

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

**Rich Get Richer Phenomenon**  
**Lecture – 122**  
**Implementing Rich-getting-richer Phenomenon (Barabasi-Albert Model)-2**

(Refer Slide Time: 00:05)

---

**Implementing Rich-getting-richer Phenomenon**  
**(Barabasi-Albert Model)-2**

**Anamika Chhabra**

In the previous video, we discuss the main idea behind Rich-getting-richer Phenomena. And we also discuss this steps that we are going to follow for implementation. In this video we are going to start off with implementation.

(Refer Slide Time: 00:19)

```
rich-getrich_recording.py  existing_functions.py
1 import networkx as nx
2 import random
3 import matplotlib.pyplot as plt
4
5 def display_graph(G, i, ne): #i is the new node added, ne is the list of
6     pos = nx.circular_layout(G)
7     if i == '' and ne == '':
8         new_node = []
9         rest_nodes = G.nodes()
10        new_edges = []
11        rest_edges = G.edges()
12    else:
13        new_node = [i]
14        rest_nodes = list(set(G.nodes()) - set(new_node))
15        new_edges = ne
16        rest_edges = list(set(G.edges()) - set(new_edges) - set([(b,a) for (a,b) in new_edges]))
17    nx.draw_networkx_nodes(G, pos, nodelist = new_node, node_color = 'g')
18    nx.draw_networkx_nodes(G, pos, nodelist = rest_nodes, node_color = 'r')
19    nx.draw_networkx_edges(G, pos, edgelist = new_edges, edge_color = 'g')
20    nx.draw_networkx_edges(G, pos, edgelist = rest_edges, edge_color = 'r')
21    plt.show()
```

So, let me import some packages that we are going to need, we are going to need the network. We are going to take random number, so we will meet random package. We are also going to display the networks, so let us import matplotlib ok.

So, let us start our main function ok. So, if you remember from the previous video, if you remember the outline if you have not seen the video, you may go back and see the steps at we are going to follow here. So, we require the values of n, m and m0, n is the total number of users, m is the number of edges that every new node will get attached to, m0 is the number of nodes in the initial network that we will start with.

Let us take the value of n from the user. So, we can use the function raw input and in case you are using Python 3.x you can use input function instead of raw input. So, let us use this function, raw input and as the user entered the value of n all right. Next, we need the value of m0 as I told you can initialize m0 to be some 3, 4, 5 whatever you want. Let me take m0 based on n that the user will pass. So, I plan to take a random value between 2 is that is the minimum number of nodes that should be there in the initial network and n divided by 5; to n divided by 5 I will choose the random number between 2 to n/5. So, assume n the user entered n to be 100 then I will take a random value between 2 and 20 that is how I will start the initial network.

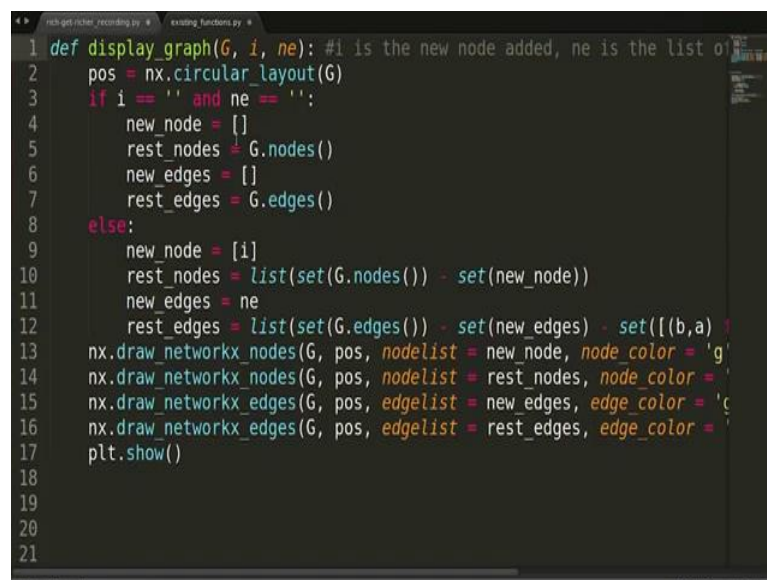
In fact, if the user enters a very high value of n that this might not be a good idea, you can it is it is actually better to fix up a value of n0 to be 3, 4 or something. For example,

user enters  $n$  to be 1000 then your initial networks will be between a random number between 2 and 200 right. It is fine you can take any value of  $m_0$ . And if you remember the only thing is that  $m$  should be less than equal to  $m_0$ . So, here we will take  $m$  to be  $m_0 - 1$ .  $m = m_0 - 1$ . And we have to start with an initial network with  $m_0$  nodes.

So, as of now we have not means started the graph. So, let us start the initial graph with  $m_0$  nodes. And the condition is that every node should have at least one edge. So, I will have a path graph because, there every node has at least one edge you can do any other thing like, you can add the edges one by one just making sure that an every node has one link at least. You can manually add the edges, so it is up to you. I will take a path graph, I will write `nx.path` graph of  $m_0$  nodes ok.

Now, to start off I wish to see how the graph looks like. So, that we can see the changes that that keep happening to the graph at every time stamp, I am going to make use of a function `display graph` that we created in one of the previous videos. So, I am going to just take the function from there and I will briefly explain it here, I will not write the function line by line here.

(Refer Slide Time: 04:37)

A screenshot of a Python code editor with a dark background. The code defines a function `display_graph(G, i, ne)`. The function takes a graph `G`, a new node `i`, and a list of new edges `ne`. It uses `nx.circular_layout(G)` for node positions. It then branches into two cases: if `i` and `ne` are empty, it resets the node and edge lists; otherwise, it adds the new node and edges to the existing sets. Finally, it uses `nx.draw_networkx_nodes` and `nx.draw_networkx_edges` to visualize the graph with different colors for new and existing elements, and `plt.show()` to display it.

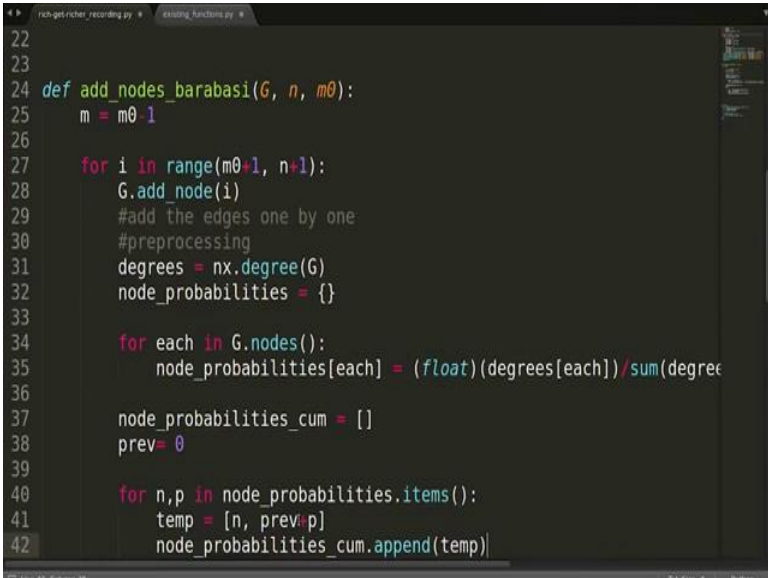
```
1 def display_graph(G, i, ne): #i is the new node added, ne is the list of new edges
2     pos = nx.circular_layout(G)
3     if i == '' and ne == '':
4         new_node = []
5         rest_nodes = G.nodes()
6         new_edges = []
7         rest_edges = G.edges()
8     else:
9         new_node = [i]
10        rest_nodes = list(set(G.nodes()) - set(new_node))
11        new_edges = ne
12        rest_edges = list(set(G.edges()) - set(new_edges) - set([(b,a) for (a,b) in new_edges]))
13    nx.draw_networkx_nodes(G, pos, nodelist = new_node, node_color = 'g')
14    nx.draw_networkx_nodes(G, pos, nodelist = rest_nodes, node_color = 'b')
15    nx.draw_networkx_edges(G, pos, edgelist = new_edges, edge_color = 'g')
16    nx.draw_networkx_edges(G, pos, edgelist = rest_edges, edge_color = 'b')
17    plt.show()
18
19
20
21
```

So, let me call that function `display graph` before that let me copy it. So, this is the function that we created one of the previous videos, I am copying it and pasting it here, I brief out for the once you are not see in the previous videos.

