```
"""
Implementation of problem 4.2
"""
import numpy as np
from numpy import *

# Node class that stores the compatability function at that node,
# the list of neighbor nodes it sends/receives messages to/from,
# the list of incoming messages and outgoing messages
class Node():
   # stores vector for node's singleton compatibility function
   compat = []
   # stores list of neighbors
   neigh = []

   # these are dictionaries that map a node's neighbors
   # to the message it's sending/receiving
   in_msgs = {}
   out_msgs = {}

   def __init__(self, compat):
      self.compat = compat

   # debugging functions
   def __repr__(self):
      string = "{neigh: " + str(self.neigh) + ", in_msgs: " + str(self.in_msgs) + ",
out_msgs: " + str(self.out_msgs) + "}\n"
      return string

   def __str__(self):
      string = "{neigh: " + str(self.neigh) + ", in_msgs: " + str(self.in_msgs) + ",
out_msgs: " + str(self.out_msgs) + "}\n"
      return string

# Tree class represents a tree as a set of nodes. Each node
# has a list of neighbors which represents the possible edges
# that exist in the tree.
class Tree():
   nodes = []
   size = 0

   def __init__(self, max_node):
      self.size = max_node

      for i in range(max_node):
         self.nodes.append(Node(self.single_compat(i)))

      self.nodes[0].neigh = [1,2]
      self.nodes[1].neigh = [0,3,4]
      self.nodes[2].neigh = [0,5]
      self.nodes[3].neigh = [1]
      self.nodes[4].neigh = [1]
      self.nodes[5].neigh = [2]

      # initialize messages for all edges uniformly at first
      # for incoming and outgoing edges
      for i in range(self.size):
         self.nodes[i].out_msgs = {}
         self.nodes[i].in_msgs = {}
         for j in self.nodes[i].neigh:
            self.nodes[i].out_msgs[j] = np.array([1,1])
            self.nodes[i].in_msgs[j] = np.array([1,1])
```

```
    # defines the compatibility function for a given node
    def single_compat(self, s):
        if s%2 == 0:
            return np.array([0.7, 0.3])
        else:
            return np.array([0.1, 0.9])

    # defines the compatibility function for a given edge (or pair of nodes)
    def edge_compat(self, s, t):
        if s == t:
            return 1.0
        else:
            return 0.45

    # runs sum-product algorithm on tree
    def sum_prod(self):
        numIter = 20
        for k in range(numIter):
            # for each node s
            for s in range(self.size):
                # for each node t that has an edge with s
                for t in self.nodes[s].neigh:
                    self.nodes[t].out_msgs[s] = np.array([0,0])
                    # compute for x_s = 0 and x_s = 1
                    for idx in range(2):
                        # get edge compatibility function
                        edge_compat0 = self.edge_compat(idx,0)
                        edge_compat1 = self.edge_compat(idx,1)
                        edge_compat = np.array([edge_compat0, edge_compat1])

                        # get t's singleton compatibility function
                        compat_t = self.nodes[t].compat

                        # compute final product of edge and singleton compatibility function
                        final_compat = compat_t[idx]*edge_compat

                        # compute product of all the received messages from neighbors
                        # that are NOT the one you are sending a message to
                        vec_prod = np.array([1,1])
                        for u in self.nodes[t].neigh:
                            if u != s:
                                vec_prod = vec_prod*self.nodes[u].out_msgs[t][idx]

                        result = final_compat * vec_prod
                        self.nodes[t].out_msgs[s] =  self.nodes[t].out_msgs[s] + result
                    self.nodes[s].in_msgs[t] = self.nodes[t].out_msgs[s]

        # compute marginals from formula:
        # p(x_s) = psi(x_s)* prod_over_neighbors(M*_(t->s)(x_s))
        for i in range(self.size):
            marg = np.prod(self.nodes[i].in_msgs.values(),0)
            marg = self.nodes[i].compat*marg
            marg_norm = marg/np.sum(marg)
            print "p(",i, ") = ", marg_norm

if __name__ == '__main__':
    t = Tree(6)
    t.sum_prod()
```

```
===============
CONSOLE OUTPUT:

p( 0 ) =  [ 0.56497099  0.43502901]
p( 1 ) =  [ 0.10106613  0.89893387]
p( 2 ) =  [ 0.57151058  0.42848942]
p( 3 ) =  [ 0.06281947  0.93718053]
p( 4 ) =  [ 0.54515564  0.45484436]
p( 5 ) =  [ 0.13357467  0.86642533]
```