

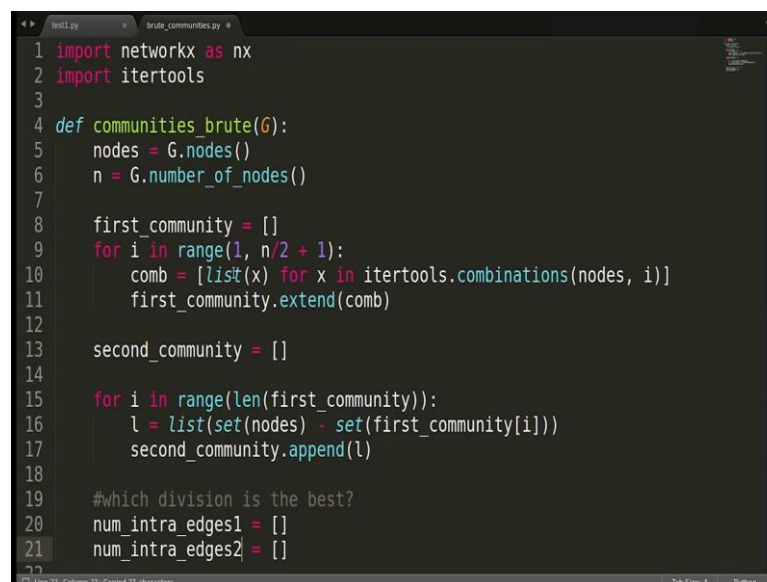
Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture - 35
Strong and Weak Relationships
Finding Communities in a graph
(Brute Force Method)

Hi everyone! In this video we are going to divide the nodes of a given graph into 2 communities using a Brute Force approach. Now Brute Force approach means we will be trying all possible divisions of the nodes of the graph into 2 parts and after that we will check which of these divisions is the best? Now how do we check which of these divisions is the best? As per the definition of communities, the nodes which are a part of a community; they form a lot of connections to the nodes of the same community and they form very less connections with the nodes of the other communities.

So, here we are going to make use of this property and we are going to find the best division out of all possible divisions and that is what you will display. So, let us get started.

(Refer Slide Time: 00:59)



```
1 import networkx as nx
2 import itertools
3
4 def communities_brute(G):
5     nodes = G.nodes()
6     n = G.number_of_nodes()
7
8     first_community = []
9     for i in range(1, n/2 + 1):
10         comb = [list(x) for x in itertools.combinations(nodes, i)]
11         first_community.extend(comb)
12
13     second_community = []
14
15     for i in range(len(first_community)):
16         l = list(set(nodes) - set(first_community[i]))
17         second_community.append(l)
18
19     #which division is the best?
20     num_intra_edges1 = []
21     num_intra_edges2 = []
22
```

So, let me import networkx. So, let me create a function here, let us pass the graph here. So, we are given the graph object, our aim is to divide the nodes of this graph into 2

partitions. So, let us first get the nodes of this graph. So, we can make use of this function `G.nodes` you might be knowing that this returns a list of the nodes of the graph let us also check the total number of nodes.

So, we can write `G.number_of_nodes`, now our aim is to divide the nodes of this graph into 2 communities; let us call them first community and second community now on. So, we must divide the nodes this list of nodes into first community and second community. Since you have to try all possible cases, we would have to try all those cases where the first community has one node and the second community has $n - 1$ node. All those cases where the first community has 2 nodes and the second committee has $n - 2$ nodes; and all those cases were the first community has 3 nodes and second community has $n - 3$ nodes and so on.

So, we must check for all these possible cases. So, if we find the combination of nodes that are going to fall in the first community, it is very easy to get the nodes which are going to fall in the second community because they are going to be all the remaining nodes. So, our aim is to get all the nodes that may fall into the first community; basically, all the combinations of the nodes that may fall in the first community. So, how do we do that? To get the combination, there is a function in the package called `itertools`. So, I am going to import that package and function name is combinations.

