CPSC 320: A Five Step Problem Solving Process Solutions*

In this worksheet, you will practice five useful steps for designing and analyzing algorithms, starting from a possibly vague problem statement. These steps will be useful throughout the course. They could also be useful when you find yourself thinking on your feet in an interview situation. And hopefully they will serve you well in your work post-graduation too! You will also gain experience with the design and analysis of graph algorithms, starting with graph search algorithms and then moving on to finding the diameter of a graph.

1 Graph Diameter

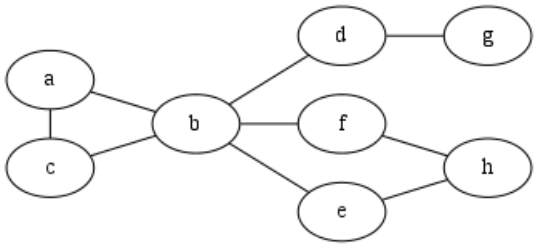
The *diameter* of a connected, undirected, unweighted graph is the largest possible value of the following quantity: the smallest number of edges on any path between two nodes. In other words, it's the largest number of steps required to get between two nodes in the graph. Your task is to design an efficient algorithm to find the diameter.

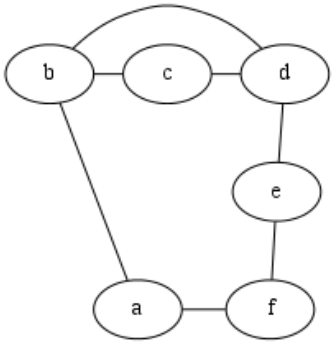
Step 1: Build intuition through examples.

In this section you will both build intuition about the problem we are looking at, and review useful and important graph algorithms you learned about in CPSC 221. For each of the following graphs:

1. Find all the articulation points (if any) in the graph.
2. Give the diameter of the graph.
3. Draw out the rooted tree generated by a breadth-first search of the graph from node a (draw dashed lines for edges that aren't part of the tree).
4. Draw out the rooted tree generated by a depth-first search of the graph from node a (with the same use for dashed lines).

Recall that an articulation point of a graph is a vertex whose removal disconnects the graph.



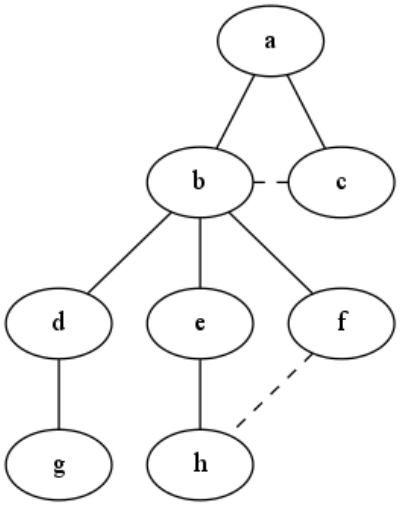


SOLUTION: For the graph on the left:

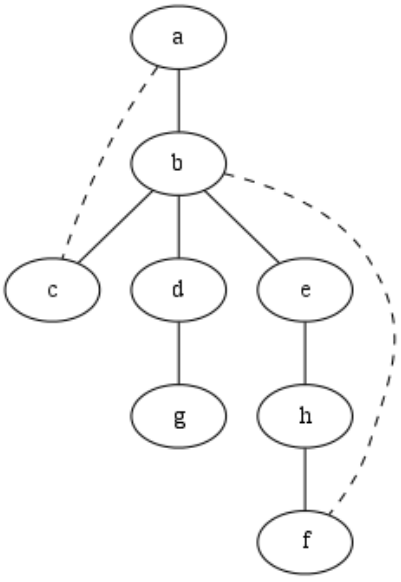
1. `b` and `d` are articulation points. Removing `b` (and all edges incident on `b`) disconnects the graph into three components! (The ones containing `a`, `d`, and `f`, each of which also contains at least one other node.) Removing `d` disconnects the graph into two components, of which one just has `g`.

2. The diameter of the left graph is 4. The shortest path between nodes `g` and `h` (via `d`, `b`, and one of `e` or `f`) is 4 steps long. This actually only shows a lower bound on the diameter, but if you try all the other pairs of nodes, you'll find the shortest paths between them are all shorter.

3. Here's ours. Yours should look much the same, with possible minor differences in order of nodes at the same level and which edge of (f, h) and (e, h) is dashed:



4. Here's ours. Yours may look quite different depending on the order you chose to visit nodes. (DFS generally can have more radically differing shapes depending on order.) We visited alphabetically earlier neighbours first.

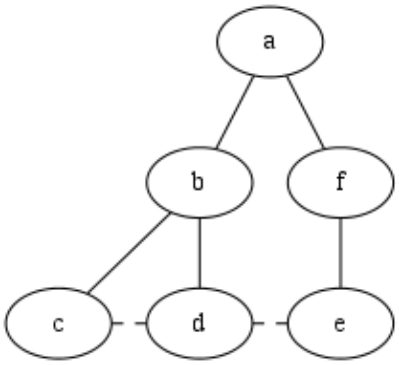


For the graph on the right:

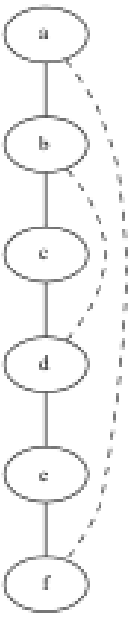
1. There are no articulation points in this graph. (There are at least two disjoint paths between any pair of nodes; so, removing just one node won't break any other pair's connectivity.)

