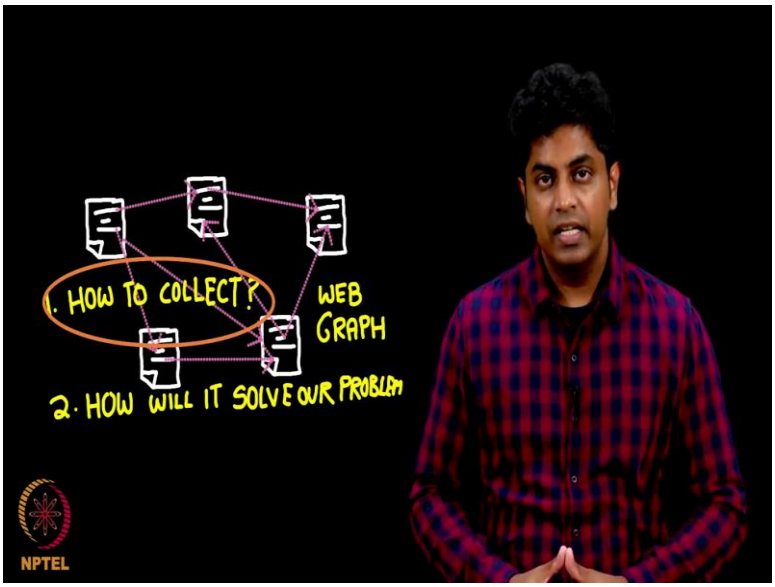
Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Link Analysis
Lecture - 76
Collecting the Web Graph

So, let us be organised. We are only asking these two questions. Let us answer them one at a time.

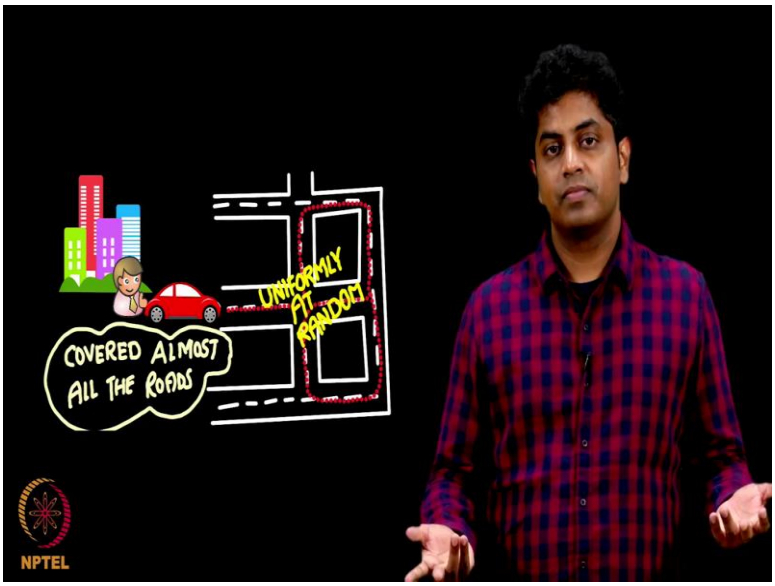
(Refer Slide Time: 00:15)



There is this question of web graph which will solve our problem; our problem of searching the web looks like unrelated you see. A graph of who points to whom which page points to what, has nothing to do with your search problem. So, first question is how we collect this web graph? I am saying this web graph will solve the problem. How will it solve the problem? Two things: firstly, how do you collect this web graph; secondly, how do you think this web graph will solve your problem.

Let us go step by step. So, first question. How do we collect this web graph? Here is a technique to collect the web graph. You should imagine a small situation.

(Refer Slide Time: 01:03)