(Refer Slide Time: 03:39)

```
In [2]: import itertools

In [3]: itertools.combinations([1,2,3,4],2)
Out[3]: <itertools.combinations at 0x7fe4b975c208>

In [4]: for i in itertools.combinations([1,2,3,4],2):
...:     print i
...:
(1, 2)
(1, 3)
(1, 4)
(2, 3)
(2, 4)
(3, 4)

In [5]: for i in itertools.combinations([1,2,3,4],2):
...:     print list(i)
```

Let me show you the functionality of this function first. So, let me show you on ipython console a how this work function works, let me import itertools here and let me show you the functioning of the combinations function `itertools dot combination`. So, we pass 2 parameters in this function; first is the list from which we want the combinations and second is some number that we pass which goes from one to the length of the list. So, length of the list is 4 here, we can pass any number from 1 to 4 here let me pass 2 over here.

So, what it does is it returns all possible combinations out of this list which are of length 2. So, let me run this, this another point to note here is that this function does not return a list it returns an object, which has tuples of the combination as in the combinations are in the form of tuples. So, when if you want to see that we will write maybe we can use for loop for i in let me use this for i in this object print i. So, it has returned the combinations in the form of tuples, since we are working in list, we would want to convert them into list. So, we can do that by just writing list over here. So, now, it is returning in the form of list.

(Refer Slide Time: 05:15)

```
(3, 4)
In [5]: for i in itertools.combinations([1,2,3,4],2):
        print list(i)
...:
[1, 2]
[1, 3]
[1, 4]
[2, 3]
[2, 4]
[3, 4]

In [6]: for i in itertools.combinations([1,2,3,4],3):
        print list(i)
...:
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]

In [7]:
```

Similarly, if you want all the combinations which are of length 3 then you will write like this. So, our aim here is to get all possible combinations of the nodes list which are of length 1 2 3 and so on. So, let us get back here we are going to put all the combinations in a list, let me create a list first say first community. Now I am going to start a loop for i

in range one to what is the length of our list that is n . So, I will write $n/2 + 1$. I will explain why I get this. So, I will store these combinations that this function will written in the form of a list and since it is it returns the tuples, I will pass convert them into list. So, I will write list of x for x in whatever it returns.

Let me also pass the parameters. So, the parameters are going to be the nodes list and i . So, as you can see the i is ranging from 1 up to $n/2 + 1$. So, why did we do $n/2$ here? This is because the case is where the first community has k nodes, and the second community has the remaining $n - k$ nodes and the cases where the first community has $n - k$ nodes and the second community have k nodes. So, these cases are going to be the same these divisions are basically the same. So, there is no point of checking all the n to 1 for the first community combinations. So, that is why we are starting from 1 and we are stopping at $n/2$ that is if the list nodes list has 10 elements. So, we will start from 1, 2, 3, 4, up to 5 and we are not going to check for 6, 7, 8, up to 10 cases, this is because they are going to give us the same divisions and there is no point checking for the extra cases.

And the reason why we put plus one here is that you might be knowing the functionality of this range function if you pass 1, n in range it goes from 1 to $n - 1$. So, since we wanted to go exactly up to $n/2$ we are going to add plus 1 over here. So, what is this comb variable going to have; this is basically going to have a list this comb is a list of lists where all the list is having i elements. So, as I showed you on the screen comb is going to have all these 3 all these 4 lists for the value 3 and for the value 2 it is going to have all these list, and after this our aim is to add all these combinations to this list first community. So, what I am going to do here is first community dot extend and I am going to add all the list that I am getting from this comb based.

