Social Networks

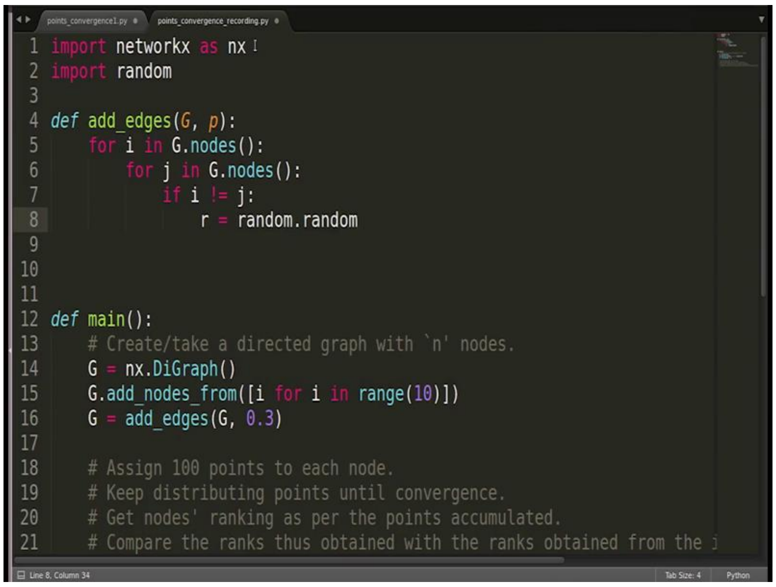
Prof. S. R. S. Iyengar
Prof. Anamika Chhabra
Department of Computer Science
Indian Institute of Technology, Ropar

Link Analysis

Lecture - 81
Implementing Page Rank Using Points Distribution Method – 2

Let us start with the Implementation of Page Rank Using Points Distribution Method, as we discussed in the previous video.

(Refer Slide Time: 00:12)



```python
import networkx as nx
import random

def add_edges(G, p):
    for i in G.nodes():
        for j in G.nodes():
            if i != j:
                r = random.random



def main():
    # Create/take a directed graph with `n' nodes.
    G = nx.DiGraph()
    G.add_nodes_from([i for i in range(10)])
    G = add_edges(G, 0.3)

    # Assign 100 points to each node.
    # Keep distributing points until convergence.
    # Get nodes' ranking as per the points accumulated.
    # Compare the ranks thus obtained with the ranks obtained from the i
```

I have pasted to 5 steps from the previous video over here and we will take the steps one by one. So, let me create a main function here ok. So, let me import networkx first. So, let us take these steps one by one. So, the first step is create or take a directed graph with n nodes as I told you previously. So, we can use a generator for creating a directed graph that is some function from network, we can make use of. For example, we have a function gnr graph which gives us a directed graph. We can use any function, or we can create our own function to create a directed graph as well. In this video we are going to created at directed graph ourselves.

So, let us see how we can do that. So, as a first step I am just creating an empty directed graph by using this function G = nx.DiGraph ok. This graph is empty, but it is a directed graph. Now, let us add the nodes and then we will add the at the edges. Let us add some nodes to it. So, I will write G.add_nodes from. So, we will add a list, I will write i for i in range, let me add some 10 nodes here. We can add more as well, but I want to show you the conversions, if you took take and a number of nodes it will take more nodes, it will take more time for convergence; you can always change that. So, I am taking 10 nodes at this point.

Now, we have added the nodes, we must now add the edges. So, I am going to use a technique of course, you can use many different methods to create a graph. The method that I am going to use here is that I will be randomly adding edges between the nodes, that is also well known technique to create a random graph which I think we will be discussing in the next weeks video. But I will be just briefing out the technique here because we will be creating this directed graph using that technique.

So, for that purpose I will be creating a function, let me create a function add edges ok. I will pass the graph here which has nodes and no edges as of now. So, I will be probabilistically adding the edges, I will briefly explain you the method as well. I am passing a probability value of 0.3 here and this function will be written in graph where the edges have been added.

So, let me create this function here I will create it here so, I will write. So, as a parameter this function requires the value of p which is a probability value. So, let me tell you what we are you doing here. So, we are going to take all possible edges that can ever we added to the graph and we will toss I mean, and we will add these edges to the graph after tossing a coin. By that what I mean is we will take an edge and we will toss a coin, if a coin turns out to be head we will add that edge and if the coin turns out to be tail we will not add that edge.

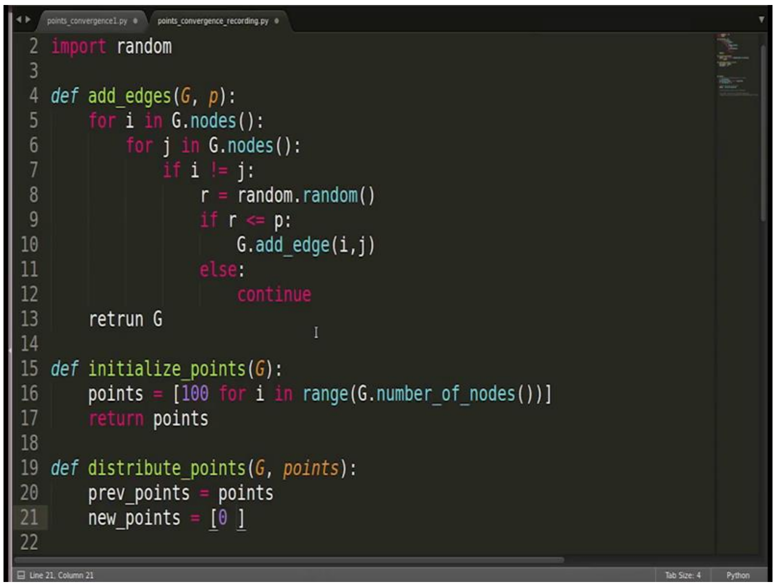
So, here in this case as you can see that we are passing a probability value p here and the value of p we have passed as 0.3; you can pass any value here, you can pass any value between 0 to 1 over here. So, by passing this probability value what we mean is that, you can assume that the coin is biased by a probability 0.3 which means with the probability 0.3 the coin will turn out to be head. And, with the probability 0.7 the coin will turn out to be a tail.

So, we will take an edge, we will toss the coin which is biased by this probability value and if a coin turns out to be head, we will add that edge otherwise we will not add that edge. So, this is the technique that we are going to follow. This is a well-known technique which is used to basically create random graphs. We are using this technique; you can use any other technique as well.

So, let us see how we can implement this. So, we have to take all possible edges, how can we do that. I can start loop I can write for i in G.nodes. So, I got all the nodes here because, we want the edges I will start another loop here. So, I will write for j in G.nodes ok. So, i comma j is we will give us all possible edges, we are now going to add self edges. So, I am going to check over here whether i is equal to j or not. So, I will write if i != j only then go ahead.

So, we have to implement that coin tossing here now, can you imagine how we can do that. We are going to take a random value for that we need to random package. So, let me import that, import random. So, we are going to make use of function random.random which gives us a random value between 0 and 1. Let me store that here r = random.random.

(Refer Slide Time: 05:56)



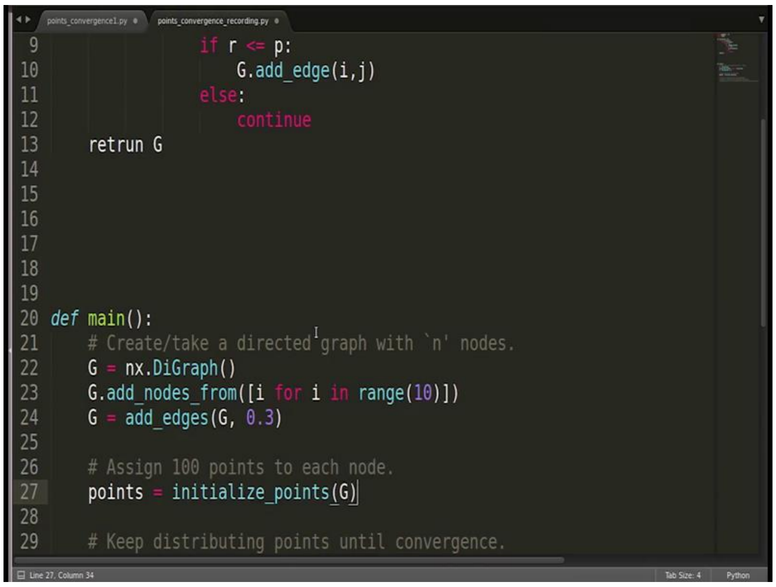
```python
import random

def add_edges(G, p):
    for i in G.nodes():
        for j in G.nodes():
            if i != j:
                r = random.random()
                if r <= p:
                    G.add_edge(i,j)
                else:
                    continue
    retrun G

def initialize_points(G):
    points = [100 for i in range(G.number_of_nodes())]
    return points

def distribute_points(G, points):
    prev_points = points
    new_points = [0 ]

```

Now, we got some value here in r, we will check whether that value r <= p value or not. If that if r <= p; that means, we got a head and if r > p ; that means, we got a tail. So, that is that how we are going to implement that coin tossing part. So, we got this random value, let us check whether r <= p; if it is we are going to add that edge. So, I will write G.add_edge, the edge is going to be (i , j) right and the edge part when r > p, we assume that that is that is a tail.

So, we will not add that edge, we will just continue here. So, this is how we are going to add the edges. Now, you know that this G is a directed graph. So, the edges that are going to be added here by using this add edge function are going to be directed edges. So, after this loop we would have added several edges, then we can return this graph. So, this is our add edge function and now we have created this graph G and we can go ahead.

So, we are done with the first point, let us implement the second one. It says assign 100 points to each node. So, we have to maintain a list points which will be storing the points for every node and we have to initialise this points list to 100. Let me create a function although you can directly write it, but I am going to create a function for this.

(Refer Slide Time: 07:35)



```
                if r <= p:
                    G.add_edge(i,j)
                else:
                    continue
    retrun G






def main():
    # Create/take a directed graph with `n' nodes.
    G = nx.DiGraph()
    G.add_nodes_from([i for i in range(10)])
    G = add_edges(G, 0.3)

    # Assign 100 points to each node.
    points = initialize_points(G)

    # Keep distributing points until convergence.
```

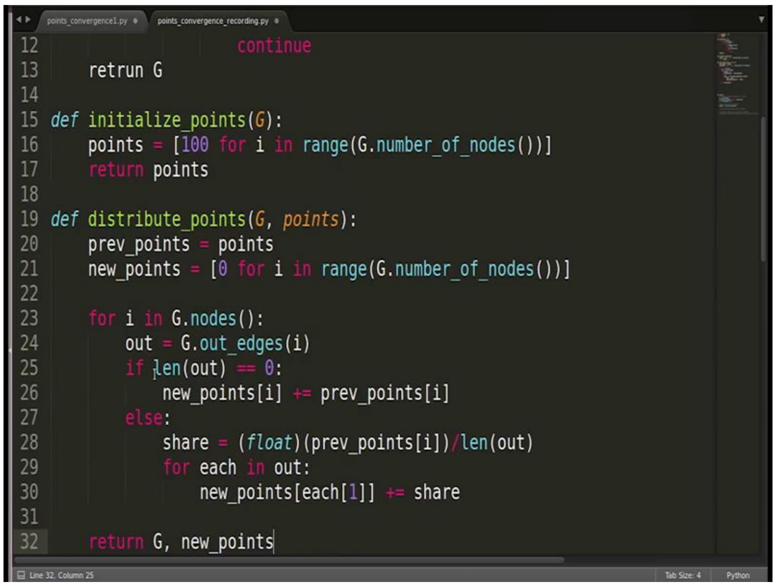
So, I will write points is equal to initialize points and I will pass the graph here ok. So, let me create this function ok. So, this function is just going to create a list points and that points list will be having 100 points for each node. So, I will write 100 for i in range, what is going to the size of this list. And, the size of this list is going to be equal to the number of nodes that are there in the graph although we know that there are 10 nodes as of now, but we can always change that. So, let me generalized it by writing

G.number_of_nodes and then we should return this list points we are done ok. Let us go here; we have done the second point. So, every node has 100 points right now.

Let us go to third step, third step says keep distributing points until convergence. Now, how do we distribute_points? Let us see how we do that. For that purpose, let me create a function distribute_points. So, I will write define distribute_points. So, this is going to be one iteration and we are going to distribute_points in this function. As you know that after the nodes distribute their points their points change. So, we are going to need previous points and new_points.

Let us pass the graph here because, we are going to need that and let us pass the points that are there as of now. As I told you previous points and new_points let me create to let me create two lists here. So, I will write previous points to be the same as points and this is just for a ease of understanding. Another list I will create new_points, this is what we will be returning. So, this list I am going to initialize to 0 and during the distribution part, this list will get more points and that is what we will written.

(Refer Slide Time: 09:58)



```
                        continue
    retrun G

def initialize_points(G):
    points = [100 for i in range(G.number_of_nodes())]
    return points

def distribute_points(G, points):
    prev_points = points
    new_points = [0 for i in range(G.number_of_nodes())]

    for i in G.nodes():
        out = G.out_edges(i)
        if len(out) == 0:
            new_points[i] += prev_points[i]
        else:
            share = (float)(prev_points[i])/len(out)
            for each in out:
                new_points[each[1]] += share

    return G, new_points
```

So, for now I will write 0 for i in range G.number_of_nodes. I have not copied G.number_of_nodes ok. So, this is the initialization of the new_points list which will be returned by this function. Now, how do we distribute? As I told you for every node, we will look at its out links. So, let me start a loop at these points, I write for i in G.nodes. Now, i is a node and we must distribute its points. How do we distribute? We look at its out links right. So, let us keep a track of the out links of this node i, the function that we can use from networkx for this purpose is out_edges.

So, I will call that function I will write G.out_edges. Now, this function is there for directed graphs only, the parameter will be the node. So, we write G.out_edges. What this function does is, for the node i it will return a list of the edges which are out_edges from this node i. Now after this, this node i has to distribute its points and equal share of its points to its neighbours right. So, we have the points that this node i has in this list previous points right. We are going to distribute these points equally amongst its neighbours.

And the number of neighbours we can get by getting the list and by getting the length of this list. Now, what happens what will happen if the list is empty, that is the node has no out link. In that case, the length of this list will be 0. So, it is a good idea to check for that condition over here. So, let me check if the length of this out is 0 not, because in that case division by 0 exception might come. So, it is better to check it here.

So, I will write if length of out is equal to 0 which means this node has no out link ok. If a node has no out link, what should happen? It will not be distributed; it will not be distributing any of its points right. So, what we write is, new_points for i are going to be plus equal to I will tell you why I have written plus equal to previous points i ok. So, I have written plus equal to because, this point is going to retain its points it is not going to give its points to anybody because there is no out link. However, it might have some in links and through those in links it might be getting more points.

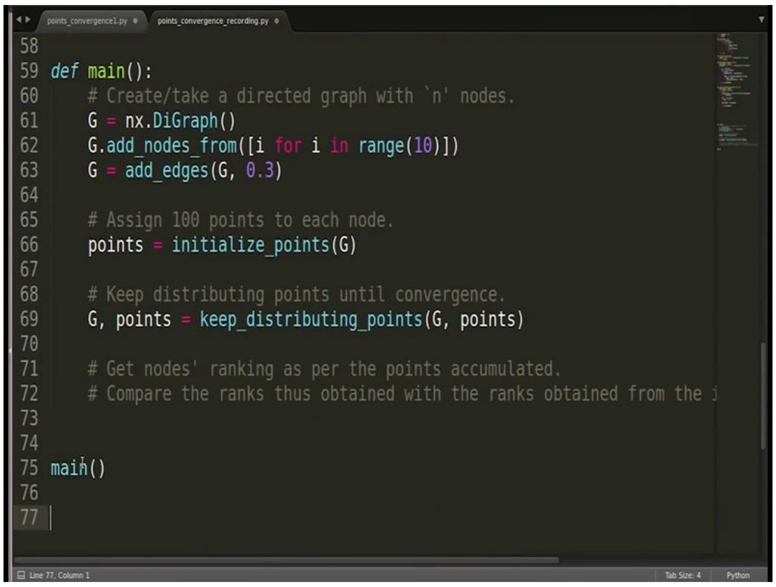
So, that is why we have to keep adding it. So, we have to write plus equal to. Now, this is what we will do in case a node has no out link. In the general case that is the nodes have out links, what are we going to do? We are going to distribute an equal share of these nodes points to its neighbours. Now, what is that share going to be? Let us create a variable share. Now, the share is going to be the number of points that are going to be distributed to every neighbour. What is this share going to be? So, as I told you if there are if there are 3 neighbours, the points the points that are going to be distributed is total divide by 3.

