CS 3510 — Algorithms, Spring 2022

Lecture 13: Strongly Connected Components

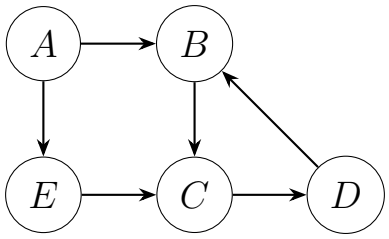
March 8, 2022

Lecturer: Frederic Faulkner *Scribe: Aditya Diwakar*

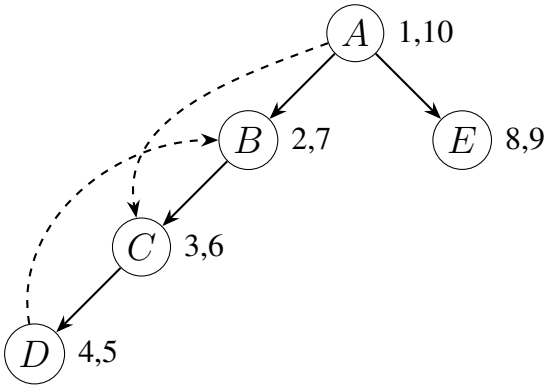
1 Quick Review

The DFS algorithm (pseudocode in the previous lecture) utilizes an explore method. The explore method will visit all nodes that are reachable from a node v (when called from v) and we can therefore find components a graph (using DFS).

We also defined an `explore` function that keeps track of pre and post numbers. The pseudocode and detailed explanation/walkthrough of an example is also in the previous lecture. For the sake of clarity, here is another example:



We can then generate the following DFS tree, see the table on the next page for how this is generated.

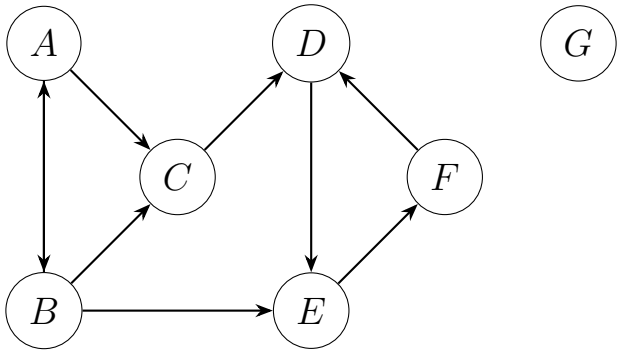


Function Call	Pre/Post Numbers	Clock	Stack
`explore(A)`	$pre = 1$	2	A, B
`explore(B)`	$pre = 2$	3	A, B, C
`explore(C)`	$pre = 3$	4	A, B, C, D
`explore(D)`	$pre = 4$	5	A, B, C, D
`explore(D)`	$pre = 4, post = 5$	6	A, B, C
`explore(C)`	$pre = 3, post = 6$	7	A, B
`explore(B)`	$pre = 2, post = 7$	8	A
`explore(A)`	$pre = 1$	8	A, E
`explore(E)`	$pre = 8$	9	A, E
`explore(E)`	$pre = 8, post = 9$	10	A
`explore(A)`	$pre = 1, post = 10$	11	`Done`

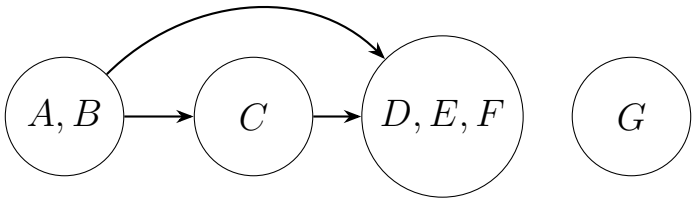
2 Components

A component is an undirected graph where from any vertex v, we can get to every vertex u in that component. In an undirected graph, any set of vertices that are connected (not a disconnected graph) is a component as every vertex is reachable since all edges are bidirectional.

In a directed graph, we are more curious about strongly connected components (SCCs) where all vertices can reach other vertex in that SCC. For example:



In the graph above, we can make the following SCCs: $\{A, B\}, \{C\}, \{D, E, F\}, \{G\}$. By finding these maximal SCCs, we can make something known as a metagraph like so:



Although it may be easy to look at graphs as humans and find strongly connected components, but how can we adapt DFS to label different SCCs?

Theorem 2.1. *If v is in a sink meta-node of the metagraph, then* `explore(v)` *will only explore vertices in the SCC of v.*

With this in mind, running `explore` on a sink vertex will find everything in that sink (from the metagraph). This is one of the SCCs. Once we run `explore`, we repeat the process *after* removing that sink (such that there is a new sink).

Theorem 2.2. *The vertex with the highest post # is in a source meta-node.*

Armed with this fact, we know that if we reverse the edges of G, then the source components become the sink components and therefore the vertex with the highest post # is the source for the reversed graph and a **sink** for the original graph.

Hence, an algorithm can be given (taking input G):

1. Reverse G to form G^R
2. Run DFS on G^R
3. L is the sorted list of post numbers
4. Find highest post number vertex v
5. Run `explore(v)`
6. Delete all visited vertices from L and G
7. Go-to step 4 as long as L is non-empty

The running time of $\mathcal{O}(|V| + |E|)$ as running DFS takes $\mathcal{O}(|V| + |E|)$, sorting by the post numbers can take $\mathcal{O}(|V| \log |V|)$ and steps 4-7 take $\mathcal{O}(|V| + |E|)$ as we are simply exploring the entire graph (and never seeing the same vertex more than once as we are removing it from L and G).

Hence, we have $\mathcal{O}(|V| + |E|) + \mathcal{O}(|V| + |E|) + \mathcal{O}(|V| \log |V|)$. Since $|E| = \mathcal{O}(|V|^2)$, then this adds up to a total runtime of $\mathcal{O}(|V| + |E|)$.

3 Challenges

1. In some town, all the roads are turned into a one way street. Mayor claims that you can get from any point in the city to another point in the city. How do we test this claim in linear time?

   Solution: Run the SCC algorithm and check there is only a single (SCC). By definition of SCC, every vertex in the SCC should be able to reach every other vertex which is the problem statement.

2. Can we reach any point in the town from the town hall? How can we test this also in linear time?

   Solution: Run `explore` on the town hall and see if the visited set is equal to the rest of the vertices. This means that every vertex is reachable.

3. And yet another claim, the mayor states anywhere you can drive from town hall, you can drive back to town hall. How can we also test this in linear time?

   Solution: Run the SCC algorithm and determine whether or not the town hall in a sink SCC. If it a sink SCC, then it means we can always get back to town hall (as this is a sink). If it is not a sink, it is possible to leave the town hall but never get back.

Remark: Linear time means $\mathcal{O}(|V| + |E|)$.