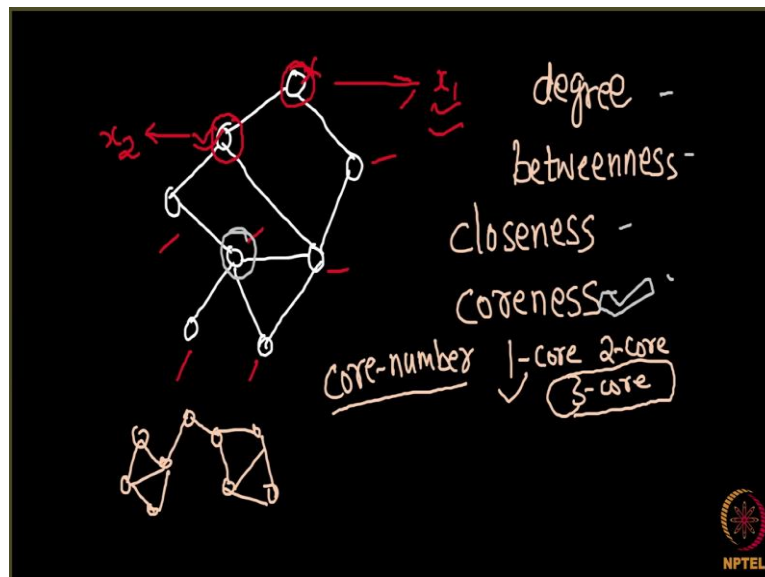


Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

How to go Viral on Web
Lecture - 165
Coding the importance of core nodes in cascading

So, next we are going to do an interesting experiment. What we are going to do is we are interested in looking at which are the most influential nodes in a network. And we have our answer also right.

(Refer Slide Time: 00:19)



So, given a network over here we saw that to find that let us say in this network over here this network over here I want to see which nodes over here are the most influential. So, what can I do is we know that how do we define influence for a seed set; given a seed we will start our independent cascade model from there and we will see that at the end how many nodes got infected.

So, to say that which nodes over here are most influential what we can do is first of all we say that this is my seed. This is my seed and then I perform the independent cascade model here and I see that at the end how many people got infected and let us say I found a value x_1 . Then I want to see the influential power of this node. So, I said the seed to be this now this is no longer the seed.

So, this over here is the seed only 1 node and then I will repeat the independent cascade model and I will see towards the end how many people got infected. I will do for this, I will do for this, do for this, do for this and do for this and at the end we will see that which of the x is maximum. And, whichever x is maximum that particular node is the node, which is the most influential in this network, but doing this kind of an experiment it is a very time consuming process right.

So, taking every node to be seed node and again and again repeating the independent cascade model and then finding out influential node is a time-consuming process. Here we have another very interesting question; can you look at this network and then guess which is the node most influential over here? I have a feeling for example, this node over here it has 1, 2, 3, 4 neighbors and no other node or maybe this node over here.

So, they have 4 neighbors. So, probably might be they are most influential, since they have more number of neighbors they can infect more people right. So, it might be the case that these nodes are most influential. So, that is an easy way of doing this same process right. So, taking every node over here and then running independent cascade model as opposed to just looking at this network and looking at its structure, looking at the properties of certain nodes and then saying that: yes, this node is influential is quite easy.

So, now the question is which nodes are the most influential over here. So, we have various centrality measures ok. What is centrality measure? It is just a complicated term. So, centrality measure is we can measure the values of nodes over here based on certain things. For example, we can look at the degree of each node right.

So, is it the degree can I say that the nodes which are having the maximum degree, they are the most influential nodes or we have another measure let us say betweenness, I hope you remember what was betweenness. So, betweenness tells me if I look at a node, if I look at a node how important is it in connecting to different parts of the network?

So, let me use another example. Let us say there is a community structure, this is one community, and this is 1 community right and there is a node over here. So, this node over here is very instrumental in connecting these 2 portions of the graph. So, this node has a very high betweenness over here right. So, are these the nodes which are having a

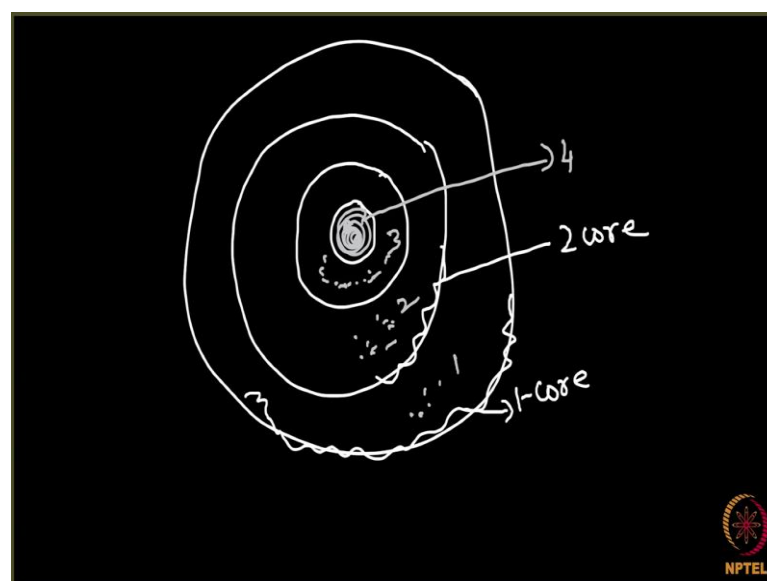
very high betweenness, which are the most influential and we will create the biggest cascade or are these the nodes having high closeness.

