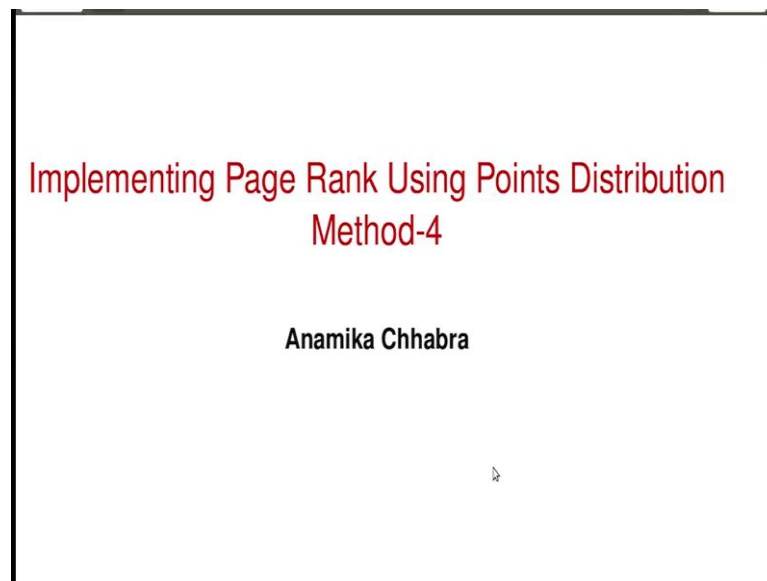


Social Networks
Prof. S. R. S. Iyengar
Prof. Anamika Chhabra
Department of Computer Science
Indian Institute of Technology, Ropar

Link Analysis
Lecture – 83
Implementing Page Rank Using Points Distribution Method – 4

(Refer Slide Time: 00:06)



In the previous video, we finished Implementing Page Rank using Points Distribution Method and it was in most of the cases coming same as the ranking that the networkx a function page rank provides us. However, in some cases the ranking was not similar; the reason for that was that our code was not handling the cases, where one node or a set of nodes have no out links. So, in that case all the points get accumulated in that node or that set of nodes. So, our code has to handle those cases. That is what we will be doing in this video.

Let us get back to our code and see what kinds of measures are required to handle those cases.

(Refer Slide Time: 00:51)

```
34
35
36 def keep_distributing_points(G, points):
37     prev_points = points
38     print 'Enter # to stop'
39     while(1):
40         G, new_points = distribute_points(G, prev_points)
41         print new_points
42
43         char = raw_input()
44         if char == '#':
45             break
46         prev_points = new_points
47
48     return G, new_points
49
50 def get_nodes_sorted_by_points(points):
51     points_array = np.array(points)
52     nodes_sorted_by_points = np.argsort(-points_array)
53     return nodes_sorted_by_points
54
```

(Refer Slide Time: 00:56)

```
67 G = nx.DiGraph()
68 G.add_nodes_from([i for i in range(10)])
69 G = add_edges(G, 0.3)
70
71 # Assign 100 points to each node.
72 points = initialize_points(G)
73 print points
74
75 # Keep distributing points until convergence.
76 G, points = keep_distributing_points(G, points)
77
78 # Get nodes' ranking as per the points accumulated.
79 nodes_sorted_by_points = get_nodes_sorted_by_points(points)
80 print 'nodes_sorted_by_points', nodes_sorted_by_points
81 # Compare the ranks thus obtained with the ranks obtained from the
82
83 pr = nx.pagerank(G)
84 pr_sorted = sorted(pr.items(), key = lambda x:x[1], reverse = True)
85 for i in pr_sorted:
86     print i[0],
87
```

So, this was our main. And here, we were calling this function `keep_distributing_points` and let us get back to this function and see what it was doing. So, `keep_distributing_points` was keeping on calling this function `distribute_points`. This `distribute_points` was basically, 1 iteration that was returning us updated points after that iteration right.

