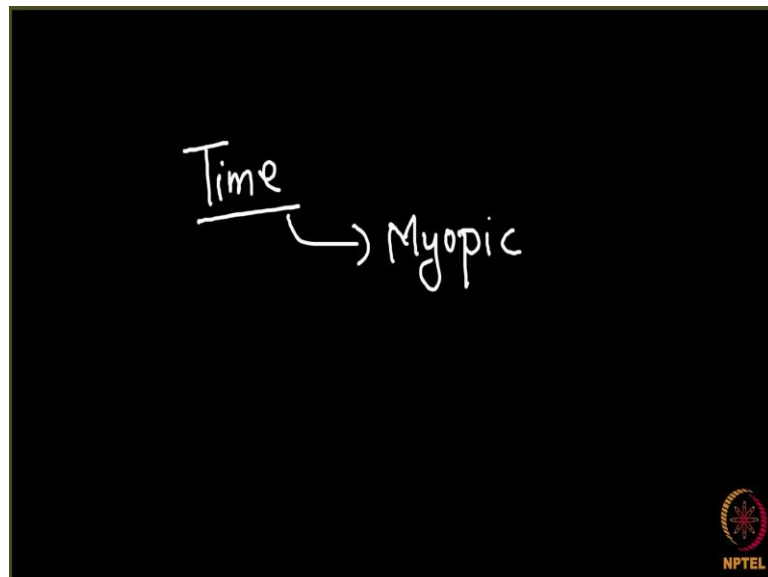


Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

How to go Viral on Web
Lecture - 158
Time Taken by Myopic Search

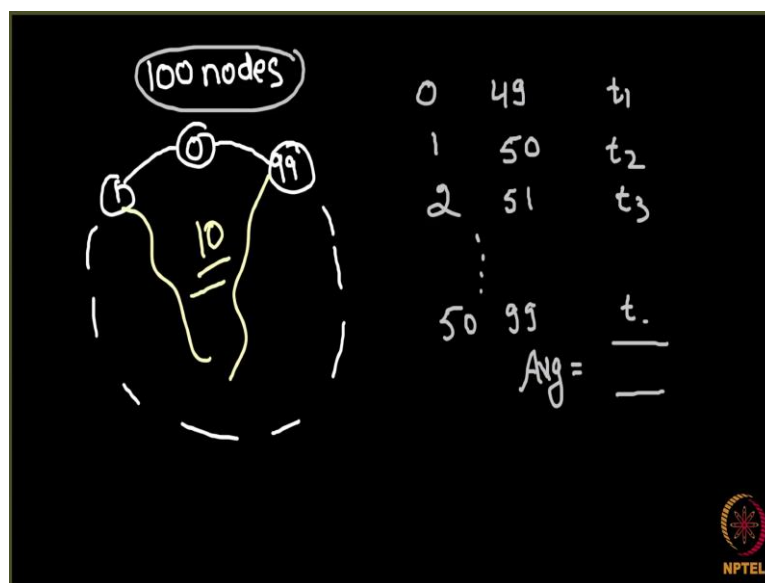
In the previous programming screencast, we compare the myopic search with optimal search.

(Refer Slide Time: 00:13)



Now next what we are interested in looking at is the Time Taken by the Myopic Search. So, if I perform my myopic search on a network of let us say 100 nodes what is the time taken? On a network of 200 nodes what is the time taken so on and so forth; is this time linear in the number of nodes or logarithmic in the number of nodes or what. So, for doing this what we are going to do is first of all let us take a network having 100 nodes.

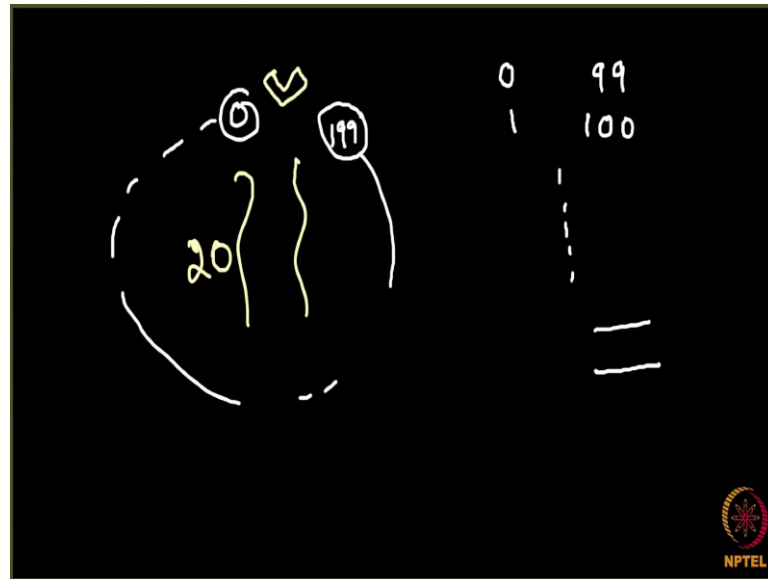
(Refer Slide Time: 00:41)



So, we take a network having 100 nodes as previous. So, we have nodes here 0 1 so on and so forth and here is 99 and we are going to do the same procedure what we did before. We will take the diametrically opposite points and look at how much time does myopic search take to work on this diametrically opposite points. So, we will be taking the point let us say 0 comma 49 and then we will see the time taken by the myopic search. Then for 1 comma 50 what is the time taken, for 2 comma 51 what is the time taken so on and so forth till 50 comma 99 what is the time taken.

