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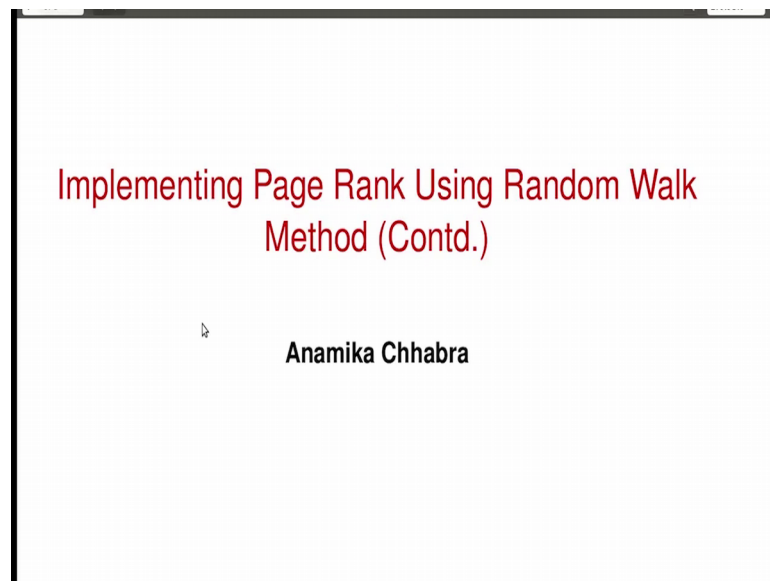**Prof. S. R. S. Iyengar**
Prof. Anamika Chhabra
Department of Computer Science
Indian Institute of Technology, Ropar

Link Analysis
Lecture – 85
Implementing Page Rank Using Random Walk Method – 2

(Refer Slide Time: 00:07)

In this video we are going to Implement Page Rank Using Random Walk. So, let us start the code.

(Refer Slide Time: 00:12)


```
25  def main():
26      #1. create/take a directed graph
27
28      G = nx.DiGraph()
29      G.add_nodes_from([i for i in range(10)])
30      G = add_edges(G, 0.3)
31
32      #2. Peform a random walk
33      |
34
35      #3. Get sorted nodes as per points accumulated during random walk.
36          I
37      nodes_sorted_by_RW = get_nodes_sorted_by_RW(RW_points)
38      print 'nodes_sorted_by_random_walk', nodes_sorted_by_RW
39
40      #4. Compare results with the inbuilt Page Rank method.
41      pr = nx.pagerank(G)
42      pr_sorted = sorted(pr.items(), key=lambda x: x[1], reverse = True)
43      for i in pr_sorted:
44          print i[0]
```

As I told you, in the previous video there are 4 steps which are to be implemented. And out of those 4 steps first third and forth steps are the same that we implemented in our previous video on implementation of page rank using point distribution method. So, we are going to reuse that code and there is no point of writing the code again.

So, I have copied pasted the code which is required over here in this file. Let me show you, we will start from the first step. First step is to take a directed graph. So, as we had created the graph. How had we created? We had taken a an object of nx dot DiGraph. So, we created an empty directed graph here and after that we add nodes to it, we added 10 nodes to this graph. And after that we added edges to the node to the graph using this function add edges.

(Refer Slide Time: 01:07)

So, let us go back here. These add edges function I have copied as it is from the previous code. So, here just to brief about we are tossing coin for every edge. And if the coin turns out to be head we are adding that edge otherwise we are not keeping that edge. So, that is how we are randomly assigning edges to the graph.
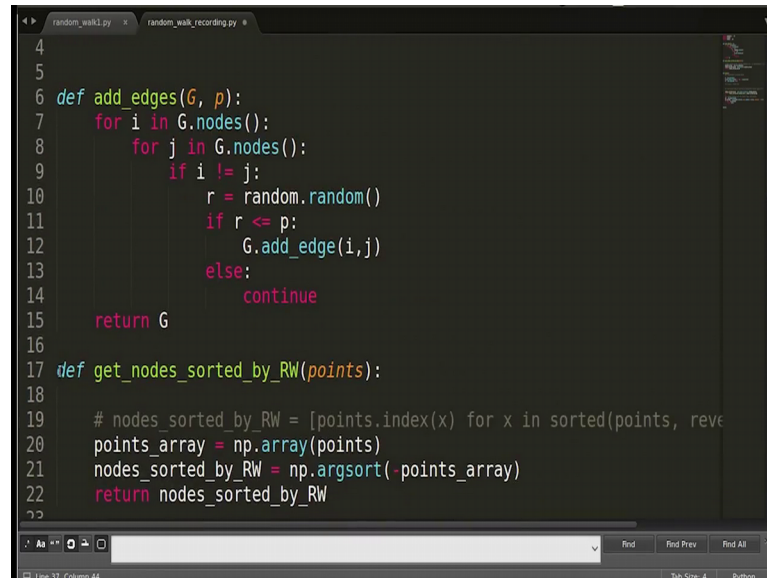
(Refer Slide Time: 01:35)

