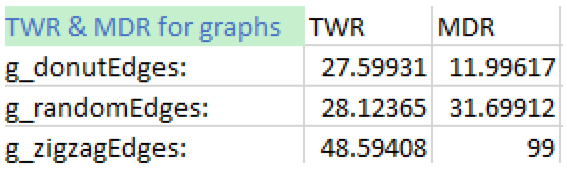
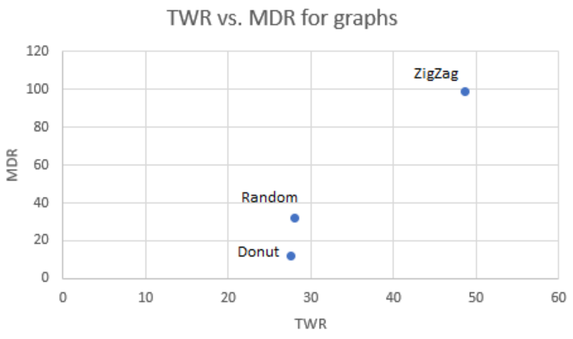
Programming Assignment 2 Part 2

1-



TWR & MDR for graphs	TWR	MDR
g_donutEdges:	27.59931	11.99617
g_randomEdges:	28.12365	31.69912
g_zigzagEdges:	48.59408	99



2-

Donut Graph:

Since this graph had the lowest MDR out of all the other graphs, then it makes sense that we choose the MST version of this graph. In addition, it appears that its TWR also was the best out of all of the graphs but not by much. Hence, it makes more sense that we focus on the MDR and therefore **pick its MST.**

Random Graph:

As for the random graph, it appears that its TWR is a significant amount lower than its MDR. Hence, we would pick its **shortest path tree** rather than its MST

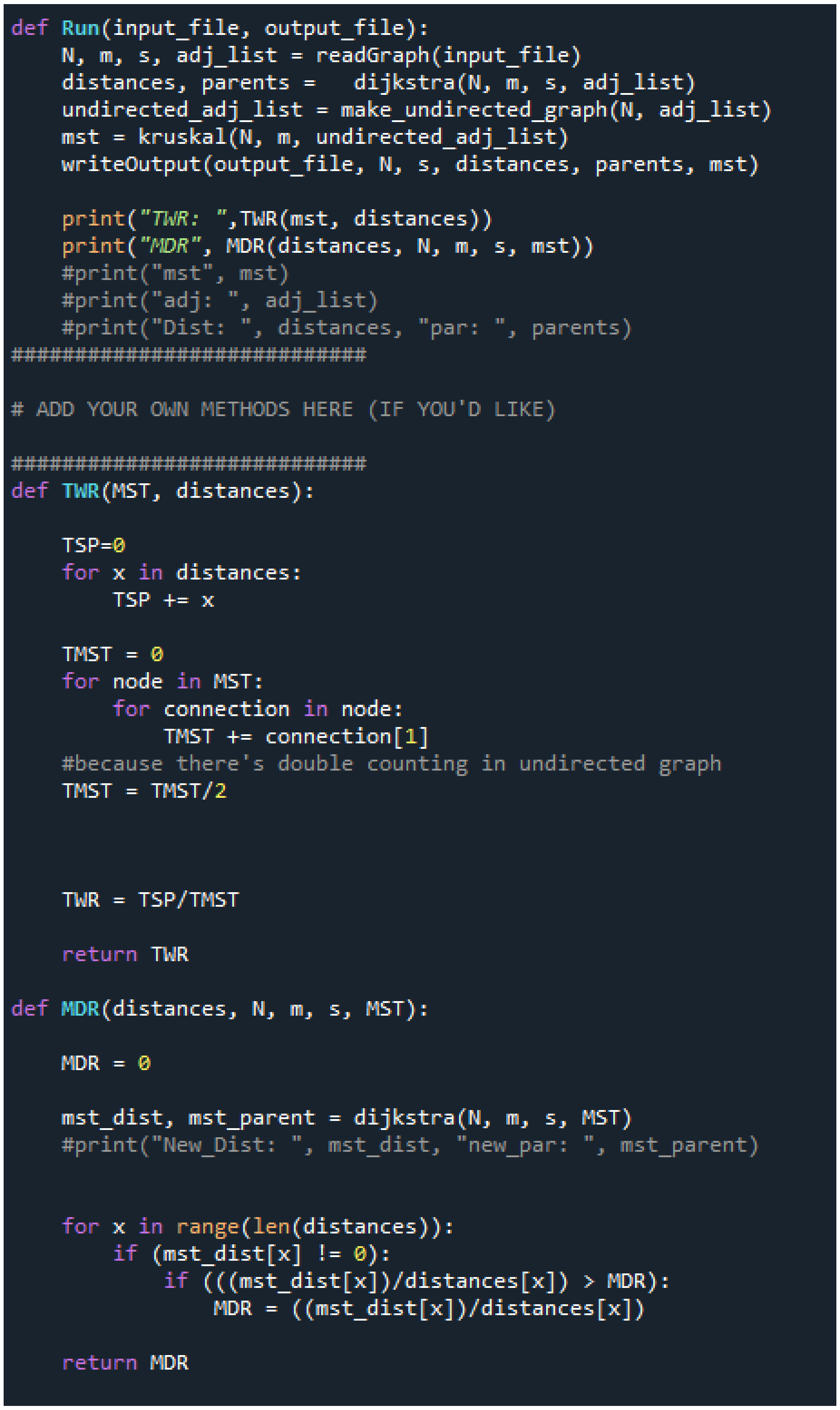
ZigZag Graph:

Overall, this graph just sucks. It does quite a lot worse than the other two graphs. However, its TWR is about half of its MDR, so for that reason I would pick the **shortest path tree** version for this graph.

Closing comments:

This programming assignment was pretty informative, and I can see now why the structure for a lot of things is usually “donut” shaped with the source node being in the middle. For example, the “donut” graph reminded me of the structure of a “Clash of Clans” village and how the source node/town hall (and often also the soldiers/barbarians) are usually placed in the middle. However, I can see why other structures would be used when one criteria is much more important than the other such as when trying to distribute resources like food and the weight of every edge is pretty small.

3-



```python
def Run(input_file, output_file):
    N, m, s, adj_list = readGraph(input_file)
    distances, parents =   dijkstra(N, m, s, adj_list)
    undirected_adj_list = make_undirected_graph(N, adj_list)
    mst = kruskal(N, m, undirected_adj_list)
    writeOutput(output_file, N, s, distances, parents, mst)

    print("TWR: ",TWR(mst, distances))
    print("MDR", MDR(distances, N, m, s, mst))
    #print("mst", mst)
    #print("adj: ", adj_list)
    #print("Dist: ", distances, "par: ", parents)
############################

# ADD YOUR OWN METHODS HERE (IF YOU'D LIKE)

############################
def TWR(MST, distances):

    TSP=0
    for x in distances:
        TSP += x

    TMST = 0
    for node in MST:
        for connection in node:
            TMST += connection[1]
    #because there's double counting in undirected graph
    TMST = TMST/2



    TWR = TSP/TMST

    return TWR

def MDR(distances, N, m, s, MST):

    MDR = 0

    mst_dist, mst_parent = dijkstra(N, m, s, MST)
    #print("New_Dist: ", mst_dist, "new_par: ", mst_parent)


    for x in range(len(distances)):
        if (mst_dist[x] != 0):
            if (((mst_dist[x])/distances[x]) > MDR):
                MDR = ((mst_dist[x])/distances[x])

    return MDR
```