Problem Set 1 **solution**

data structures and algorithms 2 - fall 2023 due september 8, 2023 at 11:00pm

problem 1 *Order class proof using limits*

Show $n^k \in o(a^n)$ for $a > 1$. Use the limit definition of the order class. If you find you need to use L'Hôpital's rule, note that $(a^n)' = (\log a)a^n$.

Solution: The key here is to apply the derivative many times until the numerator becomes a constant, which happens when the exponent of the polynomial becomes 0, i.e. $n^0 = 1$. The following shows the result after two uses of L'Hôpital's rule, then what happens if you repeat it k times:

$$\lim_{n\to} \frac{n^k}{a^n} = \lim_{n\to} \frac{kn^{k-1}}{(\log a)a^n} = \lim_{n\to} \frac{k(k-1)n^{k-2}}{(\log a)^2 a^n} = \ldots = \lim_{n\to} \frac{(k(k-1)\cdots 1)n^0}{(\log a)^k a^n} = \lim_{n\to} \frac{k!}{(\log a)^k a^n} = 0$$

since $k!$ and $(\log a)^k$ are constant with respect to n. Note that the above proof assumes k is a whole number, and that's OK for this assignment. If k is not a whole number, the exponent will become a negative value instead of 0, which still results in a constant value in the numerator and a function that goes to infinity in the denominator.

problem 2 *FlyMe Airlines*

An airline, FlyMe Airlines, is analyzing their network of airport connections. They have a graph $G = (A, E)$ that represents the set of airports A and their flight connections E between them. They define $hops(a_i, a_k)$ to be the smallest number of flight connections between two airports. They define $maxHops(a_i)$ to be the number of hops to the airport that is farthest from a_i, i.e. $maxHops(a_i) = max(hops(a_i, a_j))\,\forall a_j \in A$.

The airline wishes to define one or more of their airports to be "Core 1 airports." Each Core 1 airport a_i will have a value of $maxHops(a_i)$ that is no larger than any other airport. You can think of the Core 1 airports as being "in the middle" of FlyMe Airlines' airport network. The worst flight from a Core 1 airport (where "worst" means having a large number of connections) is the same or better than any other airport's worst flight connection (i.e. its $maxHops()$ value).

They also define "Core 2 airports" to be the set of airports that have a $maxHops()$ value that is just 1 more than that of the Core 1 airports. (Why do they care about all this? Delays at Core 1 or Core 2 airports may have big effects on the overall network performance.)

Your problem: Describe an algorithm that finds the set of Core 1 airports and the set of Core 2 airports. Give its time-complexity. The input is $G = (A, E)$, an undirected and unweighted graph, where $e = (a_i, a_j) \in E$ means that there is a flight between a_i and a_j. Base your algorithm design on algorithms we have studied in this unit of the course.

Solution: This problem is about what's called (in graph theory) *eccentricity, graph radius,* and *central vertices.*

1. To find $hops()$ for a single airport a_i, use BFS. The height of the BFS tree starting at a_i is the value we want, so modify BFS to track the level of each node and return the total depth. This value known as the eccentricity of the vertex a_i.)
2. Do this for for all vertices a_i in A. (This is known as the graph radius.) The total cost here is $\Theta(A(A+E))$.
3. As you do the last step (on in a separate loop), find the minimum of the $maxHops$ values.
4. Loop back through the list of vertices a_i and find all those that have the value equal to the minimum, and add them to the set of Core 1 airports. (These are known as central vertices.) Add any that have value that's 1 more than the minimum to the set of Core 2 airports. The cost of this is $\Theta(A)$.

The overall cost is $\Theta(A(A+E))$.

PROBLEM 3 *Counting Shortest Paths*

Given a graph $G = (V, E)$, and a starting node s, let $\ell(s,t)$ be the length of the shortest path in terms of number of edges between s and t. Design an algorithm that computes the number of distinct paths from s to t that have length exactly $\ell(s,t)$.

Solution: The key to this problem is the observation that all shortest paths from s to v consist of shortest paths from s to some node u and then an edge from u to v. Therefore, if we count the number of distinct shortest paths to u, we can do the same for its neighbor v.