(Refer Slide Time: 08:57)

```
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]

In [7]: l1 = [1,2]

In [8]: l2 = [3,4]

In [9]: l3 = [5,6]

In [10]: l1.append(l2)

In [11]: l1
Out[11]: [1, 2, [3, 4]]

In [12]: l2.extend(l3)

In [13]: l2
Out[13]: [3, 4, 5, 6]

In [14]:
```

Now, let me tell you why I am using this extend function here and not the append function, let me show you an over here why i used extend let me take some example lists; let me take another list and yet another list. So, if I do `l1.append(l2)` what will happen let me print `l1`. See how it is adding the elements my aim was basically to add this 3 4 2 this list which is having 12 already. So, when I write `l1.append(l2)` were `l2` is itself a list this is how it does.

In case I want to add only the elements we make use of the function extend. So, let me show you `l2.extend(l3)`. So, let me print `l2`, now this is how it works and this is how I wanted to be; that is why I have used this function extend this is because `comb` is a list of list and I want to add the elements only to this first community. So, we are using extend function here now we have stored all possible combinations of nodes in in this list first community, now we should also do the same for the second community as I am going to create a list here, now this is going to be very easy because all the remaining node are going to fall into this community.

Now, you see we have the list of all the nodes we have the list of nodes which are going to fall in the first community, we must get the list of remaining nodes in the second community. So, how do we do that? Basically, we must subtract the elements of first community from the elements in nodes. So, how do we do subtraction in list? If you want to do subtraction set is the appropriate data structure. So, what we can do is we can

convert the list into set and after that we can apply the subtraction operation for example, I can write $\text{set nodes} - \text{set first community}$. So, whatever it will give me it will give me the difference of the elements from nodes, $\text{nodes} - \text{first community}$ it will give me the difference of elements and that will also be in the form of set. So, I can later convert them into a list. So, that is how I will get the second community lists. So, let me store it here.

Now, since we have to do it for all the elements of first community, we have to start a loop here. So, I will write `for i in range`. So, I am going to do it for all the elements of first community. So, I will write `length of first community` all right. So, we are going to do it for all the elements of the first community and once we get the list in `l` we have to append it to the second community list. So, I will write `second community dot append l`. So, now, we have all the combinations in this list `first community` and all the combinations of the second community in this list `second community`, and please note that these 2 lists are themselves list of lists. So, we have the divisions in these lists, and we have to find which division is the best.

Now, as I told you as per the definition of community communities, the nodes which are a part of one community have a lot of connections with the nodes in the same community that is there are going to be a lot of intra community edges and there are going to be very less intercommunity edges. So, if a division is good then the number of intra community edges should be high and the number of intercommunity edges should be less. So, this is what we are going to use here. So, we are going to keep track of the number of intra and intercommunity edges for all the possible divisions that we have made so far.

(Refer Slide Time: 13:49)

```
17 second_community.append(1)
18
19 #which division is the best?
20 num_intra_edges1 = []
21 num_intra_edges2 = []
22 num_inter_edges = []
23 ratio = [] # ratio of number of intra/number of inter community edges
24
25 for i in range(len(first_community)):
26     num_intra_edges1.append(G.subgraph(first_community[i]).number_of_edges())
27
28 for i in range(len(second_community)):
29     num_intra_edges2.append(G.subgraph(second_community[i]).number_of_edges())
30
31 e = G.number_of_edges()
32
33 for i in range(len(first_community)):
34     num_inter_edges.append(e - num_intra_edges1[i] - num_intra_edges2[i])
35
36 #Find the ratio
37
```

So, what you going to do now is which division is the best. So, this is what we are going to check now, since we have to keep track of the number of intra community edges in the first community we will create a list for it and we have to keep a track of intercommunity edges in the second community we will create a list for it and we also have to keep track of the number of intercommunity edges. So, we have these 3 lists, now apart from this to check which division is the best we are going to find the ratio where (Refer Time: 14:05) in the numerator will have the sum of intra community edges, and in the denominator will have the intercommunity edges.

