DATASCI W261, Machine Learning at Scale

Assignement: week #9

Lei Yang (mailto:leiyang@berkeley.edu) | Michael Kennedy (mailto:mkennedy@ischool.berkeley.edu) | Natarajan Krishnaswami (mailto:natarajan@krishnaswami.org)

Due: 2016-03-19, 12PM PST

HW 9.0: Short answer questions

What is PageRank and what is it used for in the context of web search?

PageRank is a measure of web page quality based on the structure of the hyperlink graph. In the context of web search, PageRank is able to provide a topic-sensitive or customized ranking of webpage popularity according to query content and/or customer preference. PageRank is a measure of how frequently a page would be encountered by a stochastic process which has a asymptotic steady state. PageRank assumes a community of honest users who are not trying to “game” the measure.

What modifications have to be made to the webgraph in order to leverage the machinery of Markov Chains to compute the steady stade distibuton?

The transition matrix must be irreducible in order to leverage the Markov Chains machinery of steady state distribution. In case of dangling nodes (node with zero out-degree), the associated mass must be redistributed across the graph to other nodes, based on teleportation factor.

OPTIONAL: In topic-specific pagerank, how can we insure that the irreducible property is satified? (HINT: see HW9.4)

One approach to ensure irreducible is by requiring that nodes not reachable from in-topic nodes be removed from the network, this is hard to implement for parallel computing in that identifying the "unreachable" nodes is not easy. Alternatively, one can define a non-uniform damping factor such that all nodes will be participating in the PageRank, but the in-topic nodes will be heavily weighted according to the relative frequency in the population.

HW 9.1: MRJob implementation of basic PageRank

- Write a basic MRJob implementation of the iterative PageRank algorithm that takes sparse adjacency lists as input (as explored in HW 7).
- Make sure that you implementation utilizes teleportation (1-damping/the number of nodes in the network), and further, **distributes the mass of dangling nodes with each iteration** so that the output of each iteration is correctly normalized (sums to 1).
- [**NOTE**: The PageRank algorithm assumes that a random surfer (walker), starting from a random web page, chooses the next page to which it will move by clicking at random, with probability d, one of the hyperlinks in the current page. This probability is represented by a so-called ‘damping factor’ d, where d ∊ (0, 1). Otherwise, with probability (1 – d), the surfer jumps to any web page in the network. If a page is a dangling end, meaning it has no outgoing hyperlinks, the random surfer selects an arbitrary web page from a uniform distribution and “teleports” to that page]

As you build your code, use the test data

s3://ucb-mids-mls-networks/PageRank-test.txt Or under the Data Subfolder for HW7 on Dropbox with the same file name. (On Dropbox https://www.dropbox.com/sh/2c0k5adwz36lkcw/AAAAKsjQfF9uHfv-X9mCqr9wa?dl=0 (https://www.dropbox.com/sh/2c0k5adwz36lkcw/AAAAKsjQfF9uHfv-X9mCqr9wa?dl=0))

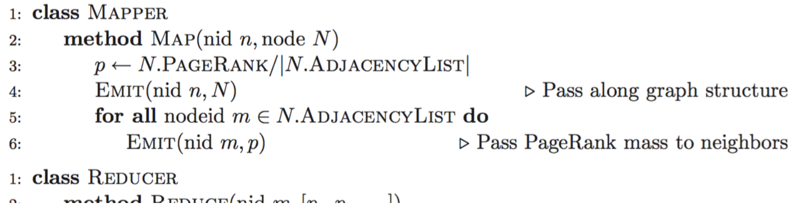
with teleportation parameter set to 0.15 (1-d, where d, the damping factor is set to 0.85), and crosscheck your work with the true result, displayed in the first image in the Wikipedia article:

https://en.wikipedia.org/wiki/PageRank (https://en.wikipedia.org/wiki/PageRank)

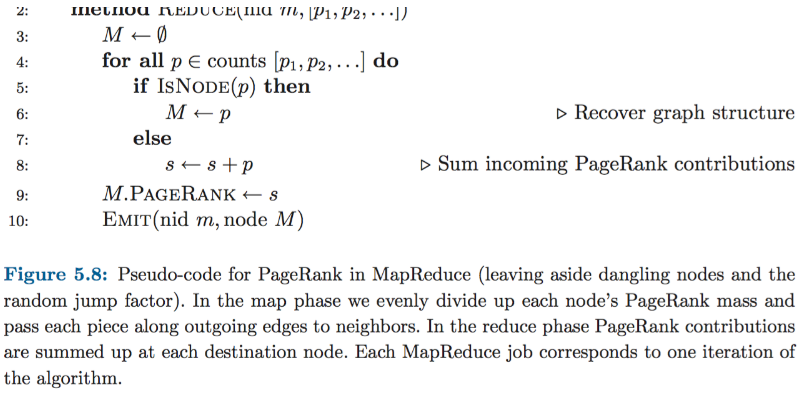
and here for reference are the corresponding PageRank probabilities:

A,0.033 B,0.384 C,0.343 D,0.039 E,0.081 F,0.039 G,0.016 H,0.016 I,0.016 J,0.016 K,0.016

PageRank with MapReduce



```
class MAPPER
    method MAP(nid $n$, node $N$)
        $p \leftarrow N.\text{PAGERANK}/|N.\text{ADJACENCYLIST}|$
        EMIT(nid $n, N$)                        ▷ Pass along graph structure
        for all nodeid $m \in N.\text{ADJACENCYLIST}$ do
            EMIT(nid $m, p$)                    ▷ Pass PageRank mass to neighbors
class REDUCER
    method REDUCE(nid $m$, [$p_1, p_2, \ldots$])
```



```
method REDUCE(nid $m$, $[p_1, p_2, \ldots]$)
    $M \leftarrow \emptyset$
    for all $p \in$ counts $[p_1, p_2, \ldots]$ do
        if ISNODE($p$) then
            $M \leftarrow p$                                  ▷ Recover graph structure
        else
            $s \leftarrow s + p$                    ▷ Sum incoming PageRank contributions
    $M$.PAGERANK $\leftarrow s$
    EMIT(nid $m$, node $M$)
```

Figure 5.8: Pseudo-code for PageRank in MapReduce (leaving aside dangling nodes and the random jump factor). In the map phase we evenly divide up each node's PageRank mass and pass each piece along outgoing edges to neighbors. In the reduce phase PageRank contributions are summed up at each destination node. Each MapReduce job corresponds to one iteration of the algorithm.