Define a new variable n_v which records the number of shortest paths from s to v for every node in V, with $n_s = 0$. Perform BFS on G. During the BFS, for every node with $\delta(s,v) = 1$, we set $n_v = 1$, since the shortest path from s in this case corresponds to the single edge from s to v.

We now consider every node with $\delta(s,v) = j$ for $j = 2, 3, \ldots, V$. For each such node v, we set n_v to be the sum of the n_u's for each neighbor u of v for which $\delta(s,u) = j - 1$.

The overall running time of this algorithm is $O(V+E)$, because BFS requires time $O(V+E)$ and the summation for each $v \in V$ amounts to inspecting every edge in the graph twice. Thus, the summation step also requires time $O(V+E)$.

(Notes: drawing a picture of the BFS search in progress makes it easier to discover and understand this solution. Also, the fact that we're counting the number of *shortest* paths makes the problem simpler, since we don't have to count paths that we know are longer than a shortest path, including those that have a cycle.)

PROBLEM 4 *Coffee*

Charlottesville is known for some of its locally-roasted coffees, each with its own unique flavor combination. Suppose a group of coffee enthusiasts are given n samples $c_1, c_2, ..., c_n$ of freshly-brewed Charlottesville coffee by a coffee-skeptic. Each sample is either a *Milli* roast or a *Shenandoah Joe* roast. The enthusiasts are given each pair (c_i, c_j) of coffees to taste, and they must collectively decide whether: (a) both are the same brand of coffee, (b) they are from different brands, or (c) they cannot agree (i.e., they are unsure whether the coffees are the same or different). Note: all n^2 pairings are tested, including $(c_i, c_i), (c_i, c_j)$, and (c_j, c_i), but not all pairs have "same" or "different" decisions.

At the end of the tasting, suppose the coffee enthusiasts have made m judgments of "same" or "different." Give an algorithm that takes these m judgments and determines whether they are *consistent*. The m judgments are *consistent* if there is a way to label each coffee sample c_i as either *Shenandoah Joe* or *Milli* such that for every taste-comparison (c_i, c_j) labeled "same," both c_i and c_j have the same label, and for every taste-comparison labeled "different," both c_i and c_j are labeled differently. Your algorithm should run in $O(m+n)$ time. Prove the correctness and running time of your algorithm. Note: you do *not* need to determine the brand of each sample c_i, only whether the coffee enthusiasts are consistent in their labelings.

Solution: The idea is to turn the same/different judgments into a graph, and then test whether the graph contains a "contradiction cycle." This later part can be done by coloring the nodes using breadth first search, and then analyzing all the edges in the graph:

1. Construct a graph where $V = \{s_1, \ldots, s_n\}$ and introduce an edge between s_i and s_j if there is a judgment between the two. Namely, the samples correspond to vertices in the graph, and the edges correspond to judgments.
2. Scan the graph to make it undirected. More precisely, for each edge $e = (x, y)$, check whether there is an edge $e_0 = (y, x)$ in the graph. If not add it. If there is, check that e and e_0 are either both "same" or both "different." If not, then stop and output `Inconsistent`.
3. Perform a BFS on every connected component of the graph. This can be done by picking an arbitrary unvisited nodes s, performing BFS from s, and repeating until all nodes are visited.
4. When starting a BFS at node s, label it arbitrarily as 0. During the BFS, when inspecting an edge (s, y), if y is unlabeled and the edge corresponds to a "same" judgments then label node y with the same label as node s and if it is "different" then label y as the opposite. The opposite of 0 is 1 and vice versa.
5. Scan every edge in E. If a "same" edge has two vertices with different labels, or a "different" edge has two vertices with equal labels, then output `Inconsistent`. Otherwise, if all checks pass, then output `Consistent`.