So, closeness is something like I take this node and I see that how close is it to other nodes in the network. For example, this node is very close to these 5 nodes and then still closer to this node and then far away from these nodes. So, you can look it up there is a very nice way of calculating this closeness centrality, but the intuition is simple. We see that how closer I am to the entire network that is closeness.

So, are these the nodes which are having a high value of closeness, which are the most influential oh we have a last measure which we discussed coreness. So, are these the nodes which are having maximum coreness, which are most influential? What do I mean by maximum coreness? Maximum coreness is we know that all the nodes in this network are in 1 core right. We define a value which is called core number, what is core number? It is nothing but the.

So, see a node can be part of 1 core, a node can be a part of 2 core, 3 core right. Whatever, is in 3 core, whatever node is in 3 core that is automatically in 1 core So, let us say there is a node which belong to three core I say that its core number is 3, right call number is 3 So, the inner you are in the network, the innermost core you are towards oh see let me put it like this ok.

(Refer Slide Time: 05:37)

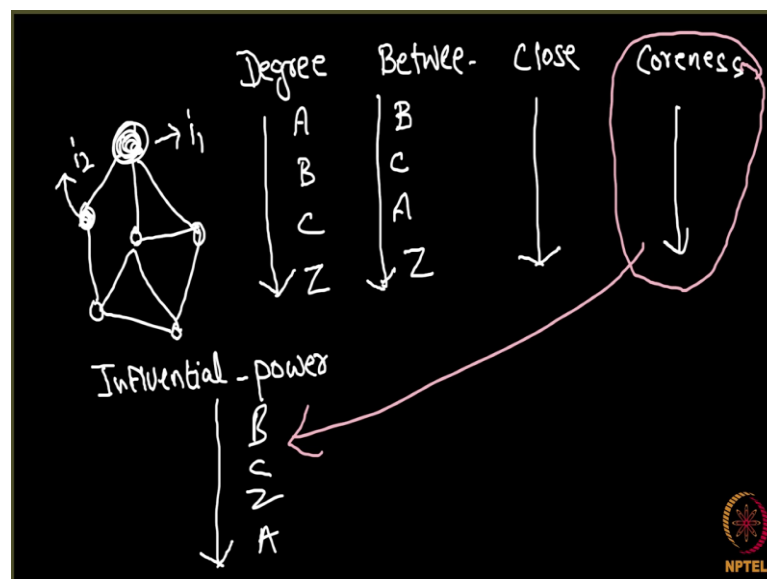


Let me say that this is the network, and then in this network, this is the so, this in this is 1 core right. So, whatever is inside this particular circle is in 1 core, whatever is inside this particular circle is in 2 core so on, and so we have 4 cores in this network. Now we know that the node which is present over here right, a node which is present over here it is a part of 1 core as well as 2 core as well as 3 core as well as 4 core right, but it is the innermost.

So, I say that the core number of this node over here is 4, the core number of all the nodes which are present over here is 3, the core number of all the nodes which are present here is 2 and which are present here is one. So, are these the nodes which lie over here which are the most influential? Has the influential power something to do with the coreness of a node which is the core number of a node?

So, this thing is what we want to look at. So, how we are going to do it is we are going to take a network. So, I take a network over here.

(Refer Slide Time: 06:47)



We will be doing it next in our next programming screen cast. So, I will take a network over here, what we are going to do? We are going to take degree of these nodes and we are going to arrange these nodes in descending order of their degree.

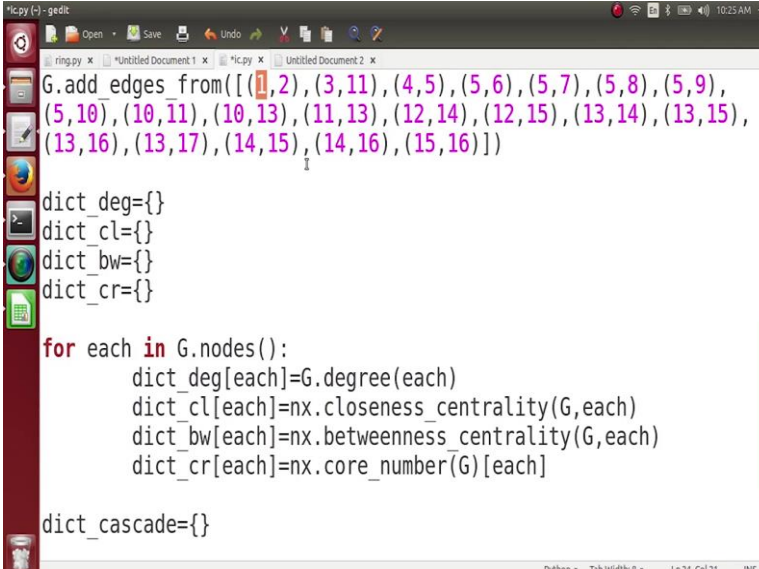
So, node which has the maximum degree comes over here and the nodes which has a minimum degree comes over here; let us say something like this. And similarly, we are

going to look at betweenness and we are going to arrange these nodes, nodes having the highest betweenness over here, least betweenness over here, similarly we are going to do for closeness, and we are going to do for coolness. And that aim is to see which out of these four measures is most related to the influential power of a node.

