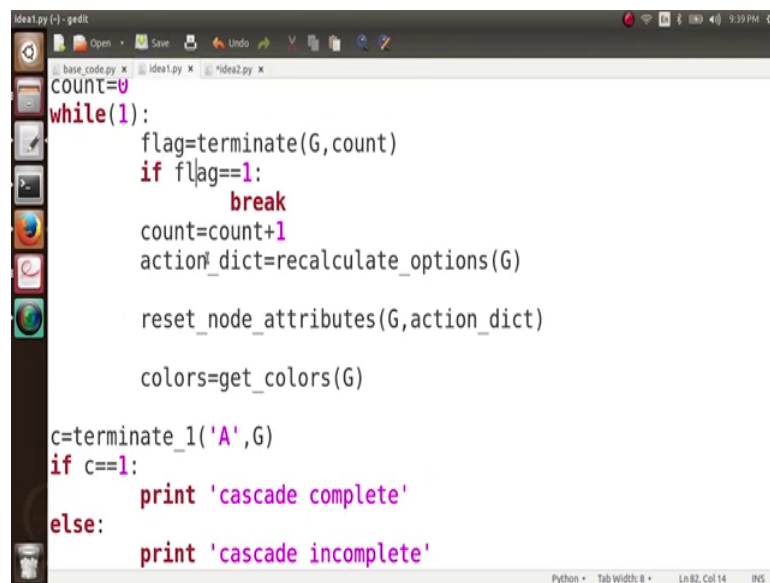


Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar
Cascading Behavior in Networks

Lecture – 98
Coding the Second Big Idea – Key People

(Refer Slide Time: 00:05)



```
idea1.py (-) - gedit
base_code.py x  idea1.py x  *idea2.py x
COUNT=0
while(1):
    flag=terminate(G,count)
    if flag==1:
        break
    count=count+1
    action_dict=recalculate_options(G)
    reset_node_attributes(G,action_dict)
    colors=get_colors(G)
c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'
```

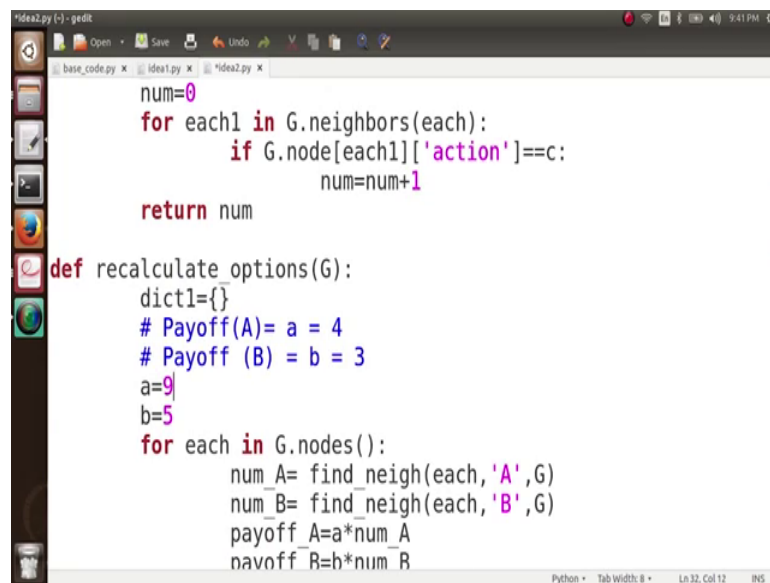
So, now we are going to implement the second idea. What was the second idea? In the first idea we looked that increasing the payoff associated with this new action or this new behaviour can cause a complete cascade. In a second idea, we want to talk about key people right. We, are going to keep the payoff associated with a particular action same, but what we are interested in?

What we are interested in looking at is, we will take this network again and we see that when we start with some particular bunch of nodes it creates a complete cascade, but we start from some other bunch of nodes, they are not able to create a complete cascade. It means that different nodes in this network have different power to create your cascade and these bunch of people which if chosen, creates your complete cascade they are known as the key people.

Again, I would like to remind you about digital marketing in choosing the right set of people to popularize your idea. So, what we are going to do in the second screen cast is,

we are going to take a network and then we are going to choose different sets of initial adopters and will look at why certain sets are able to create a complete cascade, other sets are unable to create a complete cascade. So I will take this same code and I will put it in a different file idea2 dot p y ok.

