Homework 4.
Due: Tuesday, June 29, 2021 before 11:59PM via Gradescope. Late submission with no penalty by Wednesday, June 30, 2021 before 11:59AM.

Problem 1 (pseudo-connected graphs).

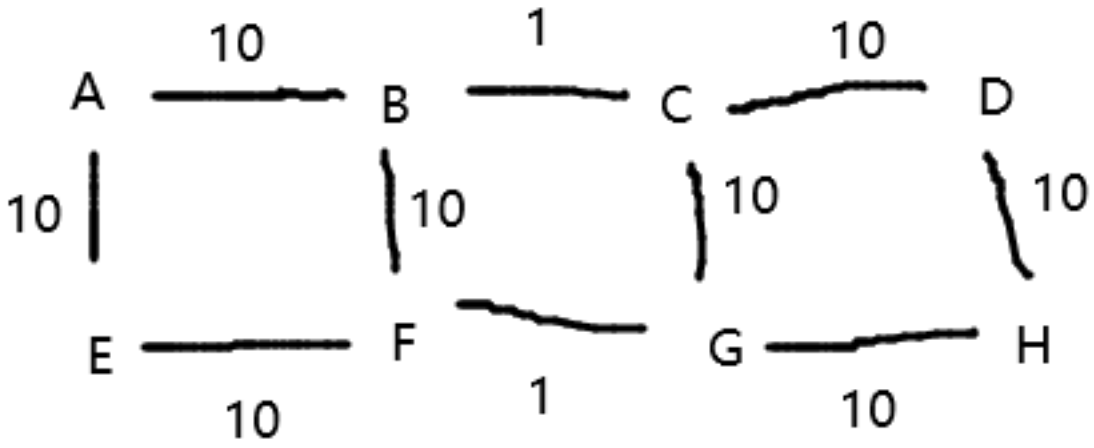
Design:
1. Run SCCs on the given graph G(V, E) and retrieve its metagraph G'.
2. Reverse G' and get graph G'_{rev}.
3. Run topological sort on G' and G'_{rev}.
4. From the left most vertex V' and V'_{rev}(representing the source strong connected component) on both graph, do $Explore(G')$ and $Explore(G'_{rev})$. Record visited vertices.
5. Check to see if there is any vertex in G'/G'_{rev} that are not in the list of visited vertices of G'/G'_{rev}. If there exist at least one vertex of that kind in any one of the lists, return false. Otherwise, return true.

Correctness:
A directed graph is psuedo-connected if it contains at most one source SCC and one sink SCC. This statement is correct because we know if two source components exist, then the vertices in the two components will have no path reaching to each other. Similarily for sink components. In all other cases, two vertices s and t will always maintain a path reaching either one or another.
The algorithm runs topolical sort on the metagraph and its reversed to get the source and sink component because we know the first left element after a topologically sorted graph is its source and the source of a reversed graph is the original graph's sink.
Now we simply just run explore subroutine on these two nodes and if we cannot reach to one of the other nodes in the graph, it means that at least one other source/sink exists. Thus showing that the graph is not psuedo-connected, and vice versa.
To see if the SCC is a sink, simply just do DFS on the SCC and if it reaches to other nodes that's not in the component, we know that the SCC is not a sink.

Runtime analysis:
Applying SCCs algorithm, topolocial sorting and explore subrountine all run at linear time $O(|V| + |E|)$ time. Therefore our runtime complexity is $O(|V| + |E|)$.

Problem 2 (MST algorithms).

1.
That algorithm fails when given this counterexample:



When edge BC and FG are cut, the weight of the two spanning trees are 30 and 30.
Connecting either BC or FG gives a final weight of 30+30+1=61.
However, we can see that a less heavy spanning tree can be made using AB, BC, CD, DH, HG, GF, FE. Which the weight is 10+1+10+10+10+1+10=52.

2.
This algorithm works because since we know that if edge e is the heaviest edge in a cycle C in the graph G, it cannot be in the MST.
Therefore, by deleting all possible heavy edges in cycle, we get a MST.