So, we will take a new list corresponding to the influential power. An influential power will actually take nodes and determine their influential power. So, I will take this node over here and look run independent cascade model from it and then see what is its influential power say; i_1 going to do this for this same node i_2 and then here i will arrange the nodes in the descending order of their influential power. So, let us say be caesar day right.

So, arrange the node here according to their influential power and then we will compare these lists. So, we will see at whether these are the nodes having highest degree which have the highest influential power or these are the nodes having highest betweenness which have highest influential power and to a surprise not surprised cannot call it supplies, but we see that actually these are the nodes having the highest value of coreness which best defines which are the most influential nodes in a network.

(Refer Slide Time: 08:39)



```
#!/usr/bin/env python
G.add_edges_from([(1,2),(3,11),(4,5),(5,6),(5,7),(5,8),(5,9),
(5,10),(10,11),(10,13),(11,13),(12,14),(12,15),(13,14),(13,15),
(13,16),(13,17),(14,15),(14,16),(15,16)])

dict_deg={}
dict_cl={}
dict_bw={}
dict_cr={}

for each in G.nodes():
    dict_deg[each]=G.degree(each)
    dict_cl[each]=nx.closeness centrality(G,each)
    dict_bw[each]=nx.betweenness centrality(G,each)
    dict_cr[each]=nx.core_number(G)[each]

dict_cascade={}
```

So, what we are going to do now is we are going to look at different-different nodes, arrange them according to certain degree closeness, betweenness, coreness and their cascading power and then see which of the centrality is matched the most with influential

power of a node. Where influential power is over noticed, if you select that particular node as seed for your infection, how many nodes are infected by the end of your process.

So, what we are going to do here now is we are going to use the same code which we have written previously for the independent cascade model ok. So, we are going to make 4 dictionaries now for holding different values. A dictionary for holding the degrees of nodes, a dictionary for holding the closeness of nodes; let us call it `cl` for simplicity and then dictionary for holding the betweenness of nodes. Let us call it `bw` and then a node a dictionary for holding the core numbers of nodes which are found by a shell decomposition and let us call that coreness denoted by `cr` ok.

So, we have these 4 dictionaries and now we are going to flood these dictionaries. How do we put in the values for these dictionaries for each in `G.nodes` what we are going to do is dictionary for degree the value corresponding to the key `each` which is the node `e` is the degree of that node, `G.degree(each)` right. And then dictionary corresponding to the closeness `each`, so there is a function in `networkx` you can directly calculate the closeness of a node and it is called `nx.closeness centrality` you can look it up and then stored closeness centrality.

And then we are going to put `G` comma `each` here and there is a similar function for betweenness `bw` is `nx` and this is simply called betweenness centrality. So, betweenness and a surprise for you I hope you did not know it earlier otherwise you would not have heard the previous programming screen cast. You can actually get the core number of a node directly with the help of `networkx` with the help of the function known as core number, which performs `k` shell decomposition on a network and gives you the core number for a `v1` node right.

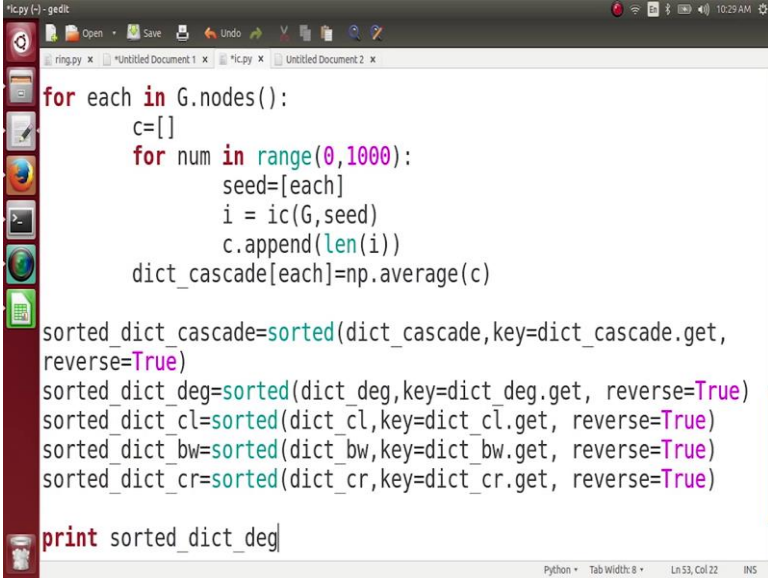
And here you see that the notation is a little bit different why. So, when you use `nx.core_number(G)` is nothing but a dictionary. In that dictionary we want, so these are these are the values. So, these are the functions which returns you the values. `nx.core_number` is a function. So, this is a function which returns your dictionary and from that dictionary you want the value corresponding to the node `each`.