(Refer Slide Time: 01:36)

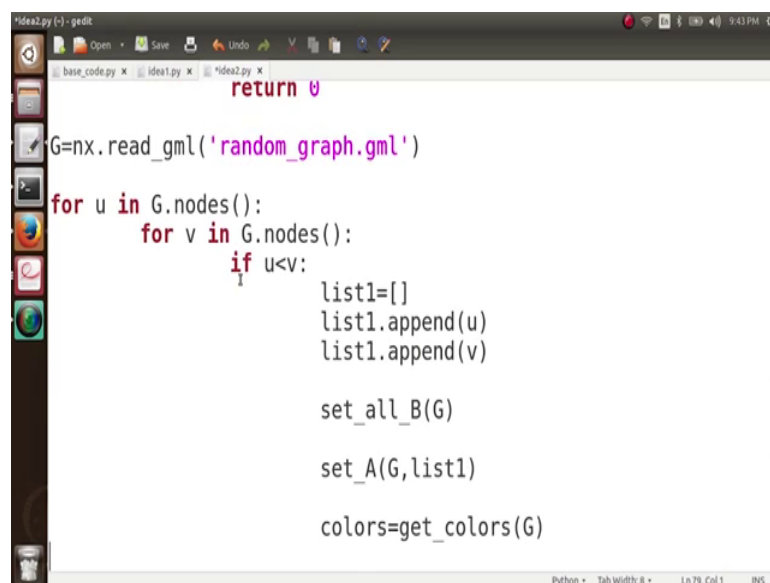


```
num=0
for each1 in G.neighbors(each):
    if G.node[each1]['action']==c:
        num=num+1
return num

def recalculate_options(G):
    dict1={}
    # Payoff(A)= a = 4
    # Payoff (B) = b = 3
    a=9
    b=5
    for each in G.nodes():
        num_A= find_neigh(each, 'A', G)
        num_B= find_neigh(each, 'B', G)
        payoff_A=a*num_A
        payoff_B=b*num_B
```

So, now what we do is, first of all I change my payoff let us say the payoff associated with a is 9 and with b is 5.

(Refer Slide Time: 01:49)



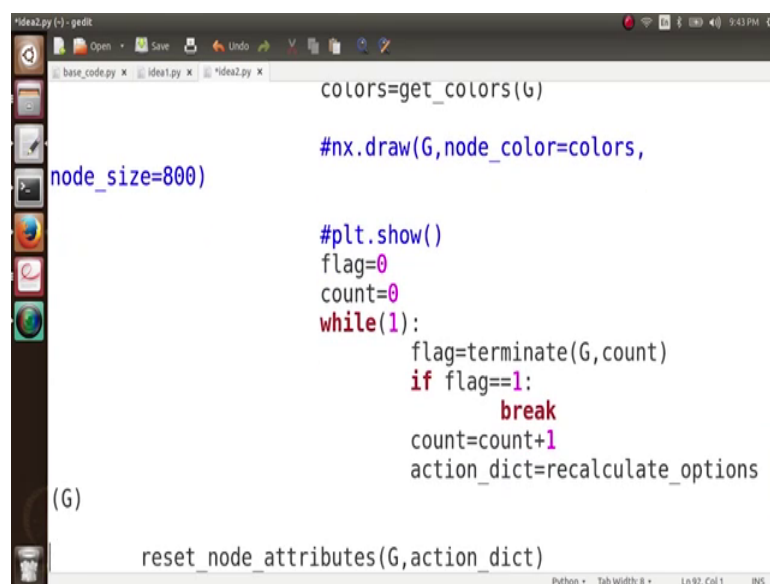
```
return 0
G=nx.read_gml('random_graph.gml')
for u in G.nodes():
    for v in G.nodes():
        if u<v:
            list1=[]
            list1.append(u)
            list1.append(v)
            set_all_B(G)
            set_A(G,list1)
            colors=get_colors(G)
```

What we want to do is, I am going to take 2 initial adopters. So, these 2 initial adopters can be any 2 nodes in the networks. So, for every possible set of 2 nodes in the network, we will start the cascade from there and we check whether this node have been able to create a complete cascade or not.

And we are using the same previous graph as before random underscore graph dot gml. So, what I am going to do here is for u in G dot nodes and then, for v in G dot nodes, I do not want my sets to be repeated. So, if I do it for like this for v in G dot nodes, v in G dot nodes 1, 2 means node 1 and node 2 will come once and node 2 and node 1 will come once, rather node 1 and here again it will choose 1 again. I do not want such kind of repetitions. So, I put a loop here node loop condition here. If u is less than v only then, it is what I am doing will be considered.