So, what this function is doing? We are passing the graph G that has to be displayed. We are passing i, i is the new node to be added and ne is the list of new edges that are going to be added. So, basically at every iteration I plan to call this function display graph and I also want to display the new edge that got added in this iteration, what did I say; I want to display the new node that got added, I and I also want to display the new edge is that got added. So, I am going to change the colours of this nodes and edges. So, that its nice to see how the nodes are getting added for that only I have created this customized function.

I am going to use a circular layout ok. So, I will use this function `nx.circular_layout(G)`. And since, I am going since i need to display the colours of the new node and new edges differently. So, what I have to do here is that. Firstly, I am putting this condition ok. So, initially there will be no new node no new edge. So, i and ne will be empty that is what will happened at the first before any iteration takes place initially right. So, new node will be. So, we will maintain a list of new nodes and we will maintain a list of new edges. Initially, they will be empty and rest of the nodes which is all the nodes in the graph, rest of the edges which is all the edges in the graph. So, we require these 4 list previous nodes, new node previous edges, new edges, these 4 list, we need ok..

(Refer Slide Time: 06:43)



```
22
23
24 def add_nodes_barabasi(G, n, m0):
25     m = m0 - 1
26
27     for i in range(m0+1, n+1):
28         G.add_node(i)
29         #add the edges one by one
30         #preprocessing
31         degrees = nx.degree(G)
32         node_probabilities = {}
33
34         for each in G.nodes():
35             node_probabilities[each] = (float)(degrees[each])/sum(degree
36
37         node_probabilities_cum = []
38         prev = 0
39
40         for n,p in node_probabilities.items():
41             temp = [n, prev, p]
42             node_probabilities_cum.append(temp)
```

In case i and ne are not empty, there are some values in them, what are we going to do is, new node list is going to have i new edges list is going to have ne, which we

passed as a parameter. Rest of the edges will be all the edges, I am sorry rest of the nodes will be all the nodes minus the new node, rest of the edges will be all the edges minus new edges minus the intersection right; minus the common edges basically, let me explain.

(Refer Slide Time: 07:17)



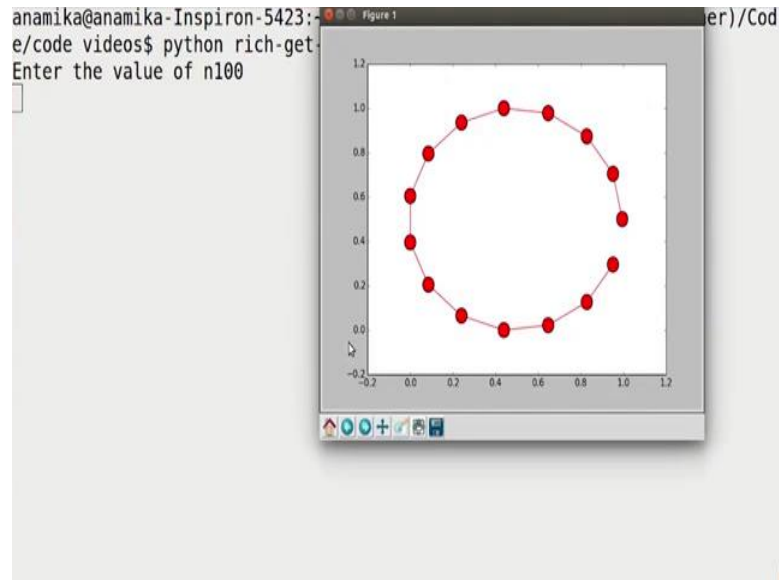
```
4
5 aph(G, i, ne): #i is the new node added, ne is the list of new edges added
6 circular_layout(G)
7 and ne == '':
8 de = []
9 nodes = G.nodes()
10 ges = []
11 lges = G.edges()
12
13 de = [i]
14 nodes = list(set(G.nodes()) - set(new_node))
15 ges = ne
16 lges = list(set(G.edges()) - set(new_edges) - set([(b,a) for (a,b) in ne
17 tworx_nodes(G, pos, nodelist = new_node, node_color = 'g')
18 tworx_nodes(G, pos, nodelist = rest_nodes, node_color = 'r')
19 tworx_edges(G, pos, edgelist = new_edges, edge_color = 'g', style = 'da
20 tworx_edges(G, pos, edgelist = rest_edges, edge_color = 'r')
21
22
23
24 input('Enter the value of n'))
```

So, it should take b a and a b to be the same edges right. So, that is why we are deducting that those edges ok. So, then we are displaying. So, since you want the graph to be customized having different nodes having different colours different edges different having different colours, you will have to make use of this function draw networkx nodes and draw networkx edges. So, separately you are calling them.

So, first function will be to display the existing nodes, second function will be to display the new nodes or vice versa. First function will be to display the new edges and second function is to display the existing edges ok. So, look at the parameters quickly we will pass the graph G and then the position that we already created and the node list that has to be given this particular colour. And we are specifying the node colour to be green here and the existing nodes, we are displaying red, it is new edges, we are display in green and you can also change the style and displaying the new edges to be dotted so that, we can precisely differentiate. And we are existing edges, we are displaying that. So, this was about the displaying part, now we are calling this function display graph and ok.

So, we are calling this function, we need 3 parameters  $G$ ,  $i$  and  $ne$ . So, let us pass  $G$  here and  $i$  initially, there is nothing, there is no node and the edges also will be empty ok. So, let us call this function let us call main to check the functioning of our existing code ok.

(Refer Slide Time: 09:33)



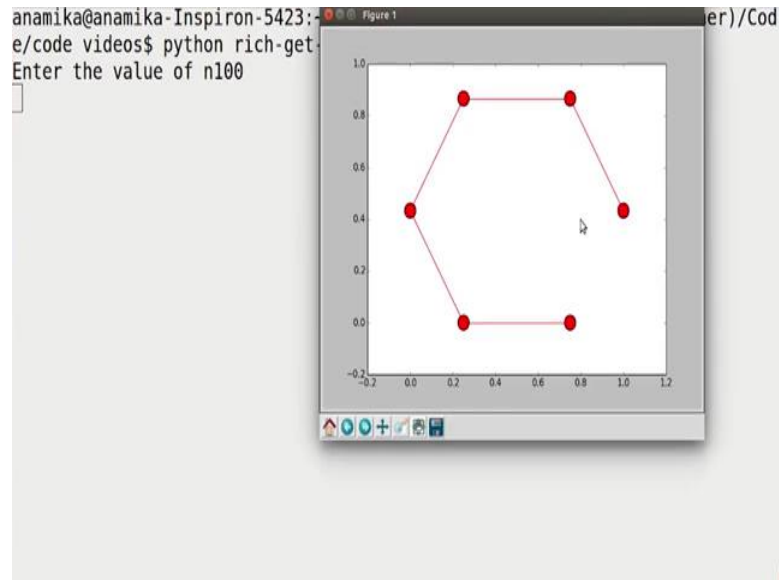
Let us call this file, enter the value of  $n$ , I am going to get 100 ok. So, this is the initial graph this is the initial graph as you can see, we had chosen  $m_0$  to be random. So, this is the number of nodes that the code has chosen. This is the initial graph and we are going to add the nodes one by one to this graph.