So, that is why we have wrote it in this manner. So, we got the dictionaries corresponding to all these force and centralities. And at the end what we need we are creating our own dictionary which holds the cascading power of node. So, we have a

dictionary for that and how do we flood these dictionary? We have to flood this dictionary using our independent cascade model. So, how do we do that? So, in this 1 by 1 we will select every node in this network as seed. So, we will take 1s seed, 2s seed and what we will do to as we know when we run this independent cascade model, it is quite a random process right. So, we will run it some 1000 times.

So, we will take 1 s seed a run independent of cascade models from this 1000 times and take the average number of nodes which are getting infected.

(Refer Slide Time: 12:57)



```
for each in G.nodes():
    c=[]
    for num in range(0,1000):
        seed=[each]
        i = ic(G,seed)
        c.append(len(i))
    dict_cascade[each]=np.average(c)

sorted_dict_cascade=sorted(dict_cascade,key=dict_cascade.get,
reverse=True)
sorted_dict_deg=sorted(dict_deg,key=dict_deg.get, reverse=True)
sorted_dict_cl=sorted(dict_cl,key=dict_cl.get, reverse=True)
sorted_dict_bw=sorted(dict_bw,key=dict_bw.get, reverse=True)
sorted_dict_cr=sorted(dict_cr,key=dict_cr.get, reverse=True)

print sorted_dict_deg
```

So, what do we do? For each in G.nodes what we are going to do is c is currently empty. And then we take a loop which runs 1000 times for numbing range to 0 to 1000 and what we are going to do we said the seed of the process to be equal to each and then we call our independent cascade model with graph G and seeds.

So, now this value i over here is nothing but a list which holds the final people infected when the seed was the 1 node each here. So, what do we do? We append this length of i inner array c right. So, some 1000 times we get different-different people who are infected for different-different processes and for all these 1000 times we append the number of people who are infected in this array c.

So, this is array c we will be consisting of 1000 entries and each entry will correspond to 1 process where this node each was infected, and some people were infected by the end

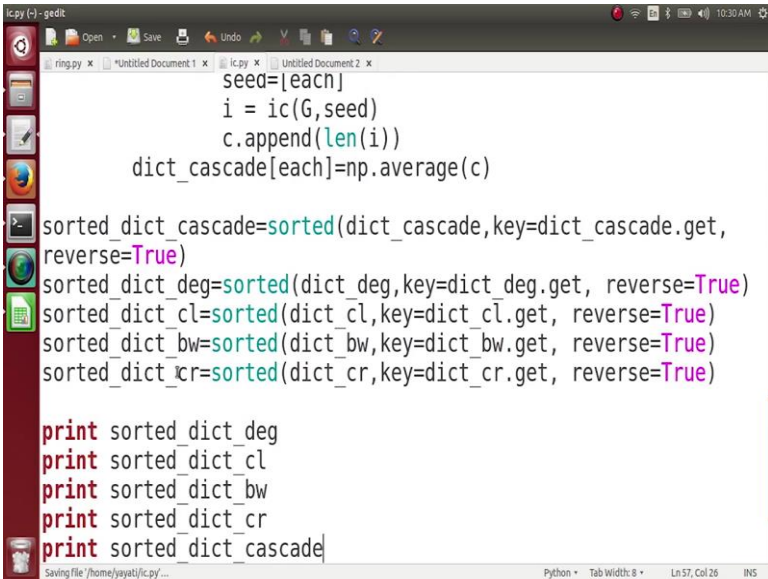
of the process. And now what I am going to append in my dictionary cascade dig the cascade each is the average, average from c, that is on an average how many people got infected, if I started my infection from this node each right.

And then what I am going to do now is I have find out I found out 5 dictionaries, 1 for degree closeness betweenness cornice and 1 for cascading and now I want to see which out of these 4 correlates the most with this 1, which is our degree for cascade.

So, what I am going to do is I am going to sort all of these dictionaries. So, sorting is easy with 1 command, so I will let first sort this dictionary corresponding to cascade So, there is a function sorted which you can call on a dictionary. So, you pass your dictionary cascade here right and we want to sort according to the keys are going to be nothing, but dictionary cascade.get and we want to sort them in a reverse order. So, we said reverse equals to true and we do this with all our dictionaries with all the remaining 4 dictionaries.

So, this cascade thing we can core degree we will replace it with degree and for closeness we replace it with cl and for betweenness we replace it with bw, bw and here also bw right and four corners we replace it with cr. And we can actually print all of these and see what is happening.