And what is the number of neighbours here? The number of neighbours is equal to the length of out here right. So, what are we going to do is we will write previous points that is what is to be distributed right, you remember that you have to only distribute the previous points. So, previous points are to be distributed in equal share so, we divide by length of out this. So, this is what is to be distributed, let me sorry let me do the float casting here so, that we get the exact values ok.

So, this is a share that has to be distributed to the neighbours. Now, where are the neighbours? The neighbours are in this list; this list is a list of edges. So, for every edge 0th value is the source that is our node i and first value that is the value at index 1 is going to be the neighbour ok. So, let us start a loop here in order to distribute the points. So, I will write for each in out which means for every neighbour, we are going to update the new_points of every neighbours. So, I will write new_points for each now each is an edge.

So, I will write each 1, each 1 is going to be the neighbour. So, new_points plus equal to share so, that is how we will be updating the points of every neighbour. I think we are done, let us return. What are we going to return? We are going to return the graph G and we are going to turn the list new_points ok. So, we have created this function, this distributes points which is going to be one iteration where every node will be distributing its points. Let us go back to main and let us see what we have to do.

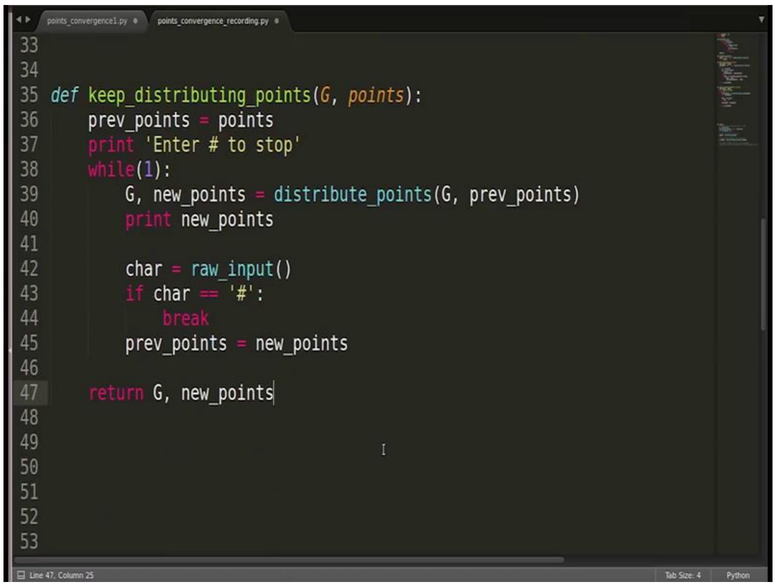
(Refer Slide Time: 15:22)



```python

def main():
    # Create/take a directed graph with `n' nodes.
    G = nx.DiGraph()
    G.add_nodes_from([i for i in range(10)])
    G = add_edges(G, 0.3)

    # Assign 100 points to each node.
    points = initialize_points(G)

    # Keep distributing points until convergence.
    G, points = keep_distributing_points(G, points)

    # Get nodes' ranking as per the points accumulated.
    # Compare the ranks thus obtained with the ranks obtained from the i


main()

```

We have to keep distributing the points until the convergence. So, we have to basically keep calling this function distribute_points. So, what we can do is let me create another function which will keep calling this function that we have just created. Let us call this function keep distributing points and the parameter will be G and the points that we have and this function should return G and points again. So, we are passing something and that is getting updated and that is what we will be returned. Now, what should this function do? Keep distributing points, this function has to keep calling the function distribute_points until we get the convergence ok. So, let us create this function ok.

(Refer Slide Time: 16:22)



```python


def keep_distributing_points(G, points):
    prev_points = points
    print 'Enter # to stop'
    while(1):
        G, new_points = distribute_points(G, prev_points)
        print new_points

        char = raw_input()
        if char == '#':
            break
        prev_points = new_points

    return G, new_points
```

So, as I told you this function has to keep calling this distribute_points function, let me start a while loop here. So, what I can write is while 1, they should keep going on ok. What should keep going on? Before that let me rename these previous points is equal to points, this is for the ease of variable name that is all ok. So, we are going to start this loop, this will keep going on and on. We have to call the function this distribute_points.

