Minimal Spanning Tree function:

First create a sorted list of edges (ascending weight)

Place every node into a separate set by itself.

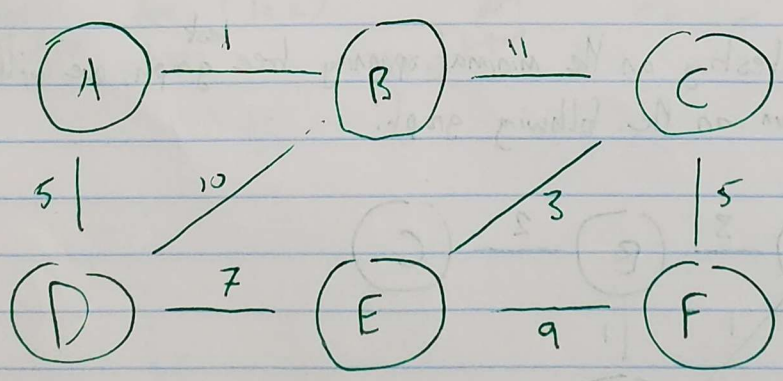
Iterate through edges; if edge connects two separate sets, connect it, and combine the two sets. If edge is between two elements in same set, remove edge.

Containers needed:

- Auto-sorted list for edges
- Array of node sets — Maybe linked list structure is better here...

Most likely: iterate through set array looking for source + destination of each edge.

Graph to test on: named g



Shortest Path function:

Some way or another, we must store some fundamental data:

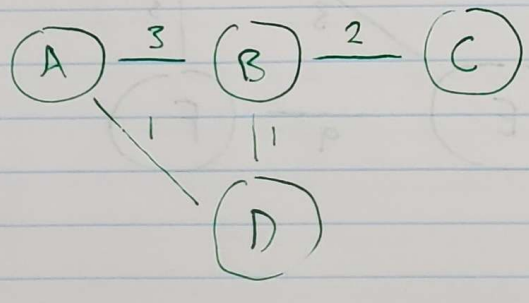
Node	Cost	Predecessor	is_visited?
A	⋮	⋮	⋮
B	⋮	⋮	⋮
C	⋮	⋮	⋮
⋮	⋮	⋮	⋮

Beginning at source_node, we must look at all neighbors and adjust cost based on the weight of the edge connecting the nodes.

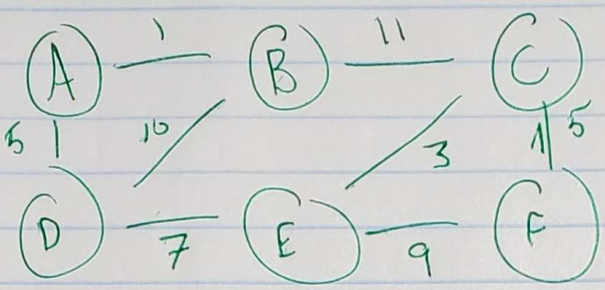
Next, choose a connected node and look at all of its neighbors. Do this for all nodes until the location is found.

Probably need to use a sorted list of edge weights again, along with a few sets: connected set – all nodes connected to source (visited)
unconnected set – all nodes not yet visited
adjacent set – all nodes unvisited but connected to checked nodes
We probably choose the next node to examine based on lowest edge weight of edge with source contained in connected (visited) set.

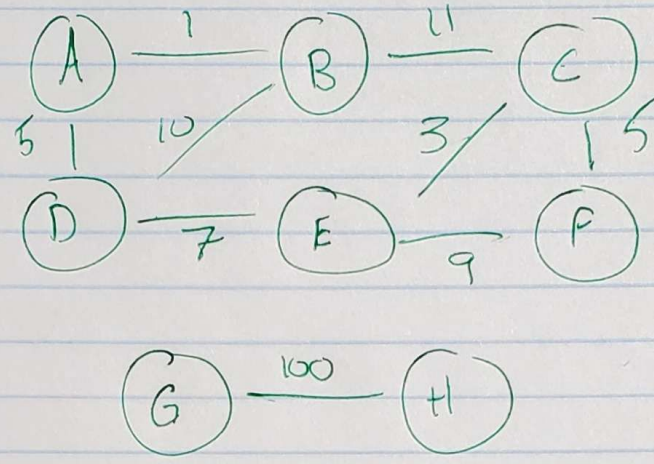
Along with testing on the minimal spanning tree graph, we will test this function on the following graph:



graph g1:



graph g2:



graph g3: