```
#!/usr/bin/env python

import sys
from graph_tool.all import *
import argparse
import json
import numpy

def assign_centrality_zone(vertex_list, centrality_zones, seen, current_zone):
	# assign a zone to vertices in vertex_list, find their neighbours that
	# have not yet been visited, increment the zone and visit unvisited
	# neighbours
	#
	# this effectively is a breadth-first traversal of the graph, assigning
	# zones that increase as a function of distance from an initial list
	# of central vertices
	#
	neighbour_list = []
	for vertex in vertex_list:
		centrality_zones[vertex] = current_zone
		seen[vertex] = True
	for vertex in vertex_list:
		for neighbour in vertex.all_neighbours():
			if not seen[neighbour]:
				neighbour_list.append(neighbour)
	max_zone = current_zone
	if neighbour_list:
		max_zone = assign_centrality_zone(neighbour_list, centrality_zones, seen,
(current_zone+1))
	return max_zone

parser = argparse.ArgumentParser(description='Given a graph, compute the most central nodes for
the largest connected component')
parser.add_argument('in_graph_filename', type=str, help='File describing input graph')
parser.add_argument('out_graph_filename', type=str, help='File to write output graph (with
centrality properties) to')
parser.add_argument('out_zone_file', type=argparse.FileType('w'), help='File to write dictionary to
that allows lookup of protein by centrality zone (in JSON format)')
parser.add_argument('out_degree_file', type=argparse.FileType('w'), nargs='?', help='(Optional) file
to write dictionary to that allows lookup of degree (number of connections) for a protein name (in
JSON format)')

args = parser.parse_args()
g = Graph(directed=False)
g.load(args.in_graph_filename, 'xml')

size = g.num_vertices()

centrality_dict = dict()
centralities = closeness(g, norm=False)

degree_by_name = dict()
```

```
for vertex in g.vertices():
	degree_by_name[g.vertex_properties['name'][vertex]] = vertex.out_degree()
	centralities[vertex] = 1/float(centralities[vertex])
	centrality = centralities[vertex]
	#centrality = 1/float(centralities[vertex])
	vertices_of_centrality_x = centrality_dict.get(centrality, [])
	vertices_of_centrality_x.append(vertex)
	centrality_dict[centrality] = vertices_of_centrality_x

sorted_centralities = sorted(centrality_dict.keys())
smallest_centrality = sorted_centralities[0]
central_nodes = centrality_dict[smallest_centrality]

centrality_zones = g.new_vertex_property("int")
seen = g.new_vertex_property("bool")
seen.a = numpy.empty(size)
seen.a.fill(False)

current_zone = 0
print "assigning vertex centrality zone"
max_zone = assign_centrality_zone(central_nodes, centrality_zones, seen, current_zone)
print "max zone:", max_zone

g.vertex_properties['centrality_zone'] = centrality_zones

vertices_by_zone = dict()
for zone in range(max_zone+1):
	vertices = find_vertex(g, centrality_zones, zone)
	print zone, len(vertices)
	vertices_by_zone[zone] = [g.vertex_properties['name'][v] for v in vertices]

#print "connected component size", size
#print "shorted total path length", smallest_centrality
#for centrality in centrality_dict:
#	print centrality, ':', [g.vertex_properties['name'][v] for v in centrality_dict[centrality]]
# print sorted_centralities

print "saving graph"
g.save(args.out_graph_filename)
#print "drawing graph"
#graph_draw(g, vertex_fill_color=centrality_zones, output='graph.pdf', output_size=(10240,10240),
vorder=centrality_zones)

#json.dump((centrality_dict, sorted_centralities), args.out_group_file)
json.dump(vertices_by_zone, args.out_zone_file)
if args.out_degree_file:
	json.dump(degree_by_name, args.out_degree_file)
```

```
#!/usr/bin/env python

import sys
from graph_tool.all import *
import argparse
import numpy
from operator import itemgetter

def safe_open(filename,mode='r'):
	try:
		file_obj = open(filename, mode)
	except IOError as e:
		sys.stderr.write('Failed to open {}: {}\n'.format(filename, e.msg))
		sys.exit(1)
	return file_obj

def load_graph_from_protein_file(input):
	g = Graph(directed=False)
	names = g.new_vertex_property("string")
	protein_set = set()
	name_to_vertex = dict()
	for line in input:
		line = line.strip()
		if line.startswith('#'):
			continue
		pair = line.split()
		if len(pair) == 2:
			for protein_name in pair:
				if protein_name not in name_to_vertex:
					# add a new vertex to the graph and add it to the
					# names property map
					vertex = g.add_vertex()
					names[vertex] = protein_name
					name_to_vertex[protein_name] = vertex
			# make sure we don't add a self-referential link
			if pair[0] != pair[1]:
				g.add_edge(name_to_vertex[pair[0]], name_to_vertex[pair[1]])
	g.vertex_properties["name"] = names
	return (g, name_to_vertex)

parser = argparse.ArgumentParser(description='Given a graph, compute the most central nodes for
the largest connected component')
parser.add_argument('protein_file', type=argparse.FileType(), help='File describing pairs of
interacting proteins')
# graph_filename is a filename because if you specify .bz2 or .gz, graph_tool automatically
compresses your graph at
# save time.
parser.add_argument('graph_filename', type=str, help='File to save largest component of graph to
(in GraphML format)')

args = parser.parse_args()
(g, name_to_vertex) = load_graph_from_protein_file(args.protein_file)
```

```
print "graph loaded", g.num_vertices(), "vertices"
(component_map, component_histogram) = label_components(g)
largest_component_size = 0
largest_component_index = 0
for i in range(len(component_histogram)):
        if component_histogram[i] > largest_component_size:
                largest_component_size = component_histogram[i]
                largest_component_index = i

print "Graph has",len(component_histogram),"components"
print "Component ID\tsize"
for (id, size) in sorted(zip(range(len(component_histogram)), component_histogram),
key=itemgetter(1), reverse=True):
        print id, "\t\t", size

in_largest_component = g.new_vertex_property('bool')
in_largest_component.a = numpy.array([(lambda x: True if x == largest_component_index else
False)(x) for x in component_map.a])
print "largest component labelled"
#g.vertex_properties['in_largest_comp'] = largest_comp


component_g = GraphView(g, vfilt=in_largest_component)
print "Pseudo-diameter of largest component is:", int(pseudo_diameter(component_g)[0])
component_g.purge_vertices()
print "vertices filtered"
component_g.save(args.graph_filename, 'xml')
```

```
#!/usr/bin/python
import json

zone_filename = "/home/emad/Desktop/comprehensive human
network/comprehensive_human_network_2014.zones.json"
input_file = open(zone_filename)
zone_dict = json.load(input_file)
max_zone = 10
for zone_num in range(max_zone):
  zone_num_str = str(zone_num)
  members_str = str(zone_dict[zone_num_str])
  members_with_single_quote = members_str.replace("", """).replace('u','')
  print "members:", zone_num_str, ":", members_with_single_quote
```