Kamil Zleliński nr. indx: 261536

TASS

Projekt 1

Mały graf - Związki pomiędzy naukowcami w określonej dziedzinie

***Zadanie**:*

Zainstaluj *Pajeka* i *pythona*, wraz z pakietem *NetworkX*. Wczytaj pobrany graf. Jeśli potrzeba, przekonwertuj dane do obsługiwanego formatu lub zaimplementuj własną procedurę wczytywania. Usuń ewentualne zduplikowane krawędzie; możesz to uwzględnić w wadze krawędzi pozostawionej. Jeśli chcesz, przekształć graf na nieskierowany. Wyznacz składowe spójne**. Ile ich jest; jaki jest rząd i rozmiar największej z nich ? Następnie dokonaj szczegółowej analizy największej składowej spójnej wg poniższych instrukcji.

Dla grafów małych:

- wykreśl graf*
- znajdź 5 wierzchołków o największej wartości: bliskości, pośrednictwa i rangi
- znajdź wszystkie największe kliki (ile i jakiego rzędu?)
- przeprowadź grupowanie aglomeracyjne, identyfikator metody grupowania single linkage (najmniejsza odległość między grupami).

O ile nie wskazano inaczej, obliczenia należy wykonać wykorzystując *NetworkX*. Zadania do wykonania zarówno w *NetworkX*, jak i *Pajeku*, zostały oznaczone **. Należy wówczas dodatkowo porównać czasy obliczeń. Zadanie do wykonania wyłącznie w *Pajeku* oznaczono *.

Dane do projektu:
http://www-personal.umich.edu/~mejn/netdata/netscience.zip

Konwersja:

Pobrany plik netscience.gml z powyższego źródła posiada rozszerzenie gml. GML nie jest formatem języka obsługiwanym przez Pajek do wczytania sieci. Plik ten wymagał więc konwersji do formatu .net. Konwersje pliku dokonałem za pomocą języka R i biblioteki igraph. Biblioteka igraph umożliwia stworzenie grafu na podstawie pliku wejściowego (funkcja *read.graph)* oraz zapisanie przetworzonego grafu do formatu obsługiwanego przez Pajeka (funkcja *write.graph)*. Biblioteka igraph dodatkowo zapewnia tworznie grafu bez zduplikowanych krawędzi (igraph nie może stworzyć grafu na podstawie danych ze zduplikowanymi krawędziami). Jeżeli więc ze wskazanego pliku udało się stworzyć graf, oznacza to, że graf nie posiada zduplikowanych krawędzi.

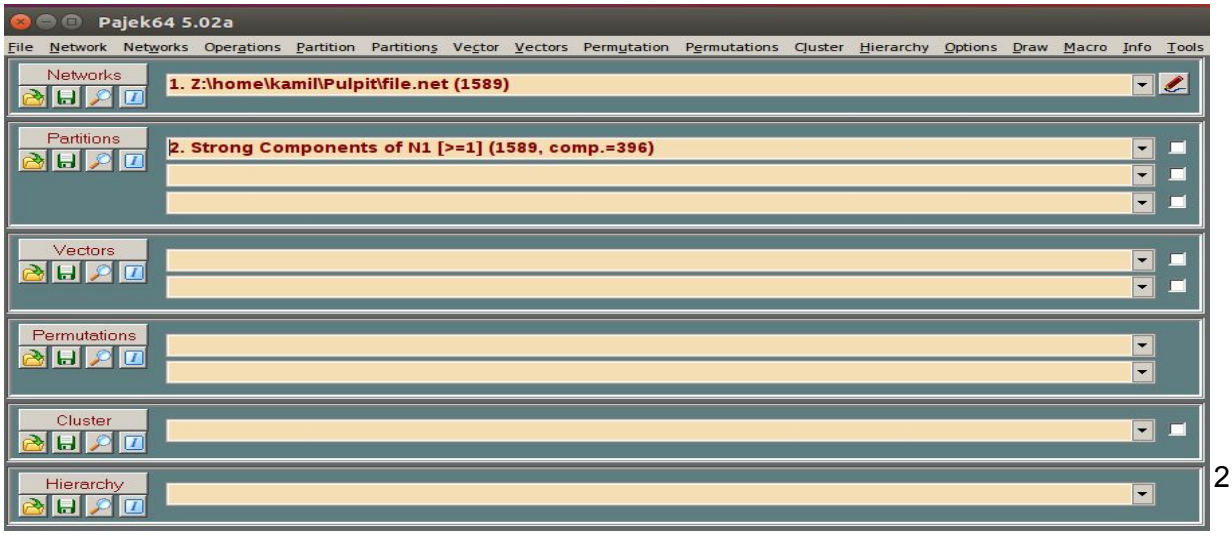
Kod:

```
library(igraph)
g<-read.graph("netscience.gml",format=c("gml"))
tmp <- tempfile()
write.graph(g, file = tmp, format = "pajek")
cat(readLines(tmp), sep = "\n")
```