Now, what will do next is we will take the average across all these times, that will give us on an average if we take 2 diametrically opposite points on this network; on an average what is the time taken by the myopic search. So, we calculate the average here. Now, this we have done for a network having 100 nodes, next what we will do we will repeat the same procedure on a network having 200 nodes. So, we will be taking a network having 200 nodes.

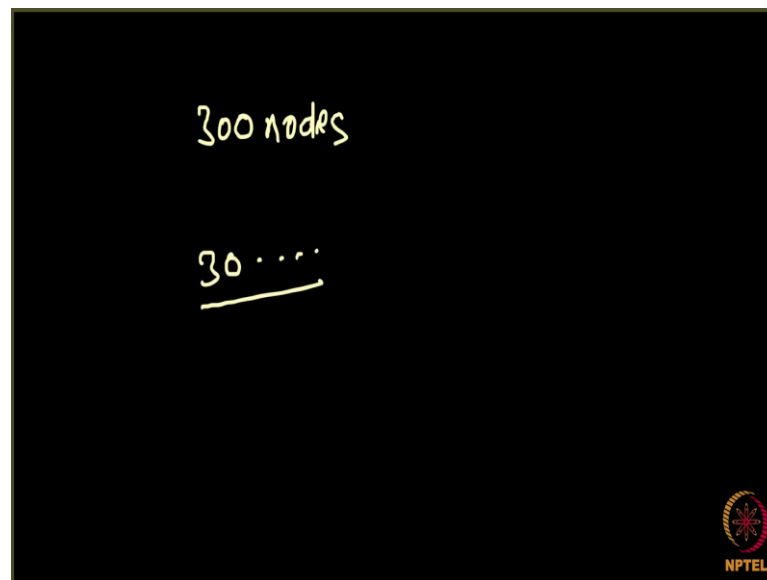
(Refer Slide Time: 01:57)



So, it will go from 0 to 199 and then we will again take the diametrically opposite points which here will be I will say 0 comma 99 and then 1 comma 100 so on and so forth. And, again we will take the average here and that will give us the time taken by myopic search on this network ok, while doing this we also take care of a small thing. So, when we have a ring a network over here right as shown you here the homophily based links and we know that there are going to be some weak ties as well.

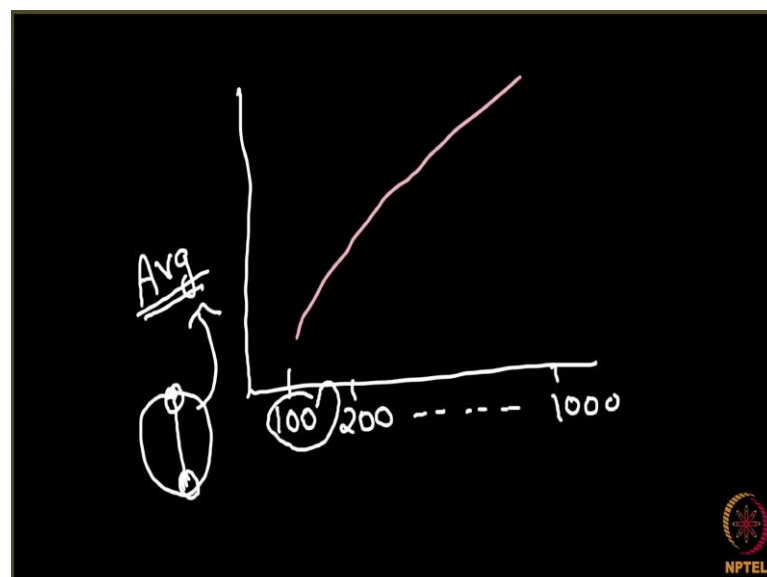
So, previously in this network we have taken 10 weak ties right, we have taken 10 weak ties. Now, if in this network is big right. So, what we are we will be going to assume in a programming screen cast is that the number of weak tie is 10 percent of the number of nodes in the network. So, for here we have 10 weak ties, here we will be making 20 weak ties in the next network which will be having 300 nodes.

(Refer Slide Time: 03:05)



We will be having 30 weak ties so on and so forth and then we will do this process. And, we will find the average over here and at the end what we will be plotting is, on the x axis we will be plotting the size of the network that is the number of nodes in the network 100 200 so on and so forth, let us say up to 1000.