2. The diameter of this graph is 3, between `c` and `f`. A fun way to double-check this: ignoring `c`, every other pair of nodes is on a single cycle of length 5; so, there will certainly be a 2-step path between them (the shorter "side" of the cycle).

3. Here's ours. As above for BFS, yours should look much the same, with possible minor differences in order of nodes at the same level. (The same edges should be dashed!)



4. Here's ours. Yours may also be a "stick" (i.e., a tree with no branches), but the order of nodes in your stick may differ. We visited alphabetically earlier neighbours first. Alternatively, if from node a you visit b and from there visit d next, node d will have two children, namely c and e, and e will have one child f.



Step 2: Develop a formal problem specification

Develop notation for describing a problem instance, a valid (potential) solution, a good solution.

SOLUTION:

- A problem instance is an unweighted, connected undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. We'll let n and m denote the number of nodes and edges of the graph, respectively, and let $V = \{1, 2, \ldots, n\}$. We'll assume that $n \geq 2$, in which case the diameter is at least 1 (a graph with one node is trivial, having diameter 0).

- In some sense, a potential solution is simply a non-negative integer, representing the diameter. However, at least one pair of nodes $u, v \in V$ will define the diameter, which is the distance (number of edges on the shortest path) from u to v. We'll denote this distance by $d(u, v)$. It seems reasonable that the pair (u, v) would be provided as part of the solution also, along with $d(u, v)$. Going even further, we may want a solution to provide a path between nodes u and v.

- A (potential) solution (k, u, v) is *good* if $k = d(u, v)$ and k is the graph diameter. It will be helpful to write a precise expression for the graph diameter, e.g., when we need to reason about algorithm correctness, so we write:

$$\text{diameter}(V, E) = \max_{1 \le i,j \le n} d(i, j). \tag{1}$$

Step 3: Identify similar problems. What are the similarities?

SOLUTION: Breadth first search seems similar, since it finds the shortest path between a node s and other nodes of an unweighted connected graph.

Step 4: Evaluate brute force.

SOLUTION: A brute force approach could be to enumerate all pairs (u, v), determine the shortest path between them, and keep track of the maximum found:

function DIAM-BRUTE-FORCE($G = (V, E)$)
  ▷ returns (k, u, v) where k is the graph diameter and $d(u, v) = k$
  $k \leftarrow 0$
  for each pair of nodes (i, j) (i.e., potential solution) **do**
    Find $d(i, j)$, the length of the shortest path between i and j (e.g., using breadth first search from i)
    if $d(i, j) > k$ **then**
      $k \leftarrow d(i, j)$
      $(u, v) \leftarrow (i, j)$
  return (k, u, v)

Now, let's figure out the worst-case runtime of DIAM-Brute-Force. The for loop iterates through all pairs (u, v), and for each runs breadth first search, which takes time $\Theta(n + m)$. There are n^2 pairs (u, v), so the algorithm takes $\Theta(n^2(n + m))$ time.

Step 5: Design a better algorithm.

1. **Brainstorm some ideas, then sketch out an algorithm.**
   Try out your algorithm on some examples.

   SOLUTION: Using BFS on every pair of nodes seems like overkill. After all, BFS already computes the shortest path to **every** node in the graph from a given node. Why not grab the longest of those and use **that** as a lower-bound on diameter?

   function DIAM-BFS($G = (V, E)$)
     ▷ returns (k, u, v) where k is the graph diameter and $d(u, v) = k$
     $k \leftarrow 0$
     for each node i **do**
       run `BFS`(i); let j be a node at the deepest level of the bfs tree
       if $d(i, j) > k$ **then**
         $k \leftarrow d(i, j)$
         $(u, v) \leftarrow (i, j)$
     return (k, u, v)

2. **Show that your algorithm is correct.**

   SOLUTION: Here, by correct we mean that (i) $k = d(u, v)$ and k is the graph diameter, as defined in equation 1 above. Our reasoning is straightforward, following the iterative structure of the algorithm. Suppose that k_l denotes the value of k after l iterations of the outer **for** loop. We claim that the value of k_l is

   $$k_l = \max_{1 \le i \le l, 1 \le j \le n} d(i, j).$$

   This follows using a straightforward induction, since the updates within the inner **for** loop on iteration $l + 1$ ensures that

   $$k_{l+1} = \max\{k_l, \max_{1 \le j \le n} d(l + 1, j)\} = \max_{1 \le i \le l+1, 1 \le j \le n} d(i, j).$$

   and so after all n iterations, k is exactly the diameter of the graph.

3. **Analyze the running time of your algorithm.**

   SOLUTION: The algorithm has n iterations of the outer for loop, and each iteration takes time $\Theta(n + m)$ for breadth first search. So the total runtime is $\Theta(n(n + m))$, a factor of n better than brute force.

Challenge problems

1. Design an algorithm to find articulation points in an undirected graph. Hint: think about how the dashed edges in a DFS relate to articulation points. It may be easiest to start with the root, which is a simpler special case than the other nodes.

2. The book described the *Independent Set* problem as one for which we do not know a polynomial-time solution. If we had a linear time algorithm for finding articulation points, describe how it could help (and how **much** it could help) for **some** graphs.

3. Here's a promising approach to find diameters: Pick a node arbitrarily. Run BFS from it. Find the node farthest from it in the BFS tree, breaking ties arbitrarily. (Goal: this node is an "extreme" node that will define the diameter.) BFS from **that** node. Return as the diameter the height of the new BFS tree.

   This algorithm is incorrect. Generate a counterexample to its correctness.