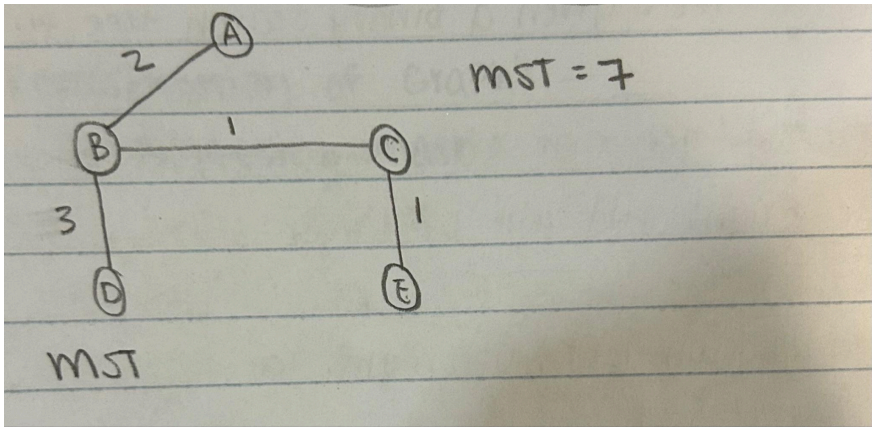
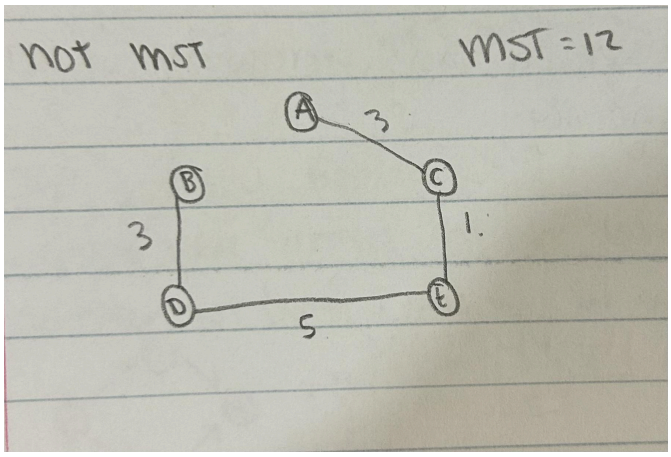
CS325: Homework 8

---

1. **Draw Minimum spanning tree**
    a. **Draw minimum spanning tree**



    b. **Draw spanning tree that is not minimum**



2. **MST Implementation: Prim's**
    a. **Implement in .py file → in gradescope**
    b. **What is the difference between prims and kruskals**

The difference between Prim's and Kruskal's is the way that the new edges / vertices are selected. In prim's, you start at any node then compare from that node to vertices that are connected to it by an edge. You pick the smallest edge then continue with the process until you have gotten all the vertices. On the other hand, with Kruskals you start with the minimal edge. From there you continue to add edges by picking the smallest edge, while ensuring it does not create a cycle. All in all, with prims you can only select edges that have a visited vertice while with Kruskals you create the MST by continuously picking the minimal edges while not creating a cycle.

3. **Apply graph traversal to solve a problem**
    a. **Describe an algorithm to solve the problem**

For this problem, we are going to want to perform graph traversal. Remembering what we had from the previous homework, I think that some aspects apply. Firstly, I would want to use BFS because it allows me to find the shortest path as it checks all directions before continuing. Also, at this stage of checking the next spot, you could implement a checker to see if you can occupy that location or if you can't. I would not use DFS just because from looking at the outputs given and thinking of a puzzle, I would want the shortest path. Also, since it is not weighed I woud not need to use something like dijkstra's.

To describe my pseudocode I would initialize an empty queue for BFS, enqueue the source cell into the queue, and use either a hashmap or dictionary to store visited vertices. I would then enter a traversing loop, which first checks that the queue isnt empty. Then, set an if the cell is the destination cell, if yes then return path. Else, then enqueue neighboring cells that havent been visited and put the current cell as visited. If the destination path is reached then store the path in the list. If the list is not empty then return list, if empty then return none which would mean you could not get to that destination cell.

    b. **Implement in .py file → in gradescope**
    c. **Time complexity**
    The time complexity would be O(rows * columns)
    d. **Extra credit**