(Refer Slide Time: 16:29)



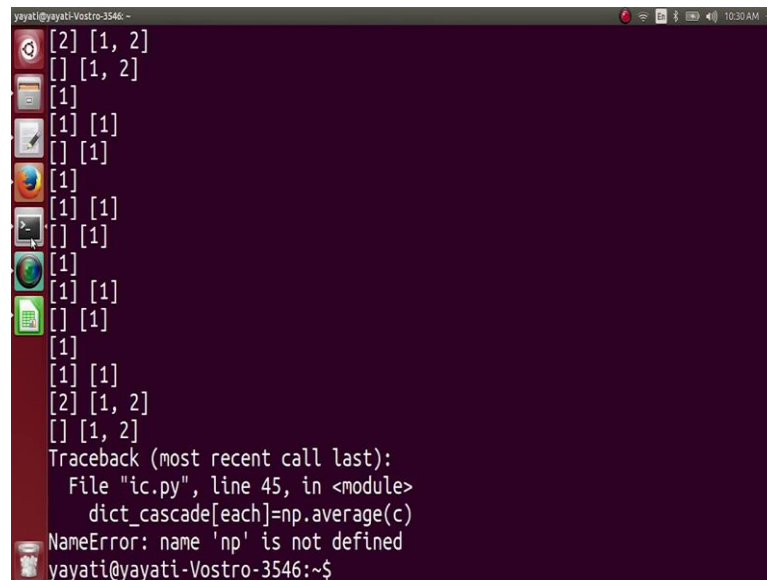
```
ic.py (-) - gedit
Open  Save  Undo  V  Print  Run  Find  Help
ring.py x  *Untitled Document 1 x  ic.py x  Untitled Document 2 x
    seed=[each]
    i = ic(G,seed)
    c.append(len(i))
    dict_cascade[each]=np.average(c)

sorted_dict_cascade=sorted(dict_cascade,key=dict_cascade.get,
reverse=True)
sorted_dict_deg=sorted(dict_deg,key=dict_deg.get, reverse=True)
sorted_dict_cl=sorted(dict_cl,key=dict_cl.get, reverse=True)
sorted_dict_bw=sorted(dict_bw,key=dict_bw.get, reverse=True)
sorted_dict_cr=sorted(dict_cr,key=dict_cr.get, reverse=True)

print sorted_dict_deg
print sorted_dict_cl
print sorted_dict_bw
print sorted_dict_cr
print sorted_dict_cascade
Saving file /home/yayati/ic.py...
```


Print let us print degree first and then let us print degree and then let us print closeness and then let us print betweenness, and then coreness, and then cascade right. Let us implement it and see no ok.

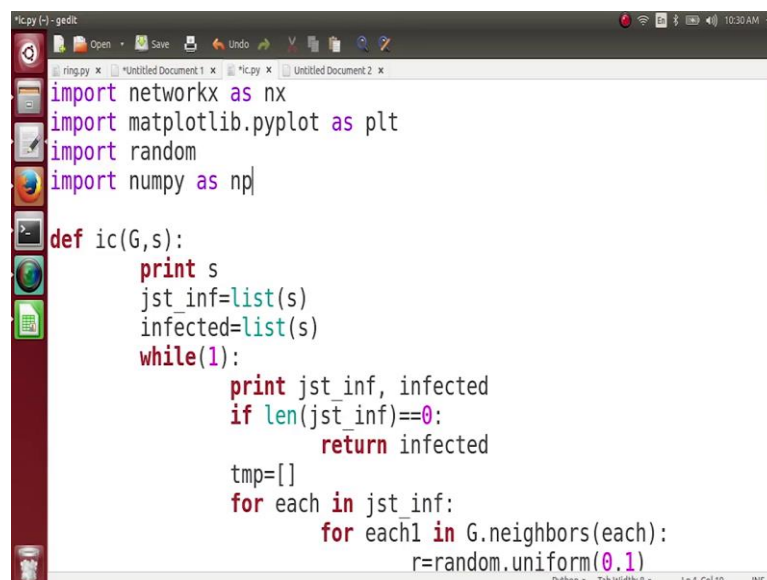
(Refer Slide Time: 16:55)



```
yayati@yayati-Vostro-3546:~$ python ic.py
[2] [1, 2]
[] [1, 2]
[1]
[1] [1]
[] [1]
[1]
[1] [1]
[] [1]
[1]
[1] [1]
[] [1]
[1]
[1] [1]
[2] [1, 2]
[] [1, 2]
Traceback (most recent call last):
  File "ic.py", line 45, in <module>
    dict_cascade[each]=np.average(c)
NameError: name 'np' is not defined
yayati@yayati-Vostro-3546:~$
```

So, we have used this numpy module for finding the average we need to import it.

