References

Collaborated with Crystal Huang.

Question 1: Tours

Suppose the New York Botanical Gardens are trying to drum up interest for their tree and shrub collections by offering tours through the arboretum, and you are tasked with reviewing their proposed routes. There is a set V of trees they want some tour to stop at, and a set E of routes from tree to tree their proposed tours would make. To make sure all the trees are visited, they want the following property:

Property 1. *For all pairs $v, u \in V$, we must have a path from u to v or a path from v to u (or both).*

They provide you with many proposed plans and a strict deadline for your feedback, so you want to develop an efficient algorithm to automate this task.

1. Suppose $G = (V, E)$ is a directed acyclic graph. Design a simple $O(|V| + |E|)$ time algorithm to determine whether or not the given graph G satisfies the desired property. For example, in Figure 1, the first example satisfies the property while the second does not (as there is no path from A to C nor from C to A). Argue the correctness of your algorithm.

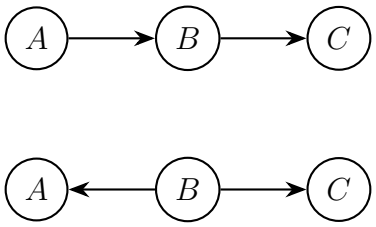

Figure 1: Illustrations for Question 1-1.

Solution:

Algorithm 1. Tours(G) with directed acyclic graph $G = (V, E)$; returns `true` if G satisfies Property 1; otherwise, `false`:

Assign a $|V|$-element array $T[1, \ldots, |V|] \leftarrow$ TopologicalSort(G). Here TopologicalSort is the well-known topological sort algorithm that returns an array containing the vertices of G in topological order. Note TopologicalSort has running time $O(|V| + |E|)$.

For $i = 2$ to $|V|$:

- if $(T[i-1], T[i]) \notin E$, via adjacency list, then return `false`.

Return `true`.

Proposition 1. *Claim.* Tours(G) has running time $O(|V| + |E|)$ for directed acyclic graph $G = (V, E)$.

Proof. Tours invokes TopologicalSort with running time $O(|V| + |E|)$. Then, the loop body runs for $|V| - 1$ iterations. On each iteration of the loop body, Tours searches the adjacency list.

The total number of edges across all $|V|$ vertices is $|E|$, giving an upper bound of $O(|V|+|E|)$ for the loop. Thus, TOURS has running time $O(|V|+|E|)$. □

Definition 1. Let $1 \leq j \leq |V|$ where $j = 1$ denotes the iteration before the loop begins, $j = i$ denotes the iteration before considering index i for $i \leq |V|$, and $j = |V|$ denotes the step immediately after the loop terminates.

Proposition 2. *Claim.* For all iterations $1 \leq j \leq |V|$, we claim that the induced subgraph containing only the vertices in $T[1, \ldots, j]$ satisfies Property 1; otherwise, Algorithm 1 terminates early and returns `false`.

Proof. We can prove this invariant by induction on j.

Basis. Consider $j = 1$. Note that we are considering the case before the first iteration of the loop. Of course, $T[1, \ldots, 1] = T[1]$. The induced subgraph containing only $T[1]$ does satisfy Property 1. So the invariant holds vacuously at initialization.

Hypothesis. Consider $j = k$ where $2 \leq k < |V|$. Assume that, at the beginning of iteration k, the induced subgraph containing only the vertices in $T[1, \ldots, k]$ satisfies Property 1.

Inductive step. Consider $j = k+1$. From the inductive hypothesis, there exists a path from $T[1]$ to $T[k]$. There are two cases:

- Suppose $(T[k], T[k+1]) \in E$. Then there exists a path from $T[1]$ to $T[k+1]$, noting $T[1]$ precedes $T[k]$ and $T[k]$ precedes $T[k+1]$ in the topological sort. Thus, Property 1 holds for the induced subgraph containing only the vertices in $T[1, \ldots, k+1]$.
- Suppose instead $(T[k], T[k+1]) \notin E$. Then Property 1 does not hold, and the Algorithm correctly returns `false`.

In all cases of the inductive step, the invariant holds.

If, at termination, $j = |V|$, then Algorithm 1 returns `true`, which is correct by the principle of mathematical induction.

The invariant holds at initialization, maintenance, and termination. Algorithm 1 is correct. □

2. Suppose now that $G = (V, E)$ is an arbitrary directed graph. Design a $O(|V| + |E|)$ time algorithm to determine whether or not the given graph G satisfies the desired property.

Solution: Algorithm 2. TOURS2(G) with directed graph $G = (V, E)$; returns `true` if G satisfies Property 1; otherwise, `false`:

Assign an n-element array $S[1, \ldots, n] \leftarrow$ STRONGLYCONNECTEDCOMPONENTS(G), where n is the number of strongly connected components in G. Here STRONGLYCONNECTEDCOMPONENTS is the well-known strongly connected components algorithm that returns a collection of sets of vertices, in topological order, representing the strongly connected components of G. Note STRONGLYCONNECTEDCOMPONENTS has running time $O(|V| + |E|)$.

