

CSE 101: Homework #2

Due on Apr 17, 2024 at 23:59pm

Professor Jones

Ray Tsai

A16848188

Problem 1

Run the SCC algorithm on the following directed graph G . When doing DFS on G^R : whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

$A : D, G$

$B : F, G, L$

$C : B, E$

$D : G, H$

$E : A, J$

$F : B$

$G : H$

$H : B, L$

$I : K$

$J : F, L$

$K : D$

$L : E, H, K$

(a) In what order are the strongly connected components (SCCs) found?

Proof. We first run DFS on G^R and get the post numbers:

A	B	C	D	E	F	G	H	I	J	K	L
24	21	4	17	23	10	19	20	15	9	16	22

Then, we run the undirected connected components algorithm on G by descending post order:

A	B	C	D	E	F	G	H	I	J	K	L
1	1	3	1	1	1	1	1	2	1	1	1

Hence, we find the following strongly connected components in the following sequence:

$$\{A, B, D, E, F, G, H, J, K, L\}, \{I\}, \{C\}.$$

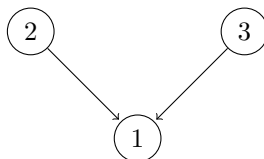
□

(b) Which are source SCCs and which are sink SCCs?

Proof. Since $\{A, B, D, E, F, G, H, J, K, L\}$ does not have outgoing edges, it is a sink. Since SCCs $\{I\}$ and $\{C\}$ don't have incoming edges, they are sources. □

(c) Draw the “metagraph” (each meta-node is an SCC of G)

Proof. The following is the metagraph of G , where node 1 represents $\{A, B, D, E, F, G, H, J, K, L\}$, node 2 represents $\{I\}$, and node 3 represents $\{C\}$.



□

Problem 2

Consider the following problem:

Given a strongly connected simple *directed* graph G , determine the total number of cycles in the graph.

Consider the following algorithm that claims to compute the total number of cycles in the graph.

For each algorithm,

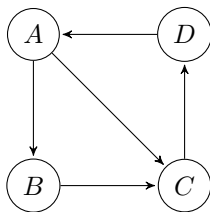
- Provide a runtime analysis (Based on $|V|$ and $|E|$)
- identify if it correctly solves the problem.
- If it is correct, provide a correctness proof. If it is not correct, provide a counterexample.

1. **Algorithm1**(G ; a strongly connected simple directed graph G .)

1. Run **DFS**(G)
2. $c = 0$
3. **for** each edge (u, v) **then**
4. **if** $post(v) > post(u)$ **then**
5. $c = c + 1$
6. **return** c

Proof. We first give a runtime analysis to this algorithm. We already know DFS takes $O(|V| + |E|)$ time. Following the DFS is a loop which iterates over all edges, which takes an additional $O(|E|)$ time. Hence, the runtime complexity of this algorithm is $O(|V| + 2|E|) = O(|V| + |E|)$.

However, the algorithm is incorrect. Consider the following graph:



Note that the graph is simple and strongly connected, as there is a directed hamiltonian cycle. Performing DFS on this graph starting from A yields the following `post` numbers:

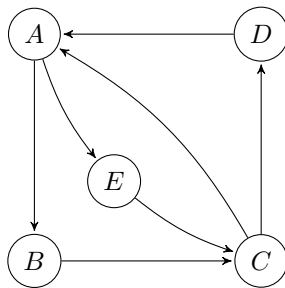
A	B	C	D
8	7	6	5

(D, A) is the only edge that meets the condition at step 4, so the algorithm outputs 1. But then there are two cycles in the graph, namely $A \rightarrow C \rightarrow D \rightarrow A$ and $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$. \square

2. **Algorithm2**(G ; a strongly connected simple directed graph G .)
 [[run graphsearch and every time you encounter a vertex you have already seen before, increment your counter, c .]]
1. $c = 0$
 2. **for all** $v \in V$:
 3. Status(v) = **U**
 4. Pick any vertex s
 5. Status(s) = **F**
 6. Initialize a Stack: $F = [s]$
 7. **while** $|F| > 0$
 8. $w = \text{pop}(F)$.
 9. For each outgoing neighbor y of w (for each $(w, y) \in E$):
 10. **if** Status(y) \neq **U**:
 11. $c = c + 1$
 12. **else:**
 13. Status(y) = **F**
 14. push(F, y)
 15. Status(w) = **X**
 16. **return** c

Proof. Since this algorithm is just graphsearch with a counter, the time complexity of the algorithm is the same as a standard graphsearch, which takes $O(|V| + |E|)$ time.

However, this algorithm is incorrect. Consider the following graph:



The graph is obviously simple and strongly connected. We first note that the graph has 4 cycles, namely $A \rightarrow B \rightarrow C \rightarrow A$, $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$, $A \rightarrow E \rightarrow C \rightarrow A$, $A \rightarrow E \rightarrow C \rightarrow D \rightarrow A$.

