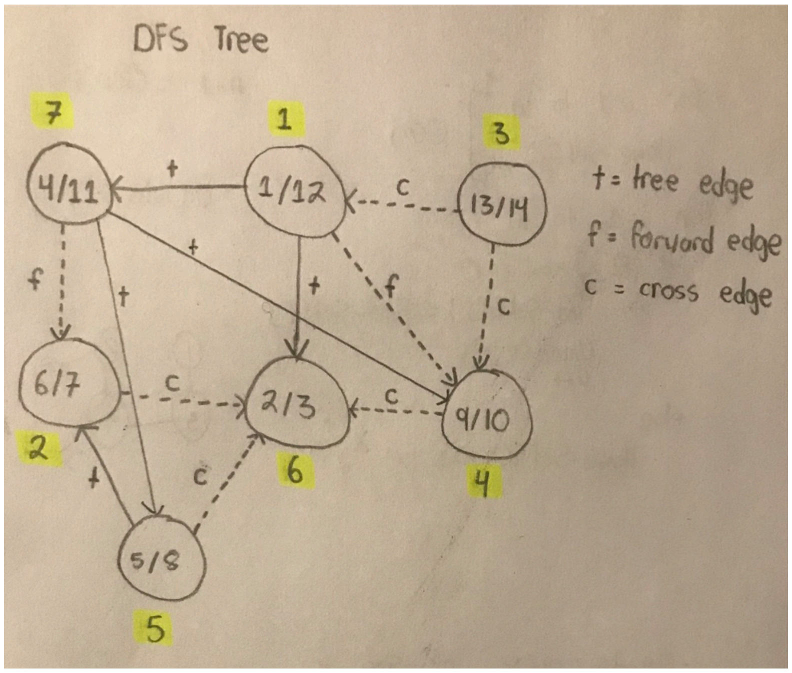
Assignment 6

Question 1

a)



(1, (6, 6), (7, (5, (2, 2), 5), (4, 4), 7), 1), (3, 3)

d[1] = 1, d[6] = 2, f[6] = 3, d[7] = 4, d[5] = 5, d[2] = 6, f[2] = 7, f[5] = 8, d[4] = 9, f[4] = 10,

f[7] = 11, f[1] = 12, d[3] = 13, d[3] = 14

b)

There are 0 back edges, 2 forward edges, and 5 cross edges.

c)

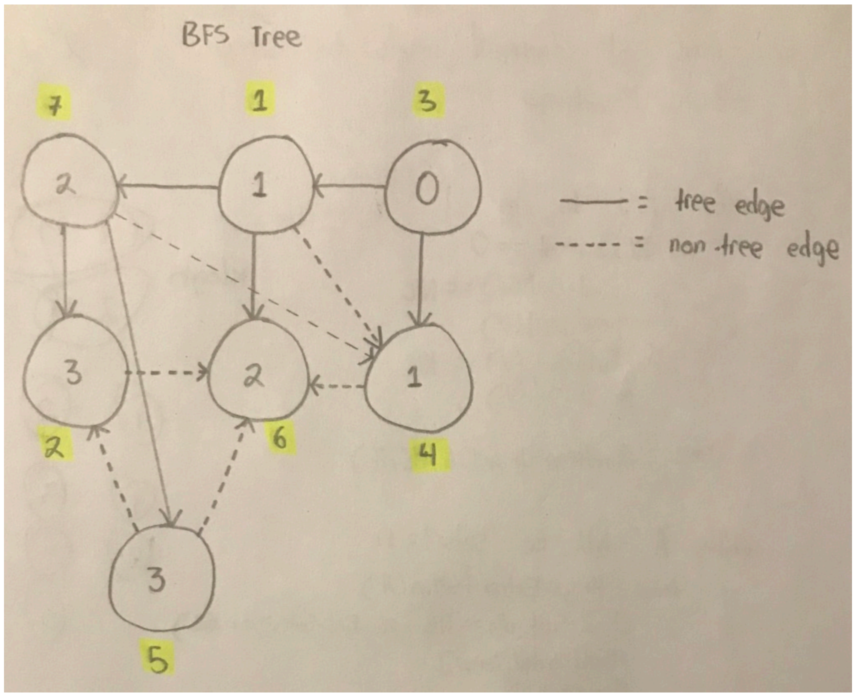
By Lemma 22.11, since there are no back edges in the DFS from part a), this means that G is acyclic. Since there are no cycles in G, this means that given any 3 courses, let's call them a, b, and c, it is impossible for a to be a prerequisite for b, b be a prerequisite for c, and c be a prerequisite for a, which means the three courses can be taken in a sequential order that satisfies the prerequisite requirements. This reasoning can be applied to every node in G so all the courses can be taken in a sequential order that satisfies the prerequisite requirements.

d)

3 -> 1 -> 7 -> 4 -> 5 -> 2 -> 6

I found this order using Topological-Sort on the DFS from part a).

e)



(3, (1, (4, 3), (6, (7, 1), 4), 6), (5, (2, 7), 5), 2)

d[3] = 1, d[1] = 2, d[4] = 3, f[3] = 4, d[6] = 5, d[7] = 6, f[1] = 7, f[4] = 8, f[6] = 9, d[5] = 10, d[2] = 11, f[7] = 12, f[5] = 13, f[2] = 14

Question 2

My algorithm will first iterate over every node from 1 to n and create a set for each node. Next it will iterate over every road in R and check the cost. If the cost is 0 then this means the road is not damaged, so both nodes connected by this road get merged together (if they do not already belong to the same set) so that they belong to the same set. Every time two nodes get merged together, the counter *visited* (*visited* is initially set to 0) will be incremented by 1, this is to keep track of the number of calls to Union the algorithm has made. If the road is damaged, the road gets added to a Min-Heap denoted by *A*. The key of each road in *A* will be the cost of repairing said road. After all the roads have been visited, a while loop begins iterating over *A* (using a maximum of k calls to Extract-Min). The while loop will stop iterating once *A* is empty or when *visited* == n - 1. Once *visited* == n - 1, this means that all the nodes in G are accessible with the current repaired and undamaged roads. Since *A* is a Min-Heap, the root of *A* always has the smallest cost which ensures the total cost of repairs will be minimized. If the two nodes in the edge of the current root of *A* belong to different sets, then this road gets added to the list *result*, the two nodes get merged together, and visited is incremented by 1. If they belong to the same set, the loop moves on to the new root of *A*. The list *result* includes the roads that are to be repaired.

My algorithm visits every road in R and only repairs the roads that will make one more node accessible then there would be if it wasn't repaired, and it does so in a way that minimizes the cost by arranging the damaged roads in a Min-Heap. Since there are n nodes, there can only be a maximum of n - 1 calls to Union, so after n - 1 calls to Union, G is guaranteed to be connected.

The first for loop iterates n times and Make-Set runs in O(1) time so the first loop runs in O(n) time. The second for loop iterates m times and either Union is called or Min-Heap-Insert is called. Union runs in O(logn) time and Min-Heap-Insert runs in O(logk) time. The while loop iterates at most k times and Extract-Min runs in O(logk) time. The total time complexity of the algorithm is therefore, O(n + mlog*m + klogk).