So, after this our graph is ready and we can perform a random walk on this. So, that is a second step performing a random walk which is to be done, we will just do that. After forming the random walk we will be having random walk points. And then we have to

rank the node based on random walk points which is what we done in the previous video. I have just pasted the code; I have just renamed this function get node sorted by random walk. I just renamed it here that is all, the functionalities entirely the same.
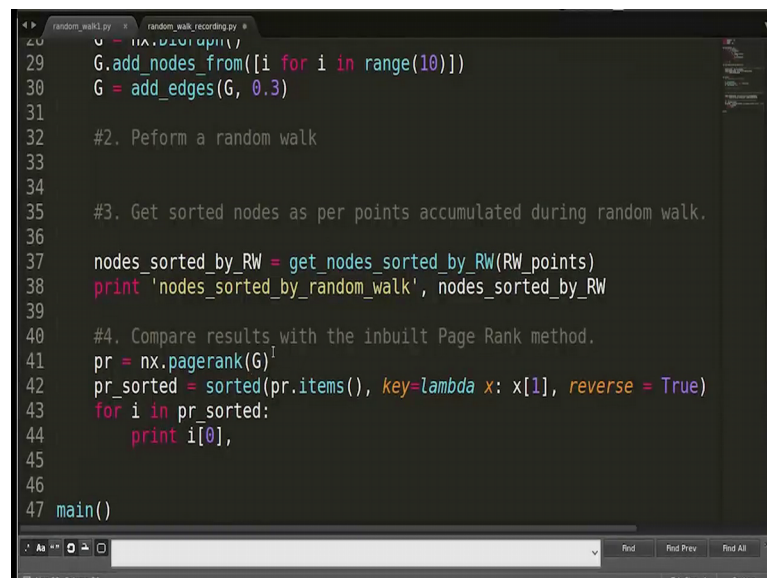
(Refer Slide Time: 02:08)


```python
 4
 5
 6  def add_edges(G, p):
 7      for i in G.nodes():
 8          for j in G.nodes():
 9              if i != j:
10                  r = random.random()
11                  if r <= p:
12                      G.add_edge(i,j)
13                  else:
14                      continue
15      return G
16
17  def get_nodes_sorted_by_RW(points):
18
19      # nodes_sorted_by_RW = [points.index(x) for x in sorted(points, reve
20      points_array = np.array(points)
21      nodes_sorted_by_RW = np.argsort(-points_array)
22      return nodes_sorted_by_RW
```

So, this function I have added as it is. So, that part is done.

(Refer Slide Time: 02:13)


```python
29      G.add_nodes_from([i for i in range(10)])
30      G = add_edges(G, 0.3)
31
32      #2. Peform a random walk
33
34
35      #3. Get sorted nodes as per points accumulated during random walk.
36
37      nodes_sorted_by_RW = get_nodes_sorted_by_RW(RW_points)
38      print 'nodes_sorted_by_random_walk', nodes_sorted_by_RW
39
40      #4. Compare results with the inbuilt Page Rank method.
41      pr = nx.pagerank(G)
42      pr_sorted = sorted(pr.items(), key=lambda x: x[1], reverse = True)
43      for i in pr_sorted:
44          print i[0],
45
46
47  main()
```

And finally, once we are done we use the page rank method from network x. And we compare the results, our results with the functions results; again this is entirely the same code ok. So, what has to be done is the second step which is performing a random walk,

now let us do that. So, this function is going to we are going to create a function random walk and that function will return random walk points ok.

(Refer Slide Time: 02:43)

So, let us write that: random walk points is equal to random walk. So, we are going to create this function random walk, we will pass the graph there ok. So, let us create this function.

(Refer Slide Time: 02:59)

Now, as I told you what do we mean by random walk? We will randomly choose, 1 node from all the nodes that, is the first step ok how do we do that? Let us create let us get a

list of nodes firstly. So, let us write node is equal to G dot nodes because, we have chosen one randomly we will choose out of these out of these list. And as and when we visit a node, we increment its counter, increment its random walk points. So, it will be nice to create a list which will be returned by this function, random walk points is equal to we have to initialize it to 0. So, we can write 0 for i in range G dot number of nodes. So, random walk points initialize to 0.