(Refer Slide Time: 03:17)

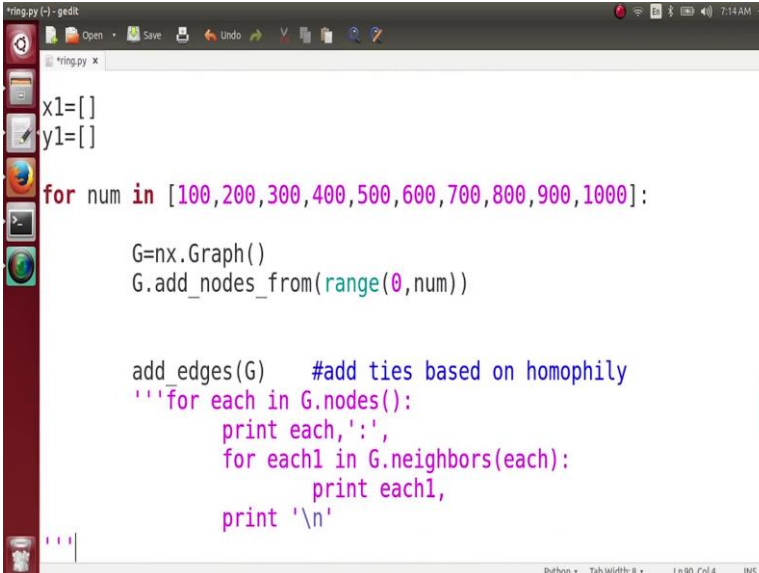


And on the y axis we will be plotting the average we have calculated. So, what is this average? It tells us that for one particular network, if we take the diametrically opposite points on this network. So, if we take this diametrically opposite points on this network

what is the average time taken by a myopic search and we will look at how does this plot look like is it linear, is it logarithmic or what. So, next what we want to do is now, we want to find out the time complexity of myopic search time which are myopic search takes for different size of networks.

So, till now we have considered only 1 network which was having 100 nodes. Now, we are going to do an experiment, we are going to build small world networks of different sizes 100 200 300 up to 1000 and we will do a myopic search there and we will see that how much time on an average does this take. So, at the end we want upload were on the x axis is the number of nodes and on the y axis is the average time taken by my myopic search. So, I make some arrays here.

(Refer Slide Time: 04:43)

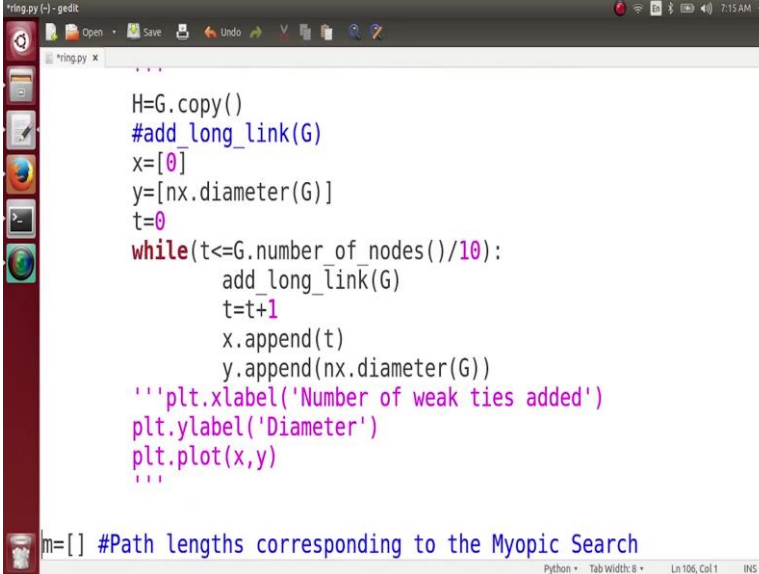


```
ring.py (-) - gedit
ring.py x
x1=[]
y1=[]
for num in [100,200,300,400,500,600,700,800,900,1000]:
    G=nx.Graph()
    G.add_nodes_from(range(0,num))

    add_edges(G) #add ties based on homophily
    '''for each in G.nodes():
        print each,':',
        for each1 in G.neighbors(each):
            print each1,
        print '\n'
```

So, I make an array x1 for the x axis which will be my number of nodes and an array y1 for the y axis which will be the time which my myopic search is checking. And what I have to do is now I have to create networks of different size and do this entire process for all these networks. So, I have to basically put all these code in a loop. So, what I am going to do is for num in and I am going to take a list here consisting of the values 100, 200, 300, 400, 500 6 up to 1000. And, then what I am going to do is the graph, I am going to create here is going to be from range 0 to num. So, first of all I will get a network on 100 nodes then 200 nodes so on and so forth and I will put everything inside this loop.

