Let’s list the pros and cons of the two different ways we currently have of returning graphing info from R

# Option 1: Correlation Matrix

* This method corresponds to the canonical way of storing a graph as a matrix.
* Does not allow us to know neighbours and is pretty much useless for creating a custom layout such as clustering.
* Perhaps we could extend this method to return additional information in addition to the correlation matrix. This info could be lists of which nodes are first neighbours and which are second neighbours (based on the genes of interest selected in the main tab).
  + The issue with this approach is that we need to hardcode the reading in of more properties from the R output. Our current method on the server called getWEightsAndDegreesFromROutput already has quite a few layer of reading to it
  + Suppose that the new info we are returning from R are lists of genes indicating which genes are first neighbours and which genes are second neighbours.
  + Furthermore, we would still have to create some sort of O(n^2) method on the server side that iterates through our nodes and checks if they are in one of these lists. If so, a node is positioned accordingly. Mind you that our method that creates all of the edges is already O(n^2) so if we were to stick to this correlation matrix approach, it isn’t to big of a deal to check whether or not our nodes are first or second neighbours

# Option 2: Lists of Weights and Degrees

* This method has some added complexity since generating edges out of it is not entirely trivial.
* Currently, we are only using this method when we want to dissect the neighbours of a selected node. So we pick a node and we get its neighbours, then we can pick one of its neighbours and get that node’s neighbours. When creating the edges, we know which nodes to create the edges from since those are the nodes that the user selected. We also know which nodes to make the edges go to since those are the weights and corresponding names that the R script returns for each level of neighbours.
* This starts to break down after the third or fourth level since one of the nodes that we need to draw an edge from could have been a couple of levels back.

# Option 3: Return Edges and Degrees from R

* An option we have yet to explore is returning edges from R. These edges are basically triples indicating source, target, and strength. On the server side, we could parse these and easily create cytoscape edges from them.
* The degrees we would be returning from R would be the degrees of the selected nodes and their neighbours. The question remains, is degree referring to their degrees in the chosen subgraph or their degree in the overall graph.

For the sake of being consistent and having general methods as opposed to specialized methods for each little thing, we should choose one of these approaches and stick with it. I think that we will stick with the first choice for now. The reason being that it’s probably better to not have to create a list for every single edge in R. What we can do is return a list of lists when it comes to creating the nodes. The first sub-list will be our selected genes, the second sub-list will be the first neighbours, and the third sub-list will be the second neighbours. This will not be difficult to do four our getRelevantSubmatrix script, but it might be a bit harder to do for our findCorrelations script.

It turns out that implementing the findCorrelations script by having it return a correlation matrix gets complicated. The complication arises due to not knowing if we should index into a column or row. We managed to get past this, however, the matrix returned contains some values that should be zeroed out. Everything that is not a column or row that was specified should be zeroed out.

For example, we specified c(“TBP-E”, “UBE2C-S”, “STAT3-E”) and got the following matrix:

UBE2C-S TBP-S EBNA1BP2-S CDC45-S TFDP1-S

TBP-E 0.4943829 0.6379173 0.5289383 0.4917726 0.0000000

STAT3-E -0.4870864 0.0000000 0.0000000 0.0000000 -0.4615142

VPS72-E 0.4930360 0.0000000 0.0000000 0.0000000 0.0000000

UBE2C-E 0.7949256 0.0000000 0.0000000 0.4967731 0.0000000

EBNA1BP2-E 0.5264976 0.0000000 0.7561553 0.0000000 0.0000000

CDC45-E 0.5020195 0.0000000 0.0000000 0.8033812 0.0000000

However, what we want is the same matrix but with only the first two rows and first column allowed to have non-zero entries. This does seem like a somewhat flimsy solution though. Let’s see if it solves the problem that occurs when doing past the second level. It does seem to solve the issue since for selected genes of: selectedGenes = c('VPS72-E', 'UBE2C-S', 'TBP-E', 'CDC45-S')

We get the following back

UBE2C-S CDC45-S VPS72-S TBP-S EBNA1BP2-S

VPS72-E 0.4930360 0.0000000 0.6086204 0.0000000 0.0000000

TBP-E 0.4943829 0.4917726 0.0000000 0.6379173 0.5289383

UBE2C-E 0.7949256 0.4967731 0.0000000 0.0000000 0.0000000

EBNA1BP2-E 0.5264976 0.0000000 0.0000000 0.0000000 0.7561553

CDC45-E 0.5020195 0.8033812 0.0000000 0.0000000 0.0000000

STAT3-E -0.4870864 0.0000000 0.0000000 0.0000000 0.0000000

SUZ12-E 0.0000000 0.4968252 0.0000000 0.0000000 0.0000000