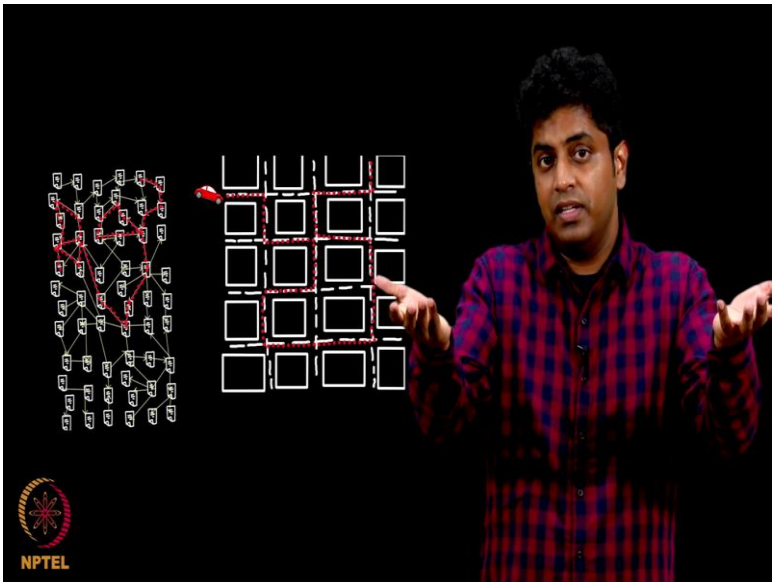


Imagine a city that you have never visited. And what I will do is I will put you to this new city; I will put you to this new city. Assume there is there are no vehicles in the city at all, you are the only one with a nice vehicle. I have given you a driver, a car road are empty; I repeat a new city you do not know of. Let us say you are not given a driver you drive the car, because if you are given a driver will know the localities. So, you are in a new city you do not know the localities, you are given the car; there is no traffic at all. What you do is you fix a camera on your car and just keep taking random turns here and there and keep going and going and going for the next one week. Assume you are given enough rest in between; you are given enough food in between; all you need to do is to simply take random turns and keep going.

Obviously you will not go a left and a right and a right and; obviously, you will not take a right and a right and a right and a right; you will come back to the same place, I said randomly. In every junction you toss a coin, you will take a decision whether you want to move towards your left or towards your right or towards straight. Or if there is a junction with 5 roads going from there, you roll a dice and you decide which side to go uniformly a track. You keep doing this for let us say, 1 week or more be even more may be 1 month. At the end of 1 month do you think there will be a road that you would not have explored?

May be yes, you would not have intuitively you feel you would not have. But do you think, you would have explored most of the roads in the city? Intuitively yes, you feel you would have if you take a car and keep going randomly wherever is an intersection, you break the tie by choosing a road uniformly at random and then move in that road and keep doing this for about a month's time in the city. You would have covered almost all roads right almost all roads you would have covered. Now what does this tell you? Does it tell you anything about the question that I just now asked?

(Refer Slide Time: 03:31)

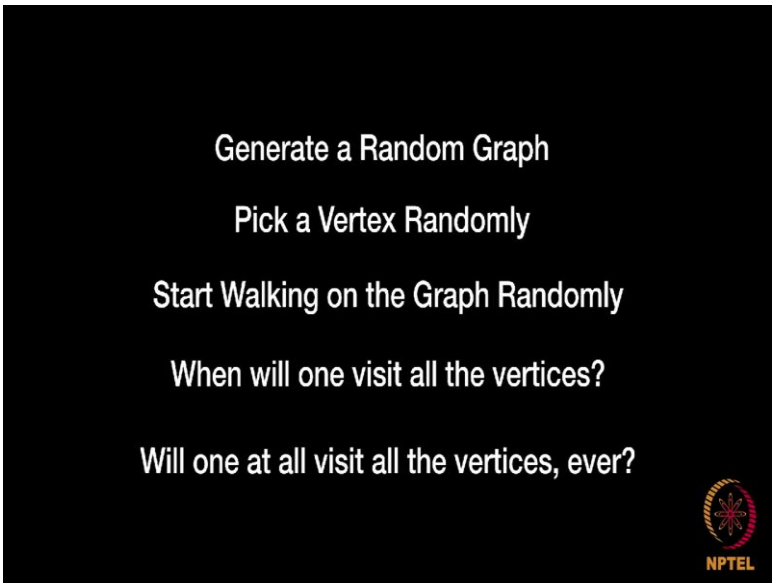


Is there a way in which you can explore this entire web graph and capture it? I gave you an example and you have this question in your hand, how do you get the web graph. The car, new city, taking random turns example; do you think you can link these two things? And do you think a question here has a solution which I stated in the form of an example?

(Refer Slide Time: 04:01)

So let us now do a quick experiment. I am curious to see if one takes a random walk on a given graph, how much time will it take for one to reach the entire graph by that I mean all vertices in the graph?