Pajek

1.Wczytywanie danych:

Uzyskany plik .net został wczytany przez import w zakładce Networks



General info:

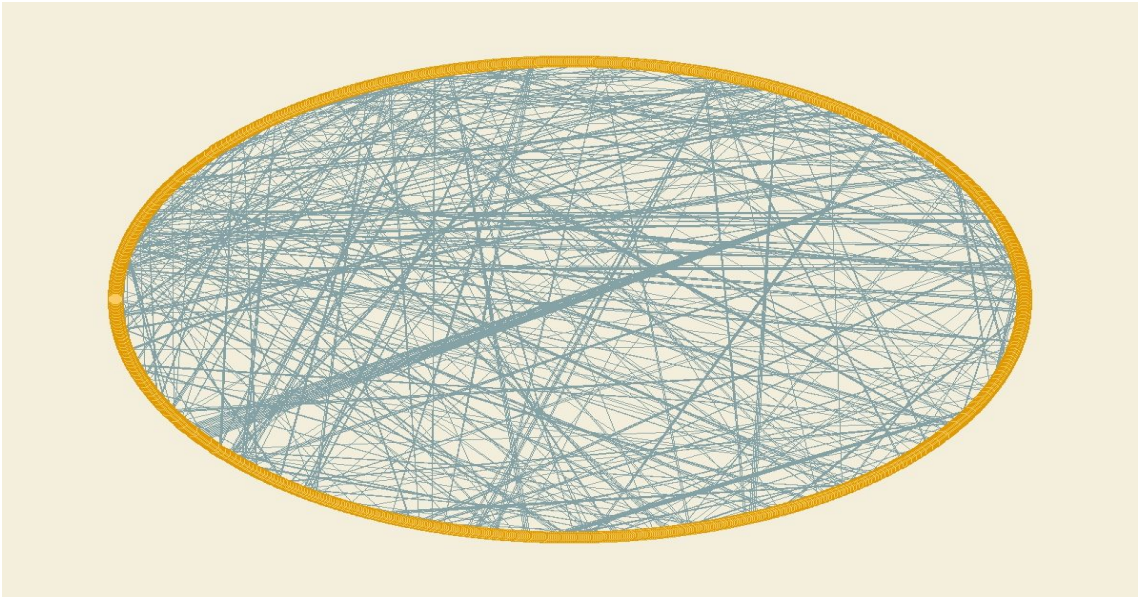
```
Reading Network   ---    Z:\home\kamil\Pulpit\file.net
-----------------------------------------------------
 Working...
          4333 lines read.
 Time spent:  0:00:00
-----------------------------------------------------
1. Z:\home\kamil\Pulpit\file.net (1589)
Number of vertices (n): 1589
-----------------------------------------------------
                                Arcs         Edges
-----------------------------------------------------
Total number of lines                    0          2742
-----------------------------------------------------
Number of loops                        0            0
Number of multiple lines                 0            0
-----------------------------------------------------

Density1 [loops allowed]    = 0.00217195
Density2 [no loops allowed] = 0.00217332
Average Degree = 3.45122719
```