Correctness: Denote a "same" edge (judgments) as a 0 and a "different" edge as a 1. Thus, a set of judgments is inconsistent if and only if the judgments graph contains some cycle $v_{i_1}, v_{i_2}, \ldots, v_{i_1}$ with odd parity. The parity of a cycle is the xor of each of its edges. For example if $s_i \to s_j$ is "same" and $s_j \to s_i$ is "different", then the parity of the cycle (s_i, s_j, s_i) would be 1 and therefore inconsistent. Consider an odd-parity cycle $c = (s_i, s_j, \ldots, s_i)$. If c has length 2, then an inconsistency will be flagged in the third step. Otherwise, the BFS in step 5 will visit and color the nodes in c by some order. Without loss of generality, let s_i be the first node analyzed by the BFS, and (s_j, s_k) be the last edge analyzed by the BFS. This last edge of the cycle will be ignored by the BFS since the two endpoints s_j, s_k will have already been colored. Let b_1 be the parity of the path $s_i \to s_j$ and b_2 be the parity of the path $s_i \to s_k$. The BFS will color these two paths consistently. However, since c is an odd parity cycle, then $b_1 \cdot b_2 = 1$, and so the edge (s_j, s_k) will raise an inconsistency in step 6.

Running time: The algorithm runs in time $O(m + n)$; the breadth-first search step in 4 and 5 dominate the running time. The other steps take either $O(m)$ or $O(n)$ time.

PROBLEM 5 *Ancient Population Study*

Historians are studying the population of the ancient civilization of *Algorithmica*. Unfortunately, they have only uncovered incomplete information about the people who lived there during Algorithmica's most important century. While they do not have the exact year of birth or year of death for these people, they have a large number of possible facts from ancient records that say when a person lived relative to when another person lived.

These possible facts fall into two forms:

- The first states that one person died before the another person was born.
- The second states that their life spans overlapped, at least partially.

The Algorithmica historians need your help to answer the following questions. First, is the large collection of uncovered possible facts internally consistent? This means that a set of people could have lived with birth and death years that are consistent with all the possible facts they've uncovered. (The ancient records *may not be accurate*, meaning all the facts taken together cannot possibly be true.) Second, if the facts are consistent, find a sequence of birth and death years for all the people in the set such that all the facts simultaneously hold. (Examples are given below.)

We'll denote the n people as $P_1, P_2, \ldots, P_n$. For each person P_i, their birth-year will be b_i and their death-year will be d_i. (Again, for this problem we do not know and cannot find the exact numeric year value for these.)

The possible facts (input) for this problem will be a list of relationships between two people, in one of two forms:

- P_i *prec* P_j (indicates P_i died before P_j was born)
- P_i *overlaps* P_j (indicates their life spans overlapped)

If this list of possible facts is not consistent, your algorithm will return "not consistent". Otherwise, it will return a possible sequence of birth and death years that is consistent with these facts.

Here are some examples:

- The following facts about $n = 3$ people are **not** consistent: P_1 *prec* P_2, P_2 *prec* P_3, and P_3 *prec* P_1.
- The following facts about $n = 3$ people **are** consistent: P_1 *prec* P_2 and P_2 *overlaps* P_3. Here are two possible sequences of birth and death years:
  $b_1, d_1, b_2, b_3, d_2, d_3$
  $b_1, d_1, b_3, b_2, d_2, d_3$
  (Your solution only needs to find one of any of the possible sequences.)

Your answer should include the following. Clearly and precisely explain the graph you'll create to solve this problem, including what the nodes and edges will be in the graph. Explain how you'll use one or more of the algorithms we've studied to solve this graph problem, and explain why this leads to a correct answer. Finally, give the time-complexity of your solution.

Solution: Our graph G will be a digraph, where each b_i and d_i is a node. After we have added our directed edges, if the graph has no cycles then we know the facts are consistent. If it has no cycles, then a topological sort will produce a sequence of b_i and d_i values that match the facts.

Graph G will have edges as follows:
For each P_i, add edge (b_i, d_i) to model that a person is born before they die.
For each P_i *prec* P_j, add edge (d_i, b_j) to model that P_i died before P_j was born;
For each P_i *overlaps* P_j, add edges (b_i, d_j) and (b_j, d_i) to model that their life spans overlapped.

Time-complexity is that for DFS and topological sort based on the graph's size. G will have $2n$ nodes. Each person will contribute one edge because of the birth/death relationship. In the worst-case, all possible pairs overlap, adding 2 more edges for each of the $\Theta(n^2)$ possible pairs. So for G, the number of nodes is $\Theta(n)$ and the number of edges is $\Theta(n^2)$. So DFS and topological sort on a graph this size will be $\Theta(n^2)$ in the worst-case (or $O(n^2)$ in general).