For $i = 2$ to n:

- for $u \in S[i-1]$:
  - for $(u, v) \in E$, via adjacency list:
    - if $v \in S[i]$, continue to next i;
- return `false`;

Return `true`.

3. Prove that your algorithm in (2) is correct and that it runs in the required time.

Solution:

Proposition 3. *Claim.* TOURS2(G) has running time $O(|V|+|E|)$ for directed graph $G = (V, E)$.

Proof. TOURS2 invokes STRONGLYCONNECTEDCOMPONENTS(G) with running time $O(|V|+|E|)$.

Then, the outer loop body executes for $n - 1$ iterations. Since n is the number of strongly connected components, $n \leq |V|$.

For each iteration i of the outer loop, we check all vertices in $S[i]$. By construction of S, we know

$$\bigcup_{i=1}^{n} S[i] = V.$$

This means that the loop executes at most once for each vertex in V, which is $O(|V|)$. For each vertex visited, the entire adjacency list is read in the worst case. Across all vertices, this process is $O(|E|)$. We assume that sets support constant-time "contains" operations, so $v \notin S[i-1]$ is $O(1)$. This implies that the running time of the loop is $O(|V| + |E|)$.

Thus TOURS2 has running time $O(|V| + |E|)$. □

Proposition 4. *Claim.* TOURS2 is correct.

Proof. On each iteration j of the loop, we can show, by a similar inductive argument to that of Proposition 2, that there exists a path in G containing every vertex in

$$\bigcup_{i=1}^{j} S[i].$$

That is, G satisfies Property 1 on each iteration j; otherwise, Algorithm 2 terminates early and returns `false`.

This invariant relies on the observation that the StronglyConnectedComponents algorithm returns components in topological order.

Basis. Consider $j = 1$. Note that we are considering the case before the first iteration of the loop. Then $S[1]$ is strongly connected by definition, so the claim holds at initialization.

Hypothesis. Consider $j = k$ where $2 \leq k \leq n$. Assume that, at the beginning of iteration k, there exists a path in G containing every vertex in

$$\bigcup_{i=1}^{k} S[i].$$

Inductive step. Consider $j = k+1$. From the inductive hypothesis, there exists a path $(x_1, \ldots, x_p)$ connecting every vertex in

$$\bigcup_{i=1}^{k} S[i].$$

There are two cases:

- Suppose there exists an edge $(u, v) \in E$ for $u \in S[k], v \in S[k+1]$. Since $S[k+1]$ is strongly connected by definition, there exists a path $(v, y_1, \ldots, y_q)$ containing v and all other vertices $y_1, \ldots, y_q \in S[k+1]$. Since $(u, v) \in E$, there exists a path $(u, v, y_1, \ldots, y_q)$.

  Since $S[k]$ is strongly connected by definition, there exists a path $(x_1, \ldots, x_p, u)$. Therefore, there exists a path $(x_1, \ldots, x_p, u, v, y_1, \ldots, y_q)$ connecting every vertex in

  $$\bigcup_{i=1}^{k+1} S[i].$$

- Suppose instead there exists no such edge in E. Then, by construction of $S[k]$ and $S[k+1]$, there is no path connecting these components. We correctly conclude that G does not satisfy Property 1, and return `false`.

In all cases of the inductive step, the invariant holds.

If, at termination, $j = n$, then Algorithm 2 returns `true`, which is correct by the principle of mathematical induction.

The invariant holds at initialization, maintenance, and termination. Algorithm 2 is correct. □

Question 2: Topological sort

1. How many valid topological sorts does the directed graph G in Figure 2 below have? List all the valid topological sorts in the following table. One of them has been listed as an example, where node A has the last finish time and D has the first.

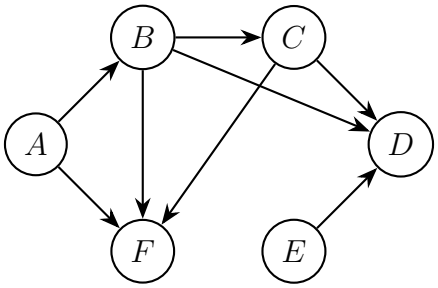

Figure 2: Directed G for topological sort.

1.	A	B	C	F	E	D
2.	A	B	C	E	D	F
3.	A	B	C	E	F	D
4.	A	B	E	C	D	F
5.	A	B	E	C	F	D
6.	A	E	B	C	D	F
7.	A	E	B	C	F	D
8.	E	A	B	C	D	F
9.	E	A	B	C	F	D

Solution: The solutions above are obtained by stepping through the topological sort algorithm in nine different ways.

2. Give an example of a graph showing that the topological sort algorithm does not work if we output vertices in order of their discovery time, instead of reverse finish time.

Solution: *False claim.* A topological sort algorithm is still correct if vertices are returned in order of discovery time.

Counterexample. Consider directed acyclic graph $G = (\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}, \{(\mathrm{B}, \mathrm{A}), (\mathrm{C}, \mathrm{A})\})$.

