Class project 1 report

R function:

1. betweenness(graph, v=V(graph), directed=True, weights=NULL, nobigint=TRUE, normalized=FALSE) The vertex and edge betweenness are (roughly) defind by the number of geodesices (shortest paths) going though a vetex or an edge.
2. ecount(graph) count the edge of graph
3. get\_diameter(graph, directed=TRUE, unconnected=TRUE, weights=NULL) get the path of diameter of graph
4. max\_cliques(graph, min=NULL, max=NULL, subset=NULL, file=NULL) These functions find all, the largest or all the maximal cliques in an undirected graph. The size of the largest clique can also be calculated.
5. simplify(graph, remove.multiple=TRUE, remove.loos=TRUE, edge.attr.comb=igraph\_opt("edge.attr.comb")) Simple graphs are graphs which do not contain loop and multiple edges.
6. alpha\_centrality(graph, nodes=V(graph), alpha=1, loops=FALSE, exo=1, weights=NULL, tol=1e-07, sparse=TRUE) alpha\_centrality calculates the alpha centrality of some (or all) vertices in a graph.
7. dfs(graph, root, neimode=c('out', "in", 'all', 'total'), unreachable=TRUE, order=TRUE, order.out=FALSE, father=FALSE, dist=FALSE, in.callback=NULL, out.callback=NULL, extra=NULL, rho=parent.frame()) Depth-first search is an algorithm to traverse a graph. It starts from a root vertex and tries to go quickly as far from as possible.
8. bfs(graph, root, neimode=c('out', 'in', 'all', 'total'), unreachable=TRUE, restricted=NULL, order=TRUE, rank=FALSE, father=FALSE, pred=FALSE, succ=FALSE, dist=FALSE, callback=NULL, extra=NULL, rho=parent.frame()) Breadth-first search is an algorithm to traverse a graph. We start from a root vertex and spread along every edge “simultaneously”.
9. is.directed(graph) determine whether a graph is directed
10. is.weighted(graph) determine whether a graph is weighted
11. is.simple(graph) Simple graphs are graphs which do not contain loop and multiple edges
12. closeness(graph, vids=V(graph), mode=c('out', 'in', 'all', 'total'), weights=NULL, normalized=FALSE) Cloness centrality measures how many steps is required to access every other vertex from a given vertex.
13. is.connected(graph) determine whether a graph is connected
14. ego(graph, order, nodes=V(graph), mode=c('all', 'out'. 'in'), mindist=0) These functions find the vertices not farther than a given limit from another fixed vertex, these are called the neighborhood of the vertex.
15. power\_centrality(graph, nodes=V(graph), loops=FALSE, exponent=1, rescale=FALSE, tol=1e-07, sparse=TRUE) power\_centrality takes a graph (dat) and returns the Boncich power centralities of positions (selected by nodes). The decay rate for power contributions is specified by exponent (1 by default).
16. which\_multiple(graph, eids = E(graph)), Which\_loop decides whether the edges of the graph are loop edges. Any\_multiple decides whether the graph has any multiple edges. Which\_multiple decides whether the edges of the graph are multiple edges. Count\_multiple counts the multiplicity of each edge of a graph.

**Functions in the Lecture Notes**

In this task, we used the following code to load a sample edge file and examined some of the functions shown in the lecture notes.

> edges1<-read.table("12831.edges")

> em <-as.matrix(edges1)

> v1 <- em[1:2478,1]

> v2<-em[1:2478,2]

> relations<- data.frame(from=v1,to=v2)

> g<-graph.data.frame(relations,directed=TRUE)

> plot(g)

#3 &4a.how to load the data into a data structure usable by the R igraph packages or Python libraries? And how you simplified the graph?

step 1

get the nodeID of every file.

delete prefix and suffix of file path.

nodeID = gsub("Edges/", "", file)

nodeID = as.numeric(gsub(".edges", "", nodeID))

IDlist = c(IDlist, nodeID)

step 2

visit every file and save node and edge in two structure.

save every node in \*IDlist\* and save every path in \*relations\*

tempEdge = data.frame(from=nodeID, to=temp)

IDlist = c(IDlist, temp)

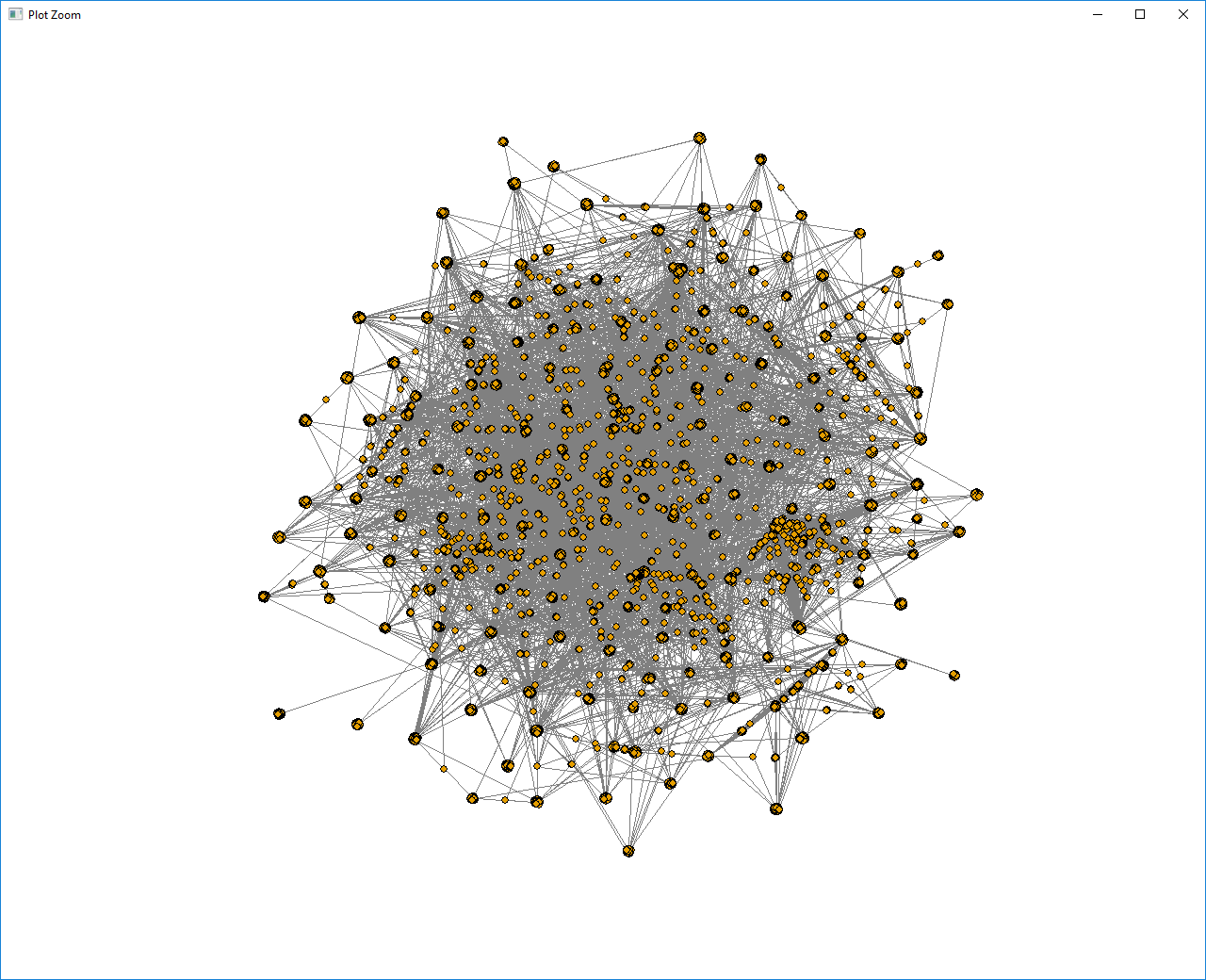
relations = rbind(relations, tempEdge)

step 3

simlify graph, remove double edges in graph.