So, whichever division has highest value of this ratio will mean that in that community inside community; there are dense connections and outside community there are sparse connections. So, we are going to find this ratio for every division that we have made so far. So, I am going to create this list as well, let me write here what this means ratio of intra divided by this is basically number of intra and number of inter community edges; alright.

Now, we have the nodes which are falling into the first community and the nodes which are fall into the second community, arrange to find the number of edges that are existing amongst the nodes in the first community and the same for the second community as well. Now how do we find the number of edges amongst the nodes in a given community because we have only as of now, we have only the list of nodes. So, for this purpose we

are going to make use of function that is subgraph, which is given which is inside the graph object. So, we will make use of a the subgraph function let me tell you the syntax for this function. So, this function is available in graph object. So, I will write `G.subgraph` and here I will pass the list of the nodes. So, in our case the list of nodes is kept in first community and second community also.

So firstly, we are doing for the first_community and for the first community also let us do it for i we are going to start loop here, for i in range basically whatever we are doing we are going to do it for all the lists given in the first community. So, we will start it for i in range will write length of first community yeah. So, I was saying you about this subgraph function this function basically takes as input a list of edges, and it returns a graph object itself which is an induced subgraph over this list of elements, over this list of nodes. So, whatever it returns is basically a graph and whatever operations you can apply on graph you can apply on the written value of this function as well. So, if you want to find the number of edges, we can just use the formula that is number of edges. Since this is itself or graph object as I told you subgraph returns an induced subgraph which is a sort of graph itself. So, you can apply this function number of edges on the return value.

Now, whatever it returns we must keep store that value in this list it is number of intra edges, because this will give us the number of intra edges in that subgraph. So, I will write number of intra edges in first_community.append all right. So, let me explain you once again what this statement is doing. So, we have the nodes that are falling into the first_community and these are here first_community right. We are starting this for loop from i goes from zero to the length of the first_community , basically we are checking it for all the possible combination. So, we have this list of nodes which we are passing into this subgraph function which. So, this function will return an object of graph itself. So, that will basically be an induced subgraph over these nodes. So, since that is a graph, we are calling this function number of edges to find the number of edges which are existing amongst those nodes.

(Refer Slide Time: 18:31)


```

7
8 community = []
9 range(1, n/2 + 1):
10 = [list(x) for x in itertools.combinations(nodes, i)]
11 t_community.extend(comb)
12
13 community = []
14
15 range(len(first_community)):
16 list(set(nodes) - set(first_community[i]))
17 nd_community.append(l)
18
19 Division is the best?
20 a_edges1 = []
21 a_edges2 = []
22 r_edges = []
23 [] # ratio of number of intra/number of inter community edges
24
25 range(len(first_community)):
26 intra_edges1.append(G.subgraph(first_community[i]).number_of_edges())
27

```

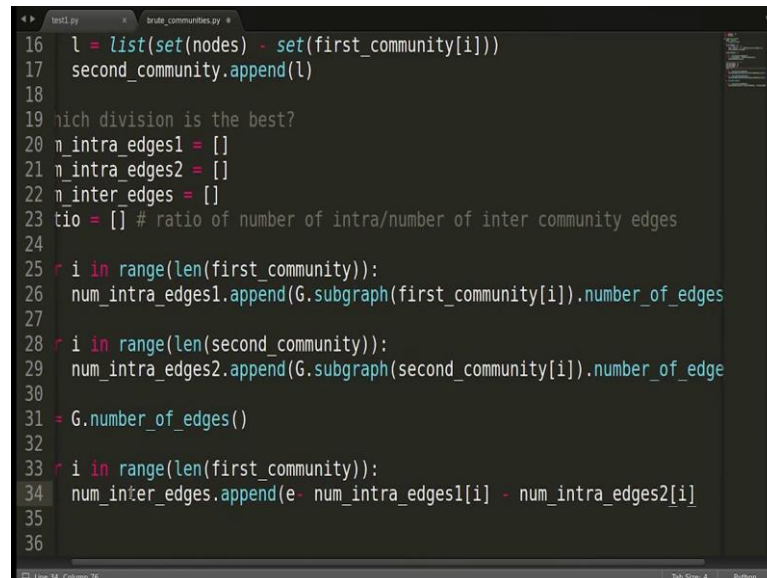
For example, if you pass one 2 3 here this will tell us the number of edges amongst these 1 2 and 3 nodes ,if there is an edge from 3 to 4 let us say, that will not be counted here that is because 4 is not a part of this subgraph. So, that is how we will get the number of edges and that information we are storing in this list as we know created this list number of intra edges one that is for first_community . So, we are appending that information here.