(Refer Slide Time: 17:05)

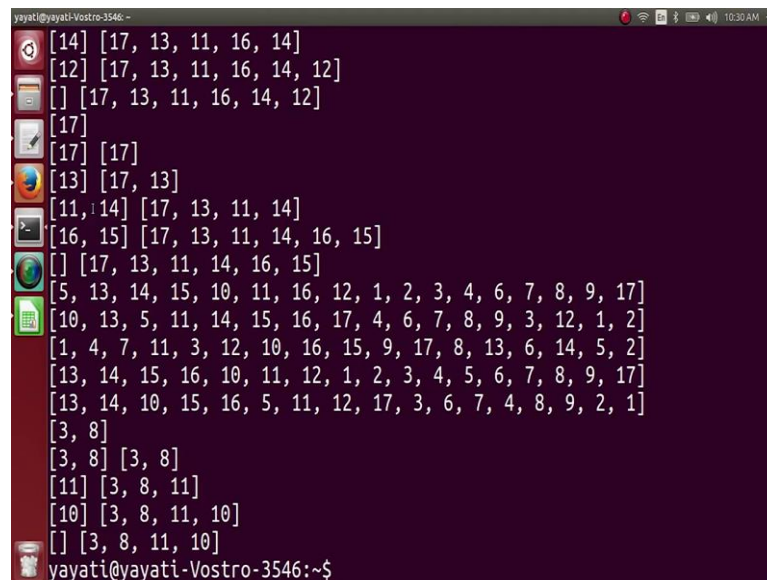


```
ic.py (-) - gedit
import networkx as nx
import matplotlib.pyplot as plt
import random
import numpy as np

def ic(G,s):
    print s
    jst_inf=list(s)
    infected=list(s)
    while(1):
        print jst_inf, infected
        if len(jst_inf)==0:
            return infected
        tmp=[]
        for each in jst_inf:
            for each1 in G.neighbors(each):
                r=random.uniform(0,1)
```

So, import numpy as np and actually we were printing some extra values where we print some extra values not sure, we are actually printing ok.

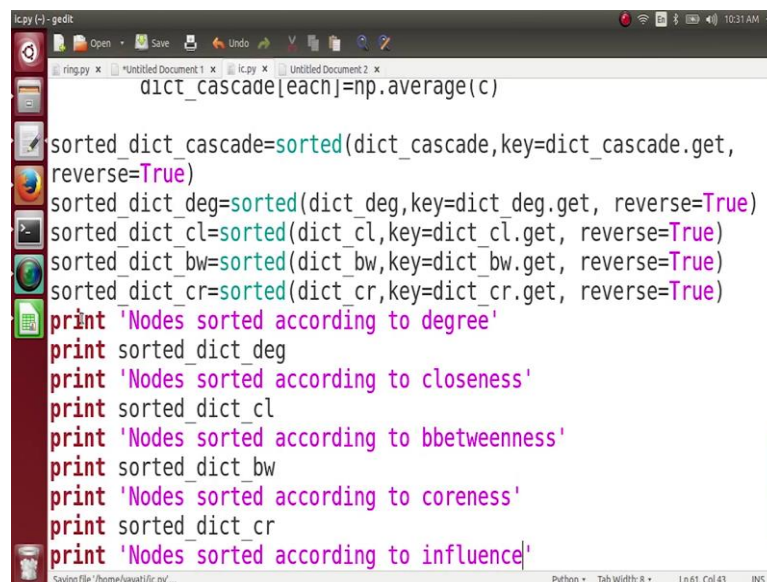
(Refer Slide Time: 17:31)



```
yayati@yayati-Vostro-3546:~$  
[14] [17, 13, 11, 16, 14]  
[12] [17, 13, 11, 16, 14, 12]  
[] [17, 13, 11, 16, 14, 12]  
[17]  
[17] [17]  
[13] [17, 13]  
[11, 14] [17, 13, 11, 14]  
[16, 15] [17, 13, 11, 14, 16, 15]  
[] [17, 13, 11, 14, 16, 15]  
[5, 13, 14, 15, 10, 11, 16, 12, 1, 2, 3, 4, 6, 7, 8, 9, 17]  
[10, 13, 5, 11, 14, 15, 16, 17, 4, 6, 7, 8, 9, 3, 12, 1, 2]  
[1, 4, 7, 11, 3, 12, 10, 16, 15, 9, 17, 8, 13, 6, 14, 5, 2]  
[13, 14, 15, 16, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 17]  
[13, 14, 10, 15, 16, 5, 11, 12, 17, 3, 6, 7, 4, 8, 9, 2, 1]  
[3, 8]  
[3, 8] [3, 8]  
[11] [3, 8, 11]  
[10] [3, 8, 11, 10]  
[] [3, 8, 11, 10]  
yayati@yayati-Vostro-3546:~$
```

So, we do not to see all these values because, so many times our process have run ok. So, we can actually if we do not need this portion of the core and let us now run it and see. And we do not know whether where one particular list is ending. So, let us say here print, so take degrees.

