Homework 2 — Graph Algorithms

Due: October 20th

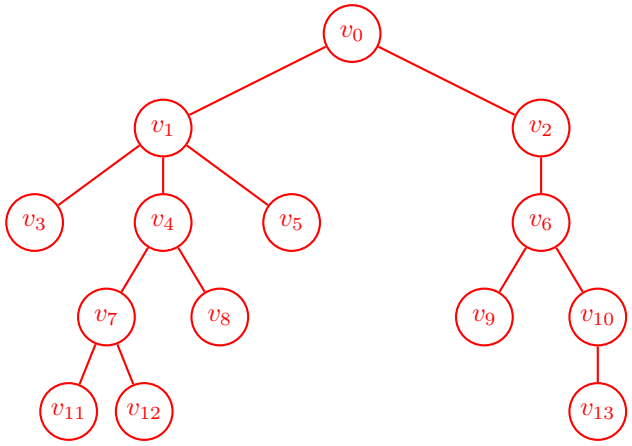
A few of these questions may appear on or help you prepare for the quiz on **October 20th**.
Note that these selected solutions and comments do not necessarily represent complete answers.

1. A *preordering* is a sequence of vertices, sorted in ascending order of their previsit numbers. Similarly, a *postordering* is a sequence of vertices sorted by postvisit number.

   Suppose that you are given the following sequences:

   - Preordering: $(v_0, v_1, v_3, v_4, v_7, v_{11}, v_{12}, v_8, v_5, v_2, v_6, v_9, v_{10}, v_{13})$
   - Postordering: $(v_3, v_{11}, v_{12}, v_7, v_8, v_4, v_5, v_1, v_9, v_{13}, v_{10}, v_6, v_2, v_0)$

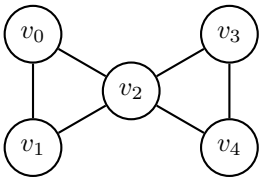
   Reconstruct a spanning tree returned by a depth-first search that produced these orderings.



2. Consider the following statement:

   Proposition: Let $G = (V, E)$ be a connected graph such that $|V| \geq 1$. There exists a vertex $v \in V$ such that $G' = (V - \{v\}, E)$ is connected.

   That is, in any connected, non-empty graph, there exists a vertex whose removal does not disconnect the graph. For example, given:



   …removing v_0 does not disconnect the graph, whereas removing v_2 does.

   (a) Prove this statement.

   (b) Give pseudocode for an efficient algorithm that finds such a vertex.

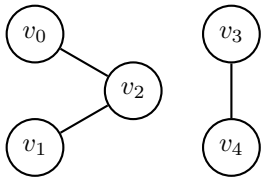
   Consider: given a spanning tree, any leaf can be removed without disconnecting the tree. Since a leaf has no children, no path from the root to another vertex can pass through it.

   (c) Give the time complexity of your algorithm.

3. Consider the following statement:

   Proposition: Let $G = (V, E)$ be a graph with exactly two connected components. There exist vertices $u, v \in V$ such that $(u, v) \notin E$ and $G' = (V, E \cup \{(u, v)\})$ is connected.

   That is, given any connected graph with exactly two components, it is possible to make the graph connected by adding a single edge. For example, given:



   ...adding the edge (v_2, v_3) connects the graph, whereas adding (v_0, v_1) does not.

   (a) Prove this statement.

   (b) Give pseudocode for an efficient algorithm that connects such a graph.

   Consider: given two spanning trees, a vertex in one tree must be able to reach its root. Adding an edge between the roots allows any vertex to access the other tree's root and, thus, its vertices.

   (c) Give the time complexity of your algorithm.

4. Consider the following statement:

   Proposition: Let $G = (V, E)$ be a directed, cyclic graph. If a depth-first search produces exactly one back edge, then there exists an edge $e \in E$ such that $G' = (V, E - \{e\})$ is acyclic.