Now, we have this list of nodes and we have to choose one out of them out of these nodes randomly. So, we can do r is equal to random dot. We have to choose 1 node randomly out of these lists; for that we can use a function random dot choice, random dot choice out of nodes ok. So, we got 1 node uniformly at random. Since, we are have visited it we have to increment its counter its point. So, we will write RW points for this node r, have to be incremented by 1.

After that we have to look at the nodes which are neighbors to this node ok. We have to choose one neighbor uniformly at random. So, let us first get a list of the neighbors of this node are. How do we get that? For a directed graph we have function G dot out edges ok. G dot out edges for r, out will contain a list of all the edges, out edges from r ok. And after that we have to choose one of the, one of the neighbors uniformly at random and we have to repeat the same process. So, basically what are we are doing, we have to keep repeating it. How many times we should do, as much as possible ok.

So, let us create a counter for that. And we can start a loop because, we have to keep repeating things, we can take a very big number probably whatever as much as you can handle. I have taken this big number. So, we will visit these many nodes. So, we will basically have these many iterations of a random walk from one node to the other. So, we can keep any number here. In the basic idea is that we should be able to visit all the nodes; we there should not be nodes which we have not visited because if you do not visited node the counter for them will be 0. And, hence we will not be able to rank the nodes properly.

So, the idea is that we should be able to visit all the nodes. And that too multiple times so, that we are able to capture the frequency of visiting in our counts; in our random walk points ok. So, what do we have to keep repeating? We are started the loop. What do we have to keep repeating? We had reached r and these are the neighbors of r, we have to

choose one uniformly at random out of them ok. So, that is what we will do. Before that what if this out is 0, the length of out is 0 that is the node which we have reached has no out links that may also happen right. And, I told you in that cases what we do? In that case we do teleportation, that we that is we choose one node uniformly at random from all the nodes.

So, let us check that if length of out is equal to 0, what am I doing? If the length of out is equal to 0. What we do? We have to choose one uniformly random let us create a new variable for that to keep the node which is currently under focus ok. This focus will contain the node which is currently under focus; which is currently being visited basically. So, we have to choose one node uniformly at random from all the nodes we will write random dot choice nodes ok. This is the list which we already created. In case the node which we have reached at has number of neighbors, what do we have to do? We have to choose one neighbor out of them.

So, maybe we can have another random variable here, we can write r 1 dot. I am sorry random dot choice. Now this choice has to be made out of what? This choice has to be made out of out. Because out is what is containing all the edges and we will choose one edge out of them. Now, what is going to be the focus? The focus is not going to be the edge. The focus is going to be the node which is on the other side. How do we get the node which is on the other side, using r 1 1 because r 1 0 comma r 1 1 is basically the edge ok yeah exactly.

So we got the focus where focus is the current node where we are currently. Since we have reached at this node we have to implement the random walk points for this node. So, we will write random walk points for focus plus is equal to 1. Now again we have to get the list of its neighbors. So, we will write out is equal to we can copy paste this, I am just set r will be replaced by focus; because focus, focus is what is a current node.

Now, this is one iteration; what is iteration having? This iteration consists of looking at the neighbors choosing one out of them randomly and then it be and then repeating the same process. That is what this while looping is doing. Let us implement the count c that is the number of times this while loop will continue ok. I think we are done. So, what we have to return? This function should return random walk points right. So, let us return

this; I think we are done. Let us go down to the main, in the main also we have implemented everything.

(Refer Slide Time: 10:05)

We only have to write this instruction rest all are the same from the previous code. I think we are good to go, let us check whether this code works.

(Refer Slide Time: 10:19)

So, let us so, as you can see the outputs 9, 2, 4, 6 and here 9, 2, 6, 4 and rest of the values are matching. So, there can be two reasons for the miss match. One could be the rank of 4 and 6 might be same, 4 and 6 here and 6, 4 and 4 here. So, they can give different

ordering. Second reason could be that we needed more iteration so, that we were able to visit all the nodes properly and we could achieve the effect. Because random walk is basically based on probability right, you randomly choose the next node. It might so, happen that you are ending up reaching one node, more than the other node whereas, they both have the same number of in links right.

So, that may happen, let us check another example. Again here a everything is same except 8 2 and 2 8 over here; we can get one more. So, in this case used as you can see it is precisely the same, everything is matching. So, it might happen because it is based on probability and it is never possible that there are 2 nodes which have exactly the same configuration in terms of the number of in links. And, we are reaching these two nodes precisely the same number of times. So, that sort of difference is going to happen. So, this was about the implementation of page rank using random walk technique.