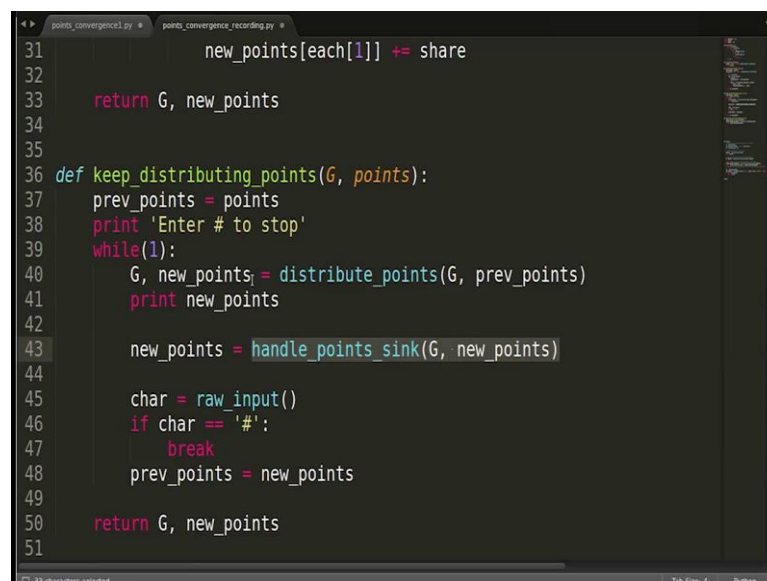
Now, after every iteration we have to make some changes to the points in order to handle the cases of sinks, point sinks. Now, how that is usually done let me explain that to you. So basically, what we do is after every iteration, we gather a fraction of points from

every node and the points that we gather, this way we then distribute these points equally amongst all the nodes right. So basically, for example, after 1 iteration we get some points with all the nodes. And after that to handle the sink cases, we take one more step and, in that step,, we let the node have 80 percent of it is points and we take some 20 percent of the points from that node. And we do that with every node. And that is how we gather some number of points, after that we divide these points that we have gathered this way amongst all the nodes equally.

Now this is the way to handle the problem of sink. You can consider this to be similar to the tax system in any country, where we take some say 30 percent of the salary from every person, if their salary is exceeding some value. So, the amount of money that we gather this way that is distributed for the development of the rest of the people or all the people in general. So, it is the same scenario here as well, the number of points that the node will be all the nodes will be keeping.

And the fraction of points that they will be letting go is basically decided by a parameter, which is usually denoted by s . So, usually we take $s = 0.8$, which means 80 percent of the nodes points will be with the nodes and 20 percent of the nodes points will be let go by all the nodes right. So, let us increment that thing here. As you can see in this function `keep_distributing_points`, we call this function and that is how we got the new points.

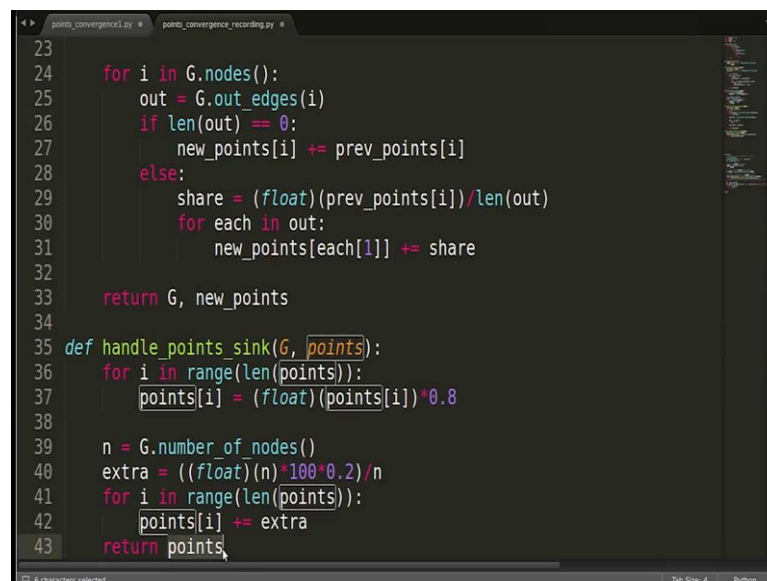
(Refer Slide Time: 03:44)



```
31         new_points[each[1]] += share
32
33     return G, new_points
34
35
36 def keep_distributing_points(G, points):
37     prev_points = points
38     print 'Enter # to stop'
39     while(1):
40         G, new_points = distribute_points(G, prev_points)
41         print new_points
42
43         new_points = handle_points_sink(G, new_points)
44
45         char = raw_input()
46         if char == '#':
47             break
48         prev_points = new_points
49
50     return G, new_points
51
```

Now, after we get the new points, let us take one more step to handle the sinks let me for that purpose create a function. So, I will write new points, is equal to handle points sink let us call this function. So, let us pass the graph and the new points here ok. So, we have to create this function, now which will be doing, whatever I just explained that is it will be taking 20 percent of the points from every node and whatever we gather this way we will be distributed to all the nodes equally.