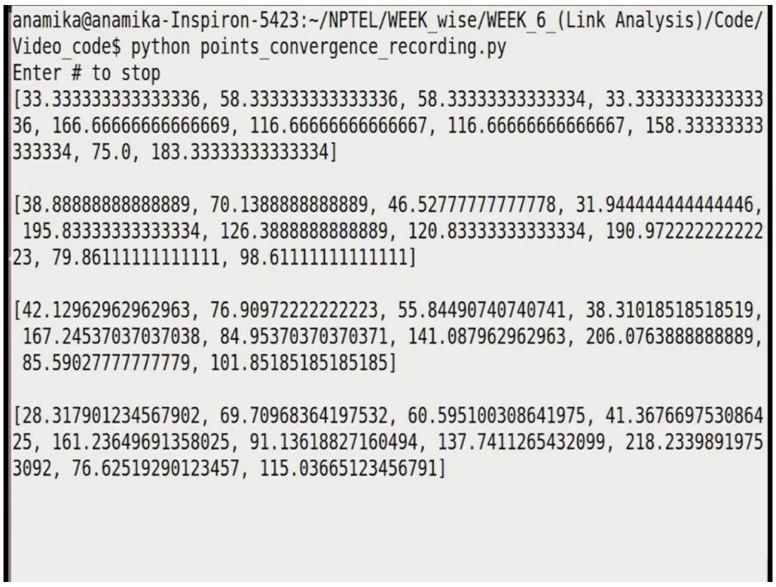
So, I will copy it and I will call it here, this function returns two things again. So, I will write G, new_points is equal to this right. What are we passing? We are passing previous points. So, let me rename these previous points done. So, we have called this function and one iteration is happened and we have got the new_points in this list new_points. If we want to keep track let us let us print this so, that we can see the updated points ok.

Now, this will keep going on and on, it has to stop somewhere basically when the convergence happens. So, you can do one thing, or you can do one or the two things, you can either through your code check whether the convergence has happened or not that is one thing. And, other simple thing that we can do is we can keep observing as we are printing it right. So, we can keep observing these points and whenever we see that convergence has happened, we can stop it.

So, for that let us let us ask the user to stop it, enter may be # to stop and let us ask the user to enter a hash when he wants it to stop. So, we will take that through the function may be raw input and if that char is equal to # we are going to stop. So, I will write break ok. Now, after one iteration, we have to assign the new_points to be the previous points and we have to repeat the process. So, it is important to write previous points is equal to new_points and after that we will go back here, and the same thing will start alright.

I think we have done, let us return let us return the same two things G and new_points. I think this function should work ok, we can, may be check the code as well if it is working fine, wait there is an error here ok. So, let us check the functionality of this code whether it is working fine or not. So, let me call main here ok.

(Refer Slide Time: 19:34)

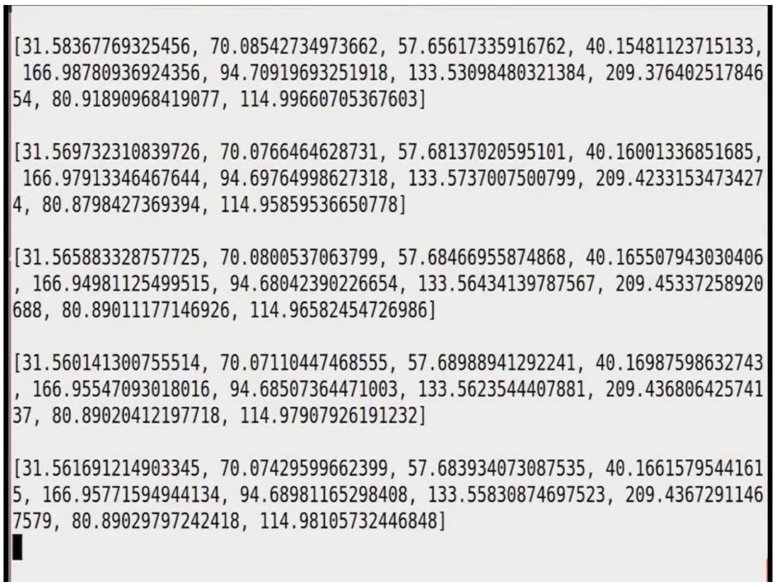


```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_6_(Link Analysis)/Code/
Video_code$ python points_convergence_recording.py
Enter # to stop
[33.333333333333336, 58.333333333333336, 58.33333333333334, 33.3333333333333
36, 166.66666666666669, 116.66666666666667, 116.66666666666667, 158.33333333
333334, 75.0, 183.33333333333334]

[38.88888888888889, 70.13888888888889, 46.52777777777778, 31.944444444444446,
 195.83333333333334, 126.38888888888889, 120.83333333333334, 190.972222222222
23, 79.86111111111111, 98.61111111111111]

[42.12962962962963, 76.90972222222223, 55.84490740740741, 38.31018518518519,
 167.24537037037038, 84.95370370370371, 141.087962962963, 206.07638888888889,
 85.59027777777779, 101.85185185185185]

[28.317901234567902, 69.70968364197532, 60.595100308641975, 41.3676697530864
25, 161.23649691358025, 91.13618827160494, 137.7411265432099, 218.2339891975
3092, 76.62519290123457, 115.03665123456791]
```