(Refer Slide Time: 04:19)

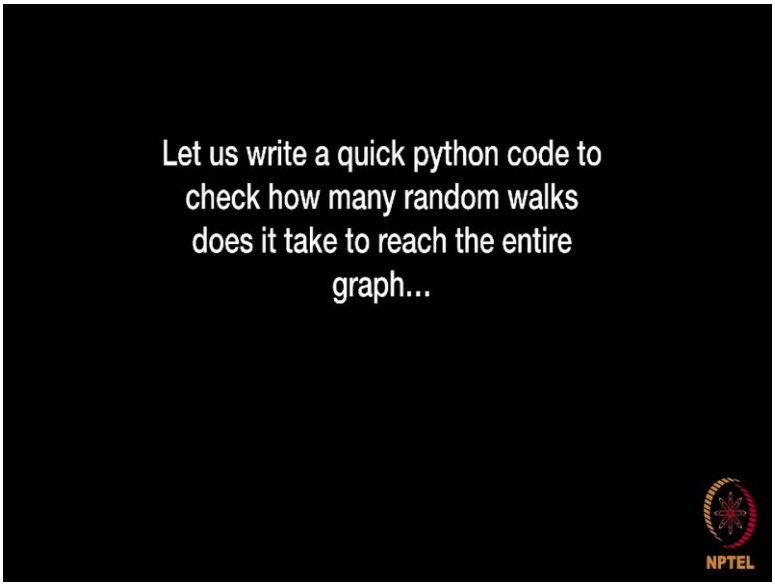


So, let us go slowly. I now need to generate a random graph on some n vertices with some probability p and then I pick a vertex randomly call it the start vertex. I will start walking on the graph randomly from this start vertex and then my question is when one will visit all the vertices in the graph. Will one at all visit all the vertices, ever? Pause for

a minute and think about this question and the what you think is your answer for this question.

Do you think it will take a long time or sometimes it is never possible to reach all the nodes by just taking a random walk? What do you mean by this? In a city, if you take a cab and then keep visiting places uniformly at random by tossing a coin and deciding whether to take a left turn or a right turn, how much time will it take for you to explore the entire city?

(Refer Slide Time: 05:21)



Let us now write a piece of code a quick python code to check how many random walks does it take to reach the entire graph. By reach the entire graph I mean reach all the vertices of the given graph.

(Refer Slide Time: 05:41)

```
1 import random
2 import matplotlib.pyplot as plt
3 import networkx as nx
4 import numpy
5
6 def walk(n,p):
7     start=random.randint(0,n-1)
8     G=nx.erdos_renyi_graph(n,p)
9     S=set([])
10    v=start
11    count=0
12    while(len(S)<n):
13        Nbr=nx.neighbors(G,v)
14        v=random.choice(Nbr)
15        S.add(v)
16        count=count+1
17    return count
18
19 l=[]
20 for i in range(20,300):
21    z=[]
22    for j in range(10):
23        z.append(walk(i,0.3))
24    l.append(numpy.average(z))
25    print i,"--->",numpy.average(z)
26 plt.plot(l)
27 plt.show()
28
~
~
"checkrandom.py" 28L, 522C written
```

Here is a piece of python code that have written. I am sure you people are, I am sure you people all of you are very familiar with python right now. You know how to write a piece of code. So, I have not gone a line by line. I have finished this code and you can probably take a pause and then take a look at this code and then judge what the code is doing. So, as you can see, I have imported a few functions import random matplotlib networkx and NumPy.

NumPy is used to compute average; it is a nice package for scientific computations. I am not using it much except for the 24th line as you can see, I am computing the average here in 24th line ok. Let me go slowly. This function walk (n, p); all it does is it generates a random graph with n nodes and p vertices starts from a random vertex in the given graph G (n, p). By that I mean an addition graph with n nodes and p probability p. And then I start with a with a vertex v which is uniformly picked from the vertex z.

And then I keep walking on this graph and then count how much time it takes until I visit all elements of the vertex set ok. Take a look at this code. It is quite self explanatory and then I what I do is I call this walk function quite often with i ranging from 20 to 300 with probability 0.3. Take a look at the walk function, you will understand what I am doing here. And every time I get an answer, what is the answer? The answer here is walk of i comma 0.3, simply returns the number of attempts number of random walks you must

take until you finish going through all the given vertices of the graph with i vertices and p equals 0.3.

And then I append this to a list called z and repeat this experiment some 10 times and then take the average of the number of random walks it takes for me to reach the entire graph. And then I do it for graphs of the size 20, 21, 22, 23 up to 300. And, then I plot on the x axis this range namely the number of vertices in the graph and in the y axis this plots the list l which stands for the number of random walks you have taken to cover the entire graph. So, let me execute this program and see what happens.