(Refer Slide Time: 04:16)

A screenshot of a code editor with a dark background. The code is written in Python and is part of a file named 'points_convergence_recording.py'. The code is as follows:

```
23
24 for i in G.nodes():
25     out = G.out_edges(i)
26     if len(out) == 0:
27         new_points[i] += prev_points[i]
28     else:
29         share = (float)(prev_points[i])/len(out)
30         for each in out:
31             new_points[each[1]] += share
32
33 return G, new_points
34
35 def handle_points_sink(G, points):
36     for i in range(len(points)):
37         points[i] = (float)(points[i])*0.8
38
39     n = G.number_of_nodes()
40     extra = ((float)(n)*100*0.2)/n
41     for i in range(len(points)):
42         points[i] += extra
43     return points
```

So, let us create this function ok. So, as a first step every nodes point will be reduced right, every nodes point will be reduced to 80 percent of what they were. So, let me start look here for `i in range(len(points))` we can generalize it to may be points and that is what it will be returning. So, for `in range(len(points))`, we must reduce the points for every node. So, we will write `points[i] = points[i] * 0.8`. So, they become 80 percent or what they were earlier let me do the casting ok.

So now, every node has reduced points reduced to 80 percent, now what should be done with the remaining 20 percent of the point that we have gathered from every node? Now if you know that what are the total number of points that the graph had initially: you can understand that then the total number of points never change. They are only sorry; they are only distributed amongst each other or the nodes will keep giving the points to each other, but the total number of points never changed ok.

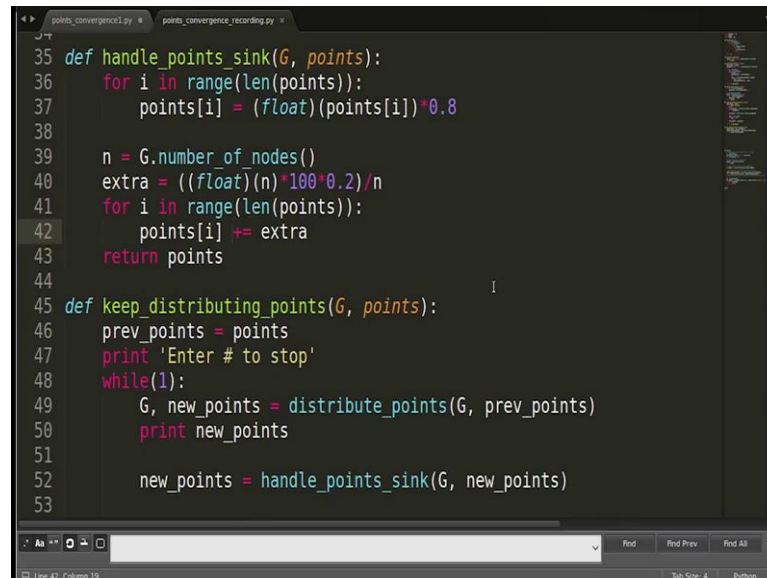
So, we are going to use that fact to know, what are the points that we have gathered? So, if every nodes point has become 80 percent, we have gathered 20 percent of the total number of points that the graph had ok. Now what is the total number of points that had? We had assigned 100 points to every node ok. Let us see the total number of nodes, I will just show that here. So, I will write `g.number_of_nodes` ok. So, I got `n` here what will be the total number of points in the graph that will be $n * 100$, because we assigned 100 points.

Now all of this these total points 20 percent is what we have now so that we can distribute to every node. So, I will write into 0.2. So, this is what we have ok. Now how many points to we have to distribute to every node? We must distribute these points equally ok. So, we will divide by `n` ok, so that we get an equal share for every node that will be the casting here ok. Let us call it `extra` now because, this is the extra points that every node will be having may be, we can took this as well ok.

So, these are the extra points that have to be assigned to every node apart from the point that they already have. So again, we will start a loop. I will write `for i in range(len(points))` again ok. So, points of every node will become plus equal to `extra` ok. So, every node is going to get extra points I think we have done. So, just return the points these are the new points ok. So, we passed old points to this function `handle_points_sink` and it will return the updated points, which will go back to this function keep distributing points here ok.

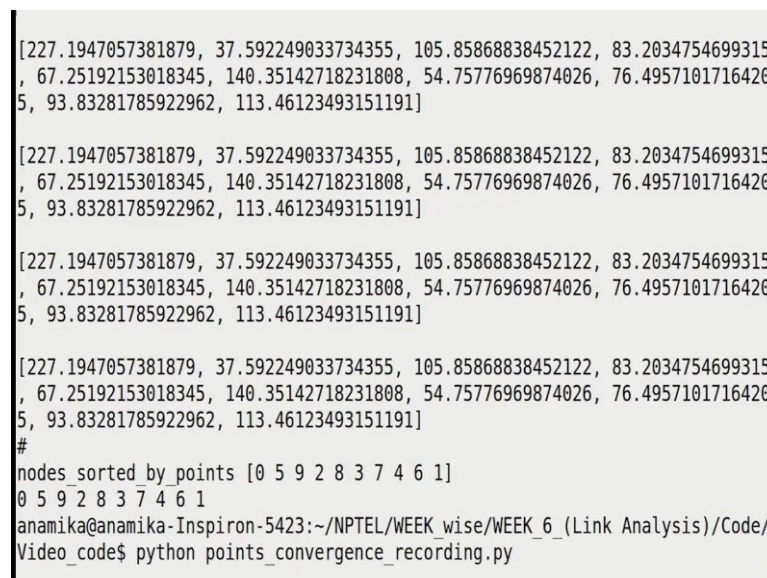
So, new points have changed, this is what we passed, and this is what we got ok. So, new points have changed since, we had not change the name. So, I do not think we have to change any other thing in the function.

