQueue.isEmpty()){ ............ E  
 if(detective[current]){ ............ c  
 current = mainQueue.remove(); ............ c  
 }  
 detective[current] = true; ............ c  
 for(int s = 0; s < adjList[current].size(); s++){ ............ V  
 if(!detective[adjList[current].get(s).vertex2]){ ............ c  
 mainQueue.add(adjList[current].get(s).vertex2); ............ c  
 distance[adjList[current].get(s).vertex2] = distance[adjList[current].get(s).vertex1] + 1; ............ c  
 daddy[adjList[current].get(s).vertex2] = adjList[current].get(s).vertex1; ............ c  
 }  
 }  
 }  
 return new SPPacket(start, distance, daddy); ............ c  
 }

O(V + E)

After careful analysis, we have come to find that BFS relies on the |V| + |E| because of its iterations being dependent on the number of vertices and edges your graph has. isStronglyConnected will run on every single vertex on the graph which means BFS will also run on every vertex of the graph. The time complexity of isStronglyConnected goes by the number of vertices on the graph, and bfsShortestPaths runs on the number of vertices and edges on the graph; consequently, since the two work together, the complete runtime analysis will be:

O(|V| \* (|V| + |E|)).