So, if u is less than v now, I create my list l. So, for every possible values of u and v what I do is, I append both of these values in my list, list dot append u and then, list dot append v. So, what is happening here? Now list l consists of my initial set, initial set of adopters. So, this particular loop condition will both view and take each and every set of 2 nodes in my network and after taking each and every set of 2 nodes in the network, we can see whether my cascade was complete or not. While I just shift everything inside these loops ok. I do not want to draw my graphs here because there will be so many sets and we cannot see the graph for each and every set.

(Refer Slide Time: 04:08)

A screenshot of a Python IDE window titled 'idea2.py (-) - gedit'. The window shows a code editor with the following Python code:

```
colors=get_colors(G)

#nx.draw(G,node_color=colors,

node_size=800)

#plt.show()
flag=0
count=0
while(1):
    flag=terminate(G,count)
    if flag==1:
        break
    count=count+1
    action_dict=recalculate_options

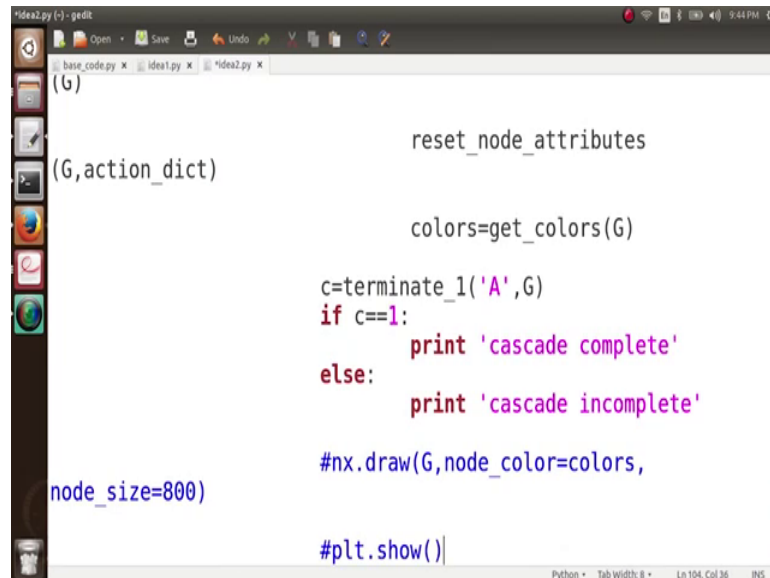
(G)

reset_node_attributes(G,action_dict)
```

The code is written in a dark-themed editor with syntax highlighting. The status bar at the bottom indicates 'Python • Tab Width: 8 • Ln 92, Col 1 • 1/5'.

It will be a time consuming process. We just want to see whether the cascade is complete or not.

(Refer Slide Time: 04:27)



```
graph TD
    G((G))
    G --> reset_node_attributes[reset_node_attributes]
    G --> colors=get_colors(G)[colors=get_colors(G)]
    G --> c=terminate_1('A',G)[c=terminate_1('A',G)]
    G --> if_c==1[if c==1:]
    if_c==1 --> print_cascade_complete[print 'cascade complete']
    if_c==1 --> else[else:]
    else --> print_cascade_incomplete[print 'cascade incomplete']
    G --> nx_draw[G, node_color=colors]
    G --> plt_show[plt.show()]
    node_size=800
```

```
(G)

reset_node_attributes

colors=get_colors(G)

c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'

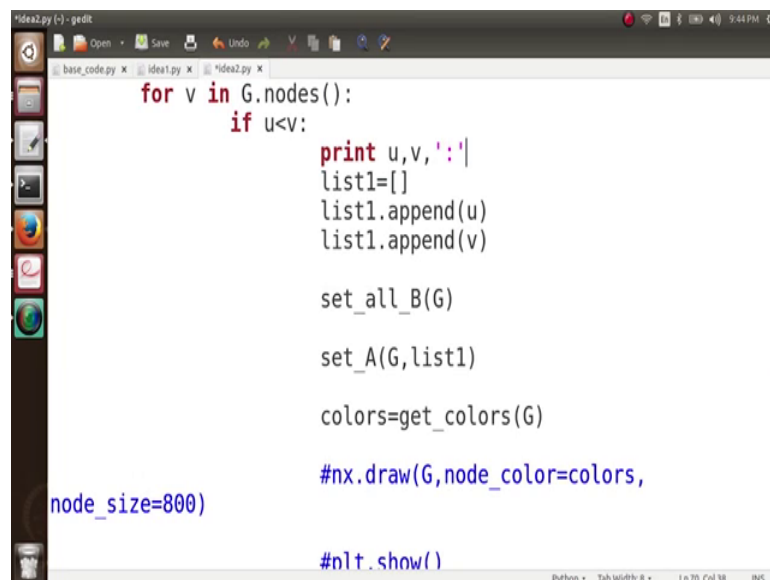
#nx.draw(G,node_color=colors,

node_size=800

plt.show()
```

