Due on April 1, 2019
69 points total

General directions: We will exclusively use Python 3 for our programming assignments, and allow only the use of modules in the Python 3 standard library unless explicitly specified otherwise on an individual assignment basis. This forbids the use of common third-party libraries such as Numpy, Sympy, etc., but not the use of math or io.

Unless specified otherwise, for the X-th homework, download the single "hwX_skeleton.py" file from the course website, and rename it to "hwX.py" on your machine. When you are done and ready to submit, upload your file named **exactly** "hwX.py" on Gradescope for assignment "HW X (Programming)." When you upload your file, the autograder will run a simple test for each function so that you can confirm it was properly uploaded and executed. Generally, if an assignment involves printing or writing a file in a specific format, there will be at least one simple test that checks your output is formatted as we expect. These tests are not worth any credit — once the due date is over, your submission will be graded by a collection of additional test cases.

All answers to non-programming questions must be typed, preferably using LaTeX. If you are unfamiliar with LaTeX, you are strongly encouraged to learn it. However, answers typed in other text processing software and properly converted to a PDF file will also be accepted. To submit your file, upload your PDF on Gradescope for assignment "HW X (PDF)." Handwritten answers or PDF files that cannot be opened will not be graded and will not receive any credit.

Finally, please read the detailed collaboration policy given on the course website. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Point values: Every problem has a specified amount of points which are awarded for the correctness of your solutions. In addition, each proof-oriented problem has an additional **style point**. In the homework handout, this is signified by a "+1" in the point value. To earn this point, your solutions should be clear, well organized, and easy to follow. This is to encourage not only perfectly correct solutions, but well presented ones.

Problem 1 (15+2 points)
Let $T = (V, E)$ be a tree with n vertices and at least one edge. For any positive integer k, let y_k denote the number of vertices with degree k.

(a) (10+1 points) Prove that the following equality holds:

$$y_1 = 2 + \sum_{k=3}^{n-1} (k-2) \cdot y_k.$$

(b) (5+1 points) Use part (a) to prove the following: if T is a rooted full binary tree, then T has $(n+1)/2$ leaves. (Recall that a *leaf* of a rooted tree is a node with zero children. Also, in a rooted full binary tree, every node has zero or two children.)

Problem 2 (15+1 points)
Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be undirected graphs. We say that G_2 contains a *copy* of G_1 if there exists an injective total function $f : V_1 \to V_2$ that satisfies the following property:

$$\forall u, v \in V_1. \{u, v\} \in E_1 \text{ implies } \{f(u), f(v)\} \in E_2.$$

In other words, the function f should map adjacent vertices in G_1 to adjacent vertices in G_2.

Let T be an arbitrary tree with k vertices. Prove the following: if G is a graph whose minimum degree is at least $k-1$, then G contains a copy of T.

Hint: Recall that any tree has at least one leaf, i.e., a vertex with degree 1. Removing this vertex (and its single incident edge) yields a tree with fewer vertices. Using this fact, proceed with induction on k.

Problem 3 (15+1 points)
Let $G = (V, E)$ be a directed graph, and let u, v be two vertices of G. We say that v is *reachable* from u if there exists a directed path from u to v in G. Furthermore, u and v are *strongly connected* if v is reachable from u and u is reachable from v. Now recall that the relation $\{(u, v) : u \text{ and } v \text{ are strongly connected}\}$ is an equivalence relation on V, and we call each equivalence class induced by this relation a *strongly connected component* of G.

Let C be a strongly connected component of G. We say that C is a *source component* if C does not contain a vertex that is reachable from a vertex outside C. Now recall that when we run DFS on G, every vertex u is assigned two positive integers $pre(u)$ and $post(u)$.

Prove the following: in any run of DFS on G, the vertex with the largest $post$-value belongs to a source component of G.

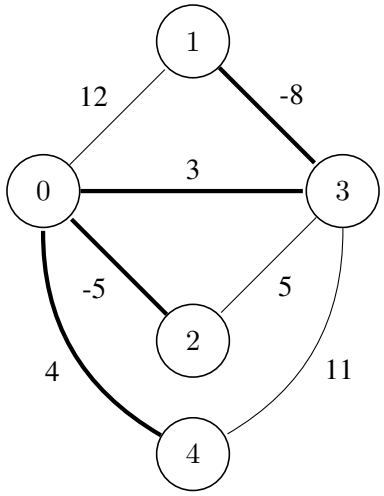
Hint: Consider the DAG formed by the strongly connected components of G.

Problem 4 (20 points)
Programming: In Lecture 16, we gave the "cycle property" regarding minimum spanning trees

(MSTs): the heaviest edge of any cycle in the graph is not in the MST. Thus, one way to find an MST is the following: while there exists a cycle C in the graph, remove the heaviest edge of C from the graph. Repeat this process until the graph is acyclic, and return this final graph.

For this problem, you will implement the algorithm above as function `findMST(M)` where `M` is a *weighted adjacency matrix*. As in the programming assignment of Homework 6, we assume that the vertex set of the input is of the form $\{0, 1, 2, \ldots, n-1\}$. The weighted adjacency matrix representation of an n-vertex undirected graph G is an $n \times n$ symmetric matrix where `M[i][j]` is `None` if there is no edge between vertex `i` and vertex `j` in G, and otherwise it is the *weight* of the edge `(i, j)` in G. (Recall that `M` is *symmetric* means `M[i][j] = M[j][i]` for every pair of vertices `i` and `j`.) See the example graph G and its corresponding matrix `M` below.

To implement `findMST(M)`, you are to use the given function `findCycle(M)` which returns `None` if there is no cycle contained in the graph G represented by `M`, otherwise it returns a cycle C represented by a list of vertices (integers between 0 and $n-1$, inclusive) such that every pair of consecutive vertices are adjacent in G, as well as the first and last vertices in C. For example, `findCycle(M)` for the matrix `M` below might return `[0,3,1]`, `[2,3,4,0]`, or any other such cycle in G represented by M. In addition, we also provide the functions `isConnected(M)`, `isSymmetric(M)`, and `createExampleGraph()`, whose details are in the skeleton file and may be useful when implementing or testing your solution. (The skeleton contains other functions as well, but you should not be calling them directly, so you can ignore them.)



Above, the 5-vertex graph G represented by the matrix `M` below. The thick edges denote the (unique) MST in G which is represented by `findMST(M)` below.

$$\texttt{M} = \begin{bmatrix} \text{None} & 12 & -5 & 3 & 4 \\ 12 & \text{None} & \text{None} & -8 & \text{None} \\ -5 & \text{None} & \text{None} & 5 & \text{None} \\ 3 & -8 & 5 & \text{None} & 11 \\ 4 & \text{None} & \text{None} & 11 & \text{None} \end{bmatrix}$$

$$\texttt{findMST(M)} = \begin{bmatrix} \text{None} & \text{None} & -5 & 3 & 4 \\ \text{None} & \text{None} & \text{None} & -8 & \text{None} \\ -5 & \text{None} & \text{None} & \text{None} & \text{None} \\ 3 & -8 & \text{None} & \text{None} & \text{None} \\ 4 & \text{None} & \text{None} & \text{None} & \text{None} \end{bmatrix}$$

For more details, see the skeleton file and the documentation of each function.