PageRank Iteration Job

In [52]:

```
%%writefile PageRankIter.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRankIter(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankIter, self).configure_options()
        self.add_passthrough_option(
            '--i', dest='init', default='0', type='int',
            help='i: run initialization iteration (default 0)')

    # mapper of first pass of the file (initialization)
    def mapper_job_init(self, _, line):
        # parse line
        nid, adj = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'adj = %s' %adj
        exec cmd
        # initialize node struct
        node = {'a':adj.keys(), 'p':0}
        rankMass = 1.0/len(adj)
        # emit node
        yield nid, node
        # emit pageRank mass
        for m in node['a']:
            yield m, rankMass

    # mapper for regular iteration (after initialization)
    def mapper_job_iter(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # distribute rank mass
        n_adj = len(node['a'])
        if n_adj > 0:
            rankMass = 1.0*node['p'] / n_adj
            # emit pageRank mass
            for m in node['a']:
                yield m, rankMass
        else:
            # track dangling mass with counter
            self.increment_counter('wiki_dangling_mass', 'mass', int(node['p']*1e10))
        # reset pageRank and emit node
        node['p'] = 0
        yield nid, node

    def debug(self):
        de = 'bug'

    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        rankMass, node = 0.0, None
        # loop through all arrivals
        for v in value:
            if isinstance(v, float):
```

```
            if isinstance(v, float):
                rankMass += v
            else:
                node = v
        # emit accumulative mass for nid
        if node:
            node['p'] += rankMass
            yield nid, node
        else:
            yield nid, rankMass

    # reducer for initialization pass --> need to handle dangling nodes
    def reducer_job_init(self, nid, value):
        # increase counter for node count
        self.increment_counter('wiki_node_count', 'nodes', 1)
        rankMass, node = 0.0, None
        # loop through all arrivals
        for v in value:
            if isinstance(v, float):
                rankMass += v
            else:
                node = v
        # handle dangling node, create node struct and add missing mass
        if not node:
            node = {'a':[], 'p':rankMass}
            self.increment_counter('wiki_dangling_mass', 'mass', int(1e10))
        else:
            node['p'] += rankMass
        # emit for next iteration
        yield nid, node

    # reducer for regular pass --> all nodes has structure available
    def reducer_job_iter(self, nid, value):
        rankMass, node = 0.0, None
        # loop through all arrivals
        for v in value:
            if isinstance(v, float):
                rankMass += v
            else:
                node = v
        # update pageRank
        node['p'] += rankMass
        # emit for next iteration
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '2',
        }
        return [MRStep(mapper=self.mapper_job_init if self.options.init else self.mapper_job_iter
                       , combiner=self.combiner
                       , reducer=self.reducer_job_init if self.options.init else self.reducer_job_
iter
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankIter.run()
```

```
Overwriting PageRankIter.py
```

In [1]:

```
##### unit test #####
#!python PageRankIter.py ./data/PageRank-test.txt --i 1 -r 'inline'  > iter1.t
#!python PageRankIter.py iter8.t --i 0 -r 'inline' > iter9.t
#!python PageRankIter.py 'hdfs:///user/leiyang/PageRank-test.txt' --i 1 -r 'hadoop' --output-dir
's3://w261.data/HW9_test'
```

PageRankDist Job

- applying damping and random jump factor
- redistribute dangling mass across the graph
- note: normalize the ranking number at last iteration

In [53]:

```
%%writefile PageRankDist.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.step import MRStep

class PageRankDist(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankDist, self).configure_options()
        self.add_passthrough_option(
            '--s', dest='size', default=0, type='int',
            help='size: node number (default 0)')
        self.add_passthrough_option(
            '--j', dest='alpha', default=0.15, type='float',
            help='jump: random jump factor (default 0.15)')
        self.add_passthrough_option(
            '--n', dest='norm', default=0, type='int',
            help='norm: normalize pageRank with graph size (default 0)')
        self.add_passthrough_option(
            '--m', dest='m', default=0, type='float',
            help='m: rank mass from dangling nodes (default 0)')

    def mapper_init(self):
        self.damping = 1 - self.options.alpha
        self.p_dangling = self.options.m / self.options.size

    # needed after initialization, after node number becomes available
    def mapper_norm(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # get final pageRank
        node['p'] = ((self.p_dangling + node['p'])*self.damping+self.options.alpha) / self.option
s.size
        yield nid, node

    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # get final pageRank
        node['p'] = (self.p_dangling + node['p']) * self.damping + self.options.alpha
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
        }
        return [MRStep(mapper_init=self.mapper_init
                       , mapper=self.mapper_norm if self.options.norm else self.mapper
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankDist.run()
```

```
Overwriting PageRankDist.py
```

In [138]:

```
##### unit test #####

#!python PageRankIter.py ./data/PageRank-test.txt --i 1 -r 'hadoop'  > iter1.t
#!python PageRankIter.py iter8.t --i 0 -r 'inline' > iter9.t


#!python PageRankDist.py iter1.t --n 1 --s 11 --j 0.15 --m 1.0 -r 'hadoop' # > iter2.t
#!python PageRankDist.py iter9.t --n 0 --s 11 --j 0.15 --m 0.41355561274607455   > iter10.t
```

PageRankSort Job

- sort pageRank with descending order

In [54]:

```
%%writefile PageRankSort.py
from mrjob.job import MRJob
from mrjob.step import MRStep
```

```
class PageRankSort(MRJob):
    #DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankSort, self).configure_options()
        self.add_passthrough_option(
            '--s', dest='size', default=0, type='int',
            help='size: node number (default 0)')
        self.add_passthrough_option(
            '--n', dest='top', default=100, type='int',
            help='size: node number (default 100)')

    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        cmd = 'node = %s' %node
        exec cmd
        yield node['p'], nid.strip('"')

    def reducer_init(self):
        self.i = 0
        self.total = 0

    def reducer(self, pageRank, nid):
        for n in nid:
            if self.i < self.options.top:
                self.i += 1
                self.total += pageRank
                yield n, pageRank/self.options.size

    def reducer_final(self):
        yield 'total mass: ', self.total/self.options.size

    def steps(self):
        jc = {
            'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.mapreduce.lib.partitio
n.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options': '-k1,1nr',
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '1',  # must be 1 for sorting
        }
        return [MRStep(mapper=self.mapper, reducer_init=self.reducer_init,
                       reducer=self.reducer, reducer_final=self.reducer_final
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankSort.run()
```

```
Overwriting PageRankSort.py
```

In [290]:

```
##### unit test #####
#!python PageRankSort.py iter10.t --s 11 --n 100 -r 'hadoop'
```

PageRankJoin Job

- find page name from index file for the top ranked pages

In [55]:

```
%%writefile PageRankJoin.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from subprocess import Popen, PIPE

class PageRankJoin(MRJob):
    #DEFAULT_PROTOCOL = 'json'

    def mapper_init(self):
        self.topRanks = {}
        # read rand list, prepare for mapper in-memory join
        cat = Popen(['cat', 'part-00000'], stdout=PIPE)
        for line in cat.stdout:
            nid, rank = line.strip().split('\t')
            self.topRanks[nid.strip('"')] = rank

    def mapper(self, _, line):
        # parse line
        name, nid, d_in, d_out = line.strip().split('\t')
        name, nid, d_in, d_out = line.strip().split('\t')
        if nid in self.topRanks:
            yield float(self.topRanks[nid]), '%s - %s' %(nid, name)

    def reducer(self, key, value):
        for v in value:
            yield key, v


    def steps(self):
        jc = {
            'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.mapreduce.lib.partitio
n.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options': '-k1,1nr',
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '1',  # must be 1 for sorting
        }
        return [MRStep(mapper_init=self.mapper_init
                       , mapper=self.mapper
                       , reducer=self.reducer
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankJoin.run()
```

```
Overwriting PageRankJoin.py
```

In [26]:

```
##### unit test #####
#!python PageRankJoin.py 'PageRankIndex' -r 'hadoop' --file 'rank' #> test.t
```

Helpers

In [4]:

```
%%writefile helper.py
#!/usr/bin/python

import requests

def getCounter(groupName, counterName, host = 'localhost'):
    # get job list
    getJobs = 'http://%s:19888/ws/v1/history/mapreduce/jobs' %host
    jobs = requests.get(getJobs).json()['jobs']['job']
    # get counters
    ts = max([job['finishTime'] for job in jobs])
    id = [job['id'] for job in jobs if job['finishTime'] == ts][0]
    getCounters = 'http://%s:19888/ws/v1/history/mapreduce/jobs/%s/counters' %(host, id)
    counterGroups = requests.get(getCounters).json()['jobCounters']['counterGroup']
    # loop through to counters to return value
    counters = [g['counter'] for g in counterGroups if g['counterGroupName']==groupName][0]
    totalValues = [c['totalCounterValue'] for c in counters if c['name']==counterName]
    return totalValues[0] if len(totalValues)==1 else None

def getCounters(groupName, host = 'localhost'):
    # get job list
    getJobs = 'http://%s:19888/ws/v1/history/mapreduce/jobs' %host
    jobs = requests.get(getJobs).json()['jobs']['job']
    # get counters
    ts = max([job['finishTime'] for job in jobs])
    id = [job['id'] for job in jobs if job['finishTime'] == ts][0]
    getCounters = 'http://%s:19888/ws/v1/history/mapreduce/jobs/%s/counters' %(host, id)
    counterGroups = requests.get(getCounters).json()['jobCounters']['counterGroup']
    # loop through to counters to return value
    counters = [g['counter'] for g in counterGroups if g['counterGroupName']==groupName]
    return {c['name']:c['totalCounterValue'] for c in counters[0]} if len(counters)==1 else []
```

```
Overwriting helper.py
```