Wczytany graf:

- Rząd : 1589 - jest to liczba wierzchołków
- Rozmiar: 2742 - jest to liczba krawędzi

Wyrysowany graf (CTRL+G):



2. Wyznaczanie składowych spójnych

Na podstawie wczytanego wykresu została utworzona partycja, grupa węzłów stworzona poprzez wyznaczenie składowych spójnych. Do utworzenia partycji zostały wybrane silnie spójne komponenty sieci. Aby ją stworzyć podjęto następujące kroki Network→Create Partition→Components→Strong z minimalnym rozmiarem partycji 1.

Utworzona partycja:

```
Reading Network   ---    Z:\home\kamil\Pulpit\file.net
-------------------------------------------------------
 Working...
          4333 lines read.
 Time spent:  0:00:00
-------------------------------------------------------
Strong Components
-------------------------------------------------------
 Working...
 Number of components: 396
 Size of the largest component: 379 vertices (23.851%).
 Time spent:  0:00:00
-------------------------------------------------------
1. Strong Components of N1 [>=1] (1589, comp.=396)
-------------------------------------------------------
Dimension: 1589
The lowest value:    1
The highest value: 396
```

Wczytany graf posiada 396 składowych spójnych, największa składowa spójna jest rzędu 379. Czas wyznaczania składowych spójnych był niezauważalny dlatego Pajek uznał go za “zerowy”.

NetworkX:

1.Wczytanie grafu :

Konwersja:

Próbując wczytać plik netscience.gml przez moduł networX natknąłeś się na problem:

Kod:

```
import networkx as nx
...
graphSmall = readGraph("netscience.gml")
```

Error:

```
networkx.exception.NetworkXError: cannot tokenize u'graph' at (2, 1)
```

Wzór pokazuje akceptowalny i nie akceptowalny format pliku:

Wrong format:

```
graph
[
  directed 0
  node
  [
    id 0
    label "Beak"
  ]
  .
  .
  .
```

Correct format:

```
graph [
  directed 0
  node [
    id 0
    label "Beak"
  ]
  .
  .
  .
```

Dlatego zdecydowałem się na wczytanie przetworzonego wcześniej pliku o rozszerzeniu .net.

Kod:

```
g = nx.read_pajek(graph_filename)
print nx.info(g)
print "Directed: ", graph.is_directed(), "\n"
```

Graf został wczytany poprawnie:

```
Name:
Type: MultiGraph
Number of nodes: 1589
Number of edges: 2742
Average degree:   3.4512
Directed:  False

Convert graph from net to gml
```

Dodatkowo udało się go skonwertować i zapisać do akceptowalnego pliku gml za pomocą polecenia : “nx.write_gml(graph, **'netsciencePajek.gml'**)”

Porównanie:

netsciencePajek.gml	netscience.gml

graph [
  multigraph 1
  node [
  id 0
  label "344"
  ]
  node [
  id 1
  label "345"
  ]
  node [
  id 2
  label "346"
  ]
  node [
  id 3
  label "347"
  ]
	graph
	[
	  directed 0
	  node
	  [
	  id 0
	  label "ABRAMSON, G"
	  ]
	  node
	  [
	  id 1
	  label "KUPERMAN, M"
	  ]
	  node
	  [
	  id 2
	  label "ACEBRON, J"
	  ]

Następnie konwertuje graf typu MultiGraph do grafu typu Graph (wczytuje go za pomocą danych z MultiGraphu. Oba grafy nie są skierowane

Jak widać na powyższym przykładzie wejściowy plik(z sieci) miał ustawioną wartość directed na false. Dlatego straciliśmy kierunek krawędzi podczas konwersji.