(Refer Slide Time: 09:56)

```
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ 100
100: command not found
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n100
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ clear
```

I can execute it again to see and this is taking these many nodes initially, also taking quite good number of nodes initially, I am sorry; I am sorry, second good number of nodes by chance ok.

(Refer Slide Time: 10:27)



I am just trying to see whether it case. So now, it took 1, 2, 3, 4, 5, 6 nodes initially.

(Refer Slide Time: 10:32)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod  
e/code videos$ python rich-get-richer_recording.py  
Enter the value of n100  
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod  
e/code videos$ python rich-get-richer_recording.py  
Enter the value of n100  
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod  
e/code videos$ python rich-get-richer_recording.py  
Enter the value of n100  
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod  
e/code videos$
```

So, it always keeps changing you can fix it or you can have the good design. So, this is the initial graph and we are going to add the nodes to this graph.



Now for adding the nodes let us create a function let us create a function add nodes since, we are going to follow Barabasi Albert model, let me name that accordingly Barabasi ok. We should pass the graph there we should pass the value of n the total number of nodes, we should pass the value of m that is the number of edges and it should be written the graph that is what we are going to we will be displaying ok.

So, we have to create this function, now you see there are  $m_0$  nodes into the existing graph already ok. So, we need the value of  $m_0$  here right because  $m_0$  is the number of nodes already. So, I should better pass  $m_0$  over here and  $m_0$  over here and let me decide the value of m in the function itself. So, I am removing this and I am passing G, n and  $m_0$ . Let me decide the value of m here, I will write m is equal to  $m_0$  minus 1 ok. So, we have  $m_0$  nodes and we have to add n minus  $m_0$  nodes more.

So, I will start a loop for every node for i in range  $m_0 + 1$  is the next node that is going to be added and we have to stop at n. So, since it is a range function, I will put n plus 1 because, it excludes last value. So, this is a loop to add nodes one by one. So, we will add the first node G.add\_node, the node is going to be i now we are added the nodes now, we have to add the edges. If you remember from the previous video, we have to do some pre processing. So, let me see that and under pre processing before that let me also write add the edges one by one ok.

Now, what is the pre processing? We need to choose the nodes based on the probabilities, which are based on the degrees. So, we need to maintain the degrees. So, let me take dictionary degrees = nx.degree(G) this written as a dictionary of dictionary, where the key is the node and the value is the degree ok. Now we also need to maintain the probabilities, which are which will be based on the degrees. So, let me create dictionary and node probabilities inside probabilities no probabilities is equal to empty dictionary this is what we are going to create, now how do we create it?

As I told you the probabilities probability of node i is equal to the degree of i divided by the sum of the degree of all the nodes. So, we should get a start loop here to assign the values to this dictionary. So, I will write for each in G.nodes node probabilities for each is going to be degree the I am sorry the degrees are there in these dictionary degrees of each, we will give us the degree of that node and that should be divided by the sum of all the degrees. Now where are all the degrees? degrees.values, if you write degrees or



values you get a list of all the degrees, we are going to sum them up and since we are dividing let me do the (Refer Time: 14:59) casting float ok, we should work.

So, we have now created the dictionary node probabilities ok if you remember from the previous video. Now, the next step is to create list that is cumulative node probabilities since, we want then, to be ordered I got create list of list and not the dictionary. Let me show, you how will do that node probabilities cumulative is equal to a list ok. So, we have to take the probabilities from these dictionary node probabilities one by one and based on that we will compute the cumulative probabilities. So, at this point let me start loop, I will write for n comma p in node probabilities.items.

Now, what do we get here? Node probabilities is a dictionary.items will give us a list of tuples and tuple we will have 2 things n , p that is the node comma the probability. So, we are caption that in n, p here and we have to store the cumulative values in the in the list of lists that is node probabilities cumulative here. So, let me initialize the probability to be initial probability to be 0 you will understand as I go on, I am plating a list temporary list, let me it temp that we will appending that we will appending to this list ok..

So initially, this list has to have node comma the cumulative probability. So, the cumulative probability for the first node is going to be basically, 0 plus it is own probability, if you remember from the previous example, we took in the previous video. So, the cumulative probability for the first node is going to be it is probability itself. So, we have going to add 0 it, and the cumulative probability for the second node is going to be the cumulative probability for the previous node plus the probability for the current node. So, we need to maintain this variable previous.

Now, this is going to be the list for the first node and we will append it to the node probabilities cumulative.append(temp) right..

(Refer Slide Time: 17:44)