In [125]:

```
##### unit test #####
#http://localhost:19888/ws/v1/history/mapreduce/jobs/job_1457742616221_0001/counters
#from helper import getCounter, getCounters
#getCounter('wiki_node_count', 'nodes', 'ec2-52-87-184-124.compute-1.amazonaws.com')
#getCounters('wiki_dangling_mass')
#getCounter('org.apache.hadoop.mapreduce.JobCounter', 'TOTAL_LAUNCHED_MAPS')
#getCounters('org.apache.hadoop.mapreduce.JobCounter', 'ec2-54-172-84-241.compute-1.amazonaws.co
m')
```

PageRank Driver

- initialize the process:
  - get node count, and dangling node count
  - redistribute loss mass, apply jump/damping factor
- get loss mass from counter
- iterately execute pageRank:
  - run pageRank process
  - get loss mass
- sort rank, normalize

In [139]:

```
%%writefile RunPageRank.py
#!/usr/bin/python

from PageRankIter import PageRankIter
from PageRankDist import PageRankDist
from PageRankSort import PageRankSort
from PageRankJoin import PageRankJoin
from helper import getCounter
from subprocess import call, check_output
from time import time
import sys, getopt, datetime, os

# parse parameter
if __name__ == "__main__":

    try:
        opts, args = getopt.getopt(sys.argv[1:], "hg:j:i:d:s:")
    except getopt.GetoptError:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index> -s <size>'
        sys.exit(2)
    if len(opts) != 5:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
            sys.exit(2)
        elif opt == '-g':
            graph = arg
        elif opt == '-j':
            jump = arg
        elif opt == '-i':
            n_iter = arg
        elif opt == '-d':
            index = arg
        elif opt == '-s':
            n_node = arg

start = time()
FNULL = open(os.devnull, 'w')
n_iter = int(n_iter)
doJoin = index!='NULL'
doInit = n_node=='0'
host = 'localhost'

print '%s: %s PageRanking on \'%s\' for %d iterations with damping factor %.2f ...' %(str(datetim
e.datetime.now()),
          'start' if doInit else 'continue', graph[graph.rfind('/')+1:], n_iter, 1-float(jump))

if doInit:
    # clear directory
    print str(datetime.datetime.now()) + ': clearing directory ...'
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/out'], stdout=FNULL)

    # creat initialization job
    init_job = PageRankIter(args=[graph, '--i', '1', '-r', 'hadoop', '--output-dir', 'hdfs:///use
r/leiyang/out'])

    # run initialization job
    print str(datetime.datetime.now()) + ': running iteration 1 ...'
    with init_job.make_runner() as runner:
        runner.run()

    # checking counters
    n_node = getCounter('wiki_node_count', 'nodes', host)
    n_node = getCounter('wiki_node_count', 'nodes', host)
    n_dangling = getCounter('wiki_dangling_mass', 'mass', host)/1e10
    print '%s: initialization complete: %d nodes, %d are dangling!' %(str(datetime.datetime.now
()), n_node, n_dangling)

    # run redistribution job
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])
    dist_job = PageRankDist(args=['hdfs:///user/leiyang/in/part*', '--s', str(n_node), '--j', jum
p, '--n', '0',
                                '--m', str(n_dangling), '-r', 'hadoop', '--output-dir', 'hdfs:///
user/leiyang/out'])
    print str(datetime.datetime.now()) + ': distributing loss mass ...'
    with dist_job.make_runner() as runner:
        runner.run()

# move results for next iteration
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

# create iteration job
iter_job = PageRankIter(args=['hdfs:///user/leiyang/in/part*', '--i', '0',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])
# run pageRank iteratively
i = 2 if doInit else 1
while(1):
    print str(datetime.datetime.now()) + ': running iteration %d ...' %i
    with iter_job.make_runner() as runner:
        runner.run()

    # check counter for loss mass
    mass_loss = getCounter('wiki_dangling_mass', 'mass', host)/1e10

    # move results for next iteration
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

    # run redistribution job
    dist_job = PageRankDist(args=['hdfs:///user/leiyang/in/part*', '--s', str(n_node), '--j', jum
p, '--n', '0',
                                '--m', str(mass_loss), '-r', 'hadoop', '--output-dir', 'hdfs:///u
ser/leiyang/out'])
    print str(datetime.datetime.now()) + ': distributing loss mass %.4f ...' %mass_loss
    with dist_job.make_runner() as runner:
        runner.run()

    if i == n_iter:
        break

    # if more iteration needed
    i += 1
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'], stdout=FNULL)

# run sort job
print str(datetime.datetime.now()) + ': sorting PageRank ...'
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/rank'], stdout=FNULL)
sort_job = PageRankSort(args=['hdfs:///user/leiyang/out/part*', '--s', str(n_node), '--n', '100',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/rank'])
with sort_job.make_runner() as runner:
    runner.run()

# run join job
if doJoin:
    print str(datetime.datetime.now()) + ': joining PageRank with index ...'
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/join'], stdout=FNULL)
    join_job = PageRankJoin(args=[index, '-r', 'hadoop', '--file', 'hdfs:///user/leiyang/rank/par
t-00000',
                                  '--output-dir', 'hdfs:///user/leiyang/join'])
    with join_job.make_runner() as runner:
        runner.run()

print "%s: PageRank job completes in %.1f minutes!\n" %(str(datetime.datetime.now()), (time()-sta
rt)/60.0)
call(['hdfs', 'dfs', '-cat', '/user/leiyang/join/p*' if doJoin else '/user/leiyang/rank/p*'])
```

```
Overwriting RunPageRank.py
```

In [140]:

```
##### unit test #####
!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.15 -i 10 -d 'NULL' -s '0'
```

```
#!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.15 -i 2 \
#-d 'hdfs:///user/leiyang/PageRankIndex' -s '11'
```

```
2016-03-18 21:21:13.094958: start PageRanking on 'PageRank-test.txt' for 10 iterations with dampin
g factor 0.85 ...
2016-03-18 21:21:13.094996: clearing directory ...
2016-03-18 21:21:16.265863: running iteration 1 ...
No handlers could be found for logger "mrjob.conf"
2016-03-18 21:21:47.714757: initialization complete: 11 nodes, 1 are dangling!
2016-03-18 21:21:49.239699: distributing loss mass ...
2016-03-18 21:22:15.480366: running iteration 2 ...
2016-03-18 21:22:48.948444: distributing loss mass 0.6523 ...
2016-03-18 21:23:16.540084: running iteration 3 ...
2016-03-18 21:23:50.296355: distributing loss mass 0.4174 ...
2016-03-18 21:24:16.599654: running iteration 4 ...
2016-03-18 21:24:49.985785: distributing loss mass 0.7042 ...
2016-03-18 21:25:16.321551: running iteration 5 ...
2016-03-18 21:25:51.455971: distributing loss mass 0.4136 ...
2016-03-18 21:26:18.735261: running iteration 6 ...
2016-03-18 21:26:51.582840: distributing loss mass 0.4254 ...
2016-03-18 21:27:19.772219: running iteration 7 ...
2016-03-18 21:27:51.399169: distributing loss mass 0.3753 ...
2016-03-18 21:28:17.655096: running iteration 8 ...
2016-03-18 21:28:49.926164: distributing loss mass 0.3812 ...
2016-03-18 21:29:16.041041: running iteration 9 ...
2016-03-18 21:29:49.249695: distributing loss mass 0.3659 ...
2016-03-18 21:30:16.576019: running iteration 10 ...
2016-03-18 21:30:47.712945: distributing loss mass 0.3660 ...
2016-03-18 21:31:10.959484: sorting PageRank ...
2016-03-18 21:31:40.667650: PageRank job completes in 10.5 minutes!

"B"     0.3632359489815919
"C"     0.36288372803185465
"E"     0.08114525762242993
"F"     0.03938466341855663
"D"     0.03938466341855663
"A"     0.032930101785246045
"K"     0.01620712734363636
"I"     0.01620712734363636
"G"     0.01620712734363636
"J"     0.01620712734363636
"H"     0.01620712734363636
"total mass: "  0.9999999999764178
```

HW 9.2: Exploring PageRank teleportation and network plots

In order to overcome problems such as disconnected components, the damping factor (a typical value for d is 0.85) can be varied. Using the graph in HW 9.1, plot the test graph (using networkx, https://networkx.github.io/ (https://networkx.github.io/)) for several values of the damping parameter alpha, so that each nodes radius is proportional to its PageRank score. In particular you should do this for the following damping factors: $[0, 0.25, 0.5, 0.75, 0.85, 1]$. Note your plots should look like the following:

https://en.wikipedia.org/wiki/PageRank#/media/File:PageRanks-Example.svg (https://en.wikipedia.org/wiki/PageRank#/media/File:PageRanks-Example.svg)

Run 10 iterations with each parameter

```
In [33]: %load_ext autoreload
         %autoreload 2
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 1 -i 3 -d 'NULL'  -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_0
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.75 -i 10 -d 'NULL' -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_1
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.5 -i 10 -d 'NULL' -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_2
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.25 -i 10 -d 'NULL' -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_3
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0.15 -i 10 -d 'NULL' -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_4
         #!python RunPageRank.py -g 'hdfs:///user/leiyang/PageRank-test.txt' -j 0 -i 10 -d 'NULL' -s '0'
         #!hdfs dfs -cat /user/leiyang/rank/p* > HW_9_2_5
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Graphing function

Graphing function

In [118]:

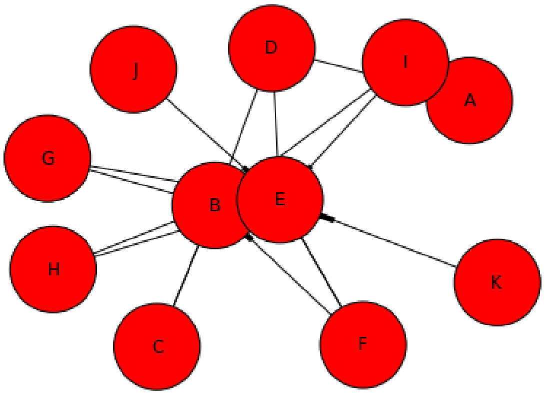
```
from matplotlib import pyplot as plt
import networkx as nx

def drawGraph(pagerank, nSize):
    # define the graph from adjacency matrix
    G = nx.DiGraph()
    with open('./data/PageRank-test.txt') as f:
        for node in f.readlines():
            source, adj = node.strip().split('\t')
            cmd = 'adj = %s' %adj
            exec cmd
            for d in adj:
                G.add_edge(source, d)
            G.node[source]['state'] = source
    G.add_node('A')
    G.node['A']['state'] = 'A'

    # define node size
    ranks = {}
    with open(pagerank) as f:
        for line in f.readlines():
            nid, rank = line.strip().split('\t')
            nid = nid.strip('"')
            if len(nid) == 1:
                ranks[nid] = float(rank)
    norm = max(ranks.values())
    size =[ranks[n]*nSize/norm for n in G.nodes()]

    # draw the graph
    pos = nx.spring_layout(G)
    nx.draw(G, pos, node_size = size)
    node_labels = nx.get_node_attributes(G,'state')
    nx.draw_networkx_labels(G, pos, labels = node_labels)
    plt.show()
```

Damping factor = 0

- total random jump

In [119]:

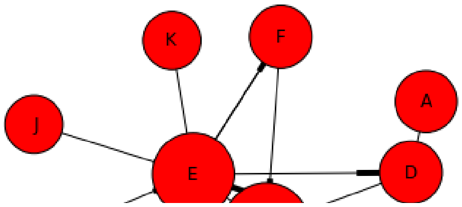
```
drawGraph('./data/HW_9_2_0', 3500)
```

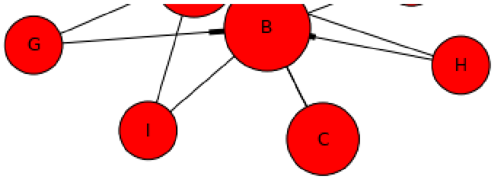


Damping factor = 0.25

In [120]:

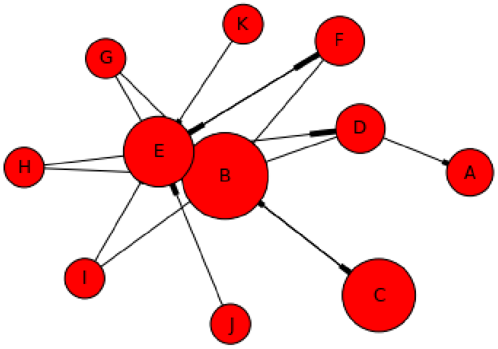
```
drawGraph('./data/HW_9_2_1', 3500)
```





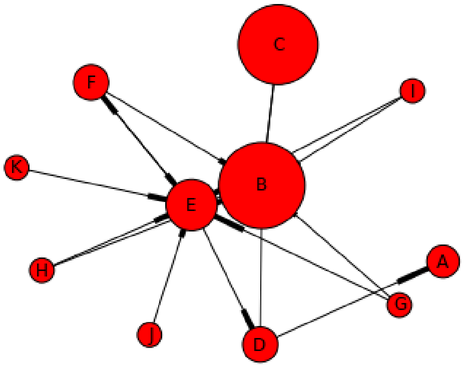
Damping factor = 0.5

In [121]:
```
drawGraph('./data/HW_9_2_2', 3500)
```



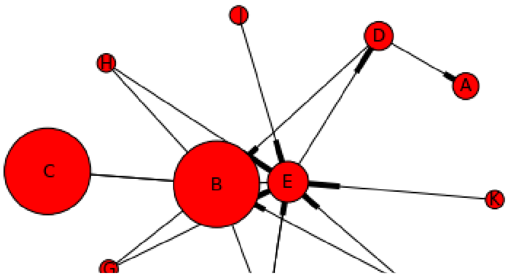
Damping factor = 0.75

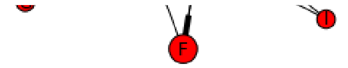
In [122]:
```
drawGraph('./data/HW_9_2_3', 3500)
```



Damping factor = 0.85

In [123]:
```
drawGraph('./data/HW_9_2_4', 3500)
```

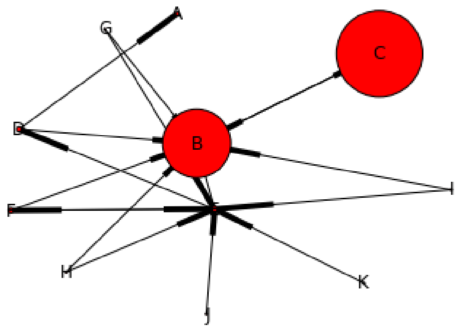




Damping factor = 1

- no teleportation at all