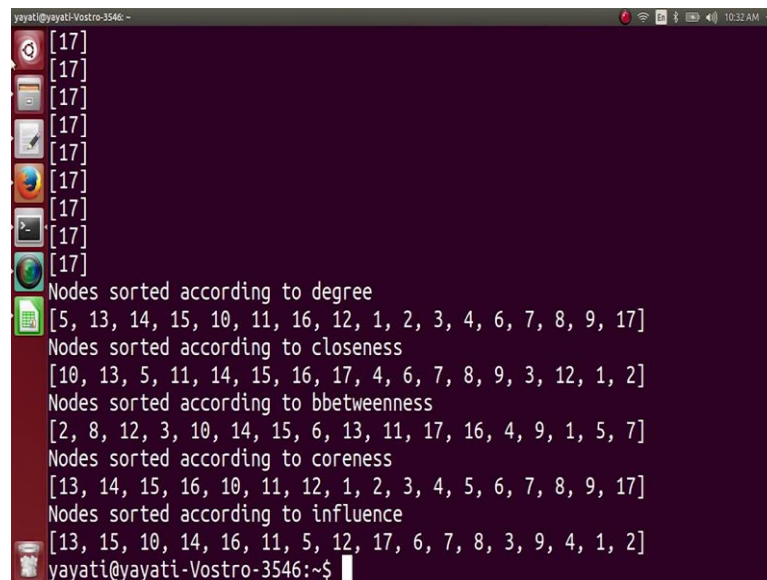
(Refer Slide Time: 17:59)



```
dict_cascade[each]=np.average(c)  
  
sorted_dict_cascade=sorted(dict_cascade,key=dict_cascade.get,  
reverse=True)  
sorted_dict_deg=sorted(dict_deg,key=dict_deg.get, reverse=True)  
sorted_dict_cl=sorted(dict_cl,key=dict_cl.get, reverse=True)  
sorted_dict_bw=sorted(dict_bw,key=dict_bw.get, reverse=True)  
sorted_dict_cr=sorted(dict_cr,key=dict_cr.get, reverse=True)  
print 'Nodes sorted according to degree'  
print sorted_dict_deg  
print 'Nodes sorted according to closeness'  
print sorted_dict_cl  
print 'Nodes sorted according to bbetweenness'  
print sorted_dict_bw  
print 'Nodes sorted according to coreness'  
print sorted_dict_cr  
print 'Nodes sorted according to influence'
```

So, these are your nodes sorted according to degree, according to closeness, according to betweenness, according to coreness and according to we can call it nothing but the influence fine.

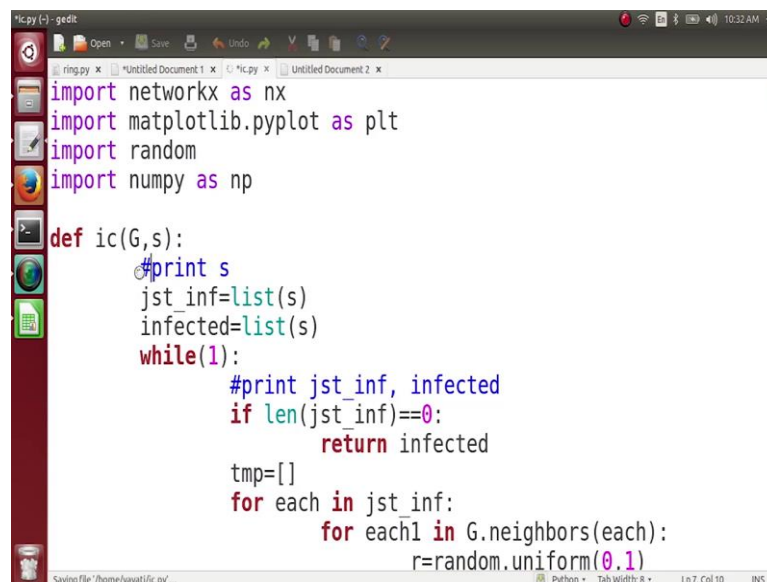
(Refer Slide Time: 18:45)



```
yayati@yayati-Vostro-3546:~$  
[17]  
[17]  
[17]  
[17]  
[17]  
[17]  
[17]  
[17]  
[17]  
Nodes sorted according to degree  
[5, 13, 14, 15, 10, 11, 16, 12, 1, 2, 3, 4, 6, 7, 8, 9, 17]  
Nodes sorted according to closeness  
[10, 13, 5, 11, 14, 15, 16, 17, 4, 6, 7, 8, 9, 3, 12, 1, 2]  
Nodes sorted according to betweenness  
[2, 8, 12, 3, 10, 14, 15, 6, 13, 11, 17, 16, 4, 9, 1, 5, 7]  
Nodes sorted according to coreness  
[13, 14, 15, 16, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 17]  
Nodes sorted according to influence  
[13, 15, 10, 14, 16, 11, 5, 12, 17, 6, 7, 8, 3, 9, 4, 1, 2]  
yayati@yayati-Vostro-3546:~$
```

So, we have many extra things printed here which we actually do not want. Somewhere this print statement is coming ok, so it is coming from here we can actually comment it and run it again. Still something is sprinting, s is sprinting here, we comment it also see here.

