Exercise 2.4.1.1: If the Adjacency Matrix (AM) of a (simple) graph has the property that it is equal to its transpose, what does this imply?

Exercise 2.4.1.2*: Given a (simple) graph represented by an AM, perform the following tasks in the most efficient manner. Once you have figured out how to do this for AM, perform the same task with Adjacency List (AL) and then Edge List (EL).

1. Count the number of vertices V and directed edges E (assume that a bidirectional edge is equivalent to two directed edges) of the graph.

2*. Count the in-degree and the out-degree of a certain vertex v.

3*. Transpose the graph (reverse the direction of each edge).

4*. Create the complement of the graph.

5*. Check if the graph is a complete graph K_n. Note: A complete graph is a simple undirected graph in which *every pair* of distinct vertices is connected by a single edge.

6*. Check if the graph is a tree (a connected undirected graph with $E = V - 1$ edges).

7*. Check if the graph is a star graph S_k. Note: A star graph S_k is a complete bipartite $K_{1,k}$ graph. It is a tree with only one internal vertex and k leaves.

8*. Delete a certain edge (u, v) from the graph.

9*. Update the weight of a certain edge (u, v) of the graph from w to w'.

Exercise 2.4.1.3*: Create the Adjacency Matrix, Adjacency List, and Edge List representations of the graphs shown in Figure 4.1 (Section 4.2.2) and in Figure 4.8 (Section 4.2.10). Hint: Use the graph data structure visualization in VisuAlgo.

Exercise 2.4.1.4*: Given a (simple) graph in one representation (AM, AL, or EL), *convert* it into another graph representation in the most efficient way possible! There are 6 possible conversions here: AM to AL, AM to EL, AL to AM, AL to EL, EL to AM, and EL to AL.

Exercise 2.4.1.5*: Research other possible methods of representing graphs other than the ones discussed in this section, especially for storing special graphs!

Exercise 2.4.1.6*: In this section, we assume that the neighbors of a vertex are listed in increasing vertex number for Adjacency List (as Adjacency Matrix somewhat enforces such ordering and there is no notion of neighbors of a vertex in Edge List). What if the neighbors are not listed in increasing vertex number in the input but we prefer them to be in sorted order in our computation? What is your best implementation?

Exercise 2.4.1.7*: Follow up question, is it a good idea to *always* store vertex numbers in *increasing order* inside the Adjacency List?

Exercise 2.4.1.8*: Think of a situation/problem where using two (or more) graph data structures *at the same time* for the same graph can be useful!