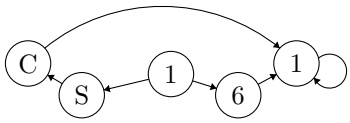
CS 161 **Problem Set 5**
Spring 2017 **Due:** May 19, 2017, 3pm

Please answer each of the following problems. Refer to the course webpage for the **collaboration policy,** as well as for **helpful advice** for how to write up your solutions.

Note: For all problems, if you include pseudocode in your solution, please also include a brief English description of what the pseudocode does.

Drawing graphs: You might try `http://madebyevan.com/fsm/` which allows you to draw graphs with your mouse and convert it into LaTeX code:



---

1. **(Examples.)** (5 points) Give one example of a directed graph on four vertices, A, B, C, and D, so that both depth-first search and breadth-first search discover the vertices in the same order when started at A. Give one example of an directed graph where BFS and DFS discover the vertices in a different order when started at A. Above, *discover* means the time that the algorithm first reaches the vertex. Assume that both DFS and BFS iterate over outgoing neighbors in alphabetical order.

   [We are expecting: a drawing of your graphs and an ordered list of vertices discovered by BFS and DFS.]

2. **(In-order traversal)** (10 points) In class, we stated the fact that Depth-First Search can be used to do in-order traversal of a binary search tree (BST). That is, given a binary search tree T containing n distinct elements $a_1 < a_2 < \cdots < a_n$, DFS can be used to return an array $[a_1, a_2, \ldots, a_n]$ containing these elements in sorted order. Call this algorithm `inOrderTraversal`. Work out the details for this algorithm, and answer the following questions:

   (a) (5 pts) Give pseudocode for `inOrderTraversal`. For your pseudocode, assume that each node in T has a key value (one of the a_i) and pointers to its left and right children, and that if a vertex has no child this is represented as a `NIL` child.

   (b) (3 pts) What is the asymptotic running time of `inOrderTraversal`, in terms of n? We're looking for a statement of the form "the running time is $\Theta(\_\_\_)$." [Hints: We notice that each node in a Tree has only one parent.]

   (c) (2 pts) Does the algorithm need to look at the values $a_1, \ldots, a_n$ (other than to return the values)? In other words, what will affect the procedure of your `inOrderTraversal`, the values of $a_1, \ldots, a_n$ or the structure of the BST?

   [We are expecting: just the answers to questions b and c, no justification needed.]

3. **(Level Averages for a Given Binary Tree)** (5 points) For a binary tree T, design an algorithm `levelAverage` to report the average values for each level of it in a descending order of levels (i.e., reporting the average value for the deepest level first). Give the pseudocode for `levelAverage`.

4. **(Using DFS to topologically sort nodes in a directed graph)** (5 points) A formal definition for the topological ordering of a graph is: for a directed graph G, we say that a topological ordering

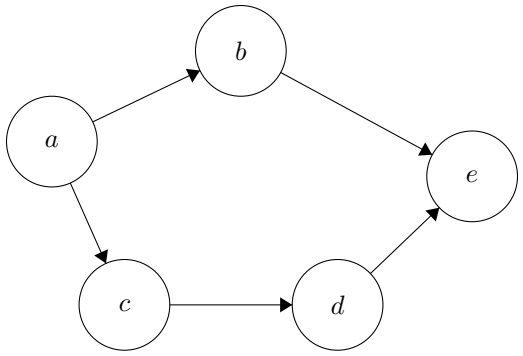


Figure 1: A directed graph.

of G is an ordering of its nodes as v_1, v_2, . . . , v_n so that for every edge (v_i, v_j) (i.e., a direct edge pointing from v_i to v_j), we have $i < j$. Now we have a directed graph shown in Figure 1, we can use the DFS-based algorithm introduced in Lecture 9 to topologically sort the nodes in it. However, as the start node is randomly picked, the reported topological ordering may be different. For the following orders, which one(s) is/are correct? For each correct topological order, please report an ordered list of vertices discovered by our algorithm.

(a) a, b, c, d, e

(b) a, b, c, e, d

(c) a, c, b, d, e

(d) a, d, b, c, e

(e) a, c, d, b, e

5. **(Bipartite Graph Recognization)** (5 points) Is the graph in Figure 2 a bipartite graph? Give your reason. **[We are expecting a brief explanation. No detailed proof is needed.]**

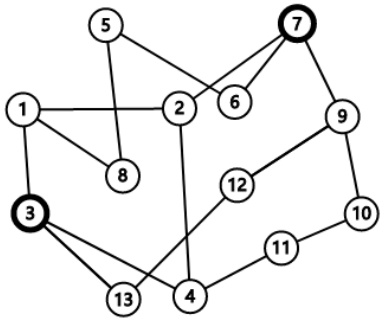


Figure 2: Example graph

6. **(Social engineering)** (10 points) Suppose we have a community of n people. We can create a directed graph from this community as follows: the vertices are people, and there is a directed edge from person A to person B if A would forward a rumor to B. Assume that if there is an edge from A to B, then A

will always forward any rumor they hear to B. Notice that this relationship isn't symmetric: A might gossip to B but not vice versa. Suppose there are m directed edges total, so $G = (V, E)$ is a graph with n vertices and m edges.

Define a person P to be *influential* if for all other people A in the community, there is a directed path from P to A in G. Thus, if you tell a rumor to an influential person P, eventually the rumor will reach everybody. You have a rumor that you'd like to spread, but you don't have time to tell more than one person, so you'd like to find an influential person to tell the rumor to.

In the following questions, assume that G is the directed graph representing the community, and that you have access to G as an array of adjacency lists: for each vertex v, in $O(1)$ time you can get a pointer to the head of the linked lists v.`outgoing_neighbors` and v.`incoming_neighbors`. Notice that G is not necessarily acyclic. In your answers, you may use to any statements or data structures we have seen in class, in the notes, or in CLRS (e.g., Red-Black Tree, Hash Table, BST and so on).

(a) (2 pts) Show that all influential people in G are in the same strongly connected component, and that everyone in this strongly connected component is influential.

[We are expecting: a short but formal proof.]

(b) (5 pts) Suppose that an influential person exists. Give an algorithm that, given G, finds an influential person in time $O(n + m)$.

[We are expecting: pseudocode, a proof of correctness, and a short argument about the runtime.]

(c) (3 pts) Suppose that you don't know whether or not an influential person exists. Use your algorithm from part (b) to give an algorithm that, given G, either finds an influential person in time $O(n + m)$ if there is one, or else returns "no influential person."

[We are expecting: pseudocode, and a short argument for both correctness and runtime. You do not need to re-write your algorithm from part (b), you can just call it.]