(Refer Slide Time: 19:07)



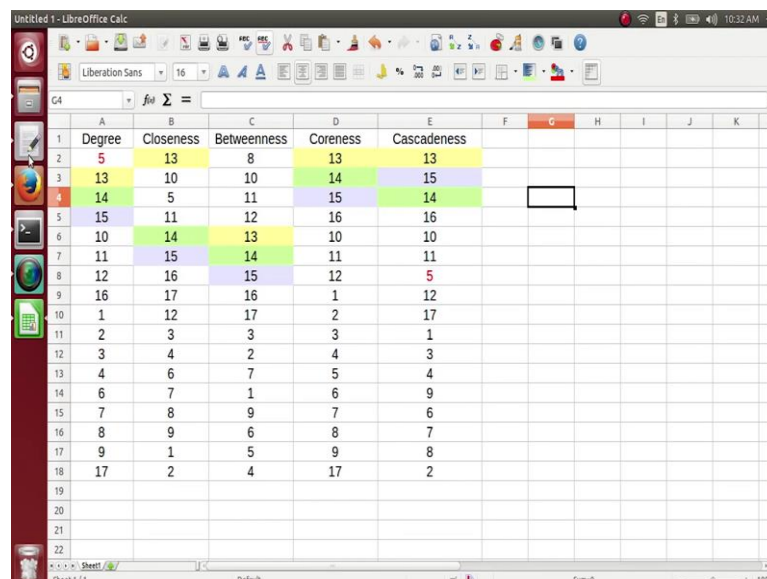
```
import networkx as nx  
import matplotlib.pyplot as plt  
import random  
import numpy as np  
  
def ic(G,s):  
    #print s  
    jst_inf=list(s)  
    infected=list(s)  
    while(1):  
        #print jst_inf, infected  
        if len(jst_inf)==0:  
            return infected  
        tmp=[]  
        for each in jst_inf:  
            for each1 in G.neighbors(each):  
                r=random.uniform(0,1)
```

(Refer Slide Time: 19:11)

```
yayati@yayati-Vostro-3546:~$ python ic.py
Nodes sorted according to degree
[5, 13, 14, 15, 10, 11, 16, 12, 1, 2, 3, 4, 6, 7, 8, 9, 17]
Nodes sorted according to closeness
[10, 13, 5, 11, 14, 15, 16, 17, 4, 6, 7, 8, 9, 3, 12, 1, 2]
Nodes sorted according to betweenness
[3, 4, 11, 14, 10, 16, 17, 13, 9, 12, 8, 15, 5, 7, 6, 2, 1]
Nodes sorted according to coreness
[13, 14, 15, 16, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 17]
Nodes sorted according to influence
[13, 15, 14, 10, 16, 5, 11, 12, 17, 6, 4, 7, 8, 3, 9, 1, 2]
yayati@yayati-Vostro-3546:~$
```

So, these are your nodes sorted according to your degrees, node sorted according to closeness between this coreness and influence. So, what I have done? I have actually run this process and I have kept these values here. So, these values might not be exactly the same.

(Refer Slide Time: 19:35)



	A	B	C	D	E	F	G	H	I	J	K
	Degree	Closeness	Betweenness	Coreness	Cascadeness						
1	5	13	8	13	13						
2	13	10	10	14	15						
3	14	5	11	15	14						
4	15	11	12	16	16						
5	10	14	13	10	10						
6	11	15	14	11	11						
7	12	16	15	12	5						
8	16	17	16	1	12						
9	1	12	17	2	17						
10	2	3	3	3	1						
11	3	4	2	4	3						
12	4	6	7	5	4						
13	6	7	1	6	9						
14	7	8	9	7	6						
15	8	9	6	8	7						
16	9	1	5	9	8						
17	17	2	4	17	2						
18											
19											
20											
21											
22											

So, for degree and closeness these are going to be the exactly the same. So, if you see a 5, 13, 14, 15 it is going to be the same 5, 13, 14, 15. For finding out the betweenness we use a random process, the algorithm uses a random process.

So, the value here might not be very same as what is here and for the coreness it is going to be the same. And for influence it is mostly going to be the same because, we have run this process a 1000 times. So, the results which we are going to get will be the same 13, 14, 15, 10 and you can see 13, 14, 15, 10 and so on. Ok let us analyze what's happening. So, here I have taken a spreadsheet, here I have put nodes according to a descending order of degree, descending order of closeness and so on.

Now I want to show you here something. Do you see here this cascade ness? It is actually nothing but the influence, so it is the influence here ok. So, if you see here you see the node which is having the highest influence is the one having the highest coreness. And actually the highest closeness it means that maybe closeness and coreness relates to influence, but if you look at the second highest node here which created the second highest cascade it as a high coreness, but it does not have a high closeness.

So, overall, you can see that this influence over here is mostly related to the coreness and degree. So, closeness if you see nodes 10, 5 and 11 here this goes quite down here and then this node edge here comes over here. So, the influence is maximally correlated with the coreness and degree here and out of coreness and degree although you can see that it is mostly related to coreness. This node 5 over here has a very high degree, but it has a very less influential power. So, this result it validates it is that: if you want to look at the maximum influential power of a node you should look at the nodes which are having the maximum coreness.

So, I have done this experiment over here for 1 network. You can actually replicate this experiment for various different networks rather you can use some real-world data sets also and verify this.