Now, the same thing you have to do for the second_community as well. So, we are just going to copy paste it and I am going to write second_community here and information has to go to this second list and here I will write second yeah, I think we are done. So, now, we have the number of intra edges in the first_community in this list and we have the number of intra edges in the second_community in this list, now we have to find the number of inter edges.

So, we can easily find the total number of edges in the graph right let us do that e is equal to G dot number of edges right. So, we got the total number of edges here and we have the intra edges in first_community and we have to intra edges in the second_community . So, we subtract these all these intra edges from the total number of edges, we will get the number of inter edges intercommunity edges. So, let us do that. So, what we will do is e minus number of intra edges in first_community ; we are going to start a loop now minus number of intra edges in the second_community .

And we have to do it for all values of i. So, I will start for i in range length of here we can use length of first_community or second_community because they are the same. So, number of intra edges or to maintain uniformity I will write first_community itself here. So, we have to keep this information in this list that is number of inter edges. So, i like I will write it here I am sorry. So, in this list number of inter edges dot append.

(Refer Slide Time: 21:34)



```
16 l = list(set(nodes) - set(first_community[i]))
17 second_community.append(l)
18
19 which division is the best?
20 n_intra_edges1 = []
21 n_intra_edges2 = []
22 n_inter_edges = []
23 tio = [] # ratio of number of intra/number of inter community edges
24
25 for i in range(len(first_community)):
26     num_intra_edges1.append(G.subgraph(first_community[i]).number_of_edges)
27
28 for i in range(len(second_community)):
29     num_intra_edges2.append(G.subgraph(second_community[i]).number_of_edges)
30
31 = G.number_of_edges()
32
33 for i in range(len(first_community)):
34     num_inter_edges.append(e - num_intra_edges1[i] - num_intra_edges2[i])
35
36
```

So, we have to append this information here. So, now, we have the number of intra edges in this list, we have the number of inter intra edges in the second_community in this list we have the number of inter edges in this list.

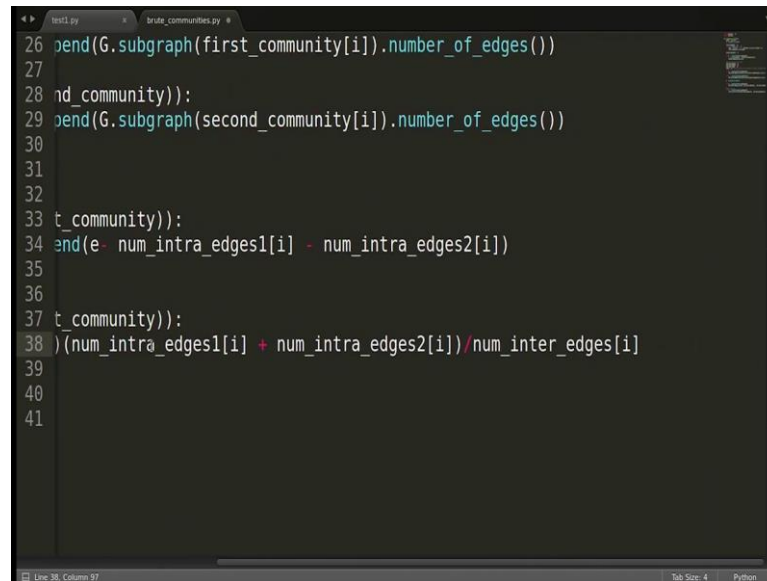
Next thing is to find is the ratio. So, let us find the ratio. So, again we have to start this loop am going to copy here.