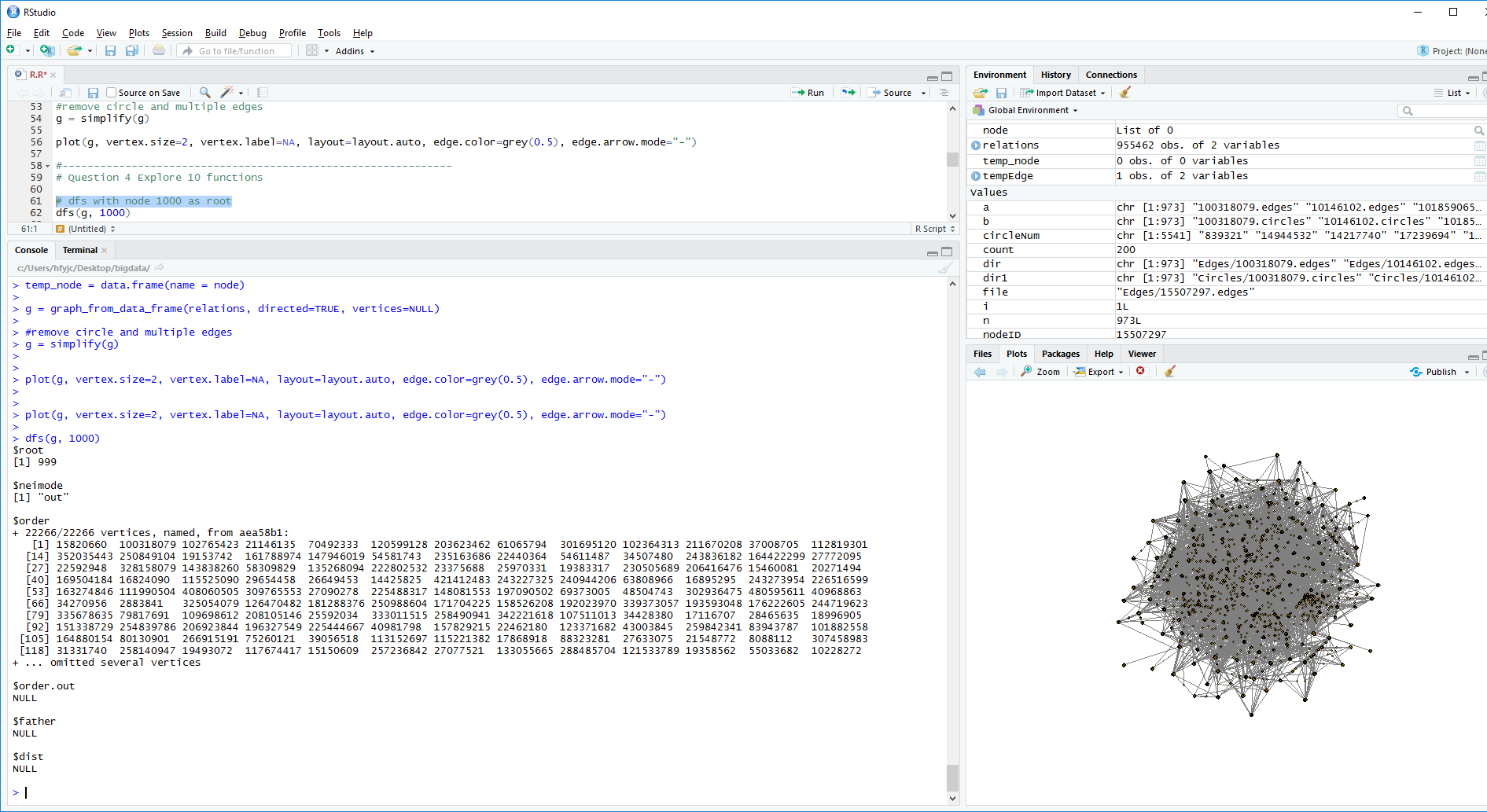
g = simplify(g)

Demonstration:

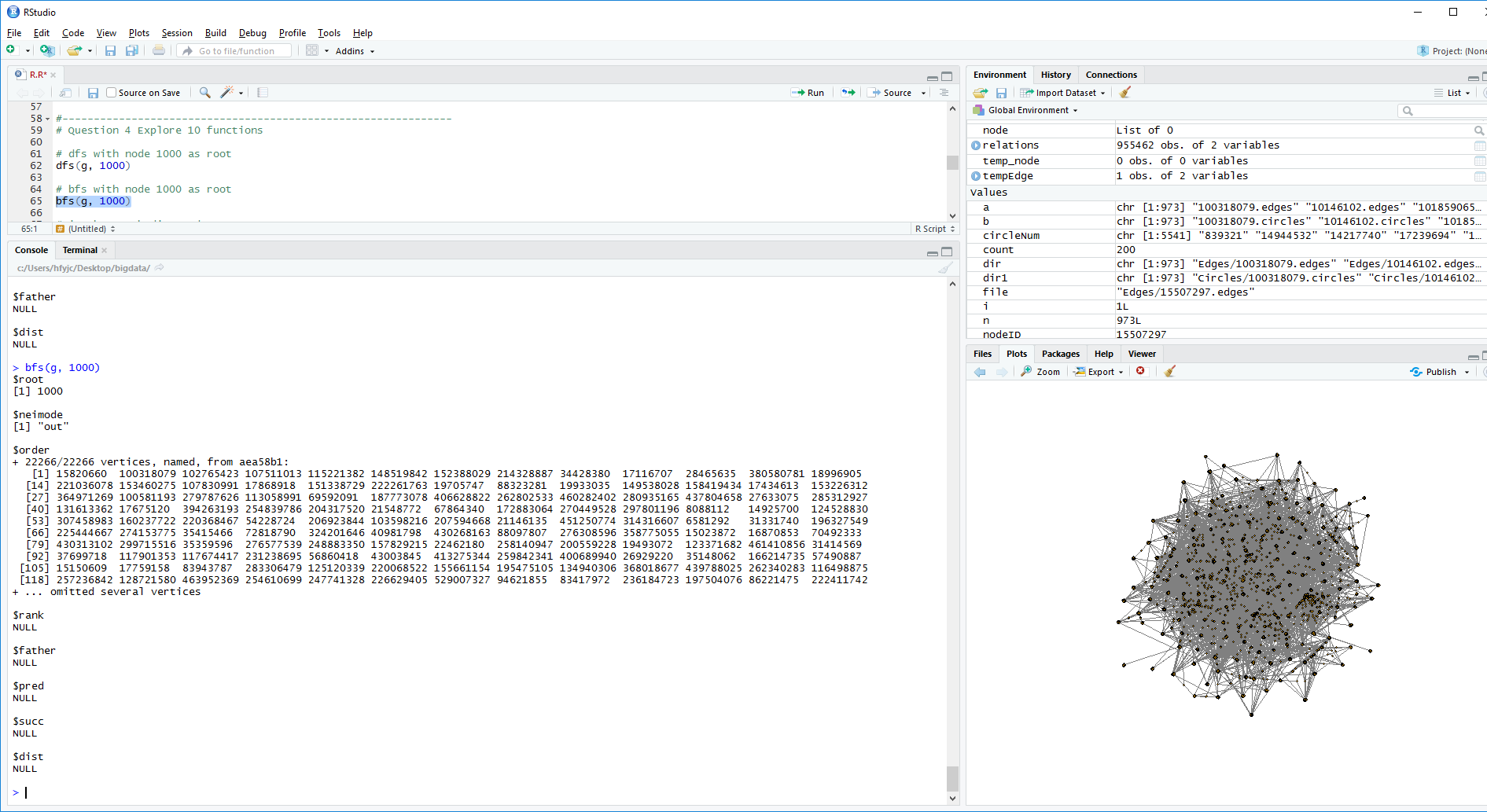


#4

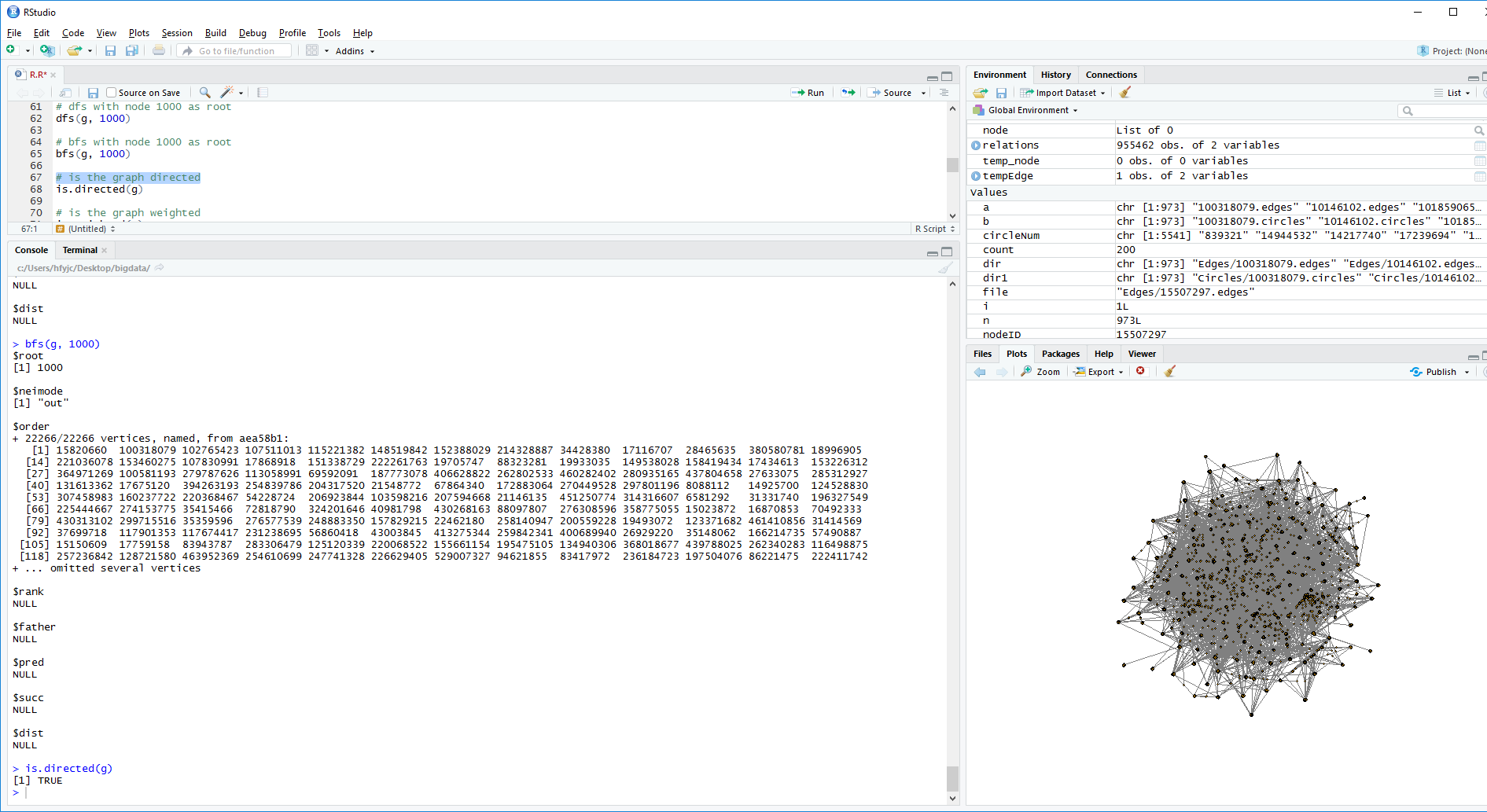
Function 1 # dfs with node 1000 as root



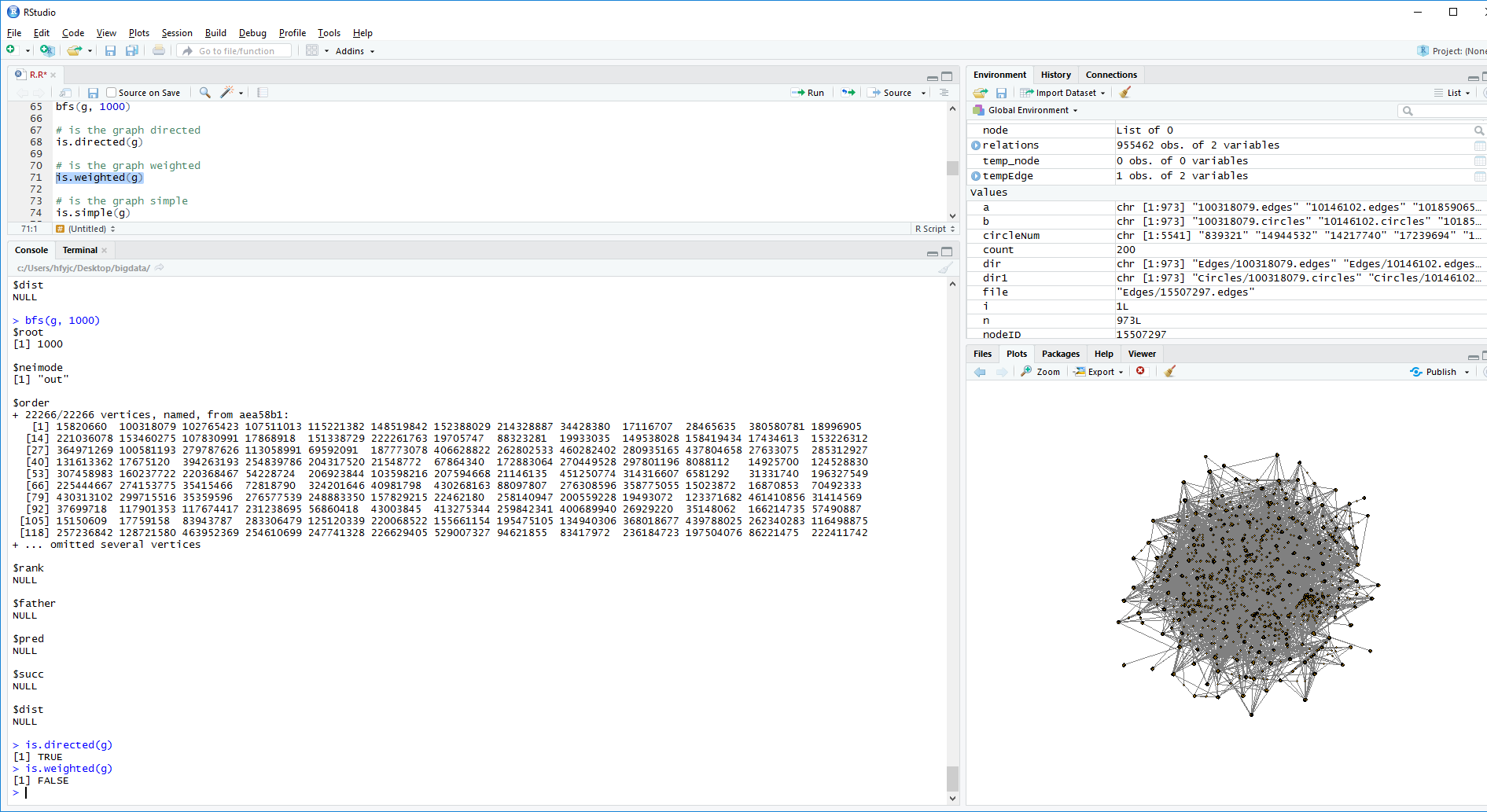
Function 2 # bfs with node 1000 as root



