Section 07: Graphs

1. Tree method

For each of the following recurrences, find their closed form using the tree method. Then, check your answer using the master method (if applicable).

(a) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/6) + n & \text{otherwise} \end{cases}$

(b) $Y(q) = \begin{cases} 1 & \text{if } q = 1 \\ 8T(q/2) + q^3 & \text{otherwise} \end{cases}$

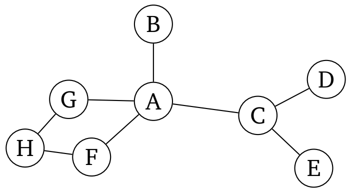
(c) $J(k) = \begin{cases} 1 & \text{if } k = 1 \\ 5J(k/5) + k^3 & \text{otherwise} \end{cases}$

(d) $S(q) = \begin{cases} 1 & \text{if } q = 1 \\ 2S(q-1) + 1 & \text{otherwise} \end{cases}$

(e) $Z(x) = \begin{cases} \log(x) & \text{if } x = 7 \\ 3Z(x/3) + 1 & \text{otherwise} \end{cases}$

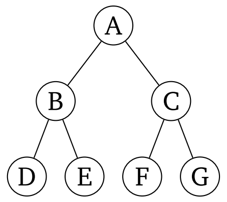
2. Graph traversal

(a) Consider the following graph. Suppose we want to traverse it, starting at node A.



If we traverse this using *breadth-first search*, what are *two* possible orderings of the nodes we visit? What if we use *depth-first search*?

(b) Same question, but on this graph:



3. Graph representations

(a) In class, we studied two different graph representations: adjacency lists, and adjacency matrices. Which representation would be more suited for each case?

   (a) Your graph represents a map of downtown Seattle, and you want to count how many blocks are between two intersections.

   (b) Your graph represents a map of downtown Seattle, and you want to check if two blocks are right next to each other.

   (c) Your graph represents available flights between cities in the U.S., and you want to know if there is a direct flight between two particular cities.

   (d) Your graph represents course prerequisites at UW, and you want to know what courses you need to have taken in order to take CSE 373.

   (e) Your graph represents a social network like Facebook. You have several friends who claim to be friends with Mark Zuckerburg, and you wish to verify whether they are correct.

(b) Suppose you are trying to implement a graph. You decide to use a similar strategy to how we implemented trees, and decide to do the following:

   (a) You create a "Graph" class with a "Vertex" private inner class. Each "Vertex" object contains any data about that vertex, as well as a list of pointers to any directly adjacent vertex.

   (b) The "Graph" class then has a single field pointing to some arbitrary vertex in the graph.

   Evaluate the strengths and weaknesses of this design.

4. Designs with graphs

Suppose you have social network data for some people (including yourself and a famous person). Write pseudocode to find answers to the following questions:

(a) How would you represent this social network with a graph?

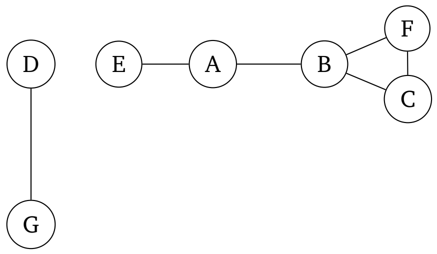
(b) Given two people, how would you determine if they were friends?

(c) How would you find the person with the most friends in the data?

(d) How would you find the length of the shortest path from yourself to the famous person?

5. Graph properties

(a) Consider the *undirected, unweighted* graph below.



Answer the following questions about this graph:

(a) Find V, E, $|V|$, and $|E|$.

(b) What is the maximum *degree* of the graph?

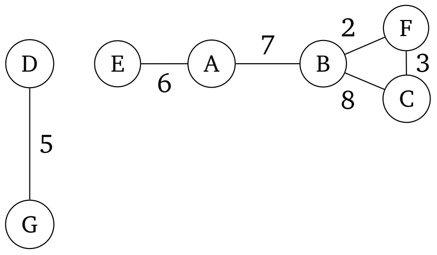
(c) Are there any cycles? If so, where?

(d) What is the maximum length simple path in this graph?

(e) What is one edge you could add to the graph that would increase the length of the maximum length simple path of the new graph to 6?

(f) What are the *connected components* of the graph?

(b) Consider the *undirected, weighted* graph below.



Answer the following questions about this graph:

(a) What is the path involving the least number of nodes from E to C? What is its cost?

(b) What is the minimum cost path from E to C? What is its cost?

(c) What is the minimum length path from E to C? What is its length?

6. Implementing graph searches

(a) Come up with pseudocode to implement *breadth-first search* on a graph, given a starting node and an adjacency list representation of the graph. Is your method recursive or not? What data structures do you use?

(b) Come up with pseudocode to implement *depth-first search* on a graph, given a starting node and an adjacency list representation of the graph. Is your method recursive or not? What data structures do you use?

7. Applying graph algorithms

(a) Given a binary tree, check whether it is a mirror of itself (i.e. symmetric around its center). For example, this binary tree is symmetric:

$$\begin{tikzcd}
 & & & 1 \arrow[dll] \arrow[drr] & & & \\
 & 2 \arrow[dl] \arrow[dr] & & & & 2 \arrow[dl] \arrow[dr] & \\
3 & & 4 & & 4 & & 3
\end{tikzcd}$$

(b) You are given a data structure of employee information, which includes the employee's **unique id**, her **importance value** and her **direct** subordinates' id. Find the total importance value of a particular employee *and* all of their subordinates.

Note that this includes subordinates whose relationship with the employee is **not direct** – for example, if employee 1 is the leader of employee 2, and employee 2 is the leader of employee 3, the total importance value of employee 1 is the sum of all three employees' importances.

(c) You have a graph with vertices representing pastures cows can graze on, and edges representing dirt roads between those pastures.

You also have several spotted and unspotted cows. The spotted cows are scared of other spotted cows, and the unspotted cows are scared of other unspotted cows. A cow on a pasture will be scared if there is a road connecting it to another pasture containing a cow of the same spottiness.

Determine if it is possible to fill all the pastures with a single cow such that none of the cows are scared.

(d) You are trying to plan a dinner party for some event. You have exactly two tables you can use to seat everybody. Your goal is to place everybody at one of the two tables such that everybody is friends with at least one other person sitting in that table.

Assuming you know who's friends with who, design an algorithm that determines where everybody will sit.

(e) Later, you change your mind: you decide that to force people to mingle, you want to make sure that each table contains mutual strangers: if somebody is sitting in table A, they are not friends with any other person who's also sitting in table A.

Again assuming you know who's friends with who, design an algorithm that will (a) determine if it's even possible to seat people this way and (b) if so, decide where everybody will sit.