(Refer Slide Time: 05:49)

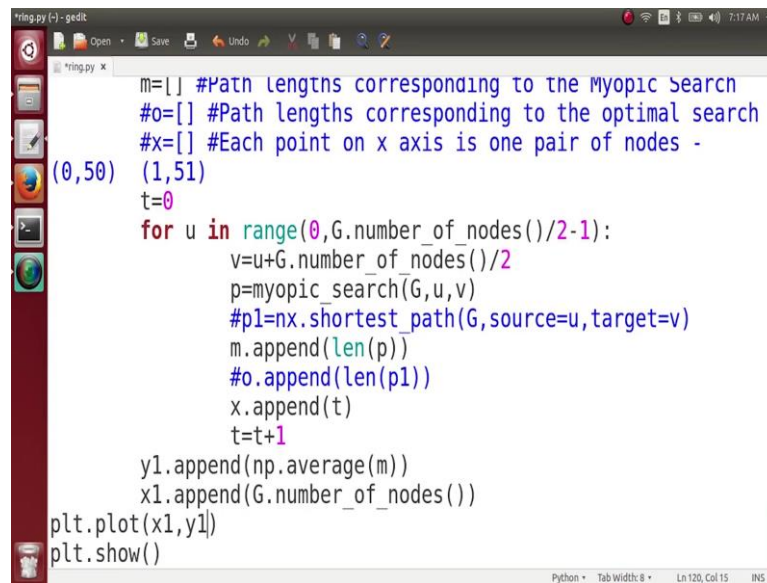


```
ring.py - gedit
ring.py x
H=G.copy()
#add_long_link(G)
x=[0]
y=[nx.diameter(G)]
t=0
while(t<=G.number_of_nodes()/10):
    add_long_link(G)
    t=t+1
    x.append(t)
    y.append(nx.diameter(G))
'''plt.xlabel('Number of weak ties added')
plt.ylabel('Diameter')
plt.plot(x,y)
'''
m=[] #Path lengths corresponding to the Myopic Search
```

And, now we are one thing to be noticed is t here t here it is telling us the number of weak ties which we have to add. So, the number of weak ties should change according to the number of nodes in the network. So, instead of 10 what I am going to do here is, I am going to make it here while $t \leq G.\text{number_of_nodes}/10$; that is I am going to have 10 percent of the edge is in the network to be weak ties.

Whatever are the number of a nodes in the network so, if there are 100 nodes so, the number of nodes so, if there are 100 nodes I will be having 10 weak ties; if there are 200 nodes I will be having 20 weak ties so on and so forth. And, then I am going to add these long range links and rest of the things remains the same.

(Refer Slide Time: 06:59)

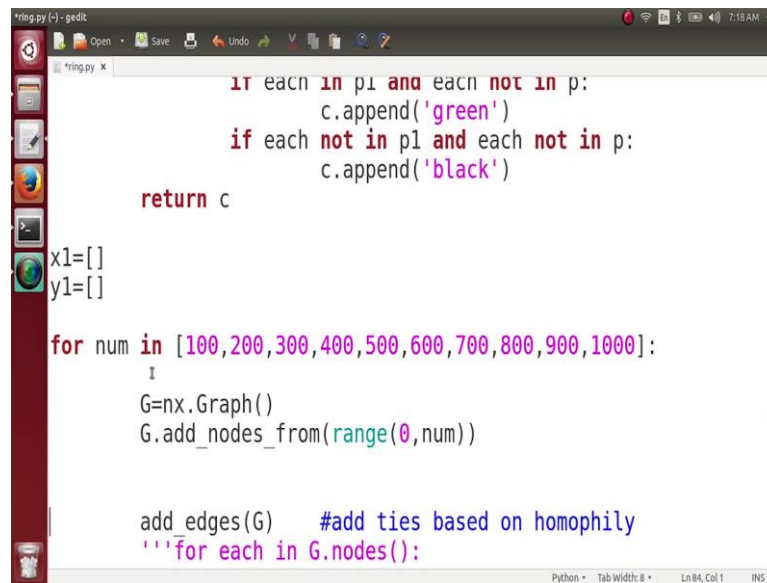


```
ring.py - gedit
ring.py
m=[] #Path lengths corresponding to the Myopic Search
#o=[] #Path lengths corresponding to the optimal search
#x=[] #Each point on x axis is one pair of nodes -
(0,50)
(1,51)
t=0
for u in range(0,G.number_of_nodes()/2-1):
    v=u+G.number_of_nodes()/2
    p=myopic_search(G,u,v)
    #p1=nx.shortest_path(G,source=u,target=v)
    m.append(len(p))
    #o.append(len(p1))
    x.append(t)
    t=t+1
y1.append(np.average(m))
x1.append(G.number_of_nodes())
plt.plot(x1,y1)
plt.show()
```

Now, mine this value u will range what is on 0 to 49, it will range from 0 to number_of_nodes/2 right. 0 to G.number_of_nodes/2 - 1 and v is going to be u plus G.number_of_nodes/2. And, then we are going to apply a myopic search here, find out the path we do not, no we are not doing an optimal search over here and then I am wrote append length of p and here is an array. So, so you can see that m is an array over here which holds the path length corresponding to the myopic search. So, you see what is happening over here, for some small world network here we are performing a myopic search. So, we have a small world network, we have done this myopic search.

And we do it for many pairs of nodes and at the end what we want to append to our y axis is nothing, but the average of everything. So, what is going to come in our y axis is I will come out of this loop, what we are going to append in our y axis is the average which is np.average. And this is numpy, numpy.average of m right and what is going to be coming across x axis is nothing, but the number of nodes G.number_of_nodes. And at the end what will be, what we will be plotting is x1 against y1 y; I hope that this code is clear. What we have done is I will quickly recap it.

