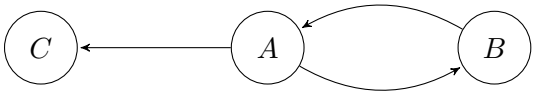
CMPSC 465 Spring 2022 | Data Structures & Algorithms Chunhao Wang and Mingfu Shao | Solution 06

1. (10 pts.) Consider this algorithm to find all connected components of a directed graph G: run DFS-with-timing on G to get the postlist (i.e., the list of vertices in decreasing post value), and then use the reverse of the postlist as a "magic ordering" to run the DFS algorithm on graph G. Design an instance to demonstrate that this algorithm is incorrect. Specifically, you will need to give a directed graph, then run above algorithm and show that the resulting $visited$ array does not give the correct connected components of G.

Solution: Consider this graph G.



DFS-with-timing on G with the alphabetical order gives us the postlist (A, C, B). If we redo DFS with the reverse of the postlist (B, C, A) as a "magic ordering", it will start DFS with B and the resulting $visited$ array is an array of all 1's. It means that $\{A, B, C\}$ is a connected component, which is not a true.

2. (10 pts.) Given a directed graph $G = (V, E)$, a vertex $t \in V$, and an edge $e = (u, v) \in E$, design an $O(|V| + |E|)$ time algorithm to determine whether there exists a cycle in G that contains both t and e.

Solution: We have that there exists a cycle in G that contains both t and e if and only if there exists a path from v to t and then from t to u in G. Therefore, we can design the following algorithm.

Algorithm. We run explore twice. The first run decides whether there exists a path from v to t in G. The second run decides whether there exists a path from t to u in G. For the first run, we run explore (G, v). If in the resulting visited array we have visited[t] = 1, (i.e., t can be reached from v in G), then we continue the second run; otherwise the algorithm terminates and return that such cycle does not exist. For the second run, we run explore (G, t). If in the resulting visited array we have visited[u] = 1, (i.e., u can be reached from t in G), then the algorithm returns that such cycle exists; otherwise there does not exist such a cycle.

Running Time. Initiation of visited array runs $O(|V|)$ time. Run explore (G, v) and explore (G, t) both take $O(|E|)$ time. Therefore, the entire algorithm runs in $O(|V| + |E|)$ time.

3. (10 pts.) Design an algorithm runs in $O(|V| + |E|)$ time which takes a directed graph as the input and determines if there is a vertex such that all other vertices are reachable from it.

Solution:
We can make the following observations before designing an algorithm:

- **If such vertex exists, it must be in the connected component that is a source in the meta-graph** (which we'll call source connected component from now on). If it's in any non-source connected component, it wouldn't be able to reach to any vertex in a source connected component which contradicts with the requirement that all other vertices are reachable from that particular vertex.
- **There can only be one source connected component.** If there are multiple source connected components, a vertex in one source connected component wouldn't be able to reach a vertex in another source connected component which contradicts with the requirement as well.

With these observations, the algorithm simply needs to check if there is only one source connected component. There are multiple ways to do that:

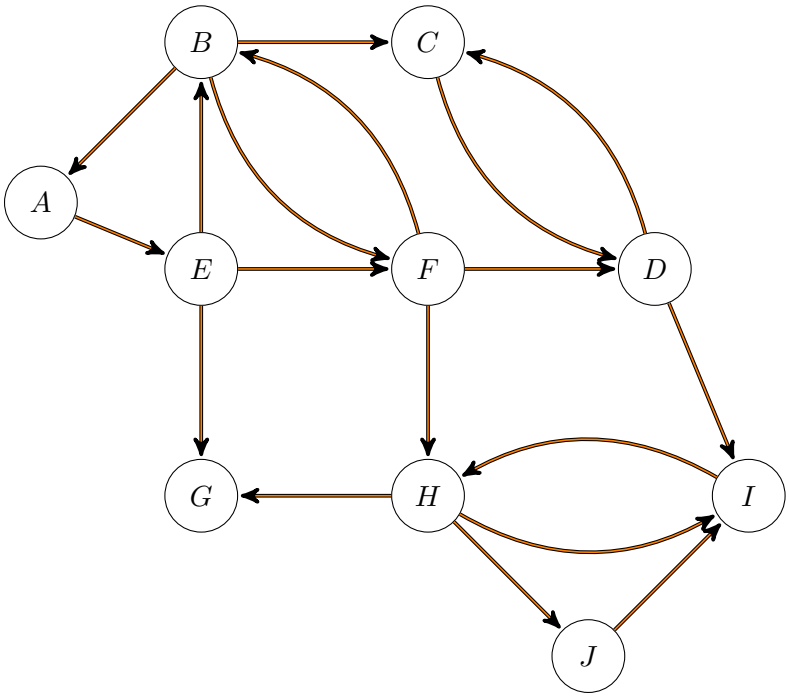
- Possible algorithm 1
  1. Based on Claim 1 in lecture A11 typed note and the fact that any meta-graph is a DAG, we know that the vertex with the largest post number is in a source connected component after running DFS-with-timing. So, we do this and get that vertex i.e. the first vertex in the postlist. Time complexity: $\Theta(|V| + |E|)$. DFS-with-timing simply records pre, post numbers for each vertex and creates a postlist as it does DFS, so it still has the same running time as the regular DFS.
  2. Run *explore* on that vertex. If all other vertices are visited in the procedure, the algorithm returns True. Otherwise, we know there is certain connected component that the source connected component containing that vertex can't reach. Since any meta-graph is a DAG, there must be at least one source connected component. Thus, it must be the case where there are multiple source components, and the algorithm can safely return False. Time complexity: $O(|V| + |E|)$.

  Overall, the running time is $O(|V| + |E|)$.
- Possible algorithm 2
  1. Run the special version of DFS used to obtain all conncected components i.e. the algorithm mentioned in page 4 of lecture A11 typed note. Time complexity: $\Theta(|V| + |E|)$.
  2. Go through array *visited* to find the vertices assigned with the largest $num\text{-}cc$. They consist of a source connected component. Time complexity: $\Theta(|V|)$.
     Correctness: The connected component containing these vertices is the last connected component we found. This implies that if we find all connected components by iteratively removing a sink connected component in the graph (as mentioned in page 3 of lecture A10 typed note), this particular connected component would be the last connected component remaining. Essentially, it is a source connected component since all non-source connected component would be removed before a source connected component.
  3. Same as step 2 in possible algorithm 1. Pick any vertex we found in the previous step and run *explore* on that vertex.

  Overall, the running time is $O(|V| + |E|)$.
- Possible algorithm 3
  1. Same as step 1 in possible algorithm 2.
  2. Construct the meta-graph for the given graph. More specifically, $V_m = \{C_1, \ldots, C_n\}$, assuming $num\text{-}cc = n$ in the end. For all $(u, v) \in E$, add (C_u, C_v) to E_m, assuming C_u is the connected component containing u and C_v is the connected component containing v. Time complexity: $\Theta(|E|)$. Note that we don't need to go through array *visited* or V.
  3. Go through E_m to find the in-degree of all vertices in the meta-graph. The vertices with in-degree 0 represent source connected components. If there is only one vertex with in-degree 0, the algorithm returns True. Otherwise, it returns False. Time complexity: $O(|E|)$ since $|E_m| \leq |E|$.

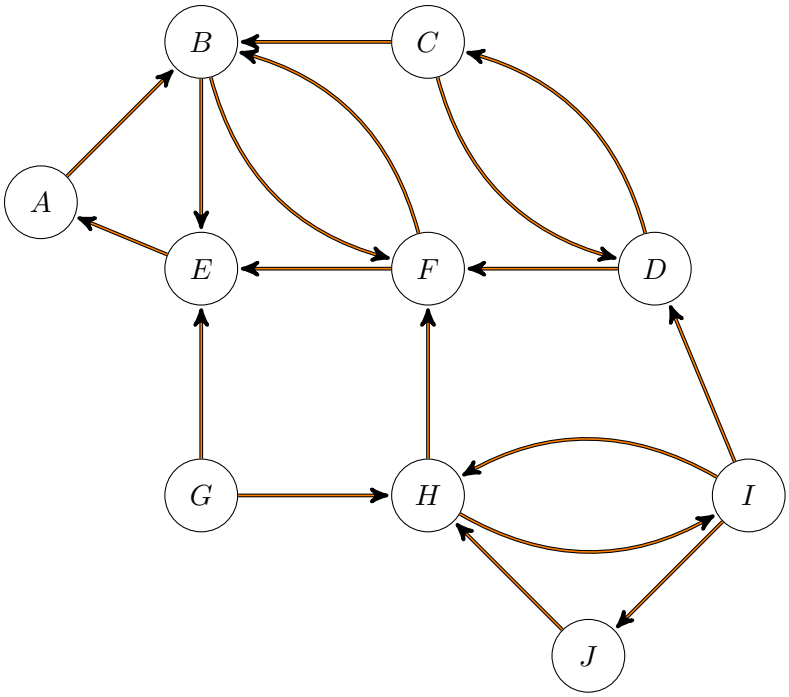
  Overall, the running time is $O(|V| + |E|)$.