Zdecydowałem się więc zmienić wejściową zmienną directed na 1, aby mógł zostać zrealizowany podpunkt zadania: przekształć graf na nieskierowany

Wczytanie grafu skierowanego:

Plik:

```
Creator "Mark Newman on Sat Jul 22 06:24:59 2006"
graph [
  directed 1
  node [
        id 0
        label "ABRAMSON, G"
  ]
  node [
        id 1
        label "KUPERMAN, M"
  ]
  node [
        id 2
        label "ACEBRON, J"
  ]
  node [
        id 3
        label "BONILLA, L"
  ]
```

Kod:

```
def readGraphGml(graph_filename):
   g = nx.read_gml(graph_filename)
   # print information about graph
   print nx.info(g)
   print "Directed: ", g.is_directed()
   return g
```

Wynik:

```
Type: DiGraph
Number of nodes: 1589
```

```
Number of edges: 2742
Average in degree:   1.7256
Average out degree:   1.7256
Directed:  True
```

Udało się skonwertować graf z nieskierowanego na skierowany (za pomocą pliku). Ale w treści zadania jest, aby wykonać tę procedurę na odwrót za pomocą biblioteki networkX.

Aby przekształcić graf skierowany na nieskierowany wystarczy wywołać metodę nx.Graph z utworzonym wcześniej grafu typu DiGraph.

Kod:

```
graph = nx.Graph(graphDi)
print nx.info(graph)
print "Directed: ", graph.is_directed(), "\n"
```

Wynik:

```
Name:
Type: Graph
Number of nodes: 1589
Number of edges: 2742
Average degree:   3.4512
Directed:  False
```

Liczba krawędzi się nie zmieniła, co oznacza, że w grafie nie istniały zduplikowane krawędzie.

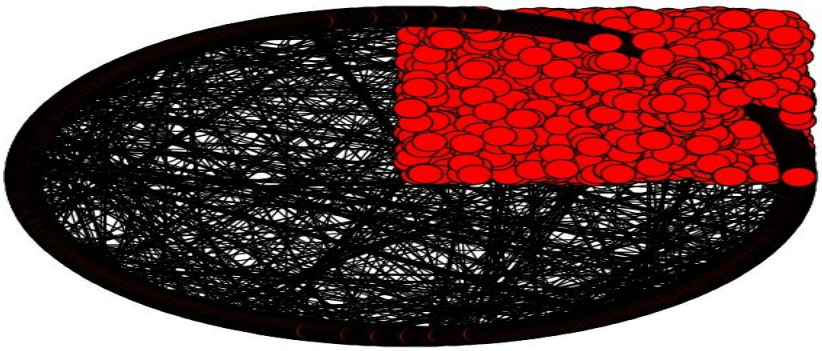
2. Wykreślenie grafu:

Za pomocą funkcji drawGraph został wykreślony graf:

```
def drawGraph(graph,filename):
   nx.draw(graph)
   nx.draw_random(graph)
   nx.draw_circular(graph)
   nx.draw_spectral(graph)
   plt.savefig(filename)
   print "Draw graph to file ",filename
```

```
Wywołanie:
drawGraph(graph,"grafJPG.jpg")
```

Wynik:



Uzyskany graf jest mało czytelny ponieważ posiada 1589 wierzchołków i 2742 połączeń między nimi. Na tak małej powierzchni jak okienko,ekran komputera nie jest w stanie się go czytelnie wyświetlić Na całym obwodzie znajdują się wierzchołki grafu (kolor czerwony), jednakże krawędzie (kolor czarny) je przykrywają. Graf został wyrysowany poprawnie.

3. Wyznaczanie składowych spójnych

Biblioteka networkX zapewnia funkcję *connected_components(Graph)*, która umożliwia wyznaczenie składowych spójnych.

Kod:

```
print "Number of components: ",
nx.number_connected_components(graph)
start = time.time()
conectedComponent = max(nx.connected_component_subgraphs(graph),
key=len)
end = time.time()
print "Duration: ",end - start,"\n"
print "Largest connected component range: ",
conectedComponent.number_of_nodes()
print "Largest connected components size: ",
```

```
conectedComponent.number_of_edges()
```

Wynik

```
Number of components:  396
Duration:  0.0755290985107

Largest connected component range:  379
Largest connected components size:  914
```

Uzyskany wynik zgadza się z wynikami uzyskanymi w program Pajek. Czas wyznaczenia składowych spójnych jest gorszy niż w programie Pajek. Wynik czasowy jest przedstawiony w sekundach. Ilość składowych spójnych wyszła taka sama (396). Rząd największej składowej spójnej wyszedł 379 (ilość wierzchołków), natomiast jej rozmiar 914 (ilość krawędzi).