```
In [124]: drawGraph('./data/HW_9_2_5', 3500)
```



HW 9.3: Applying PageRank to the Wikipedia hyperlinks network

- Run your PageRank implementation on the Wikipedia dataset for 10 iterations, and display the top 100 ranked nodes (with alpha = 0.85).
- Run your PageRank implementation on the Wikipedia dataset for 50 iterations, and display the top 100 ranked nodes (with teleportation factor of 0.15).
- Have the top 100 ranked pages changed?
  - Comment on your findings.
  - Plot the pagerank values for the top 100 pages resulting from the 50 iterations run.
  - Then plot the pagerank values for the same 100 pages that resulted from the 10 iterations run.

Top 100 PageRank on 10 iterations

```
In [108]: #!python RunPageRank.py -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -j 0.15 -i 10 \
          #-d 'hdfs:///user/leiyang/indices.txt' -s '0'
          !cat ./data/wiki_10_join
```

```
0.0014614491942438349   "13455888 - United States"
0.0006663317375798755   "1184351 - Animal"
0.0006398051875506791   "4695850 - France"
0.0005748538826286757   "5051368 - Germany"
0.0004503047142785041   "1384888 - Arthropod"
0.00044660099098876925  "2437837 - Canada"
0.0004448182480165896   "6113490 - Insect"
0.000444203734598283    "7902219 - List of sovereign states"
0.0004329952449265189   "13425865 - United Kingdom"
0.000427885336130834    "6076759 - India"
0.00042327595490625425  "4196067 - England"
0.0003981702958692759 6 "6172466 - Iran"
0.0003854336708092153   "14112583 - World War II"
0.0003631670660462813   "10390714 - Poland"
0.00034383110763123157  "15164193 - village"
0.0003383478652193903   "3191491 - Countries of the world"
0.0003293524626579213   "6416278 - Japan"
0.00032896996556415974  "6237129 - Italy"
0.00032632071986423984  "7835160 - List of countries"
0.0003250758881557822   "1516699 - Australia"
0.0003131434418030617   "13725487 - Voivodeships of Poland"
0.0003095941242456258   "9276255 - National Register of Historic Places"
0.0003080954689799150 5 "7576704 - Lepidoptera"
0.0003035425666087729   "10469541 - Powiat"
0.0002979533522103301   "5154210 - Gmina"
0.0002857902942672035   "12836211 - The New York Times"
0.00028347554322906003  "7990491 - London"
```

```
0.00028547554322906003  "7990491 - London"
0.0002690621118260582   "4198751 - English language"
0.00026401327504910687  "2797855 - China"
0.0002610656557463978   "11253108 - Russia"
0.00025755868482122047  "9386580 - New York City"
0.0002550899370909594   "3603527 - Departments of France"
0.00025104301138297773  "12074312 - Spain"
0.0002487901833478194   "3069099 - Communes of France"
0.0002454573288537552   "14881689 - moth"
0.0002448490318480156   "2155467 - Brazil"
0.00023872444275286543  "1441065 - Association football"
0.00023335074718638498  "14503460 - association football"
0.00022060503331685166  "2396749 - California"
0.00021509725578691428  "3191268 - Counties of Iran"
0.00021468682890528565  "10566120 - Provinces of Iran"
0.00021137909656219 5   "2614581 - Central European Time"
0.00021132415993817432  "11147327 - Romania"
0.00020715963504228723  "1637982 - Bakhsh"
0.00020338117266915535  "12430985 - Sweden"
0.0002026232339501364   "11245362 - Rural Districts of Iran"
0.00019701920174091924  "9355455 - Netherlands"
0.00019142274072843148  "10527224 - Private Use Areas"
0.00019074389254287321  "14112408 - World War I"
0.00018818343629020115  "2614578 - Central European Summer Time"
0.00018809311929740536  "9391762 - New York"
0.0001871031700113471   "8697871 - Mexico"
0.00018685330248519627  "6172167 - Iran Standard Time"
0.00018540138485575856  "981395 - AllMusic"
0.00017885001537615815  "6171937 - Iran Daylight Time"
0.00017834740118144265  "5490435 - Hangul"
0.00017325786783489066  "11582765 - Scotland"
0.00016954981151296857  "14725161 - gene"
0.00016767695230721733  "12067030 - Soviet Union"
0.00016731685713821892  "9562547 - Norway"
0.00016548126934061455  "994890 - Allmusic"
0.00016067308336923405  "9997298 - Paris"
0.00016052821854839323  "9394907 - New Zealand"
0.0001590426968518198   "13280859 - Turkey"
0.00015776886151301826  "10345830 - Plant"
0.00015530367287750363  "4978429 - Geographic Names Information System"
0.00015495000505848502  "12447593 - Switzerland"
0.00015322797217141157  "8019937 - Los Angeles"
0.00014889113577784647  "11148415 - Romanize"
0.00014788104548259728  "13432150 - United States Census Bureau"
0.00014712672457525987  "4344962 - Europe"
0.00014192897863341285  "1175360 - Angiosperms"
0.0001413128675411494   "12038331 - South Africa"
0.0001390960411124565   "14565507 - census"
0.00013781320406146952  "4624519 - Flowering plant"
0.00013627013406909209  "1523975 - Austria"
0.00013494958518642523  "14981725 - protein"
0.00013474185796926125  "13328060 - U.S. state"
0.00013073867121151 97  "1332806 - Argentina"
0.0001302313942366781   "10399499 - Political divisions of the United States"
0.00013006969329161375  "14963657 - population density"
0.00012834313002134818  "2578813 - Catholic Church"
0.00012828888425440012  "2826544 - Chordate"
0.00012723774168965516  "1575979 - BBC"
0.00012713810216423896  "1813634 - Belgium"
0.00012404627523796507  "2778099 - Chicago"
0.0001208387713656      "13853369 - Washington, D.C."
0.00012028369796106582  "9924814 - Pakistan"
0.00011582953101167437  "4568647 - Finland"
0.0001144336932340961   "12785678 - The Guardian"
0.00011442817412578247  "7467127 - Latin"
0.00011425740363317814  "9742161 - Ontario"
0.00011368488062447509  "3328327 - Czech Republic"
0.00011328616898369108  "10246542 - Philippines"
0.00011326731774131182  "3591832 - Denmark"
0.00011319055333767748  "5274313 - Greece"
0.00011298651394752375  "14727077 - genus"
0.00011246839612227319  "14709489 - football (soccer)"
0.00011223456345201143  "5908108 - Hungary"
0.00011220084056744828  "3973000 - Eastern European Time"
```

Top 100 PageRank on 50 iterations

In [109]:

```
#!python RunPageRank.py -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -j 0.15 -i 50 \
#-d 'hdfs:///user/leiyang/indices.txt' -s '0'
!cat ./data/wiki_50_join
```

