Instructions: Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs. Submit your files latest by tomorrow, March 22, 6pm on Moodle.

This lab-session is about graphs. Recall that a graph is denoted by $G(V, E)$ where V is a set of nodes and E is a set of edges (and each edge in turn is a pair of nodes). For this lab, we assume that the edges have some weight.

In the following exercises we will represent a graph using 3 arrays: `tl`, `hd` and `wt`. All three are of size m, or the number of edges. The array `tl` contains the tail-nodes of all edges, `hd` the head-nodes and `wt` the weight of each edge. Thus, `hd(7)` is the head-node of 7th edge, `tl(7)` is the tail-node of 7th edge, and `wt(7)` is the weight of the 7th edge. The input files net0.py, net1.py, ... also mention n, the number of nodes in the network.

You may import these files from your python code. For example, the following code shows how to import net3.py:

```
input net3 as G

print G.n
print G.tl
print G.hd
print G.wt
```

In the following exercises, we will ignore the directions of the arcs, i.e. we will assume that one can travel along an edge in either direction.

Exercise 0: Alternative Representation

Certain algorithms, like the following ones, require lists of all neighbors of a node. It may be inefficient to search neighbors of a given from the above lists again and again. So one can make a list initially. Write a Python function, `find_nhbs()` that returns an array of lists, called `nhbs`, where `nhbs[i]` is list of all neighboring nodes of node `i`. Test it on the network descriptions given on Moodle.

Exercise 1: Depth First Search

Depth-first-search is, roughly speaking, a method of traversing all vertices in a graph by going as far as possible from a starting vertex and then backtracking.

1. Implement a depth-first-search routine to traverse all the vertices in a given undirected graph. Your routine should display the list of vertices in the order they are visited by depth-first-search. Each vertex should be displayed at most once. You should start from vertex 1.
2. Test your routine on net0.py, net1.py, ..., net6.py
3. Write another routine that is a small modification of the above routine to check whether a graph has a cycle.
4. Write another routine that is a small modification of the above routine to check whether a graph is connected.

Exercise 2: Computing Minimum Spanning Trees

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set E. We are also given edge weights w_e, $e \in E$. Our goal is to compute the minimum spanning tree (i.e., smallest weighted connected acyclic subgraph) of G. Prim's algorithm is as follows:

$U \leftarrow \{1\}$, $T \leftarrow \emptyset$;
while $U \neq V$ **do**
  Among all $[i, j] \in E$ with $i \in U$ and $j \in V \setminus U$, pick an edge $[i^*, j^*]$ with the smallest weight;
  $T \leftarrow T \cup \{[i^*, j^*]\}$, $U \leftarrow U \cup \{j^*\}$;
end

Algorithm 1: Prim's Algorithm

An alternative algorithm is Kruskal's algorithm:

$C \leftarrow \emptyset$, $T \leftarrow \emptyset$;
for $i = 1$ **to** n **do**
  Let $[u, v]$ be an edge of smallest weight not belonging to C;
  $C \leftarrow C \cup \{[u, v]\}$;
  if $T \cup \{[u, v]\}$ *does not contain a cycle* **then**
    $T \leftarrow T \cup \{[u, v]\}$;
  end
end

Algorithm 2: Kruskal's Algorithm

1. [**R**] Why do the algorithms above compute the correct answer?
2. Implement Prim's algorithm in a Python function (or a language of your choice). Your implementation should take net0.py, net1.py, ..., net6.py as inputs, and display a step by step progress of the algorithm (you may turn off the display for larger problems).
3. Implement Kruskal's algorithm in a Python function (or a language of your choice). Your implementation should take net0.py, net1.py, ..., net6.py as inputs, and display a step by step progress of the algorithm (you may turn off the display for larger problems). For testing whether $T \cup \{[i, j]\}$ contains a cycle or not, use your own code from Exercise 1.