4. Pięć wierzchołków o największych wartościach

Kolejnym punktem zadania projektowego było znalezienie 5 wierzchołków o największej wartości:

- bliskości
- pośrednictwa
- rangi

Kod:

```
#Funkcja przyjmuje jako argument liste i zwraca 5 najwiekszych elementow
def getTop5VertexValue(inList):
  sortedList = sorted(inList.iteritems(),
                      key=lambda n: -n[1] ) #klucz sortowania "-" bo desc
  return sortedList[:5]

#Najwieksze wartosci
top5Betweenness =
getTop5VertexValue(nx.betweenness_centrality(conectedComponent))
top5Closeness =
getTop5VertexValue(nx.closeness_centrality(conectedComponent))
top5Rang = getTop5VertexValue(nx.pagerank(conectedComponent))
```

```
print "Top 5 - Betweenness","\n",top5Betweenness
print "Top 5 - Closeness","\n",top5Closeness
print "Top 5 - Rang","\n",top5Rang
```

Wynik:

```
Top 5 - Betweenness
[(u'78', 0.39718418135681205), (u'150', 0.34514711880253035), (u'516',
0.28602004110719526), (u'281', 0.27016272041057426), (u'216',
0.2554278416347382)]
Top 5 - Closeness
[(u'78', 0.25661914460285135), (u'281', 0.2490118577075099), (u'150',
0.24705882352941178), (u'756', 0.24308681672025723), (u'301',
0.23290203327171904)]
Top 5 - Rang
[(u'78', 0.01616158887499995), (u'33', 0.01451940202234391), (u'34',
0.010772593302124673), (u'281', 0.009201837852845281), (u'216',
0.008891863527538632)]
--------------------------------------------------------------------------------
Top 5 - Betweenness
[(u'NEWMAN, M', 0.39718418135681166), (u'PASTORSATORRAS, R',
0.34514711880253085), (u'MORENO, Y', 0.2860200411071951), (u'SOLE,
R', 0.2701627204105742), (u'BOCCALETTI, S', 0.25542784163473825)]
Top 5 - Closeness
[(u'NEWMAN, M', 0.25661914460285135), (u'SOLE, R',
0.2490118577075099), (u'PASTORSATORRAS, R',
0.24705882352941178), (u'HOLME, P', 0.24308681672025723),
(u'CALDARELLI, G', 0.23290203327171904)]
Top 5 - Rang
[(u'NEWMAN, M', 0.016161588874999952), (u'BARABASI, A',
0.014519402022343912), (u'JEONG, H', 0.010772593302124671),
(u'SOLE, R', 0.009201837852845281), (u'BOCCALETTI, S',
0.008891863527538634)]
```

Bibliotek networkX zapewnia wyliczenia dla wszystkich wierzchołków w grafie ich wartości: bliskości, pośrednictwa i rangi. Funkcja zwraca je w formie listy, dlatego kluczowym elementem w tym podpunkcie była funkcja sortowania. Funkcja *getTop5VertexValue* przyjmuje jako argument listę i sortuje ją malejąco (zapewnia to znak -).