(Refer Slide Time: 09:39)

A screenshot of a gedit text editor window titled 'ring.py (-) - gedit'. The editor contains Python code for generating a network. The code includes a function that takes two lists, p1 and p, and returns a list c. It uses 'if' and 'for' loops to iterate over elements in p1 and p, appending 'green' or 'black' to c based on certain conditions. Below this, there are two empty lists, x1 and y1. A 'for' loop iterates over a range of numbers from 100 to 1000. Inside this loop, a graph G is created using 'nx.Graph()', nodes are added from the range(0, num), and edges are added based on homophily. The code is color-coded with syntax highlighting.

```
if each in p1 and each not in p:
    c.append('green')
if each not in p1 and each not in p:
    c.append('black')

return c

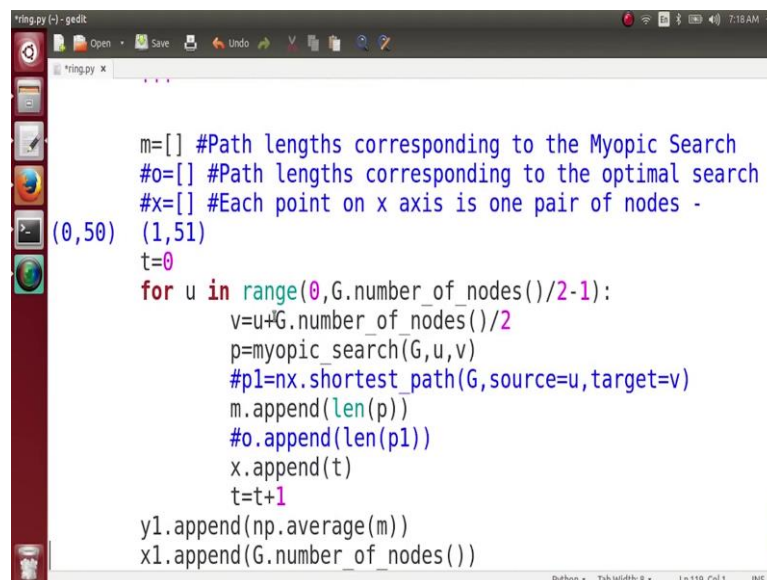
x1=[]
y1=[]

for num in [100,200,300,400,500,600,700,800,900,1000]:
    G=nx.Graph()
    G.add_nodes_from(range(0,num))

    add_edges(G) #add ties based on homophily
    '''for each in G.nodes():
```

So, here are my 2 arrays x1 and y1, where x1 is the different. So, x1 is nothing, but thing here this complete will be x1 which is the number of nodes in different networks. And, y1 will hold the average path length which myopic search takes to connect a diametrically opposite points and then we make the small world network.

(Refer Slide Time: 10:05)

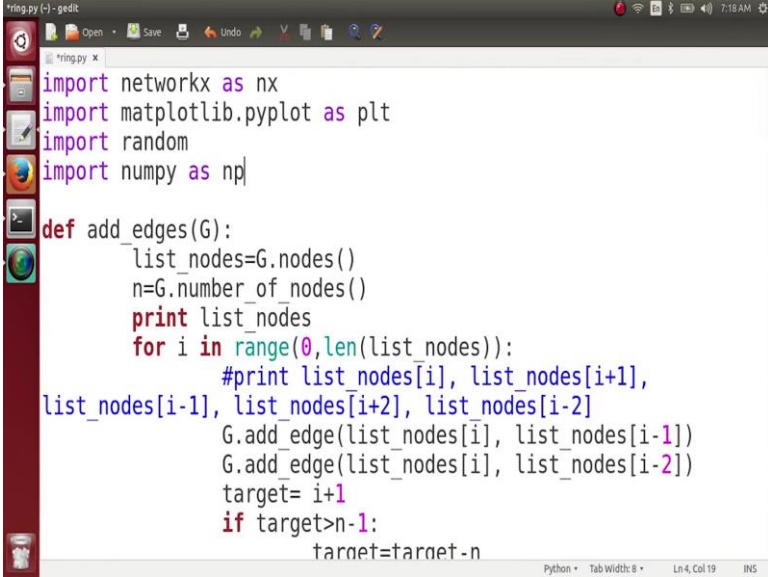
A screenshot of a gedit text editor window titled 'ring.py (-) - gedit'. The editor contains Python code for performing a myopic search and calculating path lengths. It initializes three empty lists: m (for path lengths of myopic search), o (for path lengths of optimal search), and x (for pairs of nodes). A 'for' loop iterates over a range of nodes from 0 to G.number_of_nodes()/2-1. For each node u, it calculates v as u + G.number_of_nodes()/2, performs a myopic search from u to v, and appends the length of the path to m. It also appends the length of the shortest path to o and the current node index to x. After the loop, it calculates the average of m and appends it to y1, and appends the total number of nodes to x1. The code is color-coded with syntax highlighting.

```
m=[] #Path lengths corresponding to the Myopic Search
#o=[] #Path lengths corresponding to the optimal search
#x=[] #Each point on x axis is one pair of nodes -
(0,50) (1,51)
t=0
for u in range(0,G.number_of_nodes()/2-1):
    v=u+G.number_of_nodes()/2
    p=myopic_search(G,u,v)
    #p1=nx.shortest_path(G,source=u,target=v)
    m.append(len(p))
    #o.append(len(p1))
    x.append(t)
    t=t+1
y1.append(np.average(m))
x1.append(G.number_of_nodes())
```

After making the small world network we take an array m be here and then we perform this myopic search for all the diametrically opposite points and keep a pending the path lengths in this array m. And at the end we take the average of this m and append it in y1

and this average of this m be append in y1 and x1 has nothing, but the number of nodes and we have using module numpy. So, that needs to be imported.

