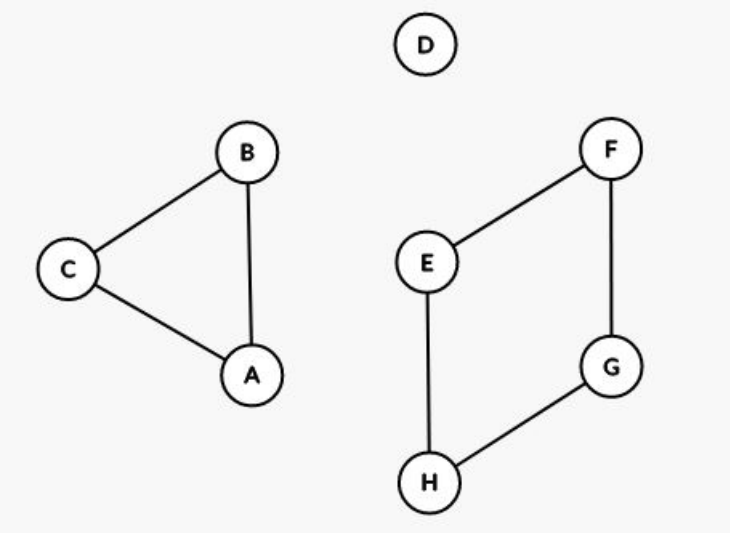
4.4 Connectivity and Strongly Connected Regions

Explanation

Undirected graphs

Connectivity and connected components are quite straightforward when it comes to undirected graphs. Indeed, an undirected graph is connected if there is a path between any two vertices. Connected regions are also referred to as connected components, which are defined as subgraphs whose edges are internally connected to themselves, but have no edges connecting to the other subgraphs. Take the following graph as an example:



In this graph, we can see that there are 3 disjoint connected components: $[A, B, C]$, $[D]$, and $[E, F, G, H]$. Notice how vertex D is itself a connected component of just one vertex, as it has no edge connecting it to any of the other nodes.

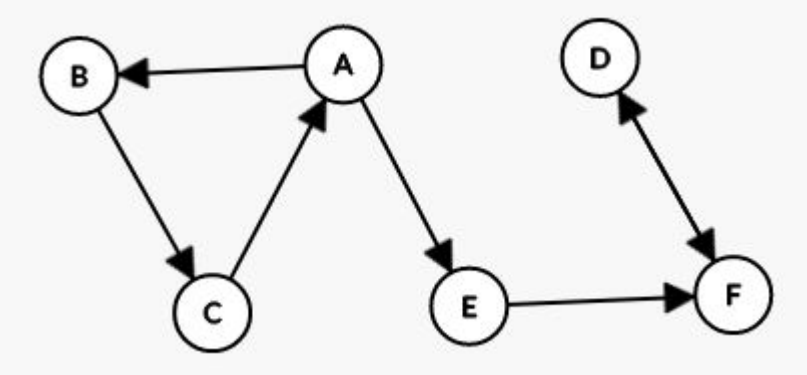
Likewise, finding connected components is straightforward in a directed graph. The outer for loop in the DFS algorithm (line 4 in the code above), is what calls each connected component of the graph to be traversed. This is because all of the nodes within each component are explored recursively in the explore function. Therefore, once the end of a subgraph is reached, then the other connected components are traversed.

Directed graphs

Directed graphs, on the other hand are little more complicated, and therefore more interesting. Connectivity in directed graphs is defined in the Dasgupta text as: Two nodes u and v of a directed graph are connected if there is a path from u to v and a path from v to u. So, similar to how there are paths from one node to any other node in a connected component in an undirected graph, the same general rule still holds for directed graphs. A strongly connected component in a directed graph is a set of vertices in which all vertices are reachable by all other vertices in that graph, and vice versa. However, this is more challenging because

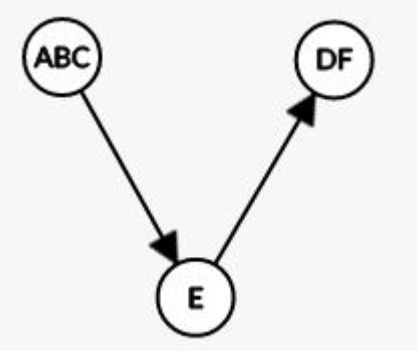
edges in a directed graph can only go one way. So, often, a more circuitous route must often be taken.

Take, for example the following graph:



In this graph, we can see that $[A, B, C]$ is a connected component. This is because for all vertices in that component, there is a path from v to u and a path from u to v. Likewise, we can see that $[D, F]$ is also a strongly connected component, as there is a path from D to F and a path from F to D. However, E is not included in either of these connected components. That's because there is no path from E to any other vertex in the graph and a path from any other vertex back to E. Therefore, E is standing alone as a solo connected component.