Let us get back to our screen and let us try calling this file ok. So, these are the points, we did not print the initial points right. They were obviously, 100 100 each we can print that as well. So, this is these are the values that we are getting after one iteration.

And we have to press enter after it, after I press enter the next the next iteration starts. And, as you can see the points are changing the first node was earlier having 33 points and now it is having 38 points. I am keeping on pressing enter and we are seeing the points how they are changing.

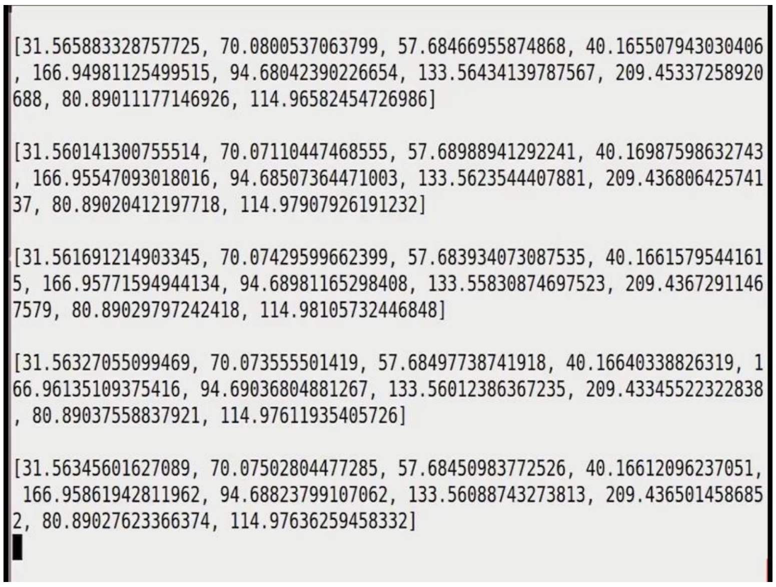
(Refer Slide Time: 20:12)



```
[31.58367769325456, 70.08542734973662, 57.65617335916762, 40.15481123715133,
 166.98780936924356, 94.70919693251918, 133.53098480321384, 209.376402517846
54, 80.91890968419077, 114.99660705367603]

[31.569732310839726, 70.0766464628731, 57.68137020595101, 40.16001336851685,
 166.97913346467644, 94.69764998627318, 133.5737007500799, 209.4233153473427
4, 80.8798427369394, 114.95859536650778]

[31.565883328757725, 70.0800537063799, 57.68466955874868, 40.165507943030406
, 166.94981125499515, 94.68042390226654, 133.56434139787567, 209.45337258920
688, 80.89011177146926, 114.96582454726986]

[31.560141300755514, 70.07110447468555, 57.68988941292241, 40.16987598632743
, 166.95547093018016, 94.68507364471003, 133.5623544407881, 209.436806425741
37, 80.89020412197718, 114.97907926191232]

[31.561691214903345, 70.07429599662399, 57.683934073087535, 40.1661579544161
5, 166.95771594944134, 94.68981165298408, 133.55830874697523, 209.4367291146
7579, 80.89029797242418, 114.98105732446848]
█
```

So, I keep on pressing the enter ok, as you can see we are sword of reaching the convergence.