(Refer Slide Time: 10:35)

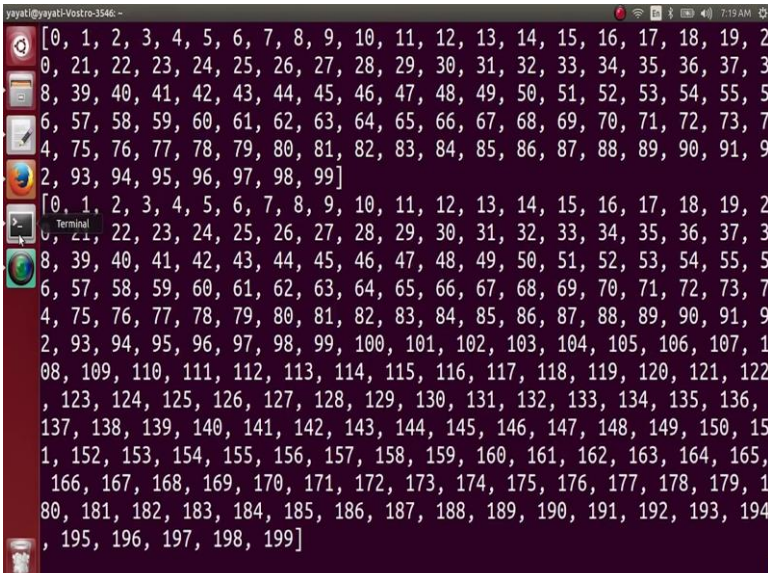


```
ring.py - gedit
import networkx as nx
import matplotlib.pyplot as plt
import random
import numpy as np

def add_edges(G):
    list_nodes=G.nodes()
    n=G.number_of_nodes()
    print list_nodes
    for i in range(0,len(list_nodes)):
        #print list_nodes[i], list_nodes[i+1],
        list_nodes[i-1], list_nodes[i+2], list_nodes[i-2]
        G.add_edge(list_nodes[i], list_nodes[i-1])
        G.add_edge(list_nodes[i], list_nodes[i-2])
        target= i+1
        if target>n-1:
            target=target-n
```

So, I import numpy as np means executes each.

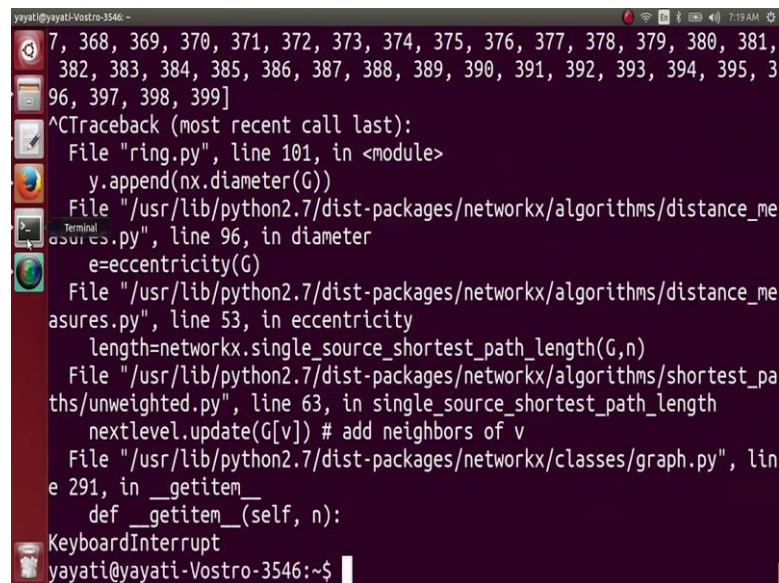
(Refer Slide Time: 10:41)



```
jyoti@jyoti-Vostro-3546:~$
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 3
8, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 5
6, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 7
4, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 9
2, 93, 94, 95, 96, 97, 98, 99]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 3
8, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 5
6, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 7
4, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 9
2, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 1
08, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122
, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 15
1, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 1
80, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194
, 195, 196, 197, 198, 199]
```

So, it will take some time.