(Refer Slide Time: 08:09)



```
35 def handle_points_sink(G, points):
36     for i in range(len(points)):
37         points[i] = (float)(points[i])*0.8
38
39     n = G.number_of_nodes()
40     extra = ((float)(n)*100*0.2)/n
41     for i in range(len(points)):
42         points[i] += extra
43     return points
44
45 def keep_distributing_points(G, points):
46     prev_points = points
47     print 'Enter # to stop'
48     while(1):
49         G, new_points = distribute_points(G, prev_points)
50         print new_points
51
52         new_points = handle_points_sink(G, new_points)
53
```

(Refer Slide Time: 08:14)



```
[227.1947057381879, 37.592249033734355, 105.85868838452122, 83.2034754699315
, 67.25192153018345, 140.35142718231808, 54.75776969874026, 76.4957101716420
5, 93.83281785922962, 113.46123493151191]

[227.1947057381879, 37.592249033734355, 105.85868838452122, 83.2034754699315
, 67.25192153018345, 140.35142718231808, 54.75776969874026, 76.4957101716420
5, 93.83281785922962, 113.46123493151191]

[227.1947057381879, 37.592249033734355, 105.85868838452122, 83.2034754699315
, 67.25192153018345, 140.35142718231808, 54.75776969874026, 76.4957101716420
5, 93.83281785922962, 113.46123493151191]

[227.1947057381879, 37.592249033734355, 105.85868838452122, 83.2034754699315
, 67.25192153018345, 140.35142718231808, 54.75776969874026, 76.4957101716420
5, 93.83281785922962, 113.46123493151191]
#
nodes_sorted by_points [0 5 9 2 8 3 7 4 6 1]
0 5 9 2 8 3 7 4 6 1
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_6_(Link_Analysis)/Code/
Video_code$ python points_convergence_recording.py
```

Let see how our program works. So, let us execute, so these are the initial values, and these are this is how the values are changing, I am just keeping on pressing enter and now I will press # ok. So, these are the values that we are getting 0592837. So, they are exactly in matching. So, we are handling the sink cases. So, the values are exactly matching.

Let me tell you one more thing sometimes there might be few differences with the with the ranking that you are getting from the networkx function as I told you, I guess one of

the reasons could be 2 of a nodes are having the same page rank another reason could be you have not let it converge, it is going on and on and it is taking a lot of iterations to reach an exact convergence point.

And you are just stopping somewhere in the middle. So, that might also lead to a slight deviation from the ranking that you are getting from networkx. Let me check one more example here ok.

(Refer Slide Time: 09:21)

```
[64.01028097141224, 58.520807692687356, 98.25552970807644, 110.8290816197897
2, 218.3354120891941, 121.90356318812216, 182.42474313462776, 93.26529858218
694, 0, 52.45528301390334]

[64.01028097141224, 58.520807692687356, 98.25552970807644, 110.8290816197897
2, 218.3354120891941, 121.90356318812216, 182.42474313462776, 93.26529858218
694, 0, 52.45528301390334]

[64.01028097141224, 58.520807692687356, 98.25552970807644, 110.8290816197897
2, 218.3354120891941, 121.90356318812216, 182.42474313462776, 93.26529858218
694, 0, 52.45528301390334]

[64.01028097141224, 58.520807692687356, 98.25552970807644, 110.8290816197897
2, 218.3354120891941, 121.90356318812216, 182.42474313462776, 93.26529858218
694, 0, 52.45528301390334]
#
nodes_sorted_by_points [4 6 5 3 2 7 0 1 9 8]
4 6 5 3 2 7 0 1 9 8
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_6_(Link_Analysis)/Code/
Video_code$
```

So, this is what we are getting let me stop here. 4653270198 here; so, we are getting the same values because, we are handling the case.

So, in case you get the slight deviations also you need do not worry because, you have understood the whole concept; how basically points distribution is similar to getting a page rank because, apart from the mathematical complexities this is what basically is the main idea behind getting the nodes rank based on the page rank values.

So, this was about the implementation of page rank using point distribution method. So, we are going to implement random walk method in the next videos.