5. Kliki

Za pomocą poniższego kodu wyznaczyłem wszystkie kliki:

Kod:

```
#Funkcja zwraca mape klikow z iloscia wystapien danego rzedu
def getCliquesSizeList(cliques):
  cliqueAmmountSizeValues = {}
  for clique in cliques:
     sizeOfClique = len(clique)
     if sizeOfClique in cliqueAmmountSizeValues:
        preValue = cliqueAmmountSizeValues[sizeOfClique]
        cliqueAmmountSizeValues[sizeOfClique] = preValue +1
     else:
        preValue = 0
        cliqueAmmountSizeValues[sizeOfClique] = preValue + 1
  return cliqueAmmountSizeValues
```

```
Wywołanie:
#Klika
cliques = list(nx.find_cliques(conectedComponent))
print "Number of cliques in graph: ",cliques.__len__()
cliquesLenList = getCliquesSizeList(cliques)
for size, number in cliquesLenList.items():
  print'Number of clique with size', size , ': ', number
```

Wynik:

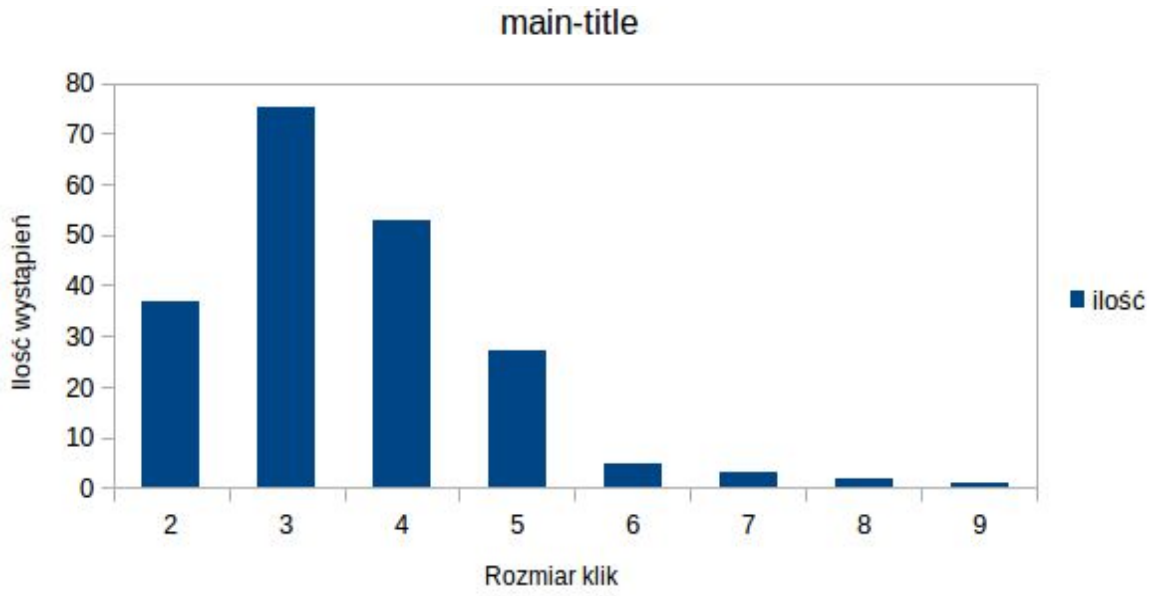
```
Number of cliques in graph:  203
Number of clique with size 2 :  37
Number of clique with size 3 :  75
Number of clique with size 4 :  53
Number of clique with size 5 :  27
Number of clique with size 6 :  5
Number of clique with size 7 :  3
Number of clique with size 8 :  2
Number of clique with size 9 :  1
```

Za pomocą gotowej funkcji w bibliotece networkX, udało mi się uzyskać listę wszystkich klik w grafie. Jest ich 203. Następnie za pomocą pętli i mapy udało mi się uzyskać sumacyjną listę wystąpień klik o danych rozmiarach.

Wystąpiło:

- 37 kilk o rozmiarze 2
- 75 klik o rozmiarze 3
- 53 kilk o rozmiarze 4
- 27 klik o rozmiarze 5
- 5 klik o rozmiarze 6
- 3 kliki o rozmiarze 7
- 2 kliki o rozmiarze 8
- 1 klik o rozmiarze 9

Wykres z excela:



Ich suma wynosi 203 więc lista ma poprawną ilość elementów. Największy wystąpiony klik byl rozmiaru 9 i wystąpił tylko raz.

