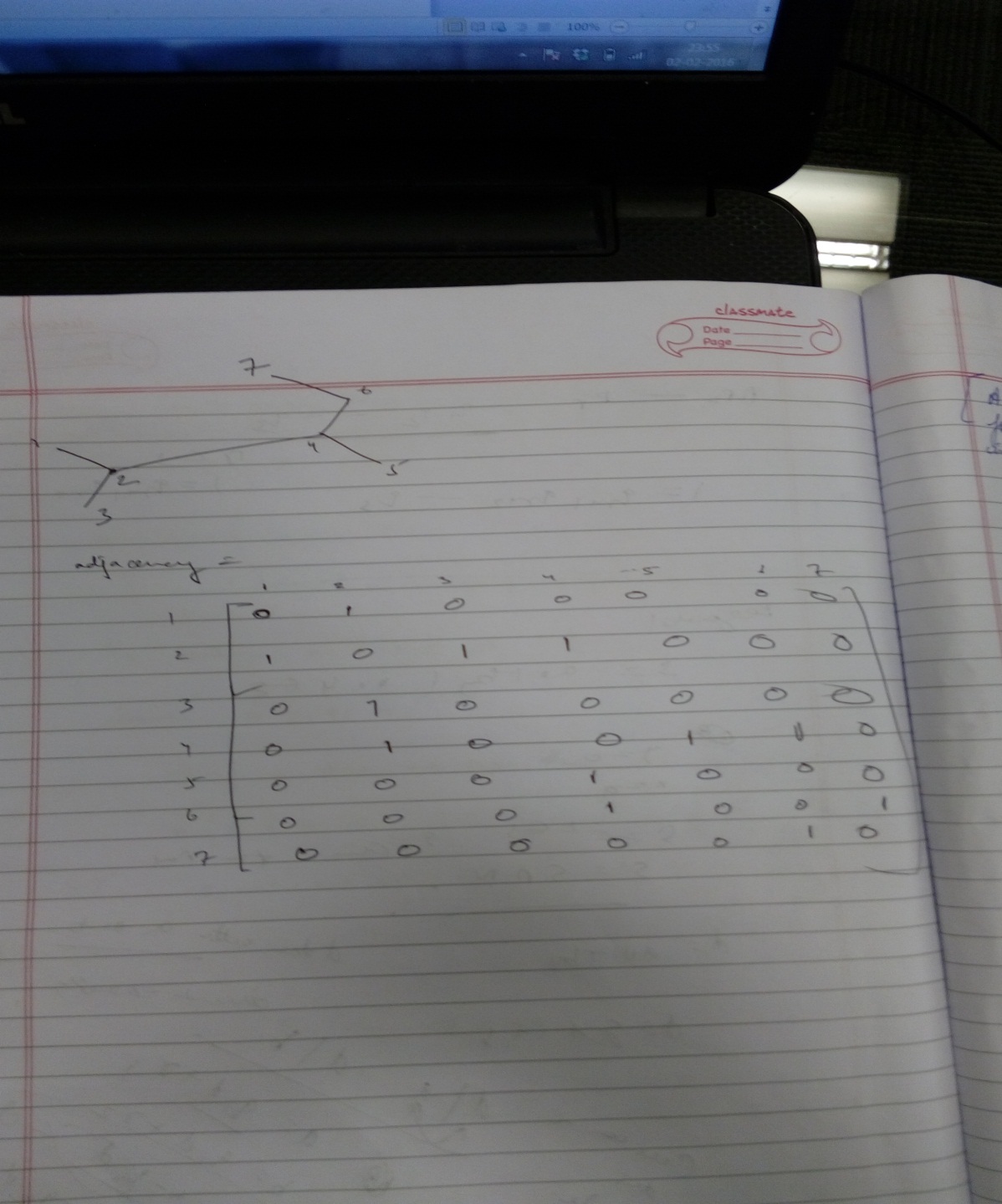
**We are trying to find exponent for a adjacency matrix of a tree**

****

Adjacency matrix of the tree we are using :

**[[0 1 0 0 0 0 0]**

**[1 0 1 1 0 0 0]**

**[0 1 0 0 0 0 0]**

**[0 1 0 0 1 1 0]**

**[0 0 0 1 0 0 0]**

**[0 0 0 1 0 0 1]**

**[0 0 0 0 0 1 0]]**

For normal computations , i.e. finding the exponent without the self loop, we are not successful. Hence self-loop is required

Cases :