   Give a proof of or a counterexample to this statement.

   Proof:

   - *Let $G = (V, E)$ be a directed, cyclic graph such that a depth-first search produces exactly one back edge e. Further suppose, for contradiction, that there exists a cycle $(u, \ldots, v, u)$ in $G' = (V, E - \{e\})$.*
   - *Without loss of generality, suppose that u is the first vertex along that cycle explored by the depth-first search. Since there exists a path $(u, \ldots, v)$, v will be a descendant of u. Thus, (v, u) is also a back edge, which contradicts the assumption that there was exactly one back edge.*
   - *Therefore, by contradiction, if a depth-first search produces exactly one back edge, then there exists an edge $e \in E$ such that $G' = (V, E = \{e\})$ is acyclic.* □

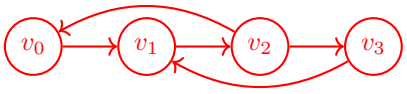
5. Consider the following statement:

   Proposition: Let $G = (V, E)$ be a directed, cyclic graph. If there exists an edge $e \in E$ such that $G' = (V, E - \{e\})$ is acyclic, then a depth-first search produces exactly one back edge.

   Give a proof of or a counterexample to this statement.

   Counterexample:

   - *Consider the following graph:*



   - *There exists an edge $e = (v_1, v_2)$ such that $G' = (V, E - \{e\})$ is acyclic, but a depth-first search rooted at v_0 produces two back edges, (v_2, v_0) and (v_3, v_1).*

6. Suppose that you are given two strings, a "source word" and a "target word". Further suppose that you wish to transforming the source word into the target word by repeatedly substituting a single character in the string, such that, after each substitution, the string is a word in the Oxford English Dictionary.

   For example, given the words "cat" and "dog", the former can be transformed into the latter by 3 substitutions: "cat" → "c**o**t" → "co**g**" → "**d**og".

   (a) Model this situation as a graph.

   Vertices represent words in the Oxford English Dictionary.

   Two vertices are connected by an edge $e = (u, v)$ if and only if words u and v have the same length and differ by exactly one character.

   (b) What algorithm can be used to determine whether or not it is possible to transform a source word into a target word?

   A depth-first search can determine whether or not there exists a path from the source word's vertex to the target word's vertex.

   (c) What algorithm can be used to determine the fewest substitutions necessary to transform a source word into a target word?

   A breadth-first search can find the shortest path from the source word's vertex to the target word's vertex.

7. Suppose that a curriculum consists of multiple courses, all of which are mandatory and some of which are prerequisites for others. Further suppose that there is no limit to the number of courses that may be taken in an academic term, and you wish to complete all courses in as few terms as possible.

   For example, under Cal Poly's 2022-2026 catalog, it is impossible to complete a B.S. in Computer Science in fewer than 6 terms, due to the prerequisite chain CSC 101 → CSC 202 → CSC 203 → CSC 248 → CSC 349 → CSC 430.

   (a) Model this situation as a graph.

   Vertices represent courses.

   Two vertices are connected by a directed edge $e = (u, v)$ if and only if course u is a prerequisite for course v.

   (b) What algorithm can be used to determine the fewest terms necessary to complete all of the courses in the curriculum?

   Note that this is not *as simple as using a directed breadth-first search to find shortest paths: a course cannot be taken until* all *of its prerequisites have been met. From a pathing perspective, a path that currently ends at vertex u cannot be allowed to pass through a successor v unless all of v's other predecessors have already been explored.*

   Further note that, despite the use of the word "fewest", shortest paths are not actually solutions. Rather the fewest terms necessary is equal to the length of the "critical path", the longest *path from a course with no prerequisites to a course that is not a prerequisite to any other.*

   (c) What algorithm can be used to determine whether or not there exists a course whose prerequisites are impossible to satisfy?

   A directed depth-first search can determine whether not the graph is acyclic; vertices along cycles represent courses that are prerequisites to themselves.