```
42     node_probabilities_cum.append(temp)
43     prev = prev + p
44
45     new_edges = []
46     num_edges_added = 0
47     target_nodes = []
48
49     while(num_edges_added < m):
50         prev_cum = 0
51         r = random.random()
52         k = 0
53         while(not(r > prev_cum and r <= node_probabilities_cum[k][1]
54             prev_cum = node_probabilities_cum[k][1]
55             k += 1
56
57         target_node = node_probabilities_cum[k][0]
58         if target_node in target_nodes:
59             continue
60         else:
61             target_nodes.append(target_node)
62
```

And the previous should now be updated. So, I will write previous is equal to previous plus plus p, because p was the probability for the current node that p appended. In order to add the edges, we will now create another loop that we make use of this cumulative probabilities list. One thing that we have to take care of is an edge that has already been added should not be added again ok.

So, we need to keep a track of the number of edges added that is one thing, second thing is we also have to keep track of the edges that I have been added. So, we need to maintain a list of the edges that we have added for a given node so, that we do not repeat the edges. So, in case and if you do not keep a track of that that is if you do not keep a track of the edges that we have already added, we might end up adding the same edge again. So, in that case it might happen that some nodes are adding lesser edges than n ok.

So, let me create a list new edges is empty; here, we will store the new edges and number of edges added number of edges added for this particular node right initialise to 0. And in order to check whether the same existing node is not going to be connected again, I am going to create list target nodes initialise to 0, I will show you how these list are going to be add going to be used.

Now, we have to add the m edges to the new node. So, I will start loop here why? Number of edges added is less than m what do we do? We have to chose an existing node based on it is probability for that we need to random number. So, I will take r is

equal to `random.uniform`, which is going to give me a random value between 0 and 1 you can also use `random.random`, you can use that this one I think just check, what is the difference between these 2 `random.random` gives value between 0 and 1 right ok.

So now so, if you remember the only difference between `random` and `uniform` is that in `random` you do not need to pass any parameter and it gives you a float value between 0 and 1 and `uniform`. You can pass the parameters `a` comma `b` where the floating point value that the function will return will be between `a` and `b`. So, if you want a random floating point between 0 and 1, you can use `uniform` with parameter 0 and 1 that was about the probabilistic of these functions.

Now, we have to choose an existing node based on this `r`. So, if you remember there is going to be your window which is cumulative probability list will provide us. So, we have to check the first node's cumulative probability, if `r` is less than or equal to cumulative probability of the first node, we will stop there otherwise; we will go to the next node, if `r` is less than or equal to the cumulative probability of the second node we will consider that else we will go on. So, let me start loop here why not `r` is greater than we need this starting point as well. So, let me let me take a variable here previous cumulative is equal to 0. So, this is just create the windows we need to keep track of the previous node's cumulative probability as well in order to find the window.

So, I will write while `r` is greater than previous cumulative, and `r` is less than or equal to node probabilities cumulative of the first node ok. So, we need keep a track of the first as well `k` is equal to 0, first node `k`. And if you remember node probabilities cumulative is a list of list, where every member list has first element as the node and the second element as the cumulative probability. So, we will write I am sorry one here right. So, while this does not happen that `r` is not into that window, we will keep going on with the next element.

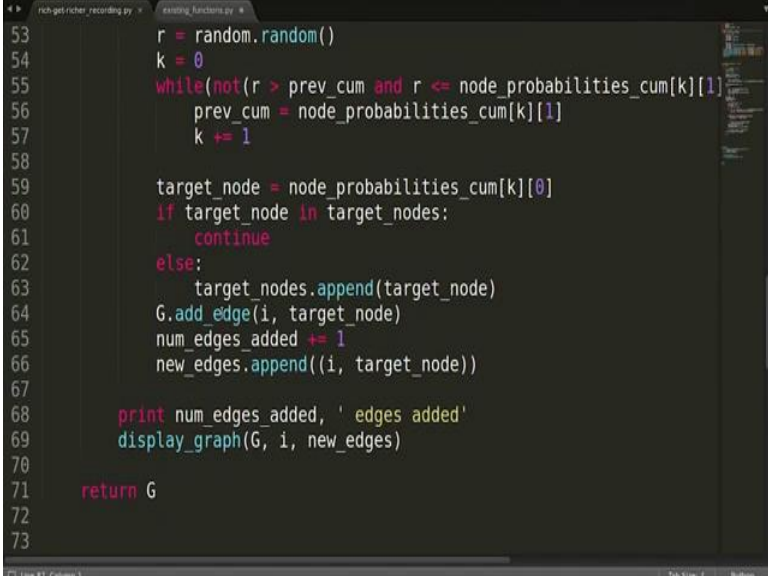
So, in our process previous cumulative is going to be updated with the cumulative probability of the current node and `k` will be implemented ok. Now whenever it finds a node, whose cumulative probability is more than this random number it will come out of the loop and when it comes out of the loop, it has found the node that is the target node. So, target node we are taking a variable target node, sorry target node is equal to now node cumulative probabilities is a list of lists and every list has first

element as a node and second implement as cumulative probability. Now we need the element here the node here. So, we will write k and here 0. So, we will get the first element that is the node that is the node.

Now, we have to check whether this node has already been connected to the new node or not? So, for that wait there is a spelling mistake target write e. So, we have already maintain this target nodes list, which will contain the all the target nodes. So, if though new node which we have chosen is already in target nodes, we will continue finding another target node if the if the target node is not there in this list, we will added there ok. So, we will write it is target node in this list target nodes, what will do? We will continue ok; we will find another target node otherwise this target node is not there in target nodes list. So, we will appended there target nodes.append this new node target node.

So basically, if the node that we have chosen is already in the list of nodes that is the target nodes. We will continue means we will go back here, and m is not implemented.

(Refer Slide Time: 25:24)



```
53 r = random.random()
54 k = 0
55 while(not(r > prev_cum and r <= node_probabilities_cum[k][1])
56       prev_cum = node_probabilities_cum[k][1]
57       k += 1
58
59 target_node = node_probabilities_cum[k][0]
60 if target_node in target_nodes:
61     continue
62 else:
63     target_nodes.append(target_node)
64     G.add_edge(i, target_node)
65     num_edges_added += 1
66     new_edges.append((i, target_node))
67
68 print num_edges_added, ' edges added'
69 display_graph(G, i, new_edges)
70
71 return G
72
73
```

So, we will repeat the whole process up to this point and in case the target node is not there in this list, we will do the rest of the processing that is connecting to this node by using the function G.add\_edge. So, we are going to add an edge from r node that is i to the target node right. So, the number of edges added will be implemented number of edges added plus equal to 1. And we also need to keep track of the new edges added

because, we are going to change the colour. So, we early maintain a list new is large sorry new edges new edges.append, we will add this edge and the edges basically i comma private node ok..

So, this is the new edge that is added. So, we can also keep track of the number of edges added. So, that we can check whether at every step we are adding same number of edges or not. Let us check print number of edges added. So, we will just keep a track of the number of edges added and after that we can display. So, this is one iteration over that is 1 node added with m edges. So, at this point you would like to display the graph take parameters, 3 parameters G comma the new node added is i the new edges added is where you keeping the this is new edges all right.

So, this should return the graph. So, I will return G. So, I think this function is done and we can start executing it ok.

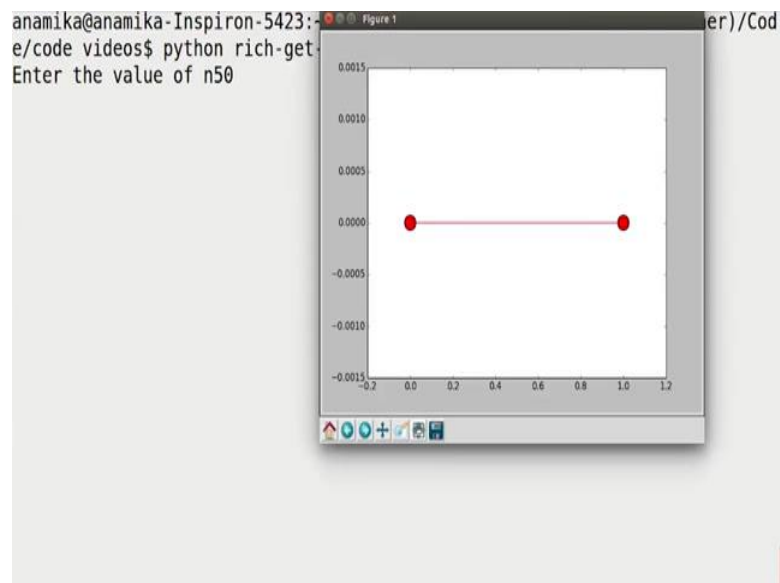
(Refer Slide Time: 27:17)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n50
```

So, this should return the graph. So, I will return G. So, I think this function is done and we can start executing it ok.

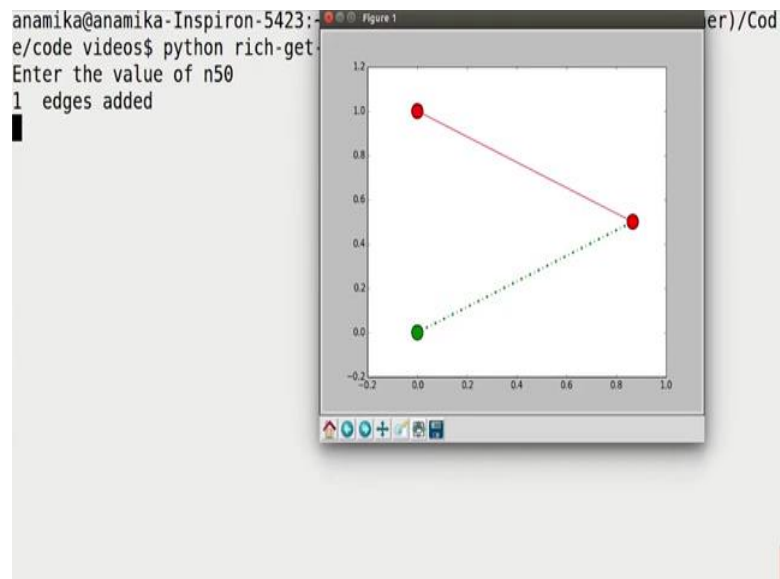
So, that is check it enter the value of n let me enter 50.