And here also we do not want to draw this graph ok.

(Refer Slide Time: 04:49)



```
graph TD
    G((G))
    G --> for_v[G.nodes()]
    for_v --> if_u<v[if u<v:]
    if_u<v --> print_uv[print u,v,':']
    if_u<v --> list1=[]
    if_u<v --> list1.append(u)
    if_u<v --> list1.append(v)
    G --> set_all_B[G]
    G --> set_A[G,list1]
    G --> colors=get_colors(G)
    G --> nx_draw[G, node_color=colors]
    G --> plt_show[plt.show()]
    node_size=800
```

```
for v in G.nodes():
    if u<v:
        print u,v,':'
        list1=[]
        list1.append(u)
        list1.append(v)

set_all_B(G)

set_A(G,list1)

colors=get_colors(G)

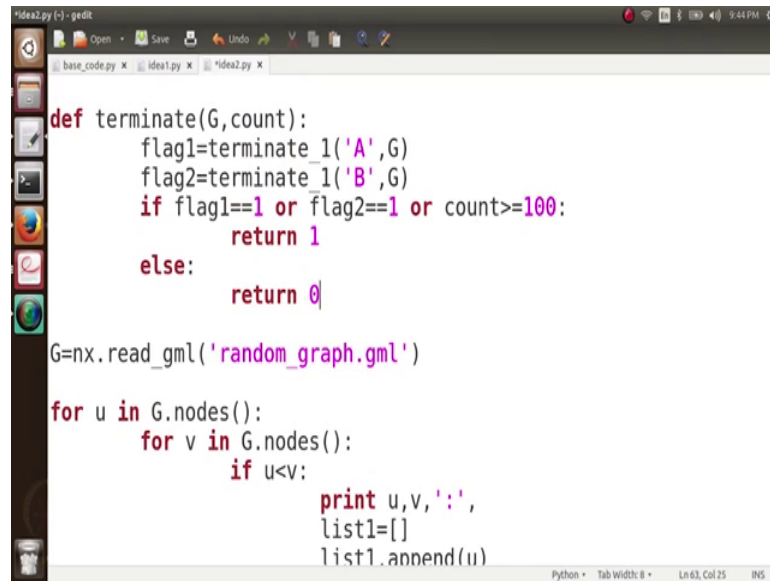
#nx.draw(G,node_color=colors,

node_size=800

plt.show()
```

And also what I do is, I print the values of u and v here. So, that I know what is happening, For which nodes we are getting the particular result ok. So, what this piece of code is doing? Let me quickly show you.

(Refer Slide Time: 05:04)



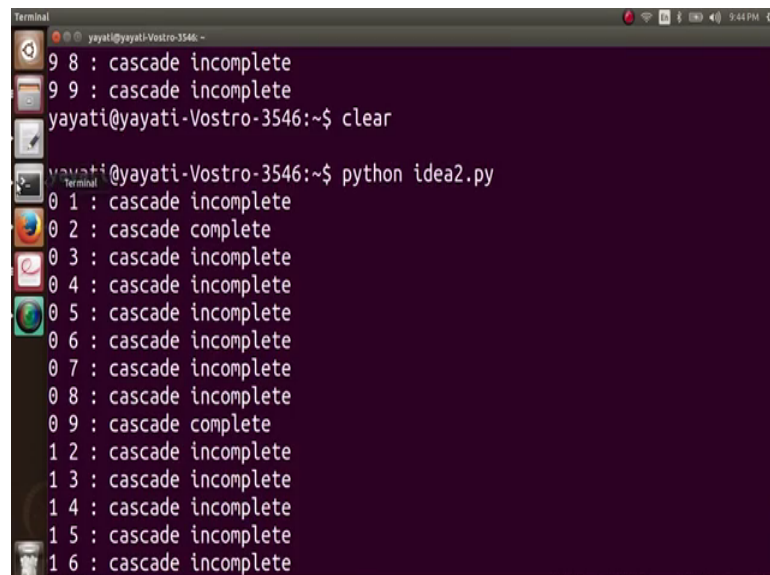
```
def terminate(G,count):
    flag1=terminate_1('A',G)
    flag2=terminate_1('B',G)
    if flag1==1 or flag2==1 or count>=100:
        return 1
    else:
        return 0

G=nx.read_gml('random_graph.gml')

for u in G.nodes():
    for v in G.nodes():
        if u<v:
            print u,v,':'
            list1=[]
            list1.append(u)
```