Another interesting aspect of connected components in directed graph is drawing a meta graph, where every connected component is a node. The following is the meta-graph of the example provided:



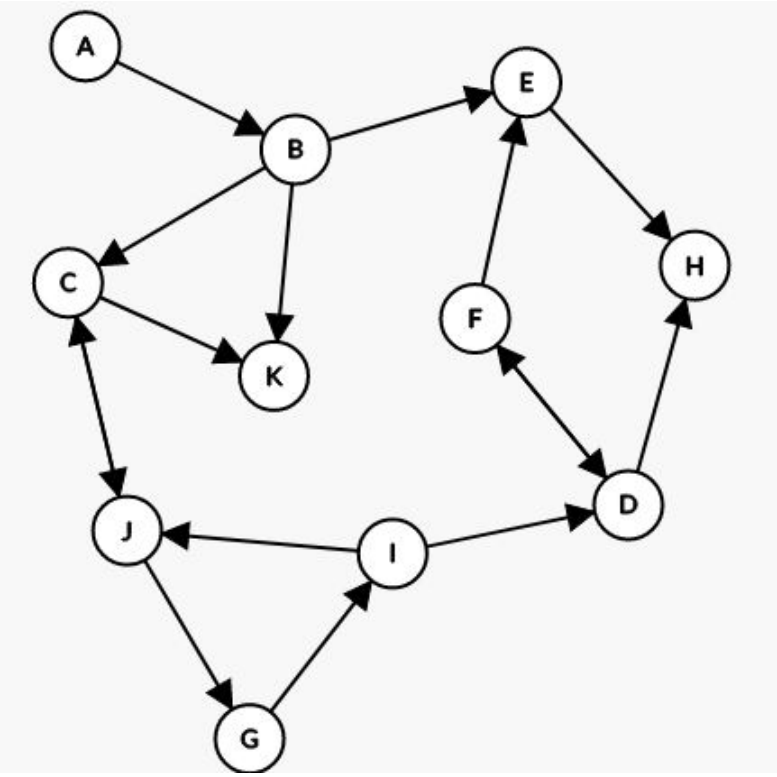
As we can see, the meta-graph of the stongly connected components of a directed graph is a dag, meaning a Directed Acyclic Graph. In this, we see each connected component is a node, and points to the proper connected components that it does in the original graph. Another thing to note is that because the meta-graph is a dag, it has source SCCs and sink SCCs. A source SCC is a connected component that has no incoming edges, only outgoing edges. Likewise, a sink SSC only has incoming nodes edges, never outgoing (I truly *love* the name "sink" SCC). In this case the source SCC is ABC and the sink SCC is DF. There can be multiple sources and sink SSCs in a metagraph, but no SSC

can be both a source *and* a sink SCC.

So, how do we find connected components? There's a relatively straightforward, linear time algorithm to do so:

1. Run depth first search on G^R (the graph with its edges reversed).
2. Run depth first search again on G (non-reversed graph), and during the DFS, process the verteces in decreasing order of their post numbers from step 1.

Exercise: Find the strongly connected components of the following graph. Provide the resulting metagraph, and list the source SCCs and sink SCCs.



Solution

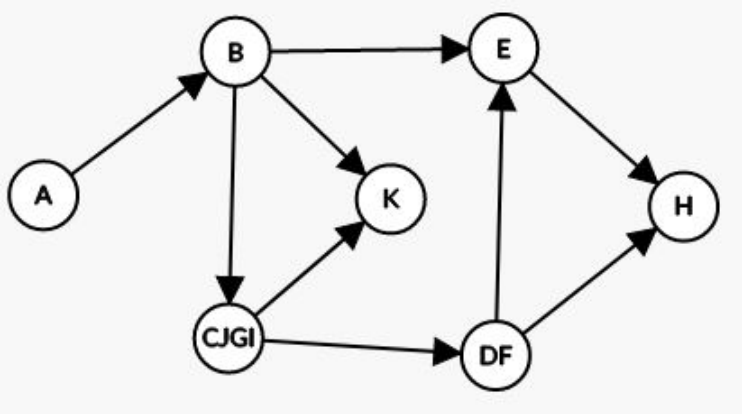
Because the work for this is mostly drawn using a pen and paper, I'll describe my process for following the strongly connected components algorithm for this graph.

- Firstly, I reversed all the edges in the graph to create G^R, starting from node A and prioritizing vertices alphabetically.
- Then, I conducted DFS on G^R, noting the pre- and post-process numbers

- Then, I conducted DFS again on G, prioritizing nodes with the highest post-process values and noted the number of what connected component it was.

For this graph, we see the connected components in order are: [K], [H], [D, FG], [E], [C, J, G, I], [B], [A].

The meta-graph is:



As we can see, the meta-graph is indeed a dag. There is one sink SCC, and that is [A]. Meanwhile, there are two sink SCCs, [K] and [H].

5 Graph Algorithms

5.1 Dijkstra's Algorithm

Explanation Dijkstra's algorithm (DA) is my absolute favorite. It solves a complex problem by taking a relatively simple process. What Dijkstra's does is it finds the shortest distance from any starting vertex to all other vertices in a weighted graph through an elegant modification of breadth first search. It does this through taking a "greedy" approach, choosing the locally shortest path to each vertex. DA also never backtracks. No time for that. It should be noted that Dijkstra's achieves this in $O(|V| \cdot |E|)$ time because it loops through all edges for each vertex, no matter if the vertex has been visited. This is because DA needs that information to determine if this new path is shorter than the previously taken path. One more thing to note is that DA only works on graphs where all edges have positive weights. More on that to come.

Let's examine the algorithm itself.

```
1. // Dijkstra's algorithm takes a graph
2. // lengths of edges, and a source vertex.
3. Dijkstras(G, l, s):
4.   for all u ∈ V
```