(Refer Slide Time: 22:03)

```
26 num_intra_edges1.append(G.subgraph(first_community[i]).number_of_edges())
27
28 for i in range(len(second_community)):
29     num_intra_edges2.append(G.subgraph(second_community[i]).number_of_edges())
30
31 e = G.number_of_edges()
32
33 for i in range(len(first_community)):
34     num_inter_edges.append(e - num_intra_edges1[i] - num_intra_edges2[i])
35
36 #Find the ratio
37 for i in range(len(first_community)):
38     ratio.append((float)(num_intra_edges1[i] + num_intra_edges2[i]) / num_inter_edges[i])
39
40 max_value = max(ratio)
41 max_index = ratio.index(max_value)
42
43 print '(', first_community[max_index], '),(', second_community[max_index], ')'
44
45 G = nx.barbell_graph(5,0)
46 communities_brute(G)
```

I am sorry. So, we have to find the ratio for all these possible combinations; ratio is going to be the divisions of number of intra edges with the number of inter edges. So, numerator is going to be the total number of intra edges, we have intra edges in 2 lists. So, we are going to sum them up the value from this list plus the value from this list. So, I will write number of intra edges 1 for the ith combination, plus number of intra edges in 2 for the ith combination. So, we have to sum them up and then we have to divide them with the number of inter edges for the ith combination. So, since I want all the precise values. So, I will write float here and this will give me the ratio and this ratio I am going to keep in the list that I have already created by the name ratio. So, I will write ratio dot append.

(Refer Slide Time: 23:15)



```
26 end(G.subgraph(first_community[i]).number_of_edges())
27
28 nd_community)):
29 end(G.subgraph(second_community[i]).number_of_edges())
30
31
32
33 t_community)):
34 end(e- num_intra_edges1[i] - num_intra_edges2[i])
35
36
37 t_community)):
38 )(num_intra_edges1[i] + num_intra_edges2[i])/num_inter_edges[i]
39
40
41
```

So, let me explain you once again what this statement is doing, we are finding the ratio of the intranet inter edges. So, on the numerator we have the total number of intra edges and in the denominator we have the total number of inter edges, and this ratio we are just keeping in this list which is named ratio and we have already created that list. Now we have all the ratio in this ratio list and our aim is to find the list the best value of this ratio that is the maximum value of this ratio. So, what we can do is. So, let us create the variable max value is equal to. So, this is a function that you can use max function and the parameter can be the list this returns us the maximum value in the list ratio.

Now, our aim is not to get the maximum value basically, our aim is to get the index at which the maximum is coming so that we get to know; what was the best division? We can make use of the index function from the list that is ratio. So, what we can write is ratio dot index max value. So, what it will return is it will return the index of the max value here. So, I will write max index is equal ratio dot index. So, this index is what we are interested in; this is because this index tells us the division which is the best division out of all possible division.

So, now we got this max division I am sorry max index that is the best index, and in the first_community and in the second_community list we have the divisions of nodes into first and second committee. So, we are going to access the division out of this list and out of this list which is falling at this index. So, what we are going to do is we are going to

basically print the final result. So, I will write `first_community` and which element the one which is falling at max index and similarly `second_community` we are going to access the element which is falling at max index.

