Homework 5

Due Wednesday, February 23rd at 11:59 pm ET on Gradescope

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 pseudocode (*),
2. a proof of correctness,
3. an analysis of the asymptotic running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class, e.g. correctness of subroutines, running time of subroutines.

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.

(*) It is fine if the English description concentrates on the high level ideas and doesn't include all the details. But the reader should not have to figure out your solution solely based on the pseudocode. You can also add comments to your pseudocode, in fact that is best practice. FYI we will share a document on good pseudocode style with you and it will also be discussed in labs and lecture.

Problem 1. *Shortest building time.*

Imagine you are the main contractor building a large skyscraper. You are promised a huge bonus if you can finish the job fast. Obviously, there are some constraints; you cannot install windows until you have built the walls. However, you can do landscaping at the same time as painting interior walls. You want to get an estimate on how long it will take you to build the skyscraper. You may assume that each task takes exactly one day to finish. You have enough resources to perform independent tasks simultaneously.

1. Describe how you can represent this problem using a directed graph. Make sure to give a precise description of how nodes, edges and directions are assigned.

2. You want to find out whether it is even possible to build the building. For this you need to know that there are no two jobs that mutually depend on each other. (e.g. j_1 needs to be finished before j_2, but j_2 needs to be finished before j_1). Formulate what property your graph needs to have so that this doesn't happen. Give a one sentence explanation why.

3. Give an algorithm that receives as input a list of m dependencies among n tasks (in the form of pairs (i, j) where task i must be performed before task j), and outputs a schedule that accomplishes all the tasks in as few days as possible. Your algorithm should output, for each task, the day on which it should be performed.

   For example, suppose that we have six tasks a, b, c, d, e, f with seven dependencies

   $$(a, b), (a, c), (a, d), (b, f), (c, f), (d, f), (e, f).$$

   Then the following outputs would be acceptable:

Task	Day
a	1
b	2
c	2
d	2
e	1
f	3

or

Task	Day
a	1
b	2
c	2
d	2
e	2
f	3

   For full credit, your algorithm should run in time $O(m + n)$. You may use algorithms from class or the textbook (Chapter 3).

   Hint: First find an order in which to do the tasks that respects all the dependencies; then make another pass to figure out the earliest day on which you can perform each job.

Problem 2. *Unique simple paths.*

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.

1. 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.

   (a) Cycles[1]

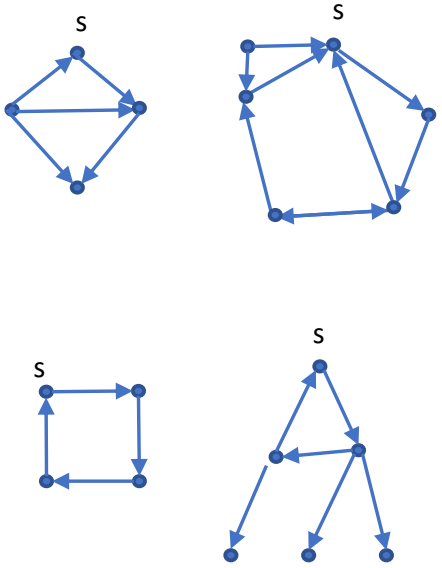
   (b) DAGs (directed acyclic graphs)

   (c) Trees

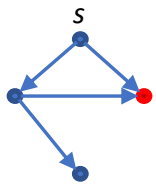
   (d) Strongly connected graphs

2. 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.]*

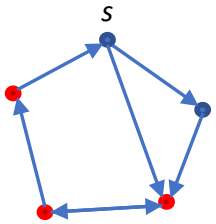
The following **do** have unique simple paths from s to every other reachable vertex



The following do **not** have unique simple paths from *s* to every other vertex



Two simple paths from s to red vertex. All other vertices are ok.



Two simple paths from s to red vertices. All other vertices are ok.

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