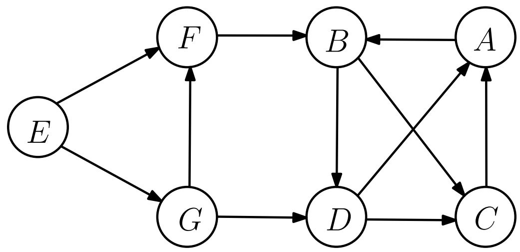
COMP 3804 — Solutions Tutorial February 16

Algorithm DFS(G):
for each vertex v
do $visited(v) = false$
endfor;
$clock = 1$;
for each vertex v
do if $visited(v) = false$
    then Explore(v)
    endif
endfor

Algorithm Explore(v):
$visited(v) = true$;
$pre(v) = clock$;
$clock = clock + 1$;
for each edge (v, u)
do if $visited(u) = false$
    then Explore(u)
    endif
endfor;
$post(v) = clock$;
$clock = clock + 1$

Problem 1: Consider the following directed graph:

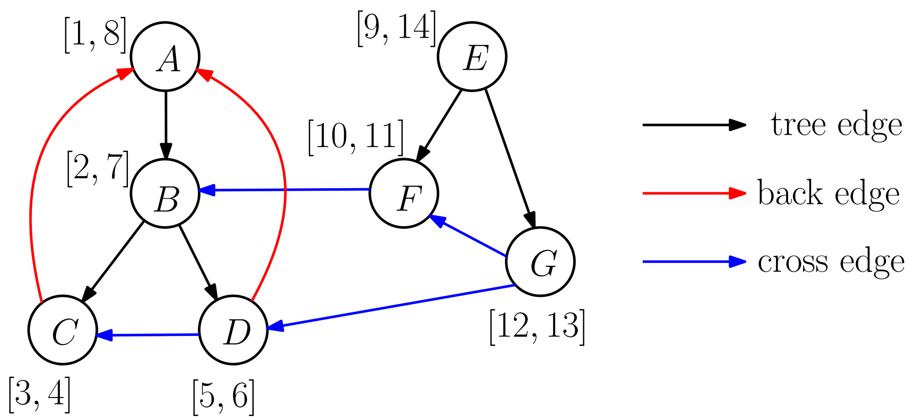


(1.1) Draw the *DFS*-forest obtained by running algorithm DFS. Classify each edge as a tree edge, forward edge, back edge, or cross edge. In the *DFS*-forest, give the *pre*- and *post*-number of each vertex. Whenever there is a choice of vertices, pick the one that is alphabetically first.

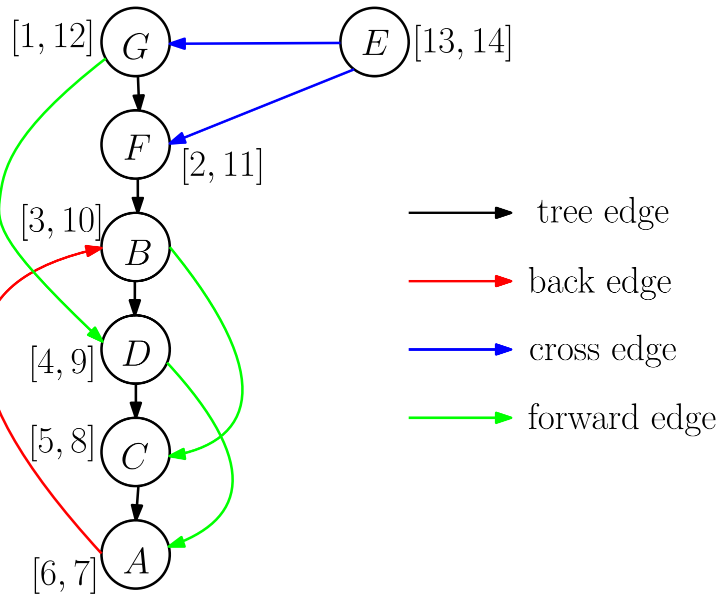
(1.2) Draw the *DFS*-forest obtained by running algorithm DFS. Classify each edge as a tree edge, forward edge, back edge, or cross edge. In the *DFS*-forest, give the *pre*- and *post*-number of each vertex. Whenever there is a choice of vertices, pick the one that is alphabetically last.