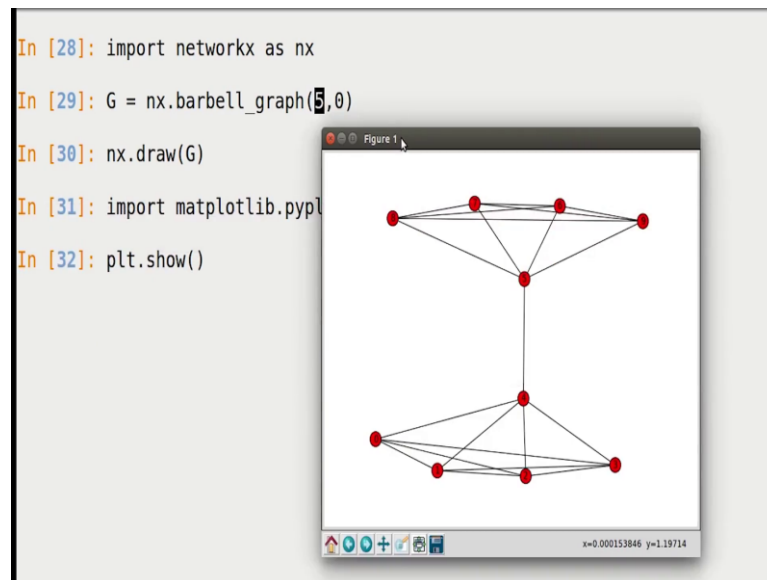
Now, I am going to print these. So, let me just do some representation thing here. So, after this is done and I will add a bracket here and then I will start the other 1 and this and then I will close it this is about the representation. So, this is how our lists 2 divisions will be printed divisions of nodes into 2 communities. Now the function is return we have to now call it and use it we want an example graph here. So, let me take an example graph here the function name is `communities brute`. So, let me call this and let me pass this graph as an example graph let us take this graph which is called barbell graph, I am going to show you what this graph looks like let us pass 2 parameters here. So, let me go back to the ipython screen and I will show you how this graph looks like.

(Refer Slide Time: 26:50)

```
In [28]: import networkx as nx
In [29]: G = nx.barbell_graph(5,0)
In [30]: nx.draw(G)
In [31]: import matplotlib.pyplot as plt
In [32]: plt.show()
```

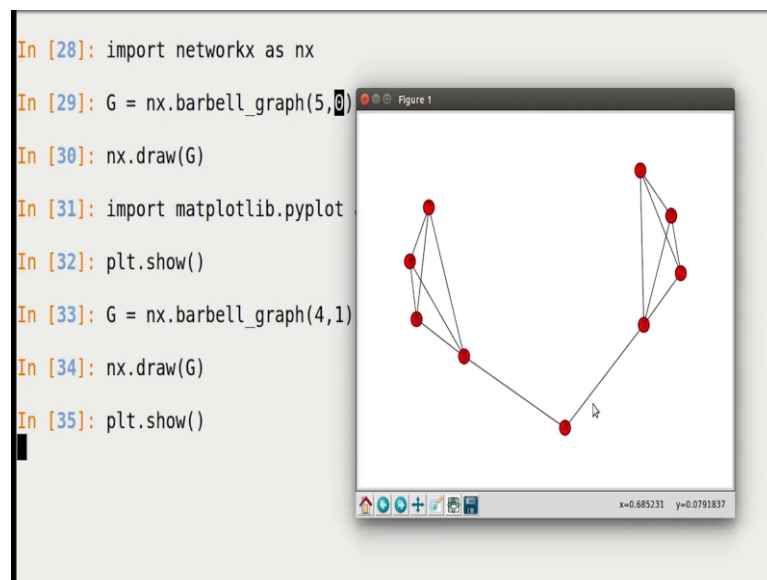
So, let us go here, here let me import the package `networkx` and I will show you how this graph looks like. So, I will take a graph `G = nx.barbell_graph`, and I am going to pass the same parameters. Now what it indicates I will show you in the visualization. So, I will write `nx.draw` to show this I need `matplotlib`. So, I will import `matplotlib` dot my plot as `plt` and I will write `plt.show`.

(Refer Slide Time: 27:39)



Now this is a graph that comes out of these 2 parameters 5 and 0. So, let me explain you what this parameter indicate; the first parameter which is 5 here it indicates that there should be 2 clicks of size 5, as you can see here first click and the second click and they should be connected by a path which has these nodes which is 0 node here.

(Refer Slide Time: 28:08)



So, this is a path which is connecting these 2 clicks and it has 0 node. So, this is how it looks like let me change the parameters a bit and show you how it comes out to be.

So, if I pass 4 and 1 let us see how it looks like. So, we passed 4 , 1 which means it should be 2 clicks of size 4 as you can see, and they should be connected with the path which has one node. So, the same is showing here.

(Refer Slide Time: 28:43)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3 (Strength of weak tie  
s)/Videos$ python brute_communities.py  
( [0, 1, 2, 3, 4] ),( [8, 9, 5, 6, 7] )  
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_3 (Strength of weak tie  
s)/Videos$
```

So, this is one node in between. So, this is a kind of barbell graph, I plan to take this graph because the kind of community is required to visible here and it is also small. So, it would not take much time for the testing. So, I am going to take the graph.

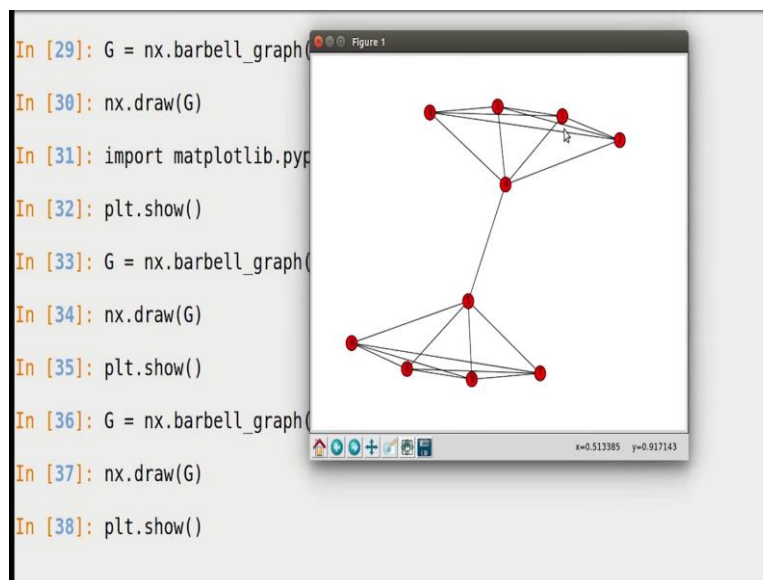
So, let us go and a test this function this is yeah. So, these are the 2 communities that we get if you remember we passed 5 , 0 which means $5 + 5$ there are going to be 10 nodes, and these 10 nodes are divided into 2 communities where the first_community have 0, 1, 2, 3, 4 and the second has 8, 9, 5, 6, 7. Let me quickly show you once again what kind of plot we were getting when we pass these parameters see this is what we got.

(Refer Slide Time: 29:20)

```
In [28]: import networkx as nx
In [29]: G = nx.barbell_graph(5,0)
In [30]: nx.draw(G)
In [31]: import matplotlib.pyplot as plt
In [32]: plt.show()
In [33]: G = nx.barbell_graph(4,1)
In [34]: nx.draw(G)
In [35]: plt.show()
In [36]: G = nx.barbell_graph(5,0)
In [37]: nx.draw(G)
In [38]: plt.show()
```

So, here you can see that communities are clearly visible that these should be the communities and our brute force algorithm is nicely able to differentiate the communities.

(Refer Slide Time: 29:28)



So, as you see 0, 1, 2, 3, 4, 5 are falling to one community, and the rest are in the second one. So, that is what we are getting here. So, the brute force algorithm is working fine, next we are going to see the Girvan Newman algorithm for community detection.