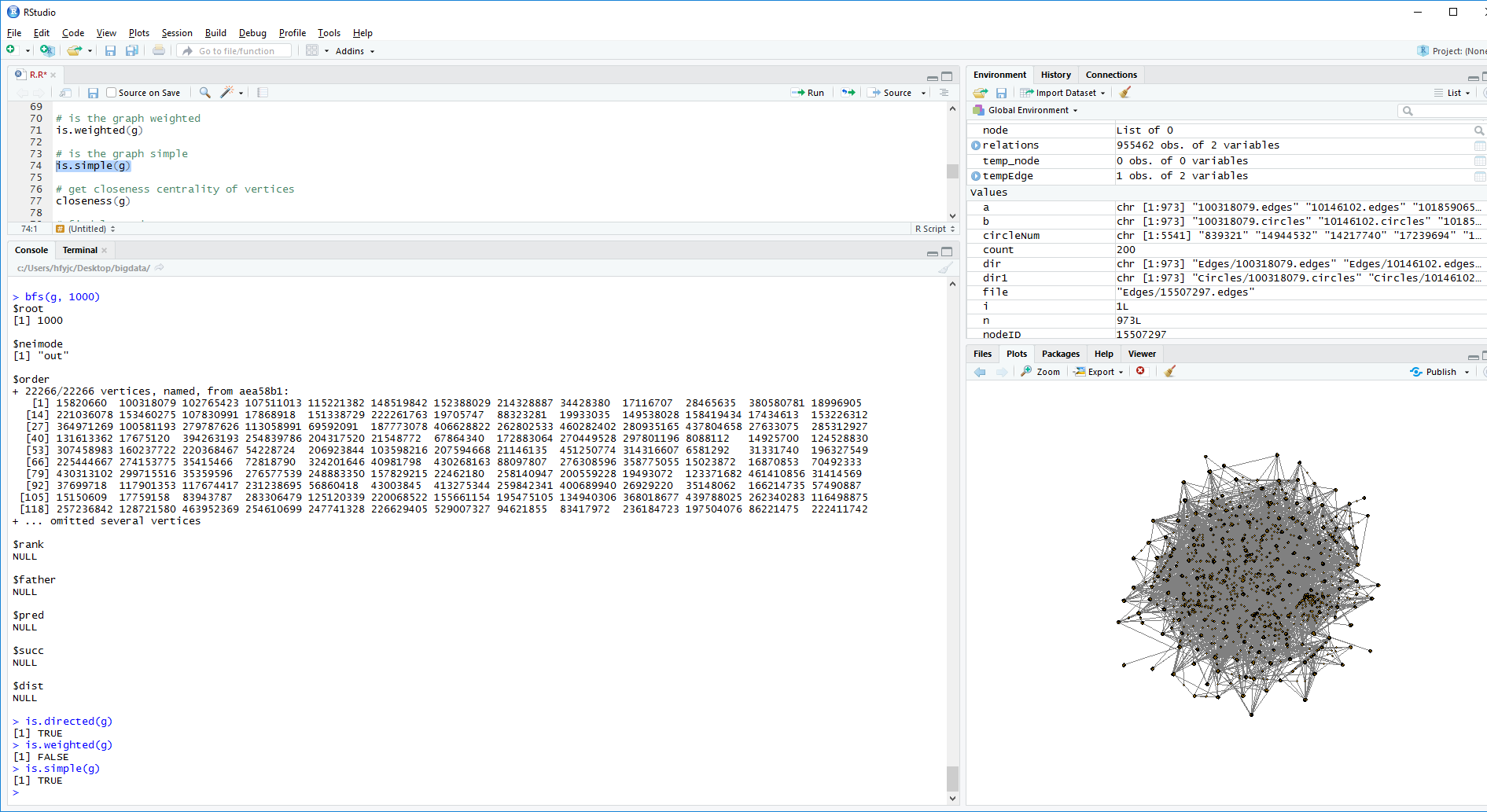
Function 3## is the graph directed



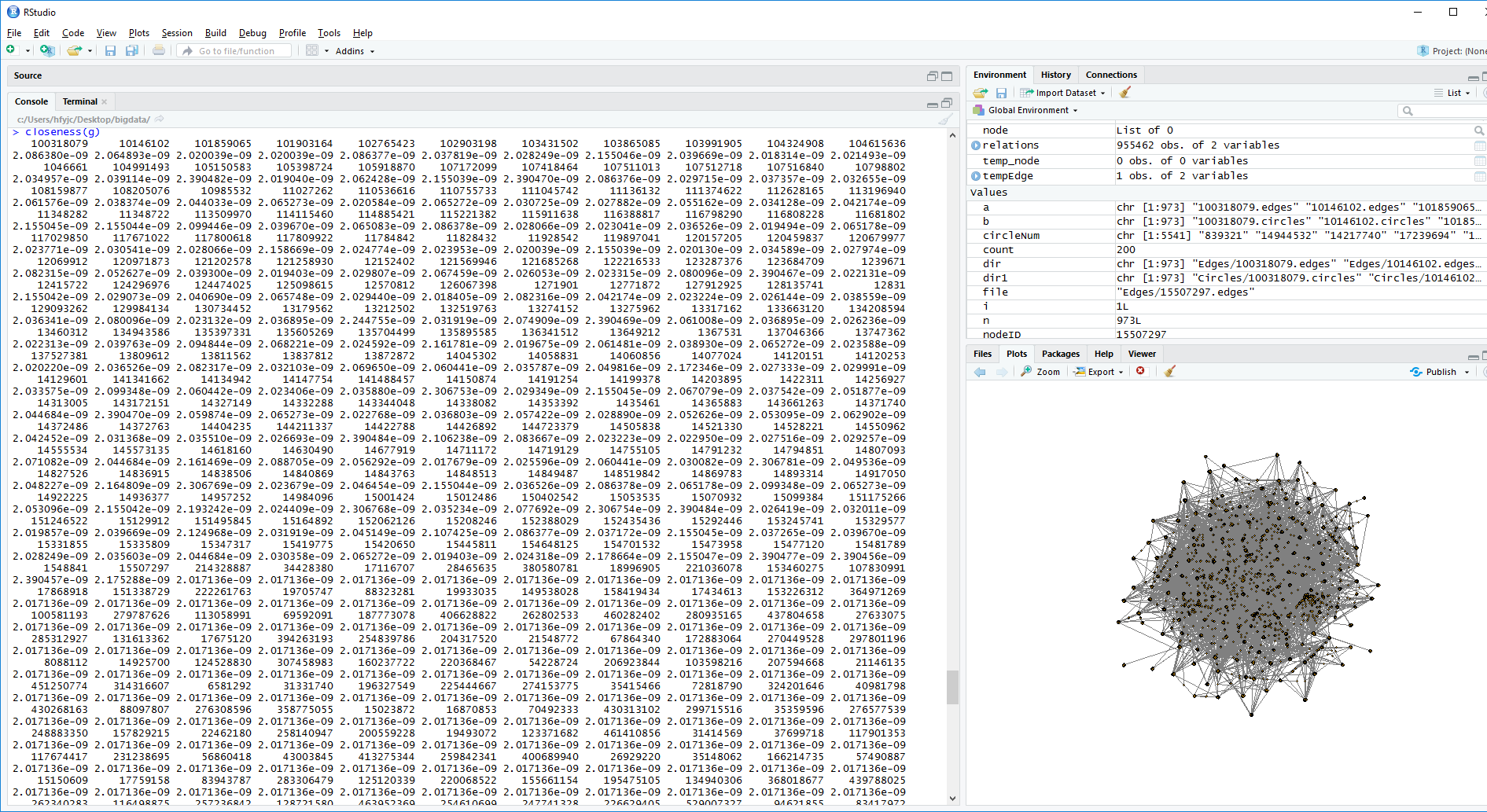
Function 4 ## is the graph weighted

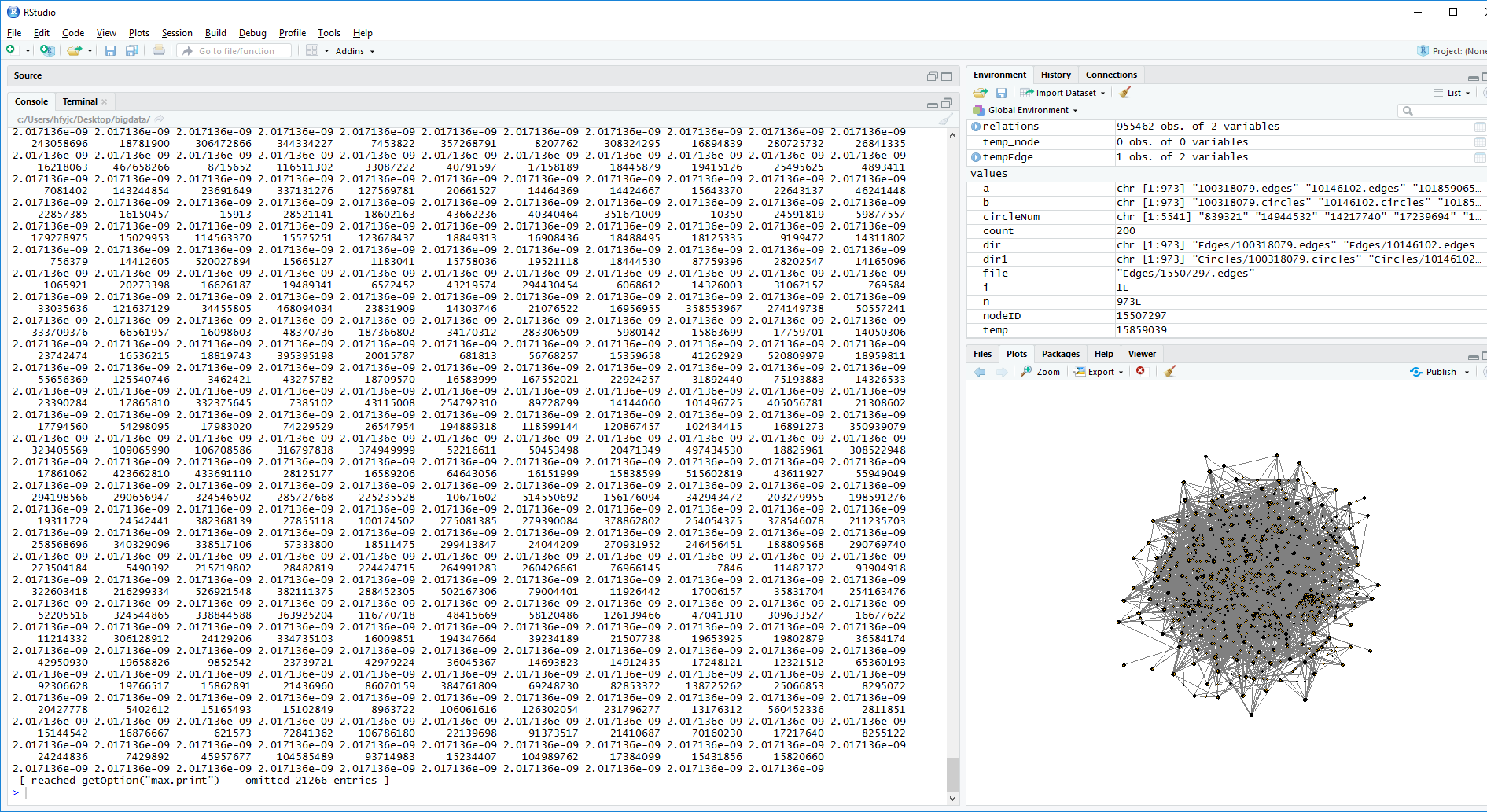


Function 5 # is the graph simple