Solution:

We start with **(1.1)**. In case there is more than one choice, we pick the alphabetically smallest one. Thus, algorithm DFS(G) starts by calling EXPLORE(A). Here is the resulting *DFS*-forest:



Next we do **(1.2)**. In case there is more than one choice, we pick the alphabetically largest one. Thus, algorithm DFS(G) starts by calling EXPLORE(G). Here is the resulting *DFS*-forest:



Problem 2: Let $G = (V, E)$ be a directed acyclic graph, and let s and t be two vertices of V.

Describe an algorithm that computes, in $O(|V|+|E|)$ time, the number of directed paths from s to t in G. As always, justify your answer and the running time of your algorithm.

Solution: We start by computing a topological sorting $v_1, v_2, \ldots, v_n$ of the vertex set. Recall that for each edge (v_i, v_j) in E, $i < j$. In other words, if we draw the vertices, in the given order, on a line, then all edges go from left to right.

If s is to the right of t in the topological sorting, then there is no directed path from s to t. Thus, we assume that s is to the left of t.

We may assume that $s = v_1$ and $t = v_n$. (If, for example, $s = v_7$, then we can remove $v_1, \ldots, v_6$, and renumber the remaining vertices. Similarly, if, for example, $t = v_{n-12}$, then we can remove $v_{n-11}, \ldots, v_n$, and renumber the remaining vertices.)

We define $P(1) = 0$ and, for each i with $2 \leq i \leq n$, $P(i)$ to be the number of directed paths from s to v_i in G. Our task is to compute $P(n)$.

For each i, let $\text{In}(i)$ be the set of indices j such that (v_j, v_i) is an edge in E. Note that $j < i$ for each such edge. The main observation is that

$$P(1) = 1$$

and for each i with $2 \leq i \leq n$,

$$P(i) = \sum_{j \in \text{In}(i)} P(j).$$

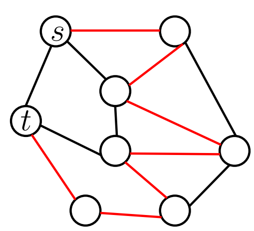
This suggests that we can compute $P(n)$ (this is the number we have to compute), by computing, in this order, $P(0), P(1), P(2), \ldots, P(n)$.

The algorithm does the following:

- Compute a topological sorting $v_1, v_2, \dots, v_n$ of the vertex set V. We have seen in class that this can be done in $O(|V| + |E|)$ time.
- Use Problem 3 from the February 9 tutorial to compute the list of incoming edges $\textsc{In}(i)$ for each vertex v_i. This takes $O(|V| + |E|)$ time.
- Initialize $P(1) = 0$. This takes $O(1)$ time.
- For $i = 2, 3, \dots, n$, do the following:
  - Initialize $P(i) = 0$;
  - For each index j in $\textsc{In}(i)$, set
  $$P(i) = P(i) + P(j).$$
  - This takes time
  $$O\left(1 + \sum_{i=2}^{n} (1 + |\textsc{In}(i)|)\right),$$
  which is $O(|V| + |E|)$.
- Return $P(n)$. This takes $O(1)$ time.

The total running time of the algorithm is $O(|V| + |E|)$.

Problem 3: A *Hamilton path* in an undirected graph is a path that contains every vertex exactly once. In the figure below, you see a Hamilton path in red. A *Hamilton cycle* is a cycle that contains every vertex exactly once. In the figure below, if you add the black edge $\{s, t\}$ to the red Hamilton path, then you obtain a Hamilton cycle.



If $G = (V, E)$ is an undirected graph, then the graph G^3 is defined as follows:

1. The vertex set of G^3 is equal to V.
2. For any two distinct vertices u and v in V, $\{u, v\}$ is an edge in G^3 if and only if there is a path in G between u and v consisting of at most three edges.