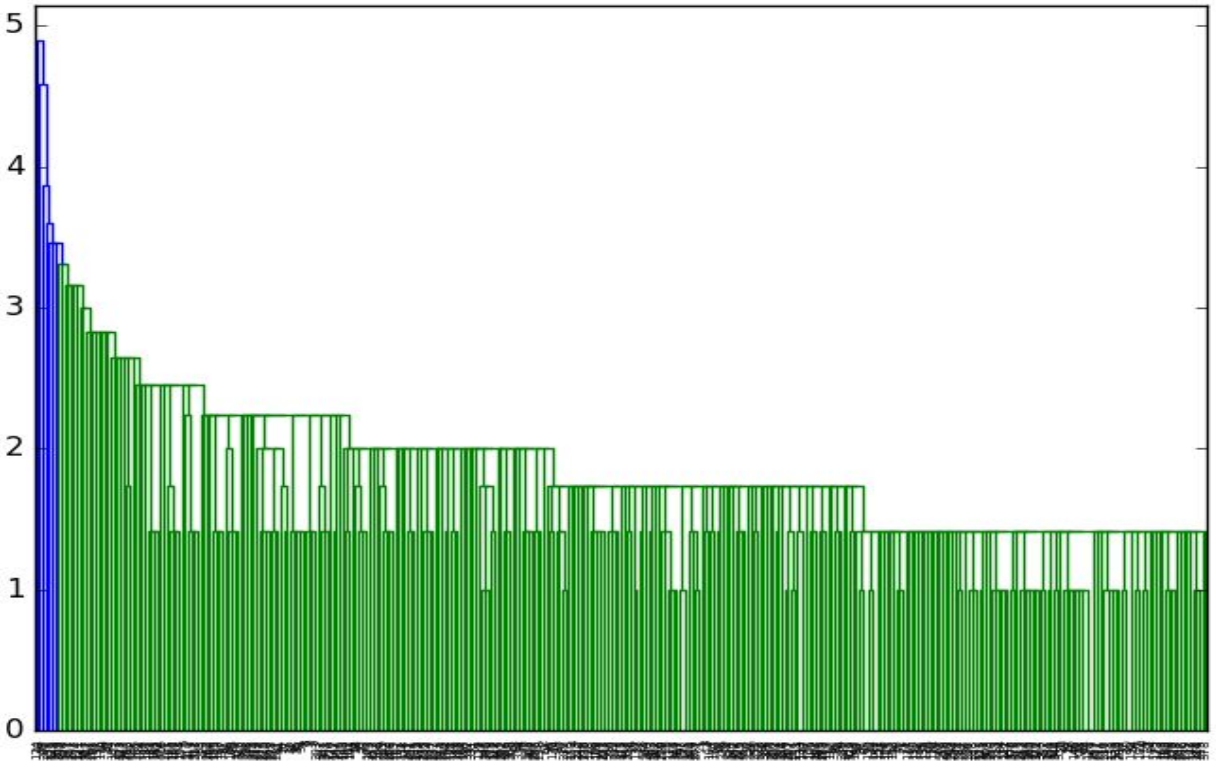
6.Single linkage

Ostatnim punktem projektu było przeprowadzenie grupowania aglomeracyjnego z identyfikatorem metody single linkage. Metoda ta nosi nazwę pojedyńczego wiązania. W wyniku jej zastosowania powstają skupienia typu “włóknistego”, co oznacza, że są połączone ze sobą tylko przez pojedyńcz obiekty, które leżą najbliżej siebie.

Kod:

```
def saveDendogramSingleLinkage(graph):
   matrix = nx.to_scipy_sparse_matrix(graph)
   matrixDense = matrix.todense()
   clusters = hierarchy.linkage(matrixDense, method='single')
   hierarchy.dendrogram(clusters)
   plt.title("dendrogram")
   plt.savefig('dendrogram.png')
   print 'Save dendrogram.png'
```

Uzyskany dendrogram:



Graf podzieliłbym na linii odległości między wierzchołkami o wartości 2.