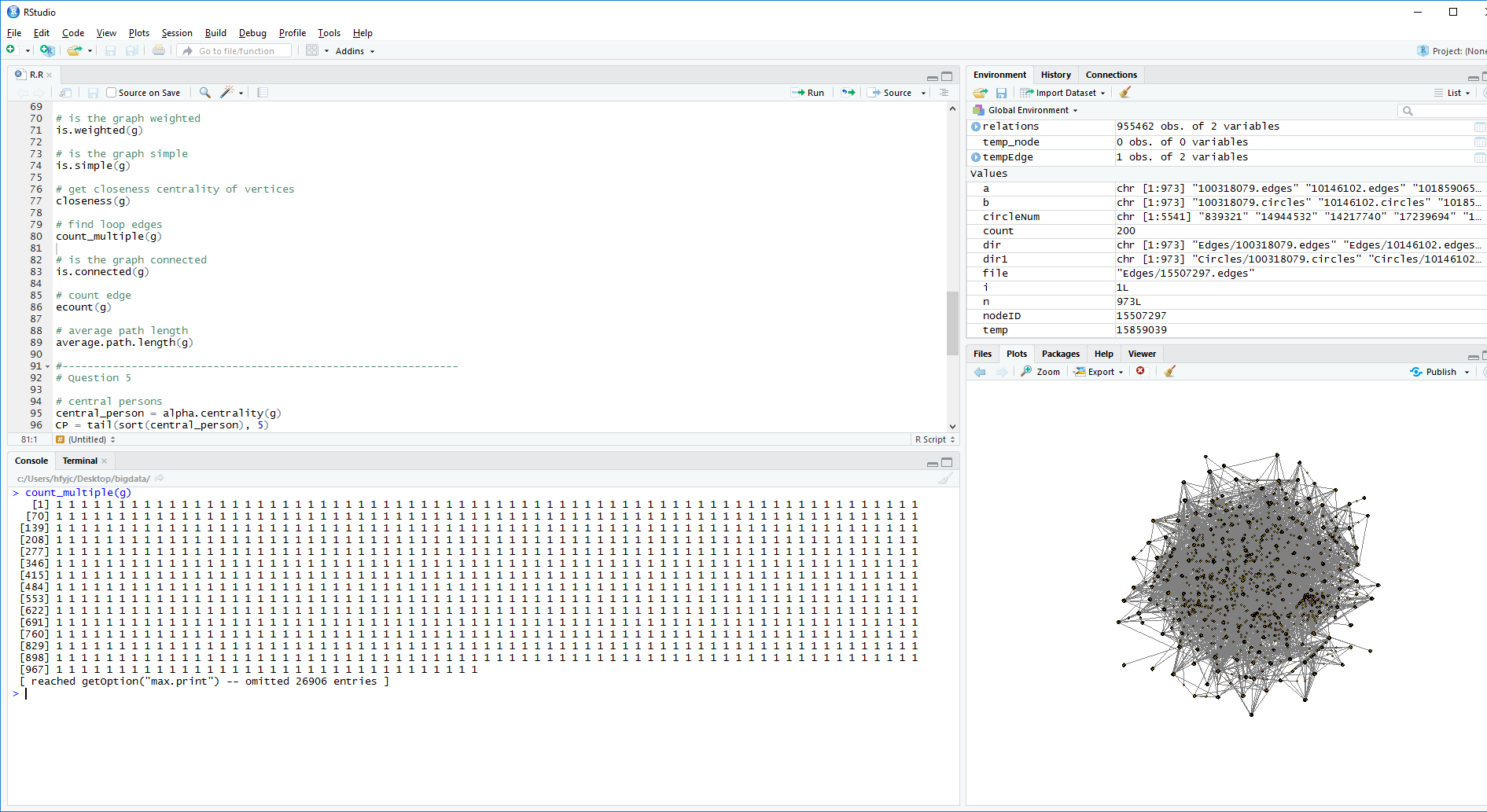


Function 6 # get closeness centrality of vertices

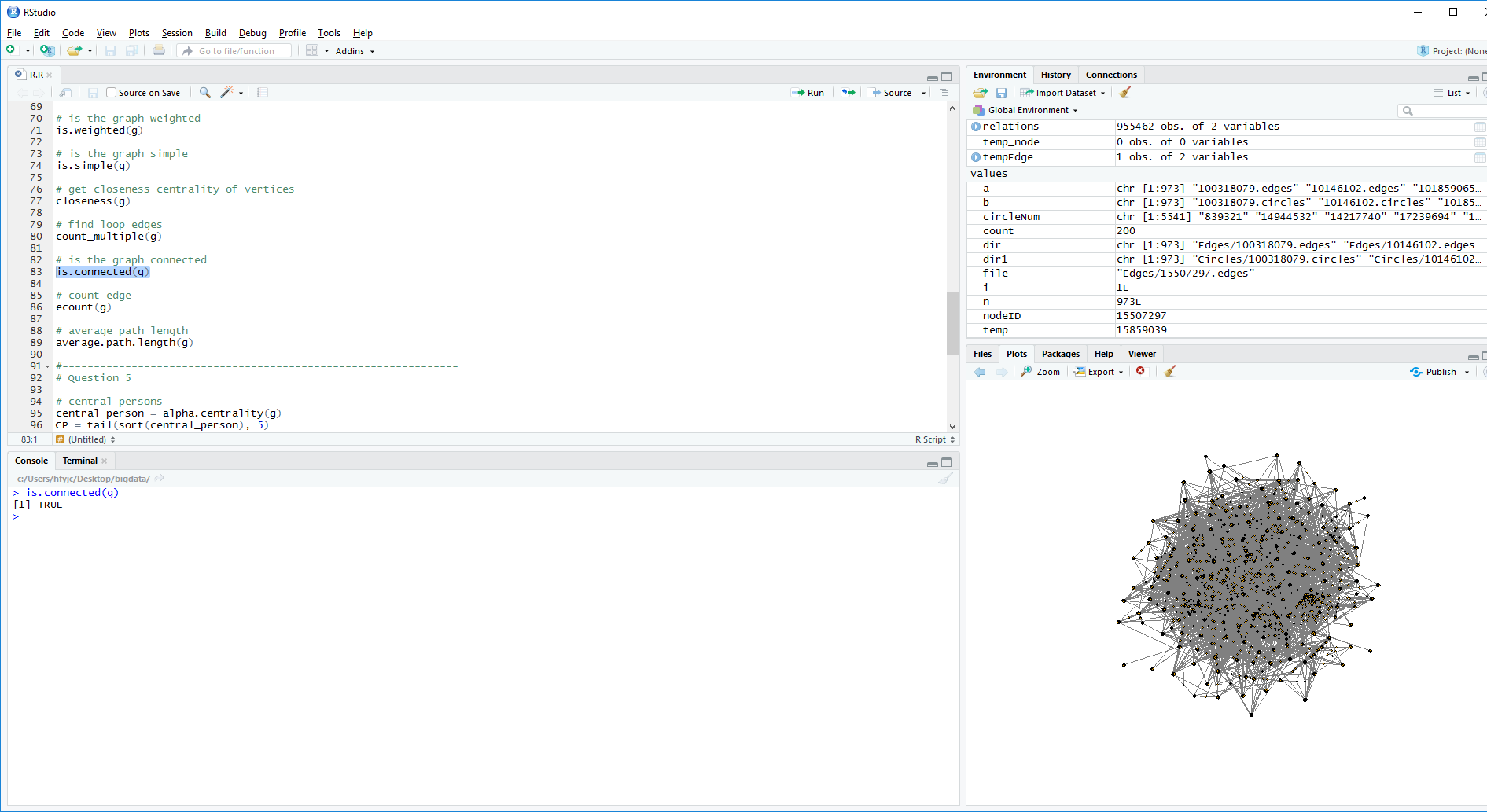




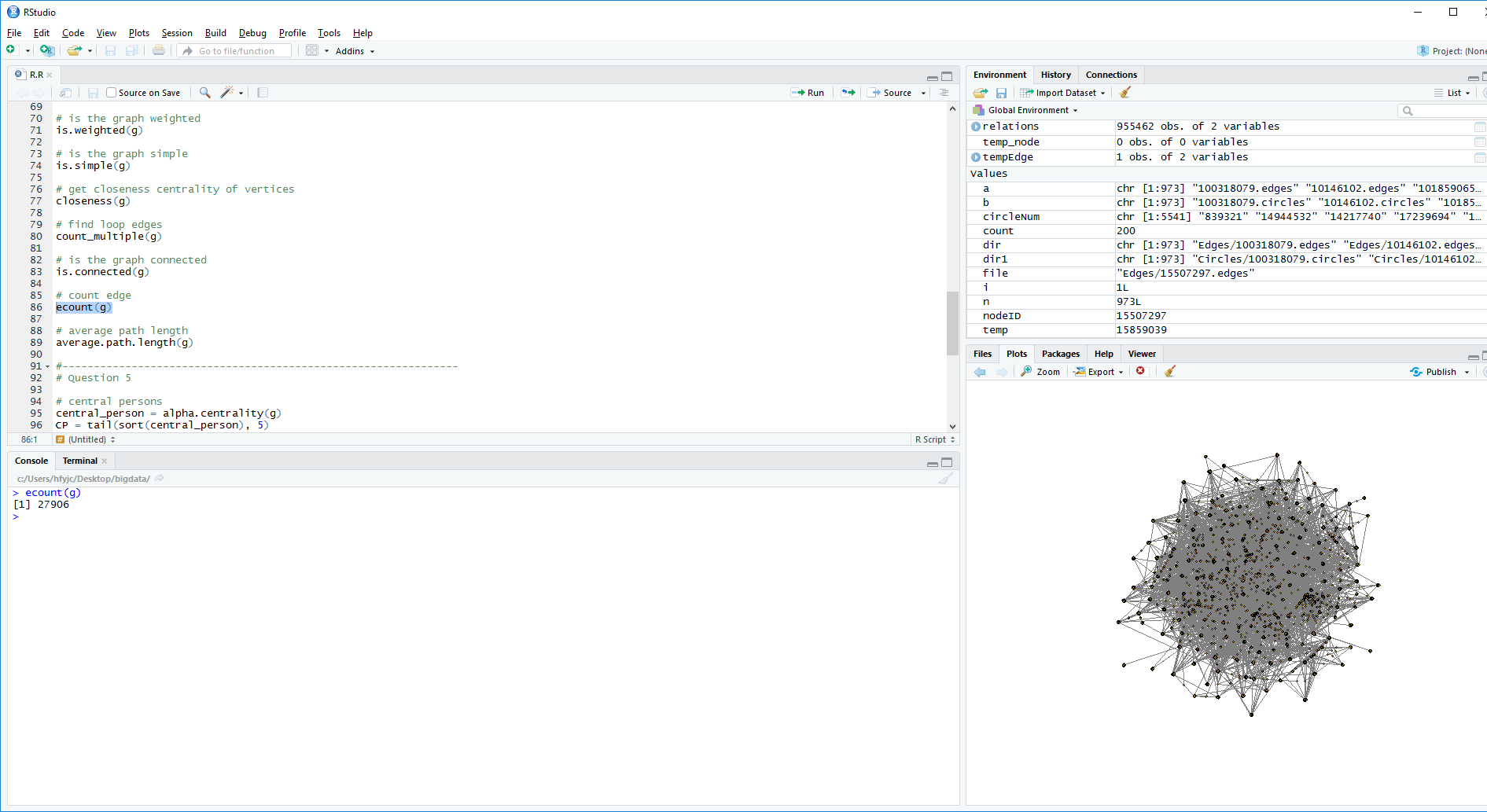
Function 7