(Refer Slide Time: 27:23)



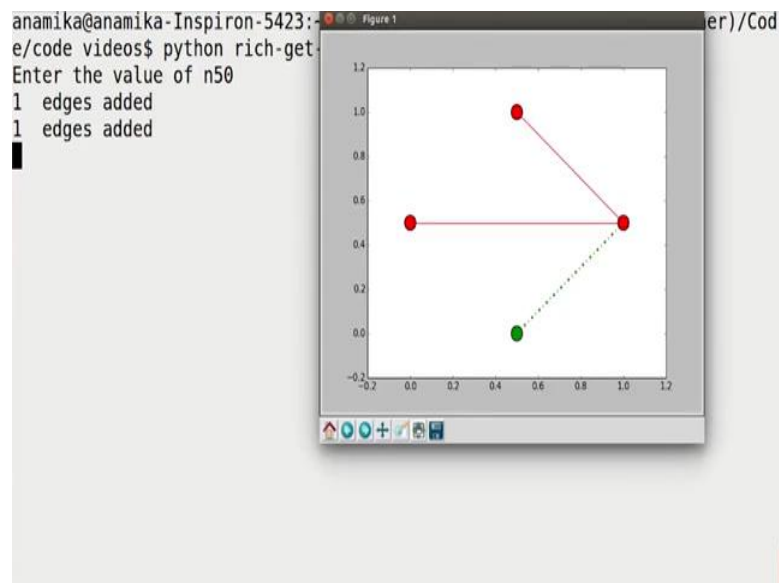
So this is the initial graph, it has chosen  $m_0$  to be true.

(Refer Slide Time: 27:32)



I will close this graph and see this is what I get in the first iteration, the new node is getting attach to one existing node. So, green is a new node and this dotted line indicates the new edges added.

(Refer Slide Time: 27:45)



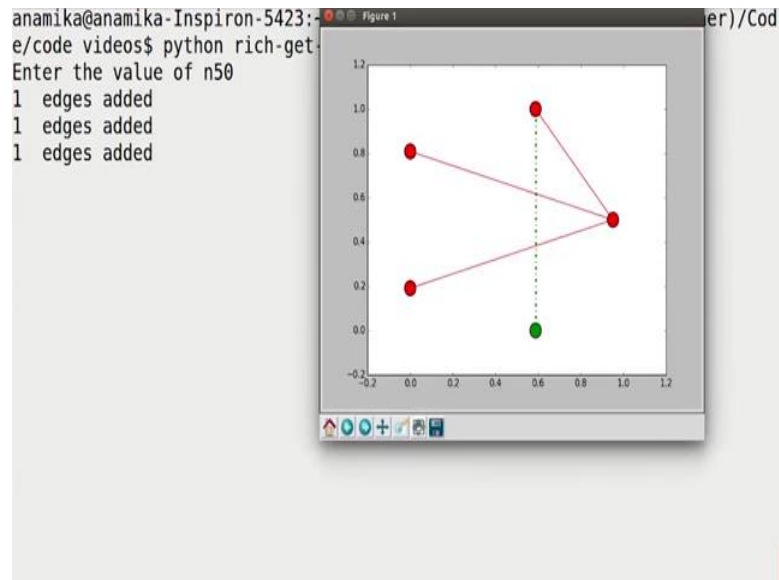
So, I am closing it and I see the second iteration the next new node is getting attach to the one of the existing edges.

So here, you can see there are 3 existing nodes, which are red and 2 of the nodes have degree 1 and 0 of the nodes has degree 2. So, the node which has degree 2 had more probability of getting attach to the new node, which is precisely what is happening having said that you cannot always be showing that always node, which has high degree will always big connecting it just a probabilistic phenomena right because, the preferential attachment is probabilistic mechanism..

The new node is free to connect to any node in the network whether it has high degree or it has just single edge. However, if new node has a choice between degree 2 and a degree 4 node then degree 4 node will be twice as likely to be connecting to the new node as compare to the node will with degree 2. So, that is what happens.

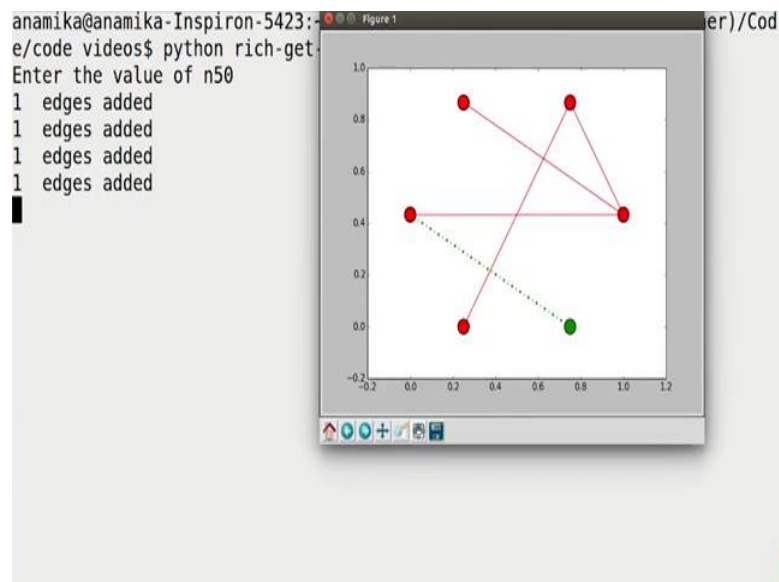


(Refer Slide Time: 28:54)



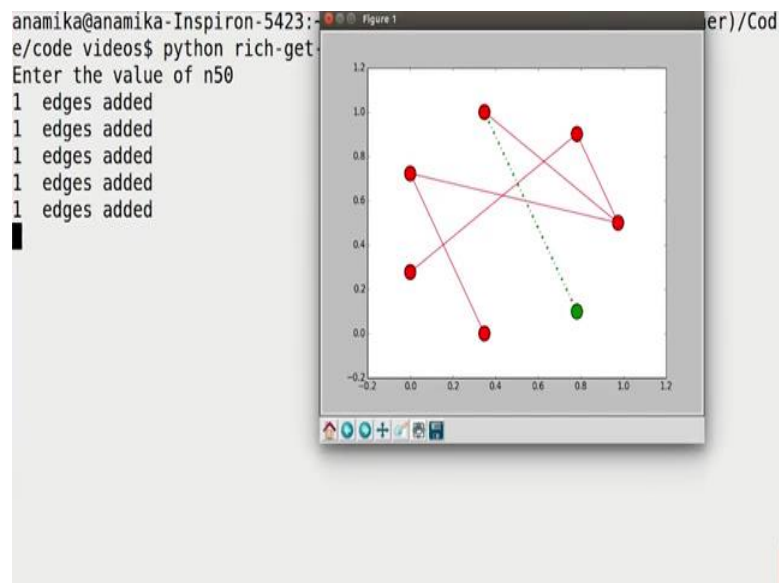
Now let us take the next iteration ok. So, the new node is getting attach to a node, which has degree 1. So, by chance it got connect to a node with less degree because, this is probabilistic. Let us check the next one.