Question 3.1: Describe a *recursive* algorithm HamiltonPath that has the following specification:

> **Algorithm** HamiltonPath(T, u, v)**:**
> **Input:** A tree T with at least two vertices; two distinct vertices u and v in T such that $\{u, v\}$ is an edge in T.
> **Output:** A Hamilton path in T^3 that starts at vertex u and ends at vertex v.

Hint: You do not have to analyze the running time. The base case is easy. Now assume that T has at least three vertices. If you remove the edge $\{u, v\}$ from T, then you obtain two trees T_u (containing u) and T_v (containing v).

1. One of these two trees, say, T_u, may consist of the single vertex u. How does your recursive algorithm proceed?

2. If each of T_u and T_v has at least two vertices, how does your recursive algorithm proceed?

Solution: Algorithm HamiltonPath(T, u, v) does the following:

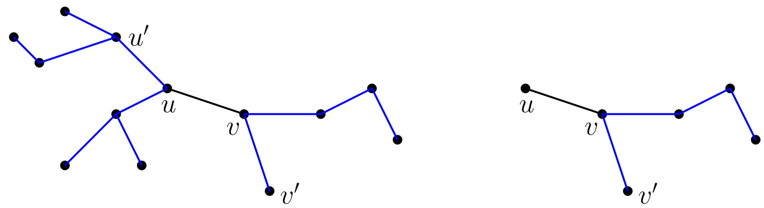
1. If T consists of two vertices: Return the path consisting of the single edge $\{u, v\}$.

2. If T has at least three vertices: Let T_u and T_v be the two trees obtained by removing the edge $\{u, v\}$ from T.

   (a) If each of T_u and T_v has at least two vertices (see the left figure below): Let u' be a neighbor of u in T_u, and let v' be a neighbor of v in T_v. Run algorithm HamiltonPath(T_u, u, u') and let P be the path returned; note that P is a Hamilton path in T_u^3 that starts at u and ends at u'. Run algorithm HamiltonPath(T_v, v', v) and let Q be the path returned; note that Q is a Hamilton path in T_v^3 that starts at v' and ends at v. Note that, since u' and v' have distance three in T, the edge $\{u', v'\}$ is in T^3. Thus, we return the path that starts by following P, then takes the edge $\{u', v'\}$, and then follows Q. This is a Hamilton path in T^3 that starts at u and ends at v.

   (b) If T_u consists of the single vertex u and T_v has at least two vertices (see the right figure below): Let v' be a neighbor of v in T_v. Run algorithm HamiltonPath(T_v, v', v) and let Q be the path returned; note that Q is a Hamilton path in T_v^3 that starts at v' and ends at v. Note that, since u and v' have distance two in T, the edge $\{u, v'\}$ is in T^3. Thus, we return the path that starts with the edge $\{u, v'\}$ and then follows Q. This is a Hamilton path in T^3 that starts at u and ends at v.

   (c) If T_u has at least two vertices and T_v consists of the single vertex v: Swap u and v and proceed as in the previous case.



Question 3.2: Prove the following lemma:

Lemma: For every tree T that has at least three vertices, the graph T^3 contains a Hamilton cycle.

Solution: Take an arbitrary edge $\{u, v\}$ in T. Algorithm $\textsc{HamiltonPath}(T, u, v)$ gives us a Hamilton path in T^3 that starts at u and ends at v. This path does not contain the edge $\{u, v\}$: This is because T has at least three vertices. If we connect the end-vertices u and v of this path using the edge $\{u, v\}$, then we obtain a Hamilton cycle in T^3.

Question 3.3: Prove the following theorem:

Theorem: For every connected undirected graph G that has at least three vertices, the graph G^3 contains a Hamilton cycle.

Solution: We run algorithm $DFS(G)$. Since G is connected, this gives us a spanning tree, say T, of G. We have seen above that T^3 contains a Hamilton cycle. Since T^3 is a subgraph of G^3, this is also a Hamilton cycle in G^3.