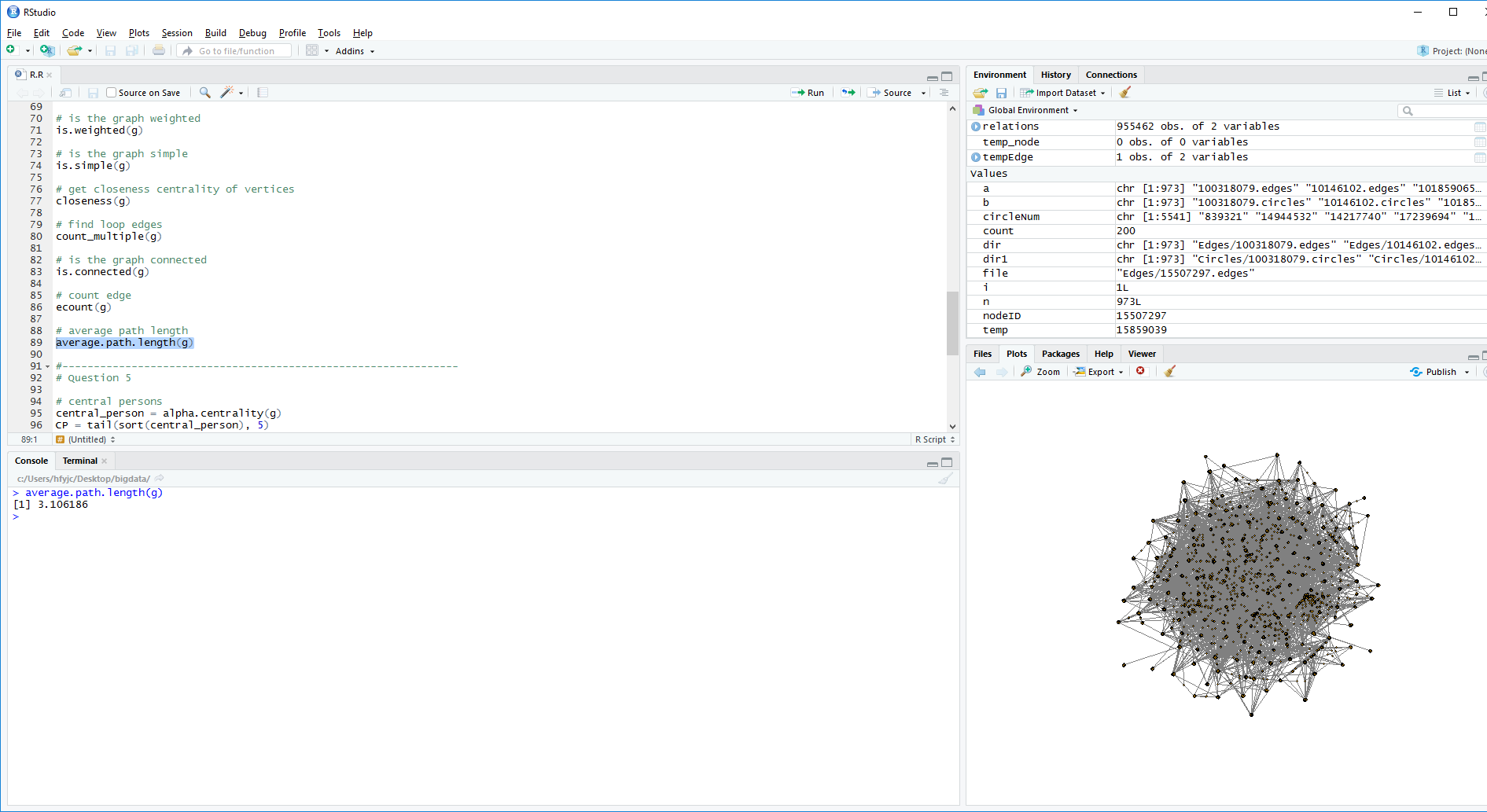
Function 8# is the graph connected



Function 9 # count edge

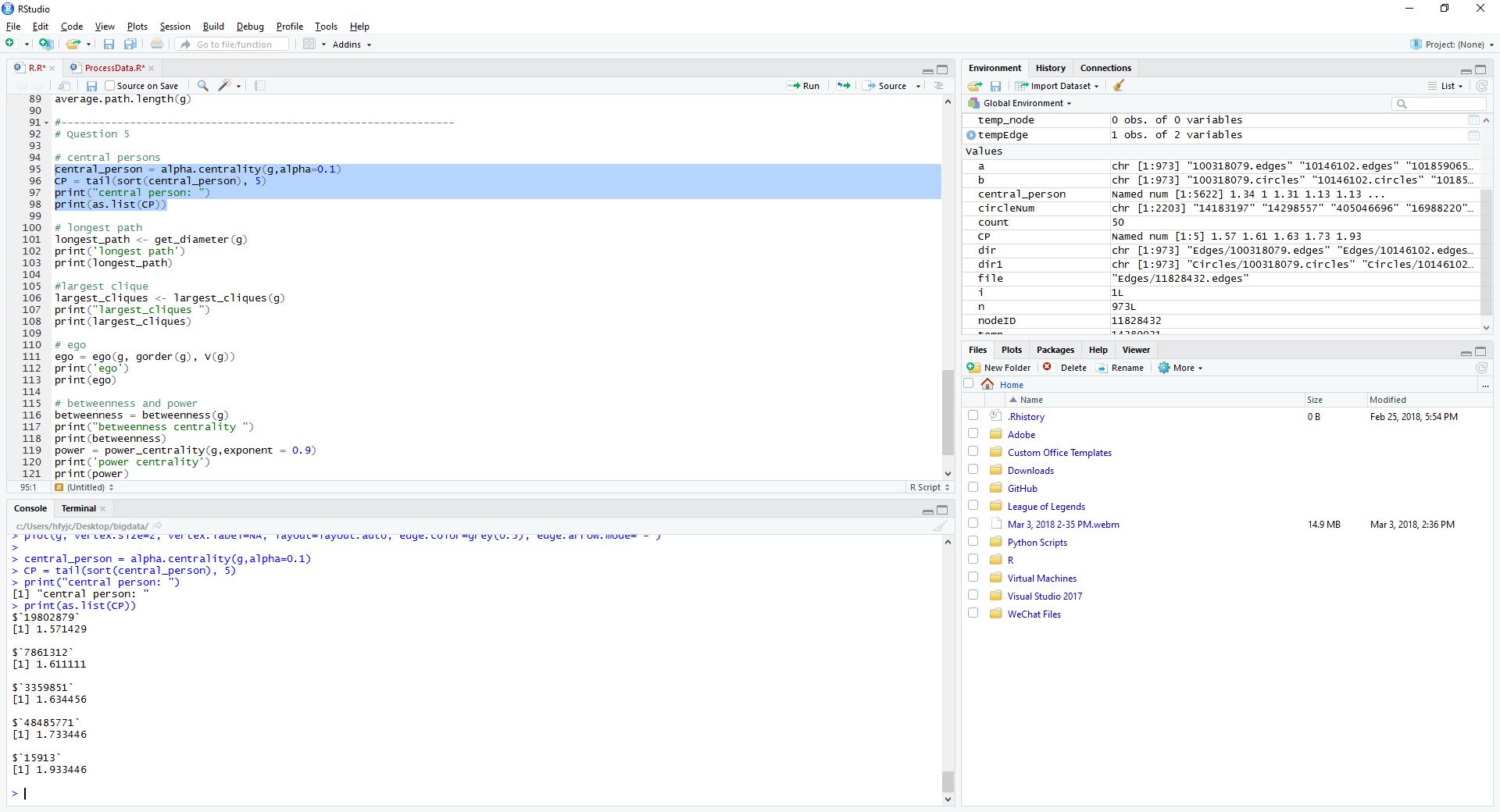


Function 10 # average path length

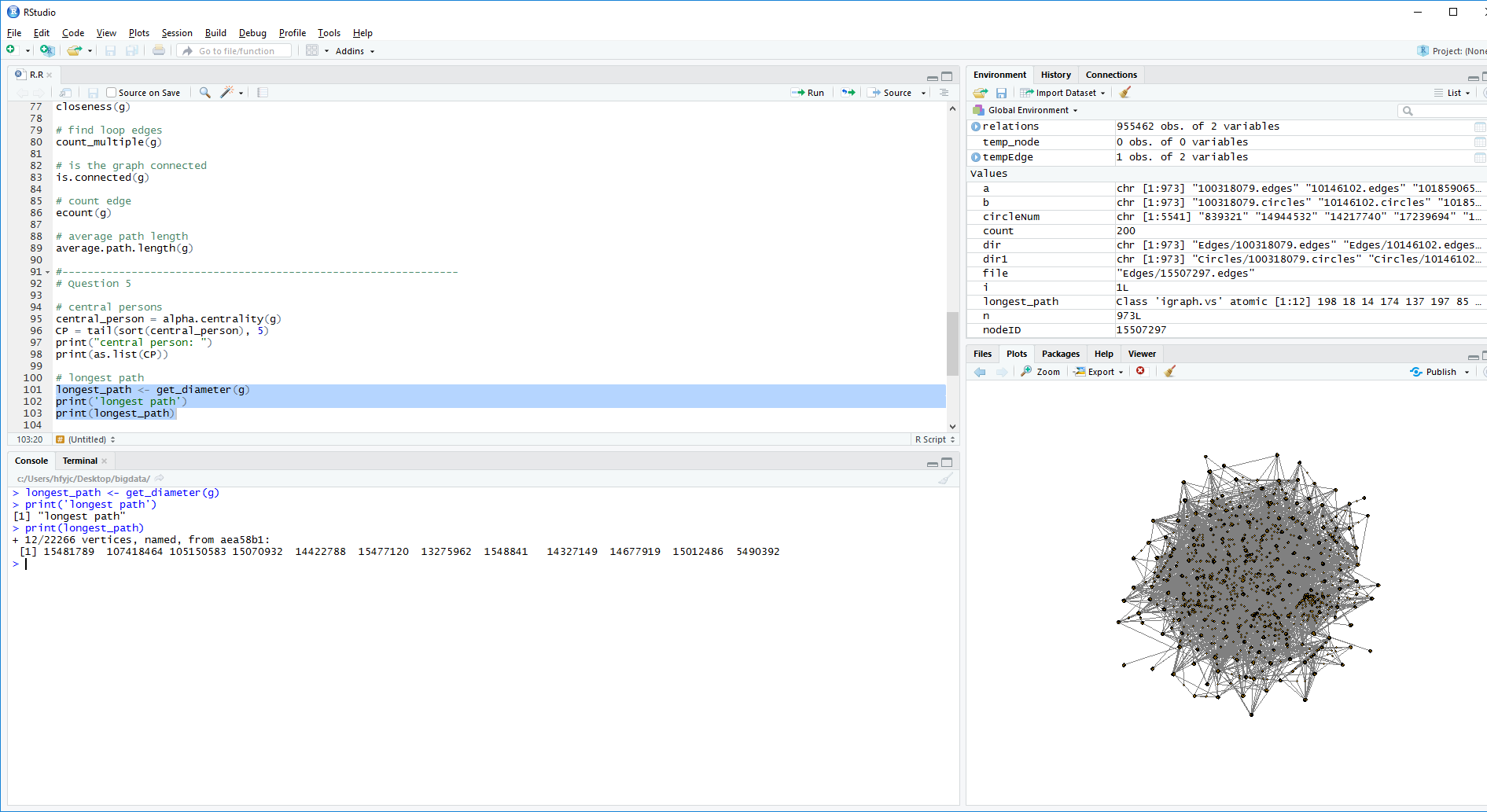


#question 5