4. (12 pts.) Run the strongly connected components algorithm on the following directed graphs G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

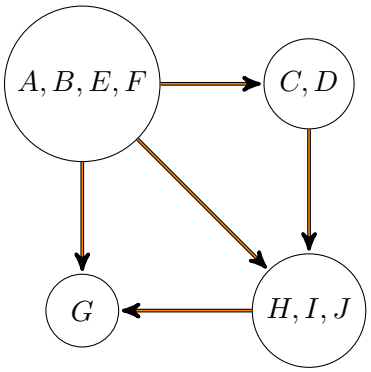


1. Give the pre and post number of each vertex in the reverse graph G^R.
2. In what order are the connected components found?
3. Which are source connected components and which are sink connected components?
4. Draw the "metagraph" (each meta-node is a connected component of G).
5. What is the minimum number of edges you must add to G to make it strongly connected?

Solution: Reverse graph G^R is as follows,



1. pre and post numbers in G^R
   A : 1, 8
   B : 2, 7
   C : 9, 12
   D : 10, 11
   E : 3, 4
   F : 5, 6
   G : 13, 20
   H : 14, 19
   I : 15, 18
   J : 16, 17
2. arranging the vertices in the decreasing order of post numbers,
   G, H, I, J, C, D, A, B, F, E
   Performing DFS on the given graph with above order will give us the connected components as follows,
   {G}, {H, I, J}, {C, D}, {A, B, E, F}
3. Source connected components: {A, B, E, F}
   Sink connected components: {G}
4. Meta graph of the given graph is as follows,



5. A graph is (strongly) connected means the meta-graph consists of a single vertex. Therefore, we should add edges from all sink components to all source components. Specifically, we can pick one vertex u in a sink component and pick one vertex v in a source component and add an edge (u, v), and we should do this for every pair of sink component and source component. In this scenario there is only one sink component and one source component, So making an edge from G to one of the four vertices in $\{A, B, C, D\}$ will make the resulting graph connected.

Rubrics:

Problem 1, 10pts

- 5 points : Provided an appropriate counterexample, especially, the graph should be directed.
- 5 points : Provided an explanation with the postlist after the first DFS and the resulting $visited$ array.
- 1 point : I don't know how to answer this question.

Problem 2, 10pts

- 10 points : correct algorithm
  - 3 points : appropriate explanation about twice explore
  - 5 points : apply explore algorithm/DFS
  - 2 points : correct return statements (e.g. return true if both b1 and b2 are true; otherwise return false) and running time analysis.
- 1 point : I don't know how to answer this question

Problem 3, 10pts

- 2 points: Found out such vertex must be in a source connected component if it exists.
- 2 points: Found out the question is asking to find whether or not there is only one source connected component.
- 3 points: Provided a correct algorithm.
- 1 point: The provided algorithm has running time $O(|V| + |E|)$.
- 2 points: Provided a correct running time analysis.
- 1 point : I don't know how to answer this question.

Problem 4, 12pts

- 3 points : correct pre and post numbers of the reverse graph
- 3 points : correct connected components and their order.
- 1 point : correct source connected component ($\{$A, B, E, F$\}$).
- 1 point : correct sink connected component ($\{$G$\}$).
- 2 points : correct metagraph with 4 nodes and 5 edges.
- 1 point : one or more edges missing in the metagraph
- 2 point : identified that one edge need to be added (from sink to source).
- 1.2 point : I don't know how to answer this question.