Bioinformatics III

First Assignment

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

April 18, 2018

Exercise 1.1: The random network

(a) Listing 1 shows source code.

Listing 1: Example Listing of source code

```python
class Node:
    def __init__(self, identifier):
        """
        Sets node id and initialize empty node list that references its connected nodes
        """
        self.id = identifier
        self.nodelist = []

    def hasLinkTo(self, node):
        """
        Returns True if this node is connected to node asked for,
        False otherwise
        """
        for i in range(0, len(self.nodelist)):
            if self.nodelist[i] == node:
                return True
        return False

    def addLinkTo(self, node):
        """
        Adds link from this node to parameter node (only if there is no link connection already
        does not automatically care for a link from parameter node to this node
        """
        if not self.hasLinkTo(node):
            self.nodelist.append(node)
            return True
        return False

    def degree(self):
        """
        Returns degree of this node
        """
        return len(self.nodelist)

    def __str__(self):
        """
        Returns id of node as string
        """
        return str(self.id)
```

(b) Listing 2 shows source code.

Listing 2: Example Listing of source code

```python
from Node import Node
```

```python
class AbstractNetwork:
    """Abstract network definition, can not be instantiated"""

    def __init__(self, amount_nodes, amount_links):
        """
        Creates empty nodelist and call createNetwork of the extending class
        """
        self.nodes = {}
        self.__createNetwork__(amount_nodes, amount_links)

    def __createNetwork__(self, amount_nodes, amount_links):
        """
        Method overwritten by subclasses, nothing to do here
        """
        raise NotImplementedError

    def appendNode(self, node):
        """
        Appends node to network
        """
        self.nodes[node.id] = node

    def maxDegree(self):
        """
        Returns the maximum degree in this network
        """
        maxdegree = 0
        for n in self.nodes.itervalues():
            if maxdegree < n.degree():
                maxdegree = n.degree()
        return maxdegree

    def size(self):
        """
        Returns network size (here: number of nodes)
        """
        return len(self.nodes)

    def __str__(self):
        '''
        Any string-representation of the network (something simply is enough)
        '''
        string = ""
        for n in self.nodes.values():
            if len(n.nodelist) == 0:
                string += str(n) + "\n"
            for ref in n.nodelist:
                if ref.id > n.id:
                    string += str(n) + " - " + str(ref) + "\n"
        return string

    def getNode(self, identifier):
        """
        Returns node according to key
        """
        return self.nodes[identifier]
```

(c) Listing 3 shows source code.

Listing 3: Example Listing of source code

```python
from AbstractNetwork import AbstractNetwork
from Node import Node
import random # you will need it :-)

class RandomNetwork(AbstractNetwork):
    """Random network implementation of AbstractNetwork"""
```

```
    def __createNetwork__(self, amount_nodes, amount_links): # remaining methods are taken from
        """
        Creates a random network
        1. Build a list of n nodes
        2. For i=#links steps, add a connection between for two randomly chosen nodes that are
        """
        for nodeid in range(0, amount_nodes):
            n = Node(nodeid)
            self.appendNode(n)

        random.seed()
        if amount_nodes > 1:
            p = 2*amount_links/(amount_nodes*(amount_nodes-1))
        else:
            p = 0

        links = 0
        while links < amount_links:
            randint1 = random.randint(0,len(self.nodes)-1)
            randint2 = random.randint(0,len(self.nodes)-1)
            n1 = self.getNode(randint1)
            n2 = self.getNode(randint2)
            if randint1 != randint2:
                if n1.addLinkTo(n2):
                    links += 1
                    n2.addLinkTo(n1)
```

Exercise 1.2: Degree Distribution

(a) Listing 4 shows source code.

Listing 4: Example Listing of source code

```
import AbstractNetwork
class DegreeDistribution:
    """Calculates a degree distribution for a network"""

    def __init__(self, network):
        """
        Inits DegreeDistribution with a network and calculate its distribution
        """
        # one further entry since 0 is degree 0 is included
        self.histogram = [0.0] * (network.maxDegree()+1)
        # increment degree distribution
        for i in range(0, network.size()):
                self.histogram[network.getNode(i).degree()] += 1.0
        # turn it into a real distribution
        for i in range(0, len(self.histogram)):
                self.histogram[i] /= float(network.size())

    def getNormalizedDistribution(self):
        '''
        Returns the computed normalized distribution
        '''
        return self.histogram
```

(b) Listing 5 shows source code.

Listing 5: Example Listing of source code

```
import matplotlib.pyplot as plt
import math

def plotDistributionComparison(histograms, legend, title):
    '''
    Plots a list of histograms with matching list of descriptions as the legend
```

```
    '''
    # adjust size of elements in histogram
    longest = 0
    # determine longest histogram
    for h in histograms:
        if(len(h) > longest):
            longest = len(h)
    # adapt other histograms
    for h in histograms:
        h.extend([0] * (longest - len(h)))

    # plots histograms
    for h in histograms:
        plt.plot(range(len(h)), h, marker = 'x')

    # remember: never forget labels! :-)
    plt.xlabel('')
    plt.ylabel('')

    # you don't have to do something here
    plt.legend(legend)
    plt.title(title)
    plt.tight_layout()# might throw a warning, no problem
    plt.show()

def poisson(k, l):
    '''
    Compute the poisson entry for k and lambda (l)
    '''
    k = float(k)
    l = float(l)
    if (k == 0):
        return(math.exp(-1.0*l))
    else:
        return (l/k)*poisson(k-1.0,l)

def getPoissonDistributionHistogram(num_nodes, num_links, k):
    '''
    Generates a Poisson distribution histogram up to k
    '''
    poissonHist = []
    lambda_ = 2.0*(float(num_links))/float(num_nodes)
    print "Lambda:", lambda_
    for i in range(0,num_links):
        if(i <= k):
            poissonHist.append(poisson(i, lambda_))
    return poissonHist
```

(c) Listing 6 shows source code.

Listing 6: Example Listing of source code

```
class Node:
    def __init__(self, identifier):
        """
        Sets node id and initialize empty node list that references its connected nodes
        """
        self.id = identifier
        self.nodelist = []

    def hasLinkTo(self, node):
        """
        Returns True if this node is connected to node asked for,
        False otherwise
        """
        for i in range(0, len(self.nodelist)):
            if self.nodelist[i] == node:
                return True
```

```python
            return False

    def addLinkTo(self, node):
        """
        Adds link from this node to parameter node (only if there is no link connection already
        does not automatically care for a link from parameter node to this node
        """
        if not self.hasLinkTo(node):
            self.nodelist.append(node)
            return True
        return False

    def degree(self):
        """
        Returns degree of this node
        """
        return len(self.nodelist)

    def __str__(self):
        """
        Returns id of node as string
        """
        return str(self.id)
```