Name: Vera Duong

ID: 109431166

CSCI 3104, Algorithms **Due March 12, 2021**

Problem Set 7 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- The easiest way to access Gradescope is through our Canvas page. There is a Gradescope button in the left menu.
- Gradescope will only accept **.pdf** files.
- It is vital that you match each problem part with your work. Skip to 1:40 to just see the matching info.

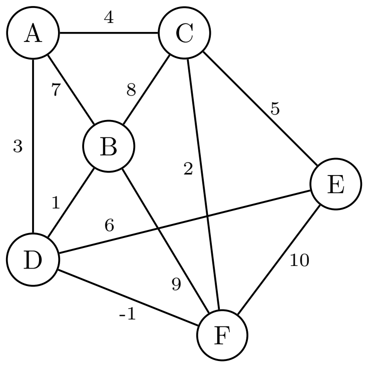
Name: Vera Duong

ID: 109431166

CSCI 3104, Algorithms **Due March 12, 2021**

Problem Set 7 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub

1. Consider the following graph:



(a) Use Kruskal's algorithm to compute the MST. It will suffice to indicate the order in which edges are added to the MST.

(b) Now use Prim's algorithm starting at node A to compute the MST, again by indicating the order in which edges are added to the MST.

(c) Is it possible to change the starting node for Prim's algorithm such that it adds edges to the MST in the same order as Kruskal's algorithm? If so, which starting node(s) would work? Justify your answer.

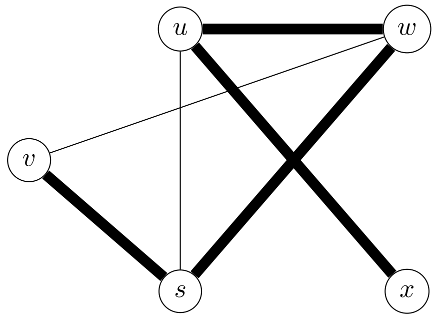
Note: For parts (a) and (b), let (Node1, Node2) represent the edge between two nodes Node1 and Node2. Therefore, your answer should have the form: (Node1, Node2), (Node4, Node5), etc.

Solution:
(a) Kruskal's is a greedy algorithm for growing an MST. It builds it by picking the smallest edge cost to largest until we've included all the nodes (with V-1 edges) and skipping edges that make a cycle. In this graph, we have **(D, F), (D,B), (C,F), (A,D), (C,E)** which creates the MST. The edges in order have a cost of -1, 1, 2, 3, and 5. We skip (A,C) because it makes a cycle, which is why (C,E) is the last edge added since it's the next smallest after (A,C).

(b) Prim's is similar to Kruskal's, however it selects the minimum that connects to the existing tree and also begins at a vertex. From Prim's our MST is **(A,D), (D,F), (D,B), (F,C), (C,E)**. From A, the smallest edge connected is (A,D). The remaining edges connected to A and D are: B, C, E, and F. We pick (D,F) next since it's the smallest. The remaining edges connected to A, D and F are: B, C and E. We pick (F,C) next. The remaining edges connected to A, B, D, and F are: E. We pick (C,E) last.

(c) It is possible because we can start at D and F and get the same order as Kruskal's, which is **(D, F), (D,B), (C,F), (A,D), (C,E)**. This is because we can see for Kruskal's, it must first have an edge added as (D,F). This means that in Prim's algorithm, we must also start at either D or F to accomplish adding the first edge of (D,F) to the MST. Since their edges have the smallest cost of all the edges (-1), we can add according to Prim's and get the same order as in part (a) with Kruskal's.

2. Consider the undirected, unweighted graph $G = (V, E)$ with $V = \{s, u, v, w, x\}$ and $E = \{(s, u), (s, v), (s, w), (u, w), (u, x), (v, w)\}$, and let $T \subset E$ be $T = \{(s, v), (s, w), (u, w), (u, x)\}$. This is pictured below with T represented by wide edges.



Demonstrate that T cannot be output by BFS with start vertex s.

Solution:

(2) BFS traverses a graph and explores all the nodes at the current depth and then moves on to the nodes at the next depth. We would mark each node connected to s as visited, and enqueue them. T is not a proper output by BFS because it does not include (s, u) as one of the first three edges (and doesn't show up in T at all), breaking the rule that we visit "layer by layer." The correct output of BFS on this graph should be: $\{(s, u), (s, v), (s, w), (u, x)\}$.

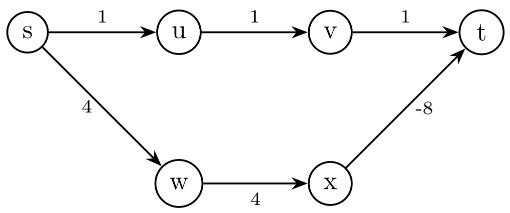
Name: Vera Duong

ID: 109431166

CSCI 3104, Algorithms **Due March 12, 2021**

Problem Set 7 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub

3. Given the following directed graph $G = (V, E)$ with starting and ending vertices $s, t \in V$, show that Dijkstra's algorithm does *not* find the shortest path between s and t.



Solution:

(3) Dijkstra's is an algorithm for finding the the shortest path between nodes on a graph. It only visits nodes that are connected to its parents that have already been visited and picks paths between those nodes in ascending edge costs, which is why the graph given does not find the shortest path with Dijkstra's due to it's layout. By starting at s, Dijkstra's would pick u since $(s, u) = 1$ which is smaller than $(s, w) = 4$. It then picks the next node with the smallest edge based off its connection to s and u. We then pick (u, v), then (v, t) to reach t since all their costs are less than 4. These costs would add up to 3, while the shortest path would add up to 0.

Dijkstra's: $(s, u), (u, v), (v, t)$ adding up to 3 (1 + 1 + 1)

Shortest Path: $(s, w), (w, x), (x, t)$ adding up to 0 (4 + 4 + -8).

4. Given a graph, implement Prim's algorithm via Python 3. The input of the Graph class is an Adjacency Matrix. Complete the Prim function. The Prim fucntion should return the weight sum of the minimum spanning tree starting from node 0.

   The file `graph.py` is provided; use this file to construct your solution and upload with the same name. You may add class variables and methods but **DO NOT MODIFY THE PROVIDED FUNCTION OR CLASS PROTOTYPES.**.

   Here is an example of how your code will be called:

   Sample input:

```
g = Graph([ [0, 10, 11, 33, 60],
            [10, 0, 22, 14, 57],
            [11, 22, 0, 11, 17],
            [33, 14, 11, 0, 9],
            [60, 57, 17, 9, 0]])

assert g.Prim() == 41
```