Please enter your name and uID below.

Name: Devin White

uID:u0775759

Collaborators, if any, and how you collaborated:

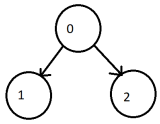
Submission notes

- Due at 11:59 pm (midnight) on Thursday, Nov 10th.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Semi-Connected Streets)

   Definitions: A **directed graph G** is semi-connected if, for every pair of vertices u and v, either u is reachable from v or v is reachable from u (or both). A **Hamiltonian Path** is a path through which all vertices in the entire graph are connected

   (a) The trivial example of this would be a disconnected DAG consisting of the verices V0 and V1 with an empty set of edges E(no edges between them) which is *technically* a graph, but not an exciting one However, as it is specified that there is a **unique** source, then One example of a DAG with a unique source that is not semi connected consists of the vertices [v0,v1,v2] with the edges v0→v1 and v0→v3. such that V0 is a unique source connected to V1 and V2 and V1 and V2 are sinks. Extending this further, a tree data structure with no Hamiltonian paths are other possible examples of a DAG that is not semi-connected, such as a Trie. This is shown below:

   (b) **A FAST** Algorithm to find whether a graph is semi connected can be done by performing a DFS/topological sort on the graph using adjacency lists . With this topological sorting, we can check whether for each vertex, there is an edge between $V_i \xrightarrow{e} V_{i+1}$ if there is no such edge, then the graph is not semi connected, otherwise, if for every V_i $V_i \xrightarrow{e} V_{i+1}$ then the graph is a semi-connected graph. This runs in O(V+E) time because topological sort/DFS only needs to check/visit each vertex and edge a single time while using/iterating through the topological order is also constant time making it all O(V+E) **For a brute force O(V(V+E)) approach**we can check every single vertex, and check each path on that vertex, seeing if at least one path connects that vertex to every other vertex that comes after it(everything besides its parent), returning false if there does not exist such a path. We then set the next vertex we check to a "source" and remove the previously checked vertex, and repeat this. This is O(V(V+E)) because for each vertex we're checking each subsequent vertex and all of the relevant edges, meaning we're performing a O(V+E) algorithm V times which comes out to O(V(V+E))

   (c) Given a DAG, A linear time algorithm to find whether a graph is semi connected can be done by performing a DFS/topological sort on the graph using adjacency lists, keeping track of the topological order. We can then go through each of the V in their topological order u→v(source→sink), and check whether for each vertex, there exists an edge between $V_i \xrightarrow{e} V_{i+1}$ if there is no such edge, then the graph is not semi connected, and we return false otherwise, if for every vertex V_i $V_i \xrightarrow{e} V_{i+1}$ in the topological sort then the graph is a semi-connected graph.(Every pair(V_i,V_{+1} must be connected for the graph to be semi-connected)
   This algorithm works because of the inherent properties of DAGS and topological sorts. By definition, a semi connected graph must have a unique topological sort(in which there is a Hamiltonian path). As a topological sort of a DAG ensures that vertices are ordered in-order of connection to other vertices(eg. if there is a path $V_i \xrightarrow{e} V_{i+1} \xrightarrow{e} V_{i+2}$ then the topological order MUST be V_i,V_{i+1},V_{i+2} this means that if V_i has no edge connecting it to V_{i+1} Then the topological sort is not unique, and there is no Hamiltonian path U,V or V,U and therefore it is not semi-connected. The running time of this algorithm is O(V+E) because it performs a DFS/Topological sort, which checks/visits each vertex and edge connected to each vertex once, while using/iterating through the topological order is also constant time

2. (Max-Min Edges)
Definitions:

- **Minimum spanning tree:** a subset of a connected, edge-weighted undirected graph which connects all the vertices together, without any cycles and using the minimum possible total edge weight Cycle: a non-empty trail in which only the first and last vertices are equal(e.g the path $v1 \rightarrow v2 \rightarrow v1$ is a cycle)
ASSUME for A) and B) that as each edge is distinct, by definition then there MUST be a distinct MST

(a) Assume for the sake of contradiction that there is a minimum spanning tree, which we'll call mMST which contains a cycle, C, and that C contains the maximum weight edge in the tree, mE such that mE ϵMST. Because mE is contained within a cycle, then by the definition above of a cycle there must be within this cycle another edge of a lower weight than mE, which we will call lE, we also know that from the definition above of a minimum spanning tree that since mEϵmMST then lE!ϵmMST. That is to say, if mE is in MST than lE CANNOT also be in mMST otherwise a cycle would be formed, conflicting with the definition of a MST
We can conclude from our assumptions above that the weight(ω), of ωlE is less than ωmE as mE is the maximal wieghted edge in the cycle. Using the definition of MSTs and these facts, We can create a new tree, called lMST, by removing mE and replacing it with lE. by replacing mE with lE, the total weight of lMST is less than mMST since ωlE $< \omega$mE. Since ωlMST ¡ ωmMST this violates the assumption that mMST is a minimum spanning tree, as by the definition of an MST above, an MST **MUST** use the minimum possible total edge weight. Therefore, it must be the case that the minimum spanning tree of an arbitrary graph, G excludes the maximum weight edge in any cycle **Thus the assertion is proven.**

(b) **Disprove:**
Suppose, (along with any needed assumptions already stated above) for the sake of contradiction that there exists a graph G that consists of the vertices V0,V1,V2,V3, with Edges E and weights ω. Assume that this graph is connected by the following edges (with weights beneath the arrows): v0$\underset{\omega 2}{\longrightarrow}$V1, v1$\underset{\omega 4}{\longrightarrow}$V2 v1$\underset{\omega 6}{\longrightarrow}$V3 v0$\underset{\omega 8}{\longrightarrow}$V2 v0$\underset{\omega 10}{\longrightarrow}$V3 v2$\underset{\omega 12}{\longrightarrow}$V3.and that the MST is the minimal edges of each cycle. If we take the minimal edge in every cycle of this graph, then we end up with an MST of total weight 16. As distinct edges make an MST unique, This is a contradiction. As can be seen by the provided vertices, the true Minimum spanning tree is of total weight 12, consisting of the edges V0$\underset{\omega 2}{\longrightarrow}$V1,V1 $\underset{\omega 4}{\longrightarrow}$V2 and V1$\underset{\omega 6}{\longrightarrow}$V3. This leaves a cycle between V0, V2, and V3, V0$\rightarrow$V2$\rightarrow$V3$\rightarrow$V0 in which **NO** cycle edges are part of the MST. This contradicts with the assertion that the MST of G includes the minimum weight edge in every single cycle of G. this is shown below with the MST in red and the weights adjacent to the path **Thus the assertion is disproven.**