Task

- [x] Create MSTHerrera
- [x] Explain logic
- [x] No redundant trees

The goal of my task was to come up with functional code that would return all minimum spanning trees given a graph, with no redundant MST's, with a basis of Prim's or Kruskal's algorithm.

Logic of Algorithm

I began working on my code off the basis of Prim's algorithm. The first challenge I ran into was how to run the finding MST multiple times while taking into consideration what it's already found beforehand. I quickly realized that recursion would be necessary as multiple iterations of the same process would need to be run. Thus I made FindSpanningTree which takes in the List of Minimum Spanning Trees I've found, a set of vertices I've already visited, and a queue with unique unvisited edges inside. MSTHerrera still begins at the zero vertex. It then populates a queue with incoming edges to zero. I then begin the recursive part of my code that will eventually populate my List of Spanning trees which I then return.

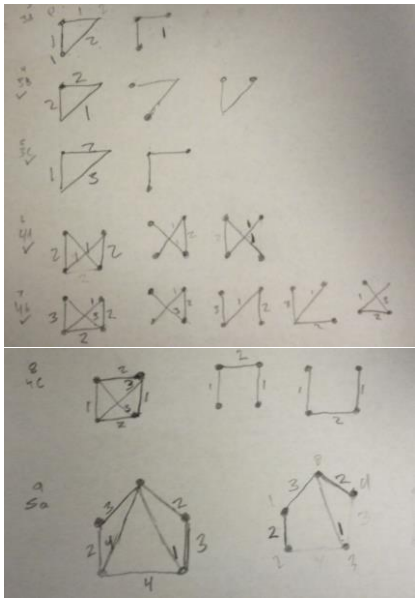
FindSpanningTree() is my recursive method. It begins with the base case by checking that if the size of the current tree (plus one) that it is currently working on is equal to the graph size, then to add that tree to the list of minimum spanning trees. The idea is that no tree will make it this far in the code unless it is unique from the all the other MST in my list of MST's already. I then continue on by removing Duplicates in my Queue that point toward a vertex already visited beforehand. Should the queue be empty after doing this, I then return to begin assessing a different tree. This is because the tree I was working on wouldn't result in a unique Tree.

After this filtering of the Queue, comes the do-while loop. Essentially as the queue isn't empty and the current nested edge is equal to the next edge on that level's queue, then continue popping and searching for a MST. These parameters prevent trying to pop a null edge and filling the current Tree with garbage edges whose weights are largest.

Testing

I found MSTAllTest on the class gibhub but realized we were missing MSTAllPrim. Putting my MSTHerrera as both the gold and system files didn't help either as the program would run my code against itself to make sure it came out with the same answers. So I resorted to the old fashioned paper and pencil method. Using Choi's Test File with print statements and my drawn out answers, I simply compared answers to make sure I was passing them all.

Hand Drawn Test Case Answers



MSTHerrera Results after running MSTAllTest w/print statements

```
0 <- 1 : 1.0
0 <- 2 : 1.0
2.0
3: true

0 <- 1 : 2.0
1 <- 2 : 1.0
3.0
0 <- 2 : 2.0
2 <- 1 : 1.0
3.0
4: true

0 <- 1 : 1.0
0 <- 2 : 2.0
5: true
```

```
0 <- 3 : 1.0
0 <- 1 : 2.0
1 <- 2 : 1.0
4.0
0 <- 3 : 1.0
3 <- 2 : 2.0
2 <- 1 : 1.0
4.0
6: true

0 <- 1 : 3.0
1 <- 2 : 1.0
1 <- 3 : 2.0
6.0
0 <- 1 : 3.0
1 <- 2 : 1.0
2 <- 3 : 2.0
6.0
```

```
0 <- 3 : 3.0
3 <- 1 : 2.0
1 <- 2 : 1.0
6.0
0 <- 3 : 3.0
3 <- 2 : 2.0
2 <- 1 : 1.0
7: true

0 <- 1 : 1.0
0 <- 2 : 2.0
2 <- 3 : 1.0
4.0
0 <- 1 : 1.0
1 <- 3 : 2.0
3 <- 2 : 1.0
8: true
```

```
0 <- 3 : 1.0
0 <- 4 : 2.0
0 <- 1 : 3.0
1 <- 2 : 2.0
8.0
 9: true
Score: 9/9
```

Closing Thoughts

I'm happy I was able to get MSTHerrera working based off of Prim's algorithm using recursion and filtering out redundant MST's. I'm sure other could find a way to complete the task faster but run time wasn't a concern of mine. After figuring out that I could easily print my graph and view where I was in the process, completing the task was much easier and doable.