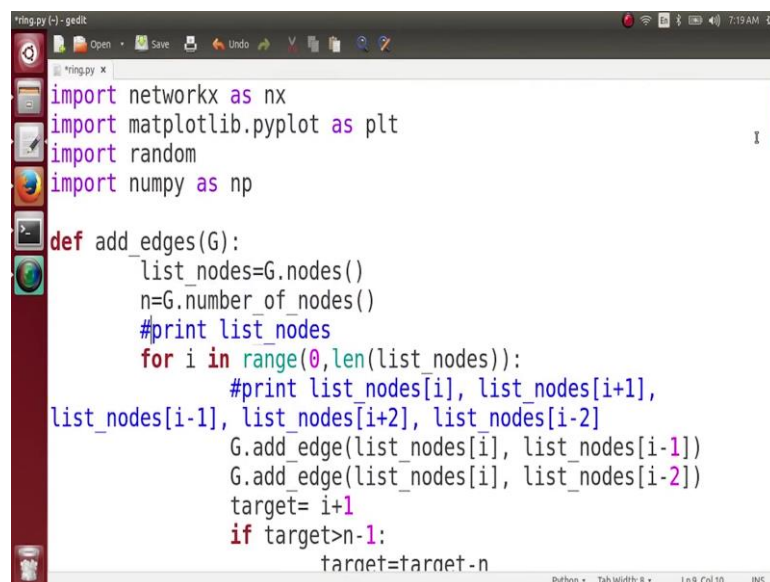
(Refer Slide Time: 10:49)

A terminal window with a dark background and light-colored text. It shows a list of numbers from 368 to 399, followed by a traceback error. The error starts with '^CTraceback (most recent call last):' and lists several files and line numbers, including 'ring.py', 'distance_measures.py', and 'graph.py'. The error ends with 'KeyboardInterrupt' and the prompt 'yayati@yayati-Vostro-3546:~\$'.

```
yayati@yayati-Vostro-3546:~$  
7, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,  
382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 3  
96, 397, 398, 399]  
^CTraceback (most recent call last):  
  File "ring.py", line 101, in <module>  
    y.append(nx.diameter(G))  
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/distance_me  
asures.py", line 96, in diameter  
    e=eccentricity(G)  
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/distance_me  
asures.py", line 53, in eccentricity  
    length=networkx.single_source_shortest_path_length(G,n)  
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/shortest_pa  
ths/unweighted.py", line 63, in single_source_shortest_path_length  
    nextlevel.update(G[v]) # add neighbors of v  
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", lin  
e 291, in __getitem__  
    def __getitem__(self, n):  
KeyboardInterrupt  
yayati@yayati-Vostro-3546:~$
```

For the sake of clarity where there is code is running correctly or not, but we are going to do is where is my first print statement; we will remove that print.

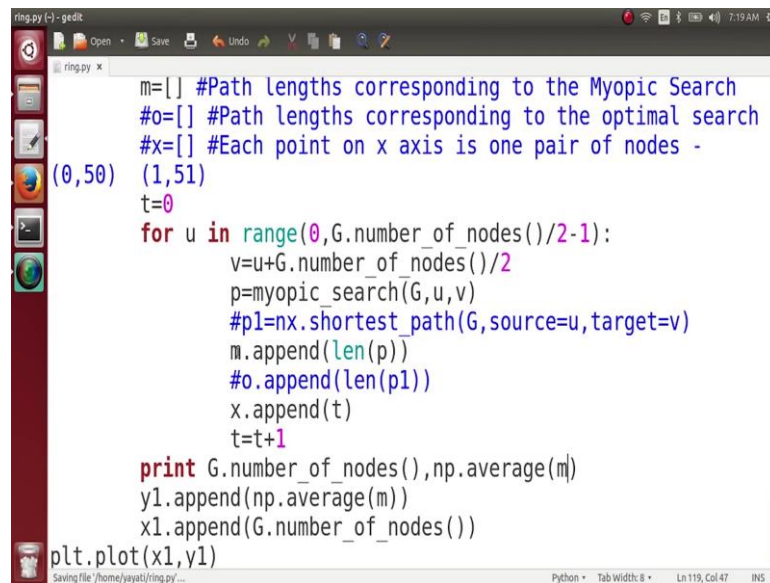
(Refer Slide Time: 11:07)

A screenshot of a code editor window titled 'ring.py - gedit'. The code is in Python and defines a function 'add_edges(G)'. It imports 'networkx as nx', 'matplotlib.pyplot as plt', 'random', and 'numpy as np'. The function 'add_edges(G)' gets the list of nodes and the number of nodes. It then iterates over the list of nodes, adding edges between nodes at indices i and i-1, and i and i-2. The code is as follows:

```
import networkx as nx  
import matplotlib.pyplot as plt  
import random  
import numpy as np  
  
def add_edges(G):  
    list_nodes=G.nodes()  
    n=G.number_of_nodes()  
    #print list_nodes  
    for i in range(0,len(list_nodes)):  
        #print list_nodes[i], list_nodes[i+1],  
list_nodes[i-1], list_nodes[i+2], list_nodes[i-2]  
        G.add_edge(list_nodes[i], list_nodes[i-1])  
        G.add_edge(list_nodes[i], list_nodes[i-2])  
        target= i+1  
        if target>n-1:  
            target=target-n
```

We do not print the list of nodes here, instead what we do here whatever we are appending we print here.