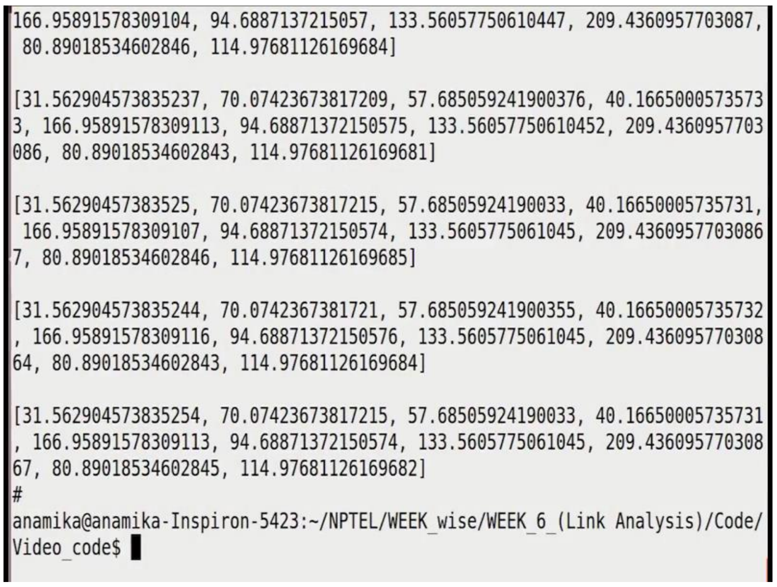
(Refer Slide Time: 20:21)



```
[31.565883328757725, 70.0800537063799, 57.68466955874868, 40.165507943030406
, 166.94981125499515, 94.68042390226654, 133.56434139787567, 209.45337258920
688, 80.89011177146926, 114.96582454726986]

[31.560141300755514, 70.07110447468555, 57.68988941292241, 40.16987598632743
, 166.95547093018016, 94.68507364471003, 133.5623544407881, 209.436806425741
37, 80.89020412197718, 114.97907926191232]

[31.561691214903345, 70.07429599662399, 57.683934073087535, 40.1661579544161
5, 166.95771594944134, 94.68981165298408, 133.55830874697523, 209.4367291146
7579, 80.89029797242418, 114.98105732446848]

[31.56327055099469, 70.073555501419, 57.68497738741918, 40.16640338826319, 1
66.96135109375416, 94.69036804881267, 133.56012386367235, 209.43345522322838
, 80.89037558837921, 114.97611935405726]

[31.56345601627089, 70.07502804477285, 57.68450983772526, 40.16612096237051,
 166.95861942811962, 94.68823799107062, 133.56088743273813, 209.436501458685
2, 80.89027623366374, 114.97636259458332]
█
```

So, it may stop, it may happen that as you keep pressing enter it will not change at all. So, nothing will change, or it may happen that values are changing by very nominal decimal points. So, you can stop wherever you want and or you can go on and on. So, it is up to you. So, I am keeping on pressing enter and I can see the values changing slightly only by decimal points. So, I can keep going on and on or I can stop.

(Refer Slide Time: 20:53)



```
166.95891578309104, 94.6887137215057, 133.56057750610447, 209.4360957703087,
 80.89018534602846, 114.97681126169684]

[31.562904573835237, 70.07423673817209, 57.685059241900376, 40.1665000573573
3, 166.95891578309113, 94.68871372150575, 133.56057750610452, 209.4360957703
086, 80.89018534602843, 114.97681126169681]

[31.56290457383525, 70.07423673817215, 57.68505924190033, 40.16650005735731,
 166.95891578309107, 94.68871372150574, 133.5605775061045, 209.4360957703086
7, 80.89018534602846, 114.97681126169685]

[31.562904573835244, 70.0742367381721, 57.685059241900355, 40.16650005735732
, 166.95891578309116, 94.68871372150576, 133.5605775061045, 209.436095770308
64, 80.89018534602843, 114.97681126169684]

[31.562904573835254, 70.07423673817215, 57.68505924190033, 40.16650005735731
, 166.95891578309113, 94.68871372150574, 133.5605775061045, 209.436095770308
67, 80.89018534602845, 114.97681126169682]
#
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_6_(Link Analysis)/Code/
Video_code$ █
```

So, I want to stop let me press hash and we are stopping. So, these are the points distribution that we are getting, and we are sort of seeing the convergence taking place as well. Now, let us get back to our code and let us see what is pending ok, next step is to get the nodes ranking as per the points accumulated. So, we have to basically rank the nodes based on the points that they have acquired overtime after number of iterations, after the convergence point.

So, I think we will do that in the next video.