1. Self loop at vertex 1

exponent is 6

[[129 116 76 143 39 47 37]

[116 232 40 59 104 133 10]

[ 76 40 52 104 9 10 29]

[143 59 104 222 11 12 67]

[ 39 104 9 11 51 67 1]

[ 47 133 10 12 67 89 1]

[ 37 10 29 67 1 1 22]]

1. Self-loop at vertex 2

exponent is 4

[[ 21 44 21 34 9 10 7]

[ 44 120 44 63 34 41 10]

[ 21 44 21 34 9 10 7]

[ 34 63 34 64 11 12 17]

[ 9 34 9 11 13 17 1]

[ 10 41 10 12 17 23 1]

[ 7 10 7 17 1 1 6]]

1. Self-loop at vertex 3

exponent is 6

[[ 52 40 76 104 9 10 29]

[ 40 232 116 59 104 133 10]

[ 76 116 129 143 39 47 37]

[104 59 143 222 11 12 67]

[ 9 104 39 11 51 67 1]

[ 10 133 47 12 67 89 1]

[ 29 10 37 67 1 1 22]]

1. Self-loop at vertex 4

exponent is 2

[[ 3 1 3 6 1 1 1]

[ 1 12 1 9 6 7 1]

[ 3 1 3 6 1 1 1]

[ 6 9 6 22 7 8 5]

[ 1 6 1 7 4 5 1]

[ 1 7 1 8 5 7 1]

[ 1 1 1 5 1 1 2]]

1. Self-loop at vertex 5

exponent is 4

[[11 1 11 23 7 1 6]

[ 1 45 1 9 30 29 1]

[11 1 11 23 7 1 6]

[23 9 23 58 25 8 17]

[ 7 30 7 25 30 23 6]

[ 1 29 1 8 23 23 1]

[ 6 1 6 17 6 1 6]]

1. Self-loop at vertex 6

exponent is 4

[[11 1 11 23 1 9 7]

[ 1 45 1 11 23 39 9]

[11 1 11 23 1 9 7]

[23 11 23 61 9 41 25]

[ 1 23 1 9 13 25 7]

[ 9 39 9 41 25 63 25]

[ 7 9 7 25 7 25 13]]

1. Self-loop at vertex 7

exponent is 6

[[ 44 1 44 95 1 9 37]

[ 1 183 1 11 95 132 46]

[ 44 1 44 95 1 9 37]

[ 95 11 95 220 9 39 104]

[ 1 95 1 9 51 74 30]

[ 9 132 9 39 74 118 67]

[ 37 46 37 104 30 67 81]]

The python code is as follows (‘’’ ‘’’ is the comment , we comment those vertex out on which we need the self loop)

**import** numpy **as** np  
  
n = 7  
  
a = [0 **for** i **in** range(n \*\* 2)]  
  
a = np.array(a, int)  
a = a.reshape((n, n))  
  
a[0][1] = 1  
a[1][0] = 1  
a[1][2] = 1  
a[1][3] = 1  
a[2][1] = 1  
a[3][1] = 1  
a[3][4] = 1  
a[3][5] = 1  
a[4][3] = 1  
a[5][3] = 1  
a[5][6] = 1  
a[6][5] = 1  
  
y = a  
ct = 100 # even on 1000 interations we dont get a positive matrix therefore we need to put self loops  
**while** ct > 0:  
 x = a  
 y = np.dot(x, y)  
 **if** 0 **not in** y:  
 **print** "exponent reached self loop not required"  
 **break** ct -= 1  
  
  
'''a[0][0] = 1 # self loop put at vertex 1  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1  
  
'''  
  
'''a[1][1] = 1 # self loop put at vertex 2  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1  
'''  
  
'''a[2][2] = 1 # self loop put at vertex 3  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1  
'''  
  
'''a[3][3] = 1 # self loop put at vertex 4  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1'''  
  
  
'''  
a[4][4] = 1 # self loop put at vertex 5  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1  
'''  
'''  
a[5][5] = 1 # self loop put at vertex 6  
y = a  
ct = 0  
while ct != 1000:  
 x = a  
 y = np.dot(x, y)  
 if 0 not in y:  
 print "exponent is " + str(ct)  
 print y  
 break  
 ct += 1  
'''  
a[6][6] = 1 # self loop put at vertex 7  
y = a  
ct = 0  
**while** ct != 100:  
 x = a  
 y = np.dot(x, y)  
 **if** 0 **not in** y:  
 **print** "exponent is " + str(ct)  
 **print** y  
 **break** ct += 1