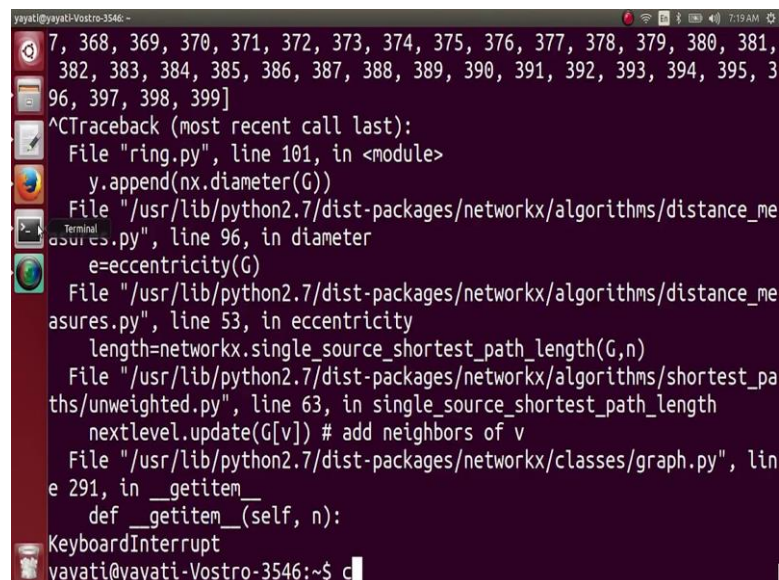
(Refer Slide Time: 11:15)



```
ring.py x
m=[] #Path lengths corresponding to the Myopic Search
#o=[] #Path lengths corresponding to the optimal search
#x=[] #Each point on x axis is one pair of nodes -
(0,50)
(1,51)
t=0
for u in range(0,G.number_of_nodes()/2-1):
    v=u+G.number_of_nodes()/2
    p=myopic_search(G,u,v)
    #p1=nx.shortest_path(G,source=u,target=v)
    m.append(len(p))
    #o.append(len(p1))
    x.append(t)
    t=t+1
print G.number_of_nodes(),np.average(m)
y1.append(np.average(m))
x1.append(G.number_of_nodes())
plt.plot(x1,y1)
```

So, we print here whatever we have appended in the x axis `G.number_of_nodes` and we print here `np.average(m)`.

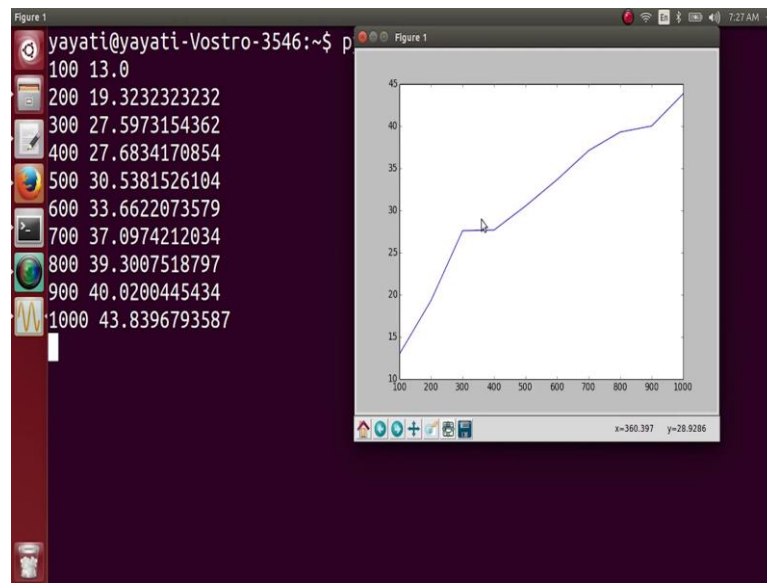
(Refer Slide Time: 11:29)



```
yayati@yayati-Vostro-3546:~$
7, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,
382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 3
96, 397, 398, 399]
^CTraceback (most recent call last):
  File "ring.py", line 101, in <module>
    y.append(nx.diameter(G))
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/distance_me
asures.py", line 96, in diameter
    e=eccentricity(G)
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/distance_me
asures.py", line 53, in eccentricity
    length=networkx.single_source_shortest_path_length(G,n)
  File "/usr/lib/python2.7/dist-packages/networkx/algorithms/shortest_pa
ths/unweighted.py", line 63, in single_source_shortest_path_length
    nextlevel.update(G[v]) # add neighbors of v
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", lin
e 291, in __getitem__
    def __getitem__(self, n):
KeyboardInterrupt
yayati@yayati-Vostro-3546:~$ c
```

So, now.

(Refer Slide Time: 11:33)



So, when I have a network was of a was consisting of 100 nodes it took 13 steps 200 nodes 19 steps so on and so forth. It will be time to run and let us see the final output and you can actually do this process for a greater number of nodes. So, when you are code instead of 100 200 300, you can take 100 150 200 250 to get more data points and have a better plot ok. So, here you see plot and what if you will plot it for more data points you see the plot more clear and you can actually observe that this is not a linear plot rather it is a lower plot. So, as you increase the number of nodes, the time which myopic search takes to execute increase is logarithmically not linearly.