Suppose that the algorithm starts at vertex A . Here is the order of edges the for loop in the algorithm will check:

$$(A, B) \rightarrow (A, E) \rightarrow (B, C) \rightarrow (C, \mathbf{A}) \rightarrow (C, D) \rightarrow (D, \mathbf{A}) \rightarrow (E, \mathbf{C}).$$

The bolded vertices represents the already visited vertices, which causes the counter to increment. But then the algorithm returns 3, which is not the number of cycles in the graph. \square

Problem 3

You are given a simple directed graph G with vertex set V , edge set E and vertex labels $L(v) \in \{0, 1\}$ as well as a starting and ending vertex s, t .

Design a reasonably efficient algorithm that determines if there is a walk from s to t such that the sequence of vertex labels in the walk have exactly one occurrence of two 1's in a row.

Proof. Consider the following algorithm:

Create a graph G' in the following way:

for each vertex v in G , create two copies v', v'' in G' . For each edge (x, y) in G ,

- If $L(x) == 1$ and $L(y) == 1$, create an edge (x', y'') in G'
- otherwise, create 2 edges $(x', y'), (x'', y'')$ in G' .

Then, run explore in G' from s' . Return TRUE if t'' is visited, and return FALSE otherwise.

We now give a justification of correctness. Suppose that the algorithm returns TRUE. Then, there is a path s' to t'' in G' . By construction, there is no edge between v' and v'' in G' , so P' is of the form $P' = (s', v'_1, \dots, v'_k, u''_1, \dots, u''_j, t'')$. We now map each vertex in the path back to the corresponding vertex in $V(G)$ (by removing the apostrophes) and obtain a new path $P \subseteq G$ from s to t , with exactly one edge (v_k, u_1) having two vertices labelled 1.

We now prove the converse. Suppose that there is a walk in G from s to t with exactly one occurrence of two 1's in a row. Then, the walk can be condensed to a path P of the form $(s, v_1, \dots, v_k, u_1, \dots, u_j, t)$, with (v_k, u_1) being the only edge such that $L(v_k) == L(u_1) == 1$. By construction, each edge $(s', v'_1), \dots, (v_{k-1}, v_k), (u''_1, u''_2), \dots, (u''_j, t'')$ are in G' . But then since $L(v_k) == L(u_1) == 1$, (v'_k, u''_1) is also in G' , which makes $P' = (s', v'_1, \dots, v'_k, u''_1, \dots, u''_j, t'')$ a path in G' . Hence, the algorithm returns TRUE.

We now give a runtime analysis of the algorithm. It takes $O(|V|)$ time to create copies of vertices and $O(|E|)$ to create the edges in G' . Running explore on G' has runtime $O(|V| + |E|)$. Hence, the total runtime of the algorithm is $O(2|V| + 2|E|) = O(|V| + |E|)$. \square

Problem 4

You are given a directed graph.

Design a reasonably efficient algorithm that *determines* if there exists a walk that goes through each vertex at least once.

Proof. Consider the following algorithm:

```

1. hasWalk = 1
2. cc = 0
3. Run SCC( $G$ )
4. for  $i = 1; i \leq cc; i++$  then
5.   for  $j = 1; j \leq cc; j++$  then
6.     Meta[ $i$ ][ $j$ ] = 0
7.   for each edge  $(u, v)$  then
8.     Meta[ccnum[ $u$ ], ccnum[ $v$ ]] = 1
9.   for  $i = 1; i \leq cc; i++$  then
10.    hasWalk *= Meta[ $i + 1, i$ ]
11. return hasWalk == 1

```

Given directed graph G , the algorithm constructs the metagraph M of G and check if there exists a path $P \subseteq M$ which chains all vertices of M in decreasing ccnum value.

We now check that if the algorithm correctly determines the existence of the desired walk. Note that a strongly connect component contains such a walk, as there exists a path between any ordered pair of vertices. In particular, for any s, t in a strongly connected component, there exists a path from s to t which passes through all vertices in the component. Hence, there exists such a walk in G if and only if there exists a walk which passes through all vertices in the metagraph M . But then the M has no cycles, so there exists such a walk in M if and only if there is a path P which passes through all vertices in M . It remains to show that P exists if and only if $(i + 1, i) \in E(M)$ for all $i \in V(M)$, $i \neq cc$. One direction is obvious, so we only need to show the existence of P implies $(i + 1, i) \in E(M)$ for all $i \in V(M)$, $i \neq cc$. Notice that the sink of P must be $u = \min V(M)$, otherwise u has an incoming edge, contradicting the nature of the SCC algorithm. Remove u from P . By the nature of the SCC algorithm, the sink of $P \setminus \{u\}$ is the next smallest element in $V(M)$, namely $u + 1$. Hence, we may recursively remove the sink of P to get the next smallest element in $V(M)$, and thus $(i, i + 1) \in E(M)$, for all $1 \leq i < cc$. Therefore, the algorithm returns TRUE if there exists a walk that goes through each vertex at least once and returns FALSE otherwise.

We now give a runtime analysis of the algorithm. We already know the SCC algorithm takes $O(|V| + |E|)$ time. The nested loop at step 4-6 iterates through all pairs of components, which is $O(|V|^2)$ at the very worst. The loop at step 7 iterates through all edges, so it takes $O(|E|)$ time. Finally, the last loop simply loops through all components, so it is $O(|V|)$ at the very worst. Hence, the algorithm has a runtime of $O(|V|^2)$. \square