```
0.0014615599816380814   "13455888 - United States"
0.0006660177936038597   "1184351 - Animal"
0.0006396773757180422   "4695850 - France"
0.0005747671982893716   "5051368 - Germany"
0.0004501232221973807   "1384888 - Arthropod"
0.00044667005168115624  "2437837 - Canada"
0.00044463224402460465  "6113490 - Insect"
0.00044387869965694206  "7902219 - List of sovereign states"
0.00043314218173262273  "13425865 - United Kingdom"
0.00042770776770628867  "6076759 - India"
0.00042341679596246464  "4196067 - England"
0.000397826042012343    "6172466 - Iran"
0.00038548623796188223  "14112583 - World War II"
0.00036266653367941786  "10390714 - Poland"
0.00034358745300642004  "15164193 - village"
0.0003380496128621886   "3191491 - Countries of the world"
0.0003292203268728178   "6416278 - Japan"
0.00032899474579557773  "6237129 - Italy"
0.00032620175381522067  "7835160 - List of countries"
0.00032511085571704744  "1516699 - Australia"
0.00031268227722189133  "13725487 - Voivodeships of Poland"
0.0003095692741012243   "9276255 - National Register of Historic Places"
0.00030798064678708647  "7576704 - Lepidoptera"
0.0003031203814549852   "10469541 - Powiat"
0.0002975477873102353   "5154210 - Gmina"
0.00028603760467156336  "12836211 - The New York Times"
0.0002836201779820457   "7990491 - London"
0.00026905355560406986  "4198751 - English language"
0.00026401414743470433  "2797855 - China"
0.0002609847438047289   "11253108 - Russia"
0.00025769760112647513  "9386580 - New York City"
0.0002549708627794919   "3603527 - Departments of France"
0.0002510220915988287   "12074312 - Spain"
0.00024867559431367516  "3069099 - Communes of France"
0.00024536414137746424  "14881689 - moth"
0.00024471986910370725  "2155467 - Brazil"
0.00023864828925466942  "1441065 - Association football"
0.00023330403431633522  "14503460 - association football"
0.00022063223474869253  "2396749 - California"
0.0002149554605041878   "3191268 - Counties of Iran"
0.0002145445586070783   "10566120 - Provinces of Iran"
0.0002112031979712803   "2614581 - Central European Time"
0.00021118711279724468  "11147327 - Romania"
0.00020703164634469528  "1637982 - Bakhsh"
0.00020330214007671839  "12430985 - Sweden"
0.00020252992610044653  "11245362 - Rural Districts of Iran"
0.00019701419936357027  "9355455 - Netherlands"
0.00019139065958623014  "10527224 - Private Use Areas"
0.0001907835867254496   "14112408 - World War I"
0.0001881715264835644   "9391762 - New York"
0.0001880220706466143   "2614578 - Central European Summer Time"
0.00018704386395116136  "8697871 - Mexico"
0.000186732570547333    "6172167 - Iran Standard Time"
0.00018522886640443312  "981395 - AllMusic"
0.00017874919317338242  "6171937 - Iran Daylight Time"
0.00017831292465815966  "5490435 - Hangul"
0.0001733486967173185   "11582765 - Scotland"
0.00016948367863767668  "14725161 - gene"
0.00016765208013989067  "12067030 - Soviet Union"
0.0001672147998973972   "9562547 - Norway"
0.00016539998297863457  "994890 - Allmusic"
0.000160696299067241    "9997298 - Paris"
0.00016052347832283404  "9394907 - New Zealand"
0.000159006634081669    "13280859 - Turkey"
0.00015761805655622823  "10345830 - Plant"
0.00015527176841381843  "4978429 - Geographic Names Information System"
0.00015493020349201966  "12447593 - Switzerland"
0.00015329019745546487  "8019937 - Los Angeles"
0.00014883434413415624  "11148415 - Romanize"
0.00014785698802403393  "13432150 - United States Census Bureau"
0.00014711081037696502  "4344962 - Europe"
0.0001418440693202807   "1175360 - Angiosperms"
0.00014129930023312176  "12038331 - South Africa"
0.00013906672709870576  "14565507 - census"
0.00013764660835955233  "4624519 - Flowering plant"
0.00013624635676277595  "1523975 - Austria"
0.00013489587624568975  "14981725 - protein"
```

```
        0.00013474291497490742  "13328060 - U.S. state"
        0.00013069327812236255  "1332806 - Argentina"
        0.0001302063045995383   "10399499 - Political divisions of the United States"
        0.00013003778509485187  "14963657 - population density"
        0.00012841221637639494  "2578813 - Catholic Church"
        0.0001282046223090247   "2826544 - Chordate"
        0.00012732350077934083  "1575979 - BBC"
        0.00012715357035669744  "1813634 - Belgium"
        0.00012410867750278315  "2778099 - Chicago"
        0.00012093630817512696  "13853369 - Washington, D.C."
        0.00012024237237818629  "9924814 - Pakistan"
        0.0001157792173901934   "4568647 - Finland"
        0.00011450767260764365  "12785678 - The Guardian"
        0.00011447345057904762  "7467127 - Latin"
        0.00011430180463041426  "9742161 - Ontario"
        0.00011359377946447987  "3328327 - Czech Republic"
        0.0001132654264571577   "10246542 - Philippines"
        0.00011323587651842911  "3591832 - Denmark"
        0.00011319290359763953  "5274313 - Greece"
        0.00011291110378232016  "14727077 - genus"
        0.00011241645992615994  "14709489 - football (soccer)"
        0.00011218707479056177  "5908108 - Hungary"
        0.00011212013199291848  "3973000 - Eastern European Time"
```

Comments

- the top 100 node are identical except for two of them swapped their place ("9391762 - New York" and "2614578 - Central European Summer Time")
- convergence is pretty fast for this graph with wiki nodes

HW 9.4: Topic-specific PageRank implementation using MRJob

Modify your PageRank implementation to produce a topic specific PageRank implementation, as described in:

http://www-cs-students.stanford.edu/~taherh/papers/topic-sensitive-pagerank.pdf (http://www-cs-students.stanford.edu/~taherh/papers/topic-sensitive-pagerank.pdf)

Note in this article that there is a special caveat to ensure that the transition matrix is irreducible. This caveat lies in footnote 3 on page 3:

```
A minor caveat: to ensure that M is irreducible when p
contains any 0 entries, nodes not reachable from nonzero
nodes in p should be removed. In practice this is not problematic.
```

and must be adhered to for convergence to be guaranteed.

Run topic specific PageRank on the following randomly generated network of 100 nodes:

s3://ucb-mids-mls-networks/randNet.txt (also available on Dropbox)

which are organized into ten topics, as described in the file:

s3://ucb-mids-mls-networks/randNet_topics.txt (also available on Dropbox)

Since there are 10 topics, your result should be 11 PageRank vectors (one for the vanilla PageRank implementation in 9.1, and one for each topic with the topic specific implementation). Print out the **top ten** ranking nodes and their topics for each of the 11 versions, and comment on your result. Assume a teleportation factor of **0.15** in all your analyses.

One final and important comment here: please consider the requirements for irreducibility with topic-specific PageRank. In particular, the literature ensures irreducibility by requiring that nodes not reachable from in-topic nodes be removed from the network.

This is not a small task, especially as it it must be performed separately for each of the (10) topics.

So, instead of using this method for irreducibility, please comment on why the literature's method is difficult to implement, and what what extra computation it will require. Then for your code, please use the alternative, non-uniform damping vector:

$v_{ji} = \frac{\beta}{|T_j|}$; if node i lies in topic T_j

$v_{ji} = \frac{1-\beta}{N-|Tj|}$; if node i lies outside of topic T_j

for β in (0,1) close to 1.

With this approach, you will not have to delete any nodes. If $\beta > 0.5$, PageRank is topic-sensitive, and if $\beta < 0.5$, the PageRank is anti-topic-sensitive. For any value of β irreducibility should hold, so please try $\beta = 0.99$ and perhaps some other values locally, on the smaller networks.

Comment on difficulty of "not reachable nodes"

With MapReduce framework, it is challenging to identify unreachable nodes from in-topic node, in that the whole graph is processed in parallel, and it's hard to track a path in order to evaluate the "reachability" of each node

Implementation Notes - two changes on the vanilla PageRank implementation:

1. during mass distribution step, in *PageRankIter* job:

- maintain an array of ranks for each topic
- distribute and accumulate rank number separately for each rank number

In [112]:

```
%%writefile PageRankIter_T.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRankIter_T(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankIter_T, self).configure_options()
        self.add_passthrough_option(
            '--i', dest='init', default='0', type='int',
            help='i: run initialization iteration (default 0)')
        self.add_passthrough_option(
            '--n', dest='n_topic', default='0', type='int',
            help='n: number of topics (default 0)')

    def mapper_job_init(self, _, line):
        # parse line
        nid, adj = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'adj = %s' %adj
        exec cmd
        # initialize node struct
        node = {'a':adj.keys(), 'p':[0]*(self.options.n_topic + 1)}
        # vanillar PageRank and topic sensitive PageRank
        rankMass = [1.0 / len(adj)] * (self.options.n_topic + 1)
        # emit node
        yield nid, node
        # emit pageRank mass
        for m in node['a']:
            yield m, rankMass

    def mapper_job_iter(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # distribute rank mass
        n_adj = len(node['a'])
        if n_adj > 0:
            rankMass = [x / n_adj for x in node['p']]
            # emit pageRank mass
            for m in node['a']:
                yield m, rankMass
        else:
            # track dangling mass for each topic with counters
            for i in range(self.options.n_topic+1):
                self.increment_counter('wiki_dangling_mass', 'topic_%d' %i, int(node['p'][i]*1e1
0))
        # reset pageRank and emit node
        node['p'] = [0]*(self.options.n_topic+1)
        yield nid, node

    def debug(self):
        de = 'bug'

    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        rankMass, node = [0]*(self.options.n_topic+1), None
        # loop through all arrivals
        for v in value:
            if isinstance(v, list):
                rankMass = [a+b for a,b in zip(rankMass, v)]
            else:
                node = v
        # emit accumulative mass for nid
        if node:
            node['p'] = [a+b for a,b in zip(rankMass, node['p'])]
            yield nid, node
        else:
            yield nid, rankMass

    # reducer for initialization pass --> need to handle dangling nodes
    def reducer_job_init(self, nid, value):
        # increase counter for node count
        self.increment_counter('wiki_node_count', 'nodes', 1)
        rankMass, node = [0]*(self.options.n_topic+1), None
        # loop through all arrivals
        for v in value:
            if isinstance(v, list):
                rankMass = [a+b for a,b in zip(rankMass, v)]
            else:
                node = v
        # handle dangling node, create node struct and add missing mass
        if not node:
            node = {'a':[], 'p':rankMass}
            for i in range(self.options.n_topic+1):
                self.increment_counter('wiki_dangling_mass', 'mass_%d' %i, int(1e10))
        else:
            node['p'] = [a+b for a,b in zip(rankMass, node['p'])]
        # emit for next iteration
        yield nid, node

    # reducer for regular pass --> all nodes has structure available
    def reducer_job_iter(self, nid, value):
        rankMass, node = [0]*(self.options.n_topic+1), None
        # loop through all arrivals
        for v in value:
            if isinstance(v, list):
                rankMass = [a+b for a,b in zip(rankMass, v)]
            else:
                node = v
        # update pageRank
        node['p'] = [a+b for a,b in zip(rankMass, node['p'])]
        # emit for next iteration
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '2',
        }
        return [MRStep(mapper=self.mapper_job_init if self.options.init else self.mapper_job_iter
                       , combiner=self.combiner
                       , reducer=self.reducer_job_init if self.options.init else self.reducer_job
_iter
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankIter_T.run()
```

```
Overwriting PageRankIter_T.py
```

In [9]:
```
##### unit test #####
#!python PageRankIter_T.py ./data/randNet.txt --i 1 --n 10 -r 'inline' > test.t
#!python PageRankIter_T.py ./data/PageRank-test.txt --i 1 --n 10 -r 'hadoop' > test.t
#!python PageRankIter_T.py test.t --i 0 --n 2 -r 'inline' > test2.t
```

2. during mass redistribution/adjustment step, in *PageRankDist* job:

$$\begin{gathered} p' = \alpha\Big(\frac{1}{|G|}\Big) \\ + (1-\alpha)\Big(\frac{m}{|G|} + p\Big), \\ |G| = N \end{gathered}$$

- instead of all nodes receiving $\frac{1}{N}$ uniformly in the teleportation term, nodes belong to topic T_j will receive $\frac{\beta}{|T_j|}$, and others receive $\frac{1-\beta}{N-|Tj|}$

- intuitively with $\beta = 0.99$, the teleportation will transition to nodes in T_j with a 99% probability, while only 1% chance to nodes outside of T_j
- because β and $|T_j|$ are both known, we will evaluate v_{ji} before hand and load all 10 of them in *mapper_init* phase, and apply it in the mapper.

In [12]:

```
%%writefile PageRankDist_T.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from subprocess import Popen, PIPE

class PageRankDist_T(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankDist_T, self).configure_options()
        self.add_passthrough_option(
            '--s', dest='size', default=0, type='int',
            help='size: node number (default 0)')
        self.add_passthrough_option(
            '--j', dest='alpha', default=0.15, type='float',
            help='jump: teleport factor (default 0.15)')
        self.add_passthrough_option(
            '--b', dest='beta', default=0.99, type='float',
            help='beta: topic bias factor (default 0.99)')
        self.add_passthrough_option(
            '--m', dest='m', default='', type='str',
            help='m: rank mass from dangling nodes')
        self.add_passthrough_option(
            '--w', dest='wiki', default=0, type='int',
            help='w: if it is wiki data (default 1)')

    def mapper_init(self):
        # load topic file and count
        T_j, self.T_index = {}, {}
        cat = Popen(['cat', 'randNet_topics.txt'], stdout=PIPE)
        for line in cat.stdout:
            nid, topic = line.strip().split('\t')
            self.T_index[nid] = topic
            T_j[topic] = 1 if topic not in T_j else (T_j[topic]+1)

        # prepare adjustment factors
        self.damping = 1 - self.options.alpha
        cmd = 'm = %s' %self.options.m
        exec cmd
        # assuming here -m is specified with a list syntax string
        self.p_dangling = [1.0*x / self.options.size for x in m]
        # for each topic, get topic bias
        self.v_ij = [[1, 1]]*(len(T_j)+1)
        N, b = self.options.size, self.options.beta
        for t in T_j:
            self.v_ij[int(t)] = [(1-b)*N/(N-T_j[t]), b*N/T_j[t]]


    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # get final pageRank
        for i in range(len(self.v_ij)):
            vij = self.v_ij[i][i==int(self.T_index[nid])]
            node['p'][i] = (self.p_dangling[i]+node['p'][i])*self.damping + self.options.alpha*vij
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
        }
        return [MRStep(mapper_init=self.mapper_init
                       , mapper=self.mapper
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankDist_T.run()
    PageRankDist_T.run()
```

```
Overwriting PageRankDist_T.py
```

In [11]:

```
##### unit test #####
# no dangling nodes for randNet data
#!python PageRankDist_T.py test.t --m '[1]*11' --s '100' --file './data/randNet_topics.txt'  -r 'h
adoop' > test2.t
```

PageRankSort job

- emit (vector_ID, pageRank)~topic_id as key~value pair
- partition on vector_ID, secondary sort (-k2,2nr) on pageRank
- print out top 10 for each vector_ID

In [2]:

```
%%writefile PageRankSort_T.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from subprocess import Popen, PIPE

class PageRankSort_T(MRJob):
    DEFAULT_PROTOCOL = 'json'
    PARTITIONER = 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner'

    def mapper_init(self):
        # load topic file and count
        self.T_index = {}
        cat = Popen(['cat', 'randNet_topics.txt'], stdout=PIPE)
        for line in cat.stdout:
            nid, topic = line.strip().split('\t')
            self.T_index[nid] = topic

