Homework 4

Due Wednesday, September 30 at 11:59 PM

Homework Guidelines

Collaboration policy Collaboration on homework problems, with the exception of programming assignments and reading quizzes, is permitted, but not encouraged. If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 5 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions.

A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

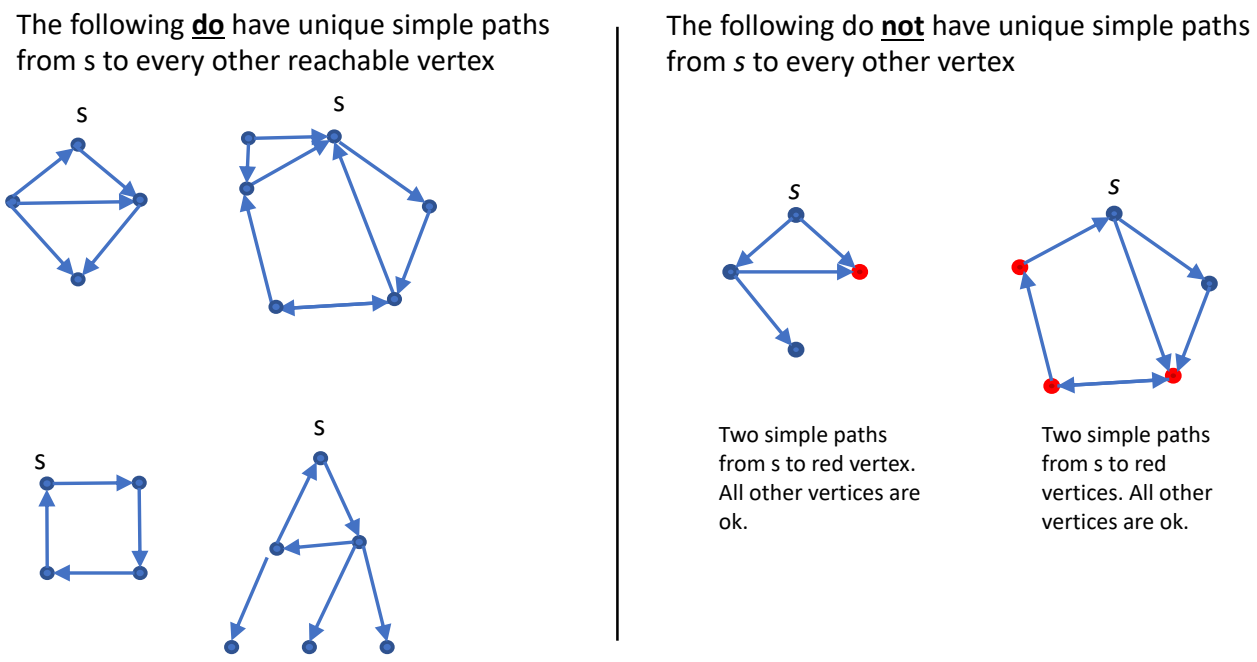
1. (**Unique simple path (15 pts)**) Given a **directed** graph $G = (V, E)$, vertex s has *unique simple paths to all vertices* if for every $v \in V$ that is reachable from s, there is at most one simple path from s to v (Recall that a path is simple if all vertices on the path are distinct).

   The figure below gives example graphs and points out pairs of vertices that do and do not have unique paths. Go over the examples to make sure you understand the definition.

   (a) For each of the following types of graphs, is it the case that in every graph of that type, every vertex s has a unique simple paths to all reachable vertices? For each type of graph, give either a short proof (one or two sentences), if yes, or a counterexample, if no.

      i. Cycles[1]
      ii. DAGs (directed acyclic graphs)
      iii. Trees
      iv. Strongly connected graphs

   (b) Give an efficient algorithm that takes a directed graph G and a vertex s as input and checks whether s has unique simple paths to all other vertices reachable from s. Your algorithm should output "no" if at least one vertex v has more than one path from s to v; otherwise it should output "yes", together with a tree of paths from s to each reachable vertex in G. For full credit, it should run in $O(m + n)$ time *[Hint: Use DFS. Think about different kinds of non-tree edges.]*.



[1] A directed graph on n vertices with directed edges of the form $(i, i + 1)$ for $i = 1, ..., n - 1$ and the edge $(n, 1)$.

2. (**Lazy Hiker (10 pts)**) Suppose you are planning a hike in a forest with many paths. You want to start and end at the parking lot without visiting the same area of the forest twice. In addition, you get lost easily, so you want to choose the hike with the least possible number of trail intersections.

   You have a map, which you can interpret as a graph G: each vertex is a trail intersection (or start/end point) and each edge is a section of trail between two intersections. Note that G is undirected.

   Write an efficient algorithm that takes as input a graph $G = (V, E)$ and a "parking lot vertex" p and outputs the a loop hike (starting and ending at p) that visits the smallest possible number of vertices while never covering the same trail segment twice. If there are no such loop hikes, your algorithm should output "no loops from p". For full credit, your algorithm should run in $O(n + m)$ time. See example graphs and correct outputs below.

   Hint 1: Run BFS starting from p and divide the BFS tree into subtrees rooted at the children of p. For each vertex v, figure out and store the subtree that it is part of.

   Hint 2: The following questions may help you get thinking in the right direction. Do not hand these in.

   - Prove that if you run BFS on an undirected graph, for every non-tree edge (u, v) either u and v are in the same BFS layer, or u and v are off by one layer (e.g. u is in layer k and v is in layer $k \pm 1$)
   - Non-tree edges create cycles. What do the numbers of the layers of the endpoints tell you about the length of the cycle and edge creates?

Example input (G, p) and correct output paths (in red)