Your name
CS 163/265, Winter 2020
Homework 1
Due Jan 17

List of collaborators:

1. Draw or describe a directed graph within which there exist two vertices u and v that belong to the same strongly connected component of each other but in which there is no cycle containing both u and v. (Here "cycle" means a walk that starts and ends at the same vertex and otherwise has no repeated vertices or edges; sometimes this is called a "simple cycle".)

■

2. Show the results of applying Tarjan's strongly connected components algorithm to the graph

```
G = {0:[ ], 1:[2,3], 2:[0,4], 3:[0], 4:[0,2], 5:[3,4]}
```

(this graph has edges from i to $2i$ (mod 6) and $3i$ (mod 6), whever those go to another vertex than i.). Your answer should include the depth first search forest, the index (dfs number) of each vertex, the lowlink (escape edge) of each vertex, and a listing of all the strongly connected components.

■

3. For a given weighted graph G and starting vertex s, define the Jarník sequence of G and s to be the sequence of edges of the minimum spanning tree, in the order that Jarník's algorithm adds them to the tree when starting at vertex s. Different starting vertices might or might not lead to different Jarník sequences.

(163 only): Find a weighted graph with four vertices that has exactly two different Jarník sequences. Find another weighted graph with four vertices that has exactly three different Jarník sequences. (Hint: both of your graphs can be trees, because the edges that are not in the MST do not affect the sequence. In your answer, all edges should have different weights from each other.)

(265 only): What is the minimum number of different Jarník sequences that an n-vertex graph can have, as a function of n? What is the maximum number? (Again, you can assume that all weights are different.)

■

4. Suppose you have a weighted undirected graph G, with two of its vertices labeled s and t, and you want to know whether there exists a path from s to t that only uses edges whose weight is at most a given number W. One way to do this would be to use the path property of minimum spanning trees: find a minimum spanning tree, look at the path from s to t in this tree, and check whether all of its edges have weight $\leq W$. Describe a simpler linear time algorithm for finding a path with this property, that does not use minimum spanning trees.

■