(Refer Slide Time: 29:09)



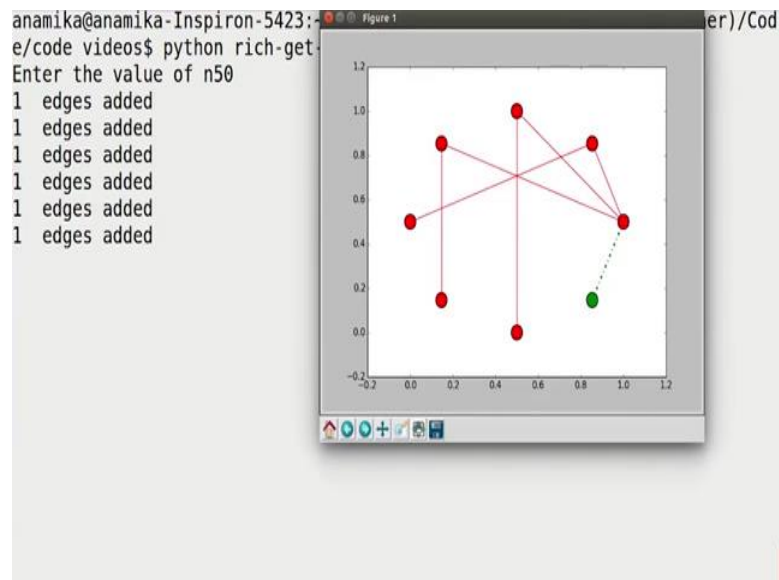
So, again the node is getting attach to a node with degree 1 because, there are few nodes.

(Refer Slide Time: 29:14)



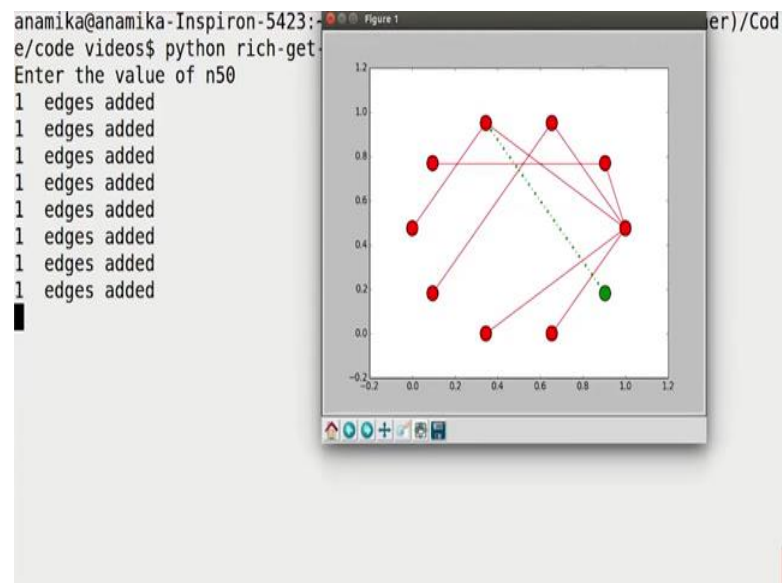
Now again the node is getting attach two by chance a node with less degree. Let us see the next one.

(Refer Slide Time: 29:20)



So now, this node has chosen an existing node high degree. Let us check the next one. Again, it has to choose the node with high degree.

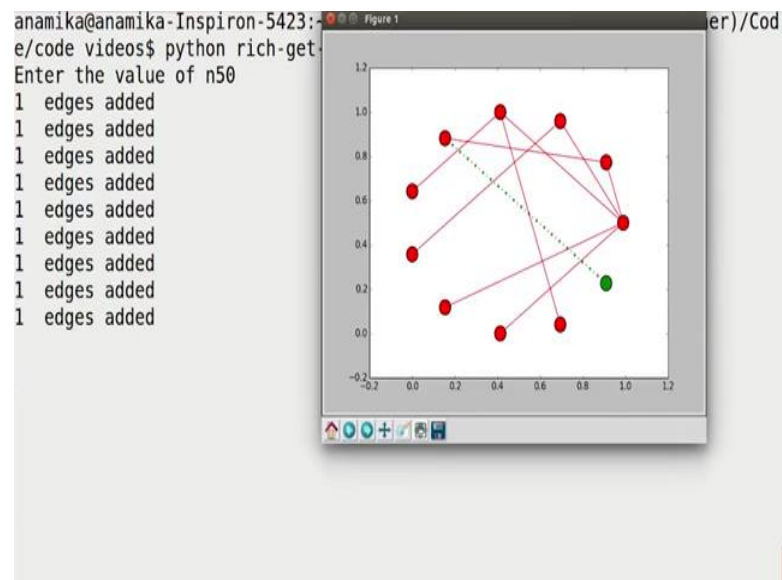
(Refer Slide Time: 29:30)



Now it has chosen our node with degree 2.

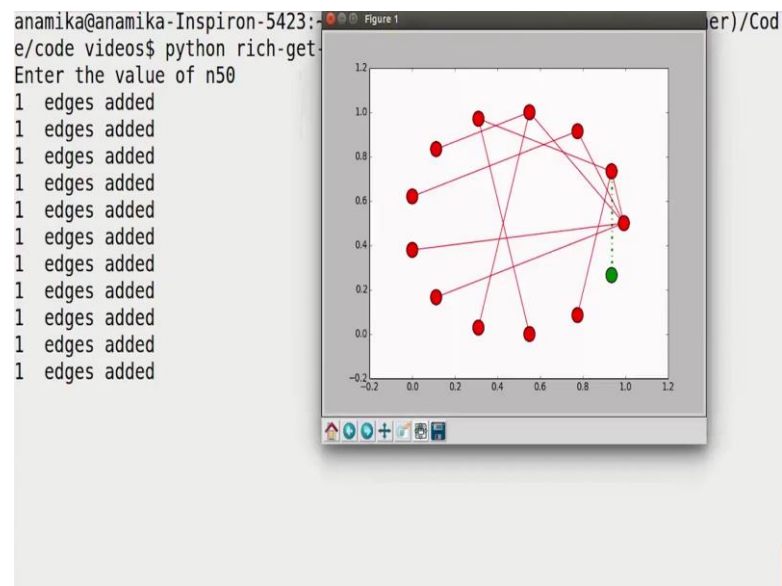
.