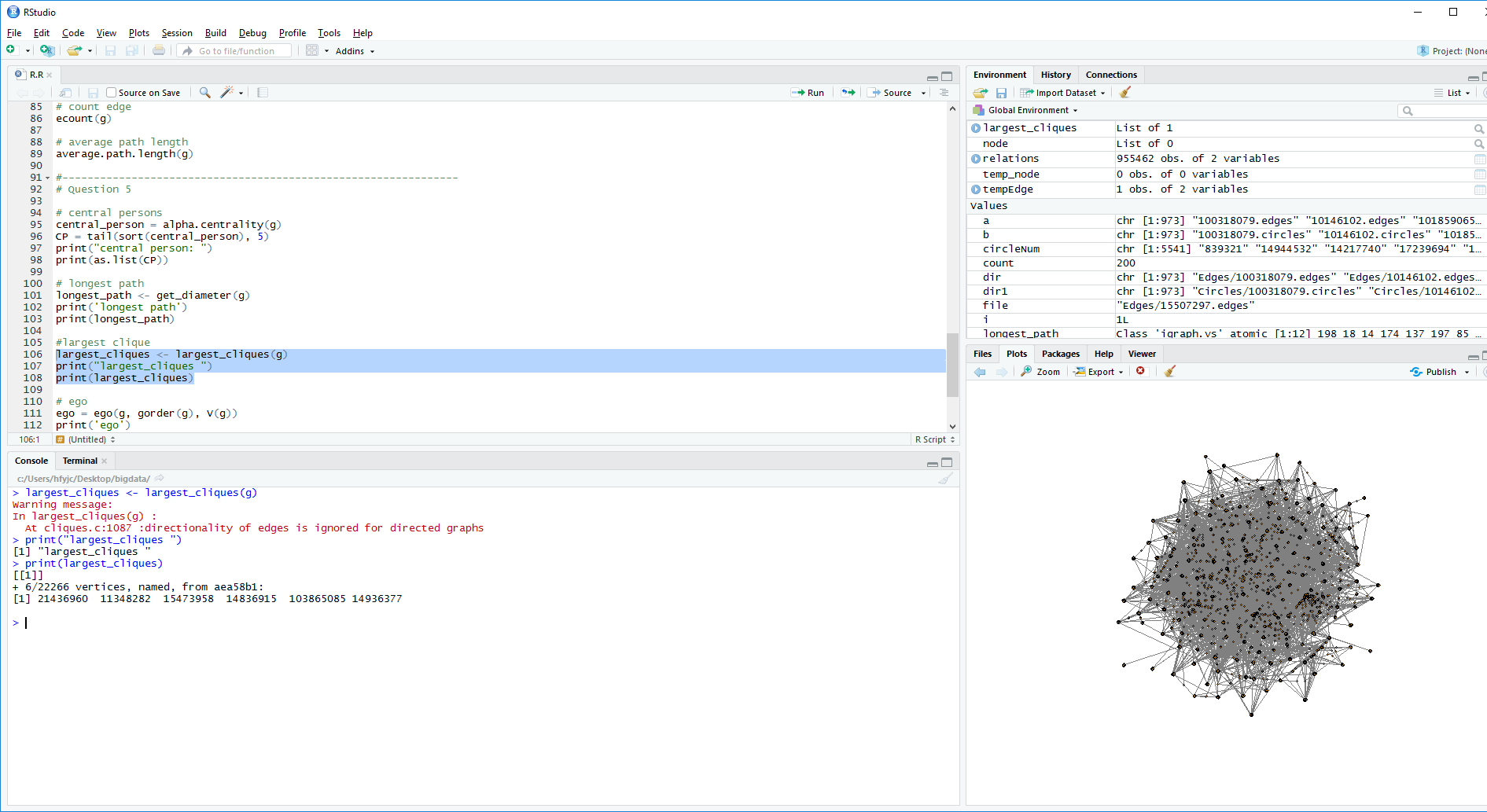
1. central persons

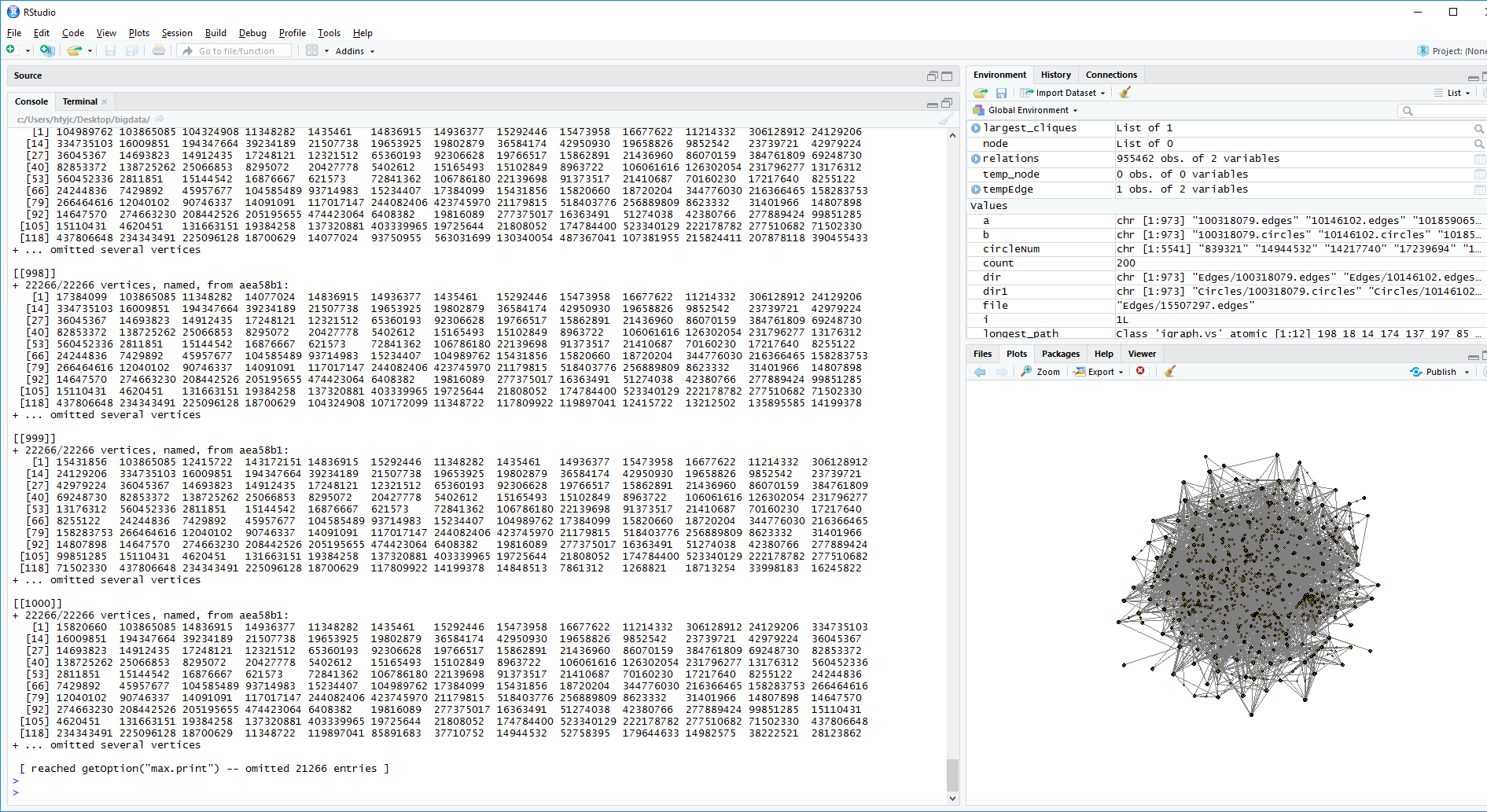


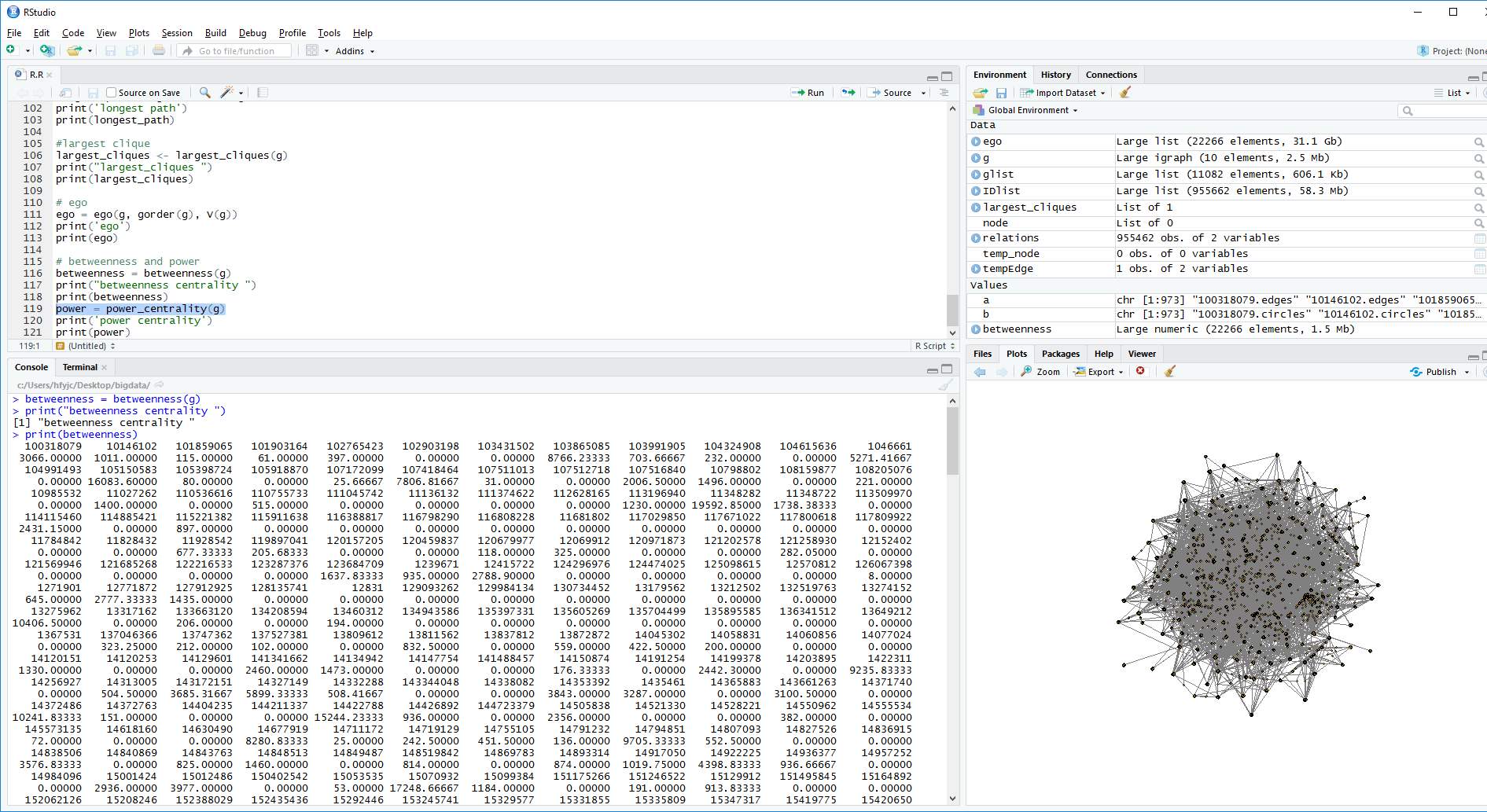
1. Longest\_path



1. Largest clique



1. Ego
2. 
3. betweenness centrality (the number of times a node acts as a bridge along the shortest path between two other nodes) and power centrality (returns the Boncich power centralities of positions).



Power

It is too large to compute