    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # emit (vector_ID, pageRank)~topic_id
        for i in range(len(node['p'])):
            yield (i, node['p'][i]), self.T_index[nid]

    def reducer_init(self):
        self.current_v = None
        self.i = 0
        self.top = 10

    def reducer(self, key, value):
        if self.current_v != key[0]:
            self.current_v = key[0]
            self.i = 0
            yield '====== Top 10 for topic %d ======' %self.current_v, ''
        if self.i < self.top:
            self.i += 1
            for v in value:
                yield key, v


    def steps(self):
        jc = {
            'mapreduce.job.maps': '3',
            'mapreduce.job.reduces': '3',
            'mapreduce.partition.keypartitioner.options': '-k1,1',
            'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.mapreduce.lib.partitio
n.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options': '-k1,1 -k2,2nr',
            'stream.num.map.output.key.fields': '2',
            'mapreduce.map.output.key.field.separator': ' ',
            'stream.map.output.field.separator': ' ',
        }
        return [MRStep(mapper_init=self.mapper_init
                      , mapper=self.mapper
                      , reducer_init=self.reducer_init
                      , reducer=self.reducer
                      , jobconf = jc
                     )
               ]
```

```
if __name__ == '__main__':
    PageRankSort_T.run()
```

```
Overwriting PageRankSort_T.py
```

In [40]:

```
##### unit test #####
#!python PageRankSort_T.py 'randNet_pr' --file './data/randNet_topics.txt' -r 'hadoop' > test.sort
2
```

Driver for topic-sensitive-pagerank

- we need to retrieve a group of counters, which represent loss mass for each topic.
- in this implementation, the counter group name is **wiki_dangling_mass**, and counter names are **mass_[i]**, with mass_0 for vanilla pageRank, and mass_1 for topic 1 etc.
- the loss values are passed as list syntax string to the job, which will transfer it to a list

In [47]:

```
%%writefile RunPageRank_T.py
#!/usr/bin/python

from PageRankIter_T import PageRankIter_T
from PageRankDist_T import PageRankDist_T
from PageRankSort_T import PageRankSort_T
from PageRankJoin import PageRankJoin
from helper import getCounter, getCounters
from subprocess import call, check_output
from time import time
import sys, getopt, datetime, os

# parse parameter
if __name__ == "__main__":

    try:
        opts, args = getopt.getopt(sys.argv[1:], "hg:j:i:d:s:")
    except getopt.GetoptError:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index> -s <size>'
        sys.exit(2)
    if len(opts) != 5:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
            sys.exit(2)
        elif opt == '-g':
            graph = arg
        elif opt == '-j':
            jump = arg
        elif opt == '-i':
            n_iter = arg
        elif opt == '-d':
            index = arg
        elif opt == '-s':
            n_node = arg

start = time()
FNULL = open(os.devnull, 'w')
n_iter = int(n_iter)
doJoin = index!='NULL'
doInit = n_node=='0'
host = 'localhost'

print '%s: %s topic sensitive PageRanking on \'%s\' for %d iterations with damping factor %.2f
...' %(str(datetime.datetime.now()),
          'start' if doInit else 'continue', graph[graph.rfind('/')+1:], n_iter, 1-float(jump))

if doInit:
    # clear directory
    print str(datetime.datetime.now()) + ': clearing directory ...'
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/out'], stdout=FNULL)

    # creat initialization job
    init_job = PageRankIter_T(args=[graph, '--i', '1', '--n', '10', '-r', 'hadoop',
                                    '--output-dir', 'hdfs:///user/leiyang/out'])

    # run initialization job
    print str(datetime.datetime.now()) + ': running iteration 1 ...'
    print str(datetime.datetime.now()) + ': running iteration 1 ...'
    with init_job.make_runner() as runner:
        runner.run()

    # checking counters
    n_node = getCounter('wiki_node_count', 'nodes', host)
    loss = getCounters('wiki_dangling_mass', host)
    loss_array = ['0']*11
    for k in loss:
        i = int(k.split('_')[1])
        loss_array[i] = str(loss[k]/1e10)
    print '%s: initialization complete: %d nodes!' %(str(datetime.datetime.now()), n_node)

    # run redistribution job
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])
    loss_param = '[%s]' %(','.join(['0']*11) if len(loss)==0 else ','.join(loss_array))
    dist_job = PageRankDist_T(args=['hdfs:///user/leiyang/in/part*', '--s', str(n_node), '--m', lo
ss_param,
                                    '--file', 'hdfs:///user/leiyang/randNet_topics.txt',
                                    '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])
    print str(datetime.datetime.now()) + ': distributing loss mass ...'
    with dist_job.make_runner() as runner:
        runner.run()

# move results for next iteration
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

# create iteration job
iter_job = PageRankIter_T(args=['hdfs:///user/leiyang/in/part*', '--i', '0', '--n', '10',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])

# run pageRank iteratively
i = 2 if doInit else 1
while(1):
    print str(datetime.datetime.now()) + ': running iteration %d ...' %i
    with iter_job.make_runner() as runner:
        runner.run()

    # check counters for topic loss mass
    loss = getCounters('wiki_dangling_mass', host)
    loss_array = ['0']*11
    for k in loss:
        i = int(k.split('_')[1])
        loss_array[i] = str(loss[k]/1e10)

    # move results for next iteration
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

    # run redistribution job
    loss_param = '[%s]' %(','.join(['0']*11) if len(loss)==0 else ','.join(loss_array))
    dist_job = PageRankDist_T(args=['hdfs:///user/leiyang/in/part*', '--s', str(n_node), '--m', lo
ss_param,
                                    '--file', 'hdfs:///user/leiyang/randNet_topics.txt',
                                    '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])

    print str(datetime.datetime.now()) + ': distributing loss mass ...'
    with dist_job.make_runner() as runner:
        runner.run()

    if i == n_iter:
        break

    # if more iteration needed
    i += 1
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'], stdout=FNULL)

# run sort job
print str(datetime.datetime.now()) + ': sorting PageRank ...'
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/rank'], stdout=FNULL)
sort_job = PageRankSort_T(args=['hdfs:///user/leiyang/out/part*', '--file', 'hdfs:///user/leiyang/
randNet_topics.txt',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/rank'])

with sort_job.make_runner() as runner:
    runner.run()

# run join job
```

```
if doJoin:
    print str(datetime.datetime.now()) + ': joining PageRank with index ...'
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/join'], stdout=FNULL)
    join_job = PageRankJoin(args=[index, '-r', 'hadoop', '--file', 'hdfs:///user/leiyang/rank/part
-00000',
                                  '--output-dir', 'hdfs:///user/leiyang/join'])
    with join_job.make_runner() as runner:
        runner.run()

print "%s: PageRank job completes in %.1f minutes!\n" %(str(datetime.datetime.now()), (time()-star
t)/60.0)


call(['hdfs', 'dfs', '-cat', '/user/leiyang/join/p*' if doJoin else '/user/leiyang/rank/p*'])
```

```
Overwriting RunPageRank_T.py
```

Run topic-sensitive-PageRank on randNet

In [50]:

```
!python RunPageRank_T.py -g 'hdfs:///user/leiyang/randNet.txt' -j 0.15 -i 2 -d 'NULL' -s '0'
```

```
2016-03-14 23:25:35.341468: start topic sensitive PageRanking on 'randNet.txt' for 2 iterations wi
th damping factor 0.85 ...
2016-03-14 23:25:35.341510: clearing directory ...
2016-03-14 23:25:38.381559: running iteration 1 ...
No handlers could be found for logger "mrjob.conf"
2016-03-14 23:26:08.768379: initialization complete: 100 nodes!
2016-03-14 23:26:10.302849: distributing loss mass ...
2016-03-14 23:26:37.074679: running iteration 2 ...
2016-03-14 23:27:10.299038: distributing loss mass ...
2016-03-14 23:27:33.786508: sorting PageRank ...
2016-03-14 23:28:08.548791: PageRank job completes in 2.6 minutes!

"====== Top 10 for topic 0 ======"      ""
[0, 1.6076271109890086] "4"
[0, 1.5757123883143969] "3"
[0, 1.