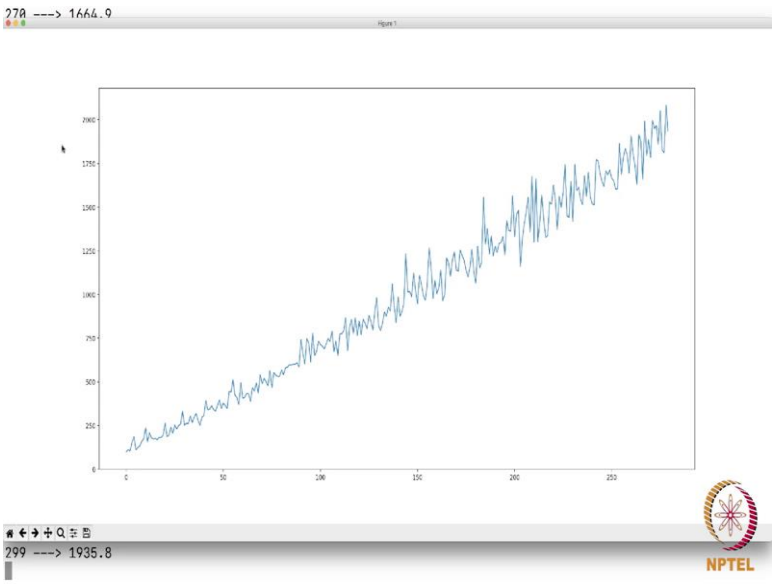
(Refer Slide Time: 08:37)

```
import random
import matplotlib.pyplot as plt
import networkx as nx
import numpy

def walk(n,p):
    start=random.randint(0,n-1)
    G=nx.erdos_renyi_graph(n,p)
    S=set([])
    v=start
    count=0
    while(len(S)<n):
        Nbr=nx.neighbors(G,v)
        v=random.choice(Nbr)
        S.add(v)
        count=count+1
    return count

l=[]
for i in range(20,300):
    z=[]
    for j in range(10):
        z.append(walk(i,0.3))
    l.append(numpy.average(z))
    print i,"--->",numpy.average(z)
plt.plot(l)
plt.show()

~
~
:!python %
```

(Refer Slide Time: 08:39)

As you can see, the these are the number of nodes and the number of at random walks it takes for you to cover the entire graph. So, let us wait and see what happens. So, 263, 4, 5, 6, 7 so, on; it goes on till 300. So, as you can see the right side is the answer which is the number of random walks it takes for you to go to the entire graph.

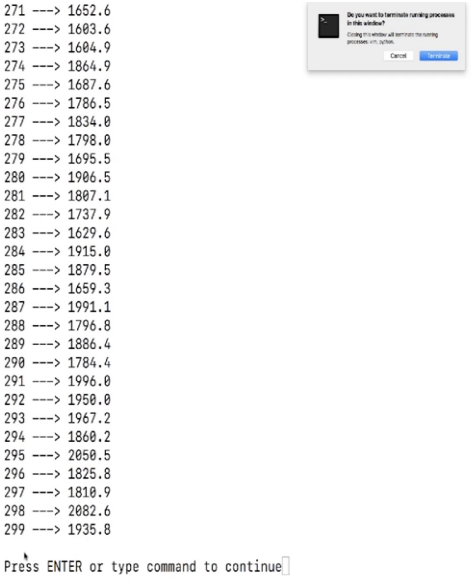
(Refer Slide Time: 09:13)



So, now I get the plot the plot looks beautiful as you can see um. The x axis is the number of nodes in the graph and y axis is the number of attempts it takes for you to number of random walks, you need to take to cover the entire graph.

What does the signify? It signifies that it is just a little bit of an effort a few times the number of nodes in a network is what it takes for you to cover the entire graph and that is a very nice observation that we make. As you can see the plot clearly says it is looking very linear, although it is not very linear a little bit of observation will tell you that it is very close to a n log n function. Anyway details a aside we will not worry much. All that I need to understand is this is very much possible.

(Refer Slide Time: 10:07)



For example, when I take a node on 299 when I take a graph on 299 nodes, when I take a graph on 299 nodes in roughly 2000 random walks, I cover the entire graph. As you can see 298 seems to have taken a little more time as you know this is averaged. So, if you have average a lot more, this will be a properly increasing function.