(Refer Slide Time: 29:34)



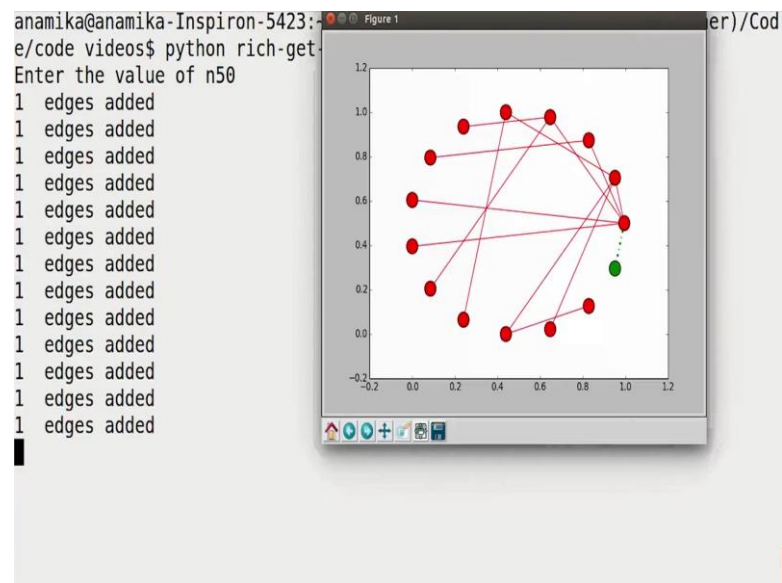
So, that keeps happening.

(Refer Slide Time: 29:36)



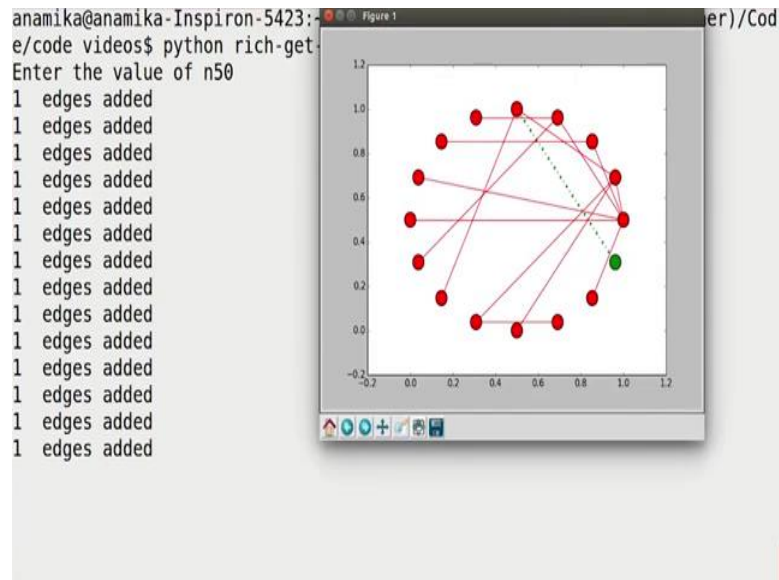
But overall, the nodes which have high degree will be connecting more to the new nodes.

(Refer Slide Time: 29:42)



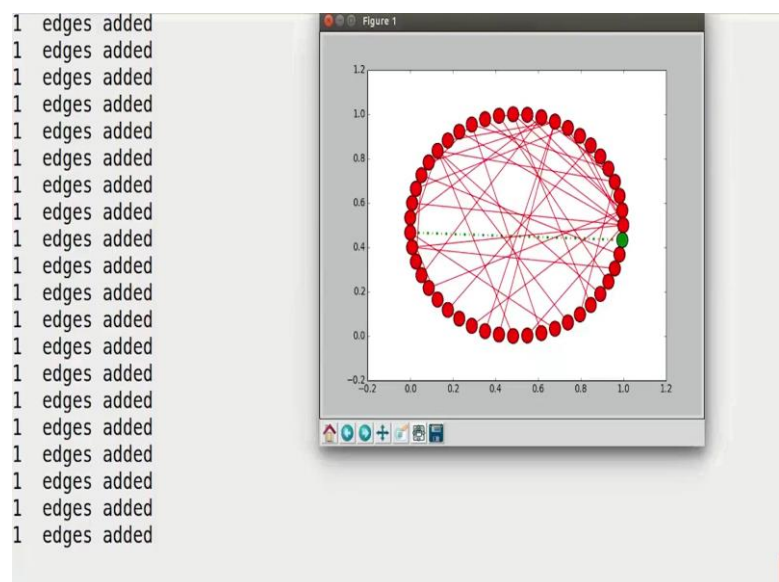
So, you can see here again happening there. So, it is again getting the connected to the node with high degree.

(Refer Slide Time: 29:47)



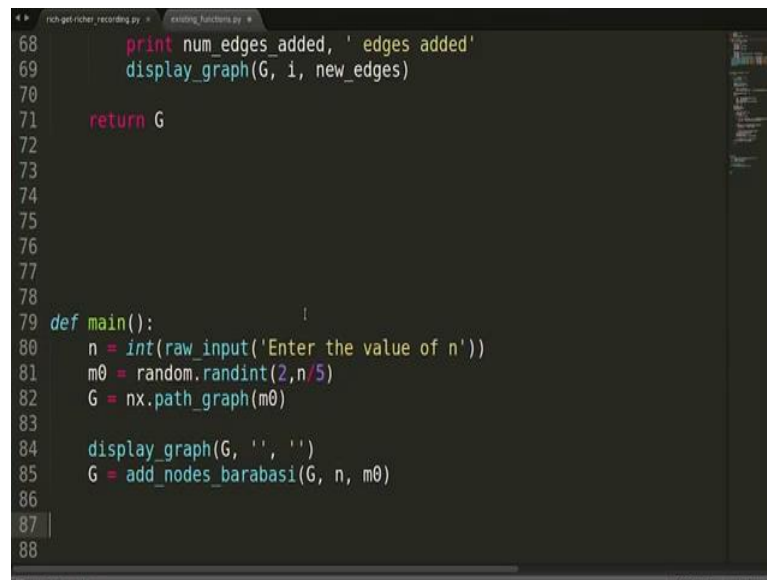
So, at each times and the network will not be symmetrical there will be more edges towards one towards some bunch of nodes, if that is not happening this network would have been symmetrical which is not so. So, I will have to keep closing this symbol 50 times. So, let me quickly do that you can keep observe with behavior ok. I should have taken the smaller network anyway.

(Refer Slide Time: 30:25)



So, this is the kind of structure that you are getting. So, a code is working fine alright.

(Refer Slide Time: 30:35)



```
68     print num_edges_added, ' edges added'
69     display_graph(G, i, new_edges)
70
71     return G
72
73
74
75
76
77
78
79 def main():
80     n = int(raw_input('Enter the value of n'))
81     m0 = random.randint(2,n/5)
82     G = nx.path_graph(m0)
83
84     display_graph(G, '', '')
85     G = add_nodes_barabasi(G, n, m0)
86
87
88
```

Now, an important property of the network start follows Barabasi Albert model or preferential attachment or Rich-getting-richer phenomena is that the degree distribution follows power law ok. That means, that the number of nodes, which less get less degree are way to high number and the number of nodes with very high degree are very less in number. So, that is the sort of property that is networks follow.

Let us try to get the degree distribution for this network, we had already created this function for plotting the degree distribution in one of the previous videos, I am going to just copy paste from there and I will briefly explain that. So, I will copy paste it, I will quickly explain this ok.

(Refer Slide Time: 31:24)

```
75 all_degrees = nx.degree(G).values() #all the degrees
76 unique_degrees = list(set(all_degrees)) #unique degree
77 unique_degrees.sort()
78 count_of_degrees = []
79
80 for i in unique_degrees:
81     c = all_degrees.count(i)
82     count_of_degrees.append(c)
83
84 print unique_degrees
85 print count_of_degrees
86
87 plt.plot(unique_degrees, count_of_degrees, 'ro-')
88 # plt.loglog(unique_degrees, count_of_degrees, 'ro-')
89 plt.xlabel('Degrees')
90 plt.ylabel('Number of nodes')
91 plt.title('Degree Distribution')
92 plt.show()
93
94 def main():
95     n = int(raw_input('Enter the value of n'))
```

So, what you are doing here is since, you want to plot the degrees the distribution of the degrees, we need to get all the degrees. So, we are using the function `nx.degree(G)`, which returns a dictionary we are taking all the values from that dictionary that is the all the degrees, we are store in this list.