So, we have a network here random underscore graph dot gml. For every possible set of 2 nodes we take them as the initial adopters in list 1. We start the cascade from these 2 nodes and we see at and whether the, it these 2 nodes the result in a complete cascade or not. So, it is a simple code.

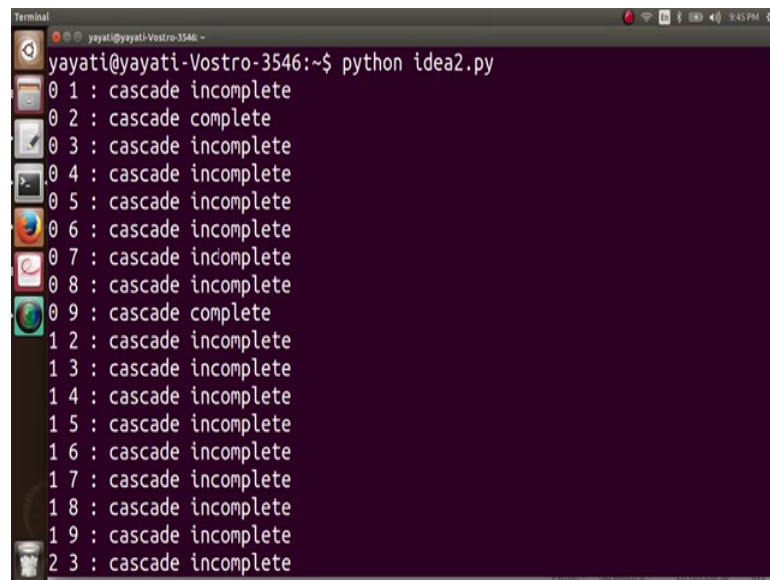
(Refer Slide Time: 05:22)



```
yayati@yayati-Vostro-3546:~$ clear
yayati@yayati-Vostro-3546:~$ python idea2.py
9 8 : cascade incomplete
9 9 : cascade incomplete
0 1 : cascade incomplete
0 2 : cascade complete
0 3 : cascade incomplete
0 4 : cascade incomplete
0 5 : cascade incomplete
0 6 : cascade incomplete
0 7 : cascade incomplete
0 8 : cascade incomplete
0 9 : cascade complete
1 2 : cascade incomplete
1 3 : cascade incomplete
1 4 : cascade incomplete
1 5 : cascade incomplete
1 6 : cascade incomplete
```

Let us run it here now ok. So, let us see this is the results so for all possible set of nodes. You see here for most of the sets the cascade remains incomplete. But there are few sets here. Like 0 2 and then, 0 9 and then 1 here 2 9.

(Refer Slide Time: 05:45)



```
Terminal
yayati@yayati-Vostro-3546:~$ python idea2.py
0 1 : cascade incomplete
0 2 : cascade complete
0 3 : cascade incomplete
0 4 : cascade incomplete
0 5 : cascade incomplete
0 6 : cascade incomplete
0 7 : cascade incomplete
0 8 : cascade incomplete
0 9 : cascade complete
1 2 : cascade incomplete
1 3 : cascade incomplete
1 4 : cascade incomplete
1 5 : cascade incomplete
1 6 : cascade incomplete
1 7 : cascade incomplete
1 8 : cascade incomplete
1 9 : cascade incomplete
2 3 : cascade incomplete
```

So, only these 3 sets they result in a complete cascade. All the other sets they results in an incomplete cascade. This shows us this that yes, there are certain key people, certain key nodes in this network and if you start your cascade from there. It tend to become a complete cascade. Proving our second idea that yes, there are key people in this network.