In a topological sort of G, B and C must precede A.

However, this modified algorithm returns B, then A, then C, which is not a valid topological sort. It could also return C, then A, then B, which is also incorrect. Hence disproven. □

Question 3: Spanning tree

Recall that for an undirected graph $G = (V, E)$, a spanning tree T is a subgraph containing all vertices V such that T is connected and acyclic.

In the following, let $e = \{u, v\} \in E$ be an edge that is *not* part of T. Prove the following properties:

1. Assume we add e to T, i.e., consider the graph T' over the vertices V that contains all edges from T and e. Show that this graph is no longer acyclic.

Solution:

Definitions. Let $G = (V, E)$ be an undirected graph. Let $T = (V, E_T)$ be a spanning tree of G, with $e \in E$ but $e \notin E_T$. Let graph $T' = (V, E_T \cup \{e\})$.

Proposition I.

Claim. T' is not acyclic.

Proof. Note $e = \{u, v\}$ for $u, v \in V$. Since T is a spanning tree, T is connected and contains u and v. Thus, there exists a path $(u, \ldots, v)$ in T. Of course $e = \{u, v\} \notin E_T$, so there exists a path $(u, w, \ldots, v)$ or $(u, \ldots, w, v)$ in T via some other vertex $w \in V$. Assume, without loss of generality, that this path is $(u, w, \ldots, v)$ in T. Since T' includes edge $e = \{u, v\}$, there exists a cycle $(u, w, \ldots, v, u)$ in T'. Ergo T' is not acyclic. □

2. Now let e' be an arbitrary edge on the cycle created by adding e to T' in part (1). Let T'' be the graph obtained by removing e', i.e., the graph obtained by replacing e' with e in the original spanning tree T. Show that T'' is connected.

Solution:

Proposition II.

Claim. From Proposition I, there exists a cycle C in T'. Let $e' \in (E_T \cup \{e\})$ be an edge on C.

Let graph $T'' = (V, (E_T \cup \{e\}) \setminus \{e'\})$. Then T'' is connected.

Proof. Since T is a tree, T is connected. Since T' is induced by adding an edge to T, T' is connected. Either $e = e'$ or $e \neq e'$.

- Suppose $e = e'$. Then $T'' = (V, E_T) = T$. So T'' is connected.
- Suppose instead $e \neq e'$. Note $e = \{u, v\}$ and $e' = \{x, y\}$ for $u, v, x, y \in V$. Now T is a tree, and thus acyclic; but, T' contains cycle C, so e must be on C. By definition, e' is on C. Therefore, $C = (u, \ldots, x, y, \ldots, v, u)$ or $C = (v, \ldots, x, y, \ldots, u, v)$. Assume, without loss of generality, that $C = (u, \ldots, x, y, \ldots, v, u)$ in T'. Although edge $\{x, y\}$ is not in T'', severing that edge from C implies that paths $(x, \ldots, u)$ and $(v, \ldots, y)$ do exist in T''. Since edge $e = \{u, v\}$ does exist in T'', there exists a path $(x, \ldots, u, v, \ldots, y)$ in T''. All other vertices in T'' are connected because T' is connected. So T'' is connected.

Ergo T'' is connected. □

3. Now complete the proof of T'' being a spanning tree by also showing that T'' is acyclic.

Solution:

Proposition III.

Claim. From Proposition I, there exists a cycle C in T'. Let $e' \in (E_T \cup \{e\})$ be an edge on C. Let graph $T'' = (V, (E_T \cup \{e\}) \setminus \{e'\})$. Then T'' is acyclic.

Proof. Either $e = e'$ or $e \neq e'$.

- Suppose $e = e'$. Then $T'' = (V, E_T) = T$. Since T is a tree, T is acyclic. So T'' is acyclic.
- Suppose instead $e \neq e'$. Note $e = \{u, v\}$ and $e' = \{x, y\}$ for $u, v, x, y \in V$. Now T is a tree, and thus acyclic; but, T' contains cycle C, so e must be on C. By definition, e' is on C. Therefore, $C = (u, \ldots, x, y, \ldots, v, u)$ or $C = (v, \ldots, x, y, \ldots, u, v)$. Assume, without loss of generality, that $C = (u, \ldots, x, y, \ldots, v, u)$ in T'. Since edge $\{x, y\}$ is not in T'', we know C does not exist in T''. There is no other cycle in T' because T is acyclic. Since C is not in T'', we know T'' is acyclic.

Ergo T'' is acyclic. $\square$

Proposition IV.

Claim. From Proposition I, there exists a cycle C in T'. Let $e' \in (E_T \cup \{e\})$ be an edge on C. Let graph $T'' = (V, (E_T \cup \{e\}) \setminus \{e'\})$. Then T'' is a spanning tree.

Proof. From Proposition II, T'' is connected. From Proposition III, T'' is acyclic. Therefore, T'' is a tree. By construction, T'' is an induced subgraph of G containing all vertices $v \in V$. Ergo T'' is a spanning tree of G. $\square$