Now, out of that list we are maintaining list of unique degrees because, what we have to check we have to check the number of nodes having a particular degree. So, we should know what the unique degrees are basically, the all possible degrees that the nodes have in that network. So, that is what will be kept in this list and then we are sorting this list because, we have to plot them. And we will keep a track of the number of nodes with degree 1, degree 2, degree 3 and so on, where the x axis will have all possible degrees in sorted order..

So, we have to maintain a count of degrees for all possible degree in the network. So, we have created this, list now in order to populate this list we have started this loop for i unique degrees, we will take first element of unique degree. And we will check in all degrees how many elements that is how may nodes have that particular degree; we will make use of a function count. So, we writing `all_degrees.count(i)`, this will return as the number of nodes having the degree(i) and we are appending that to this list count of degrees and then we are simply plotting them.



So, x axis will have unique degrees, all possible degrees sorted, and y axis will have the count of degrees that is count of nodes having that particular degree. I am get displaying them in their color and this is for labeling title that is what is the function about ok.

(Refer Slide Time: 33:29)

```
rich-get-richer_recording.py  existing_functions.py
85     print count_of_degrees
86
87     plt.plot(unique_degrees, count_of_degrees, 'ro-')
88     # plt.loglog(unique_degrees, count_of_degrees, 'ro-')
89     plt.xlabel('Degrees')
90     plt.ylabel('Number of nodes')
91     plt.title('Degree Distribution')
92     plt.show()
93
94 def main():
95     n = int(raw_input('Enter the value of n'))
96     m0 = random.randint(2,n/5)
97     G = nx.path_graph(m0)
98
99     # display_graph(G, '', '')
100    G = add_nodes_barabasi(G, n, m0)
101    raw_input()
102    plot_deg_dist(G)
103
104
105 main()
```

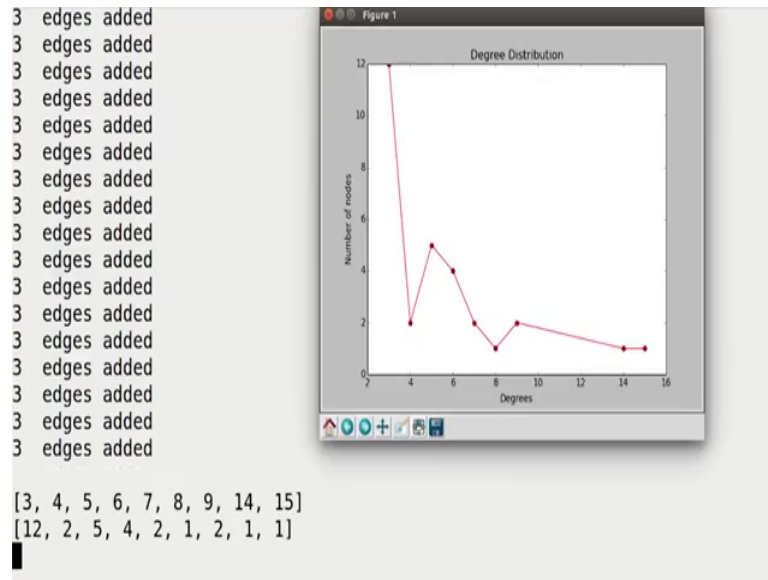
Let us call this function here i am sorry here in the end in name we are calling this function plot degree distribution G ok. So, let me let me ask the user to plus enter after getting all the plots after getting all the graphs.

(Refer Slide Time: 33:54)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich_get_richer)/Code/videos$ python rich-get-richer_recording.py
Enter the value of n
```

Let us check the functionality I will get. So, 30 this is initial graph, and these are the nodes getting added I have to press this 30 times and after that we should get the distribution get everything works fine ok.

(Refer Slide Time: 34:16)



I press enter now this is the distribution that we are getting. So, this is sort power law in the sense that the nodes with less degree are very high number and the nodes with very high degree are very less in number. So, power law is basically the sort of plot since a number of nodes is less. We are not getting it very nicely calling power law, but nevertheless it is following the that law.

If you take good number of nodes say some 100 or say 1000, you will get a nice distribution you can also check that press check that. So, let us check let us execute this code for some higher value of n. So that, we can see the distribution nicely so in order to that I need not press the I need not close the graph multiple times, I am going to remove this displaying of the graph from all the places that is all that is that is only we will calling this display graph function. So, I have commented that.

(Refer Slide Time: 35:27)

```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Code/code videos$ python rich-get-richer_recording.py
Enter the value of n1000
158 edges added
158 edges added
158 edges added
158 edges added
158 edges added
158 edges added
158 edges added
```

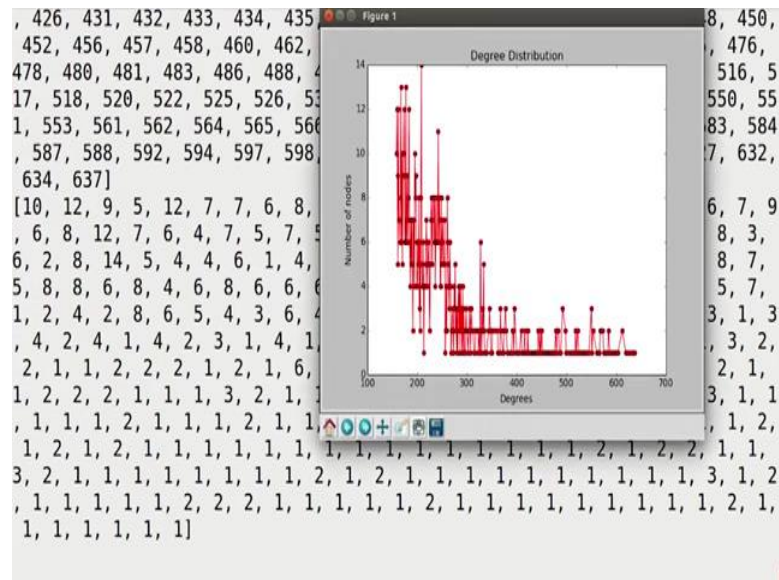
And let us call this with a high value of n say 1000.

(Refer Slide Time: 35:33)

[illegible]

So, 158 edges are getting added at every movement I think it is a very high number, I think, I should fix it on a smaller number there this is going on and on because, it has to go on for 1000 times. So, it is going on and on it is taking some chain let us see it is just finished, I am sorry.

(Refer Slide Time: 35:53)



So, I will press enter and I got the distribution. So, this is fall in solve power low as you can see right if you take a log plot of this, we will get this straight line. So, that is the whole idea about this.

So, as I told you the networks which follow preferential attachment follow power law as well and that is what we have observe there is well. So, that was the whole idea behind Rich-getting-richer phenomena. Here, we were adding the edges based on the degrees right, more degree more probability of getting connected. On the other hand, if you add the edges randomly what should happen? in that case, you will not get this power law distribution because, there is no preference to words getting a attach to a particular node, we are just randomly getting connected to the existing nodes.

So, in that case you will not the getting power law. In fact, you will getting normal distribution. So, in the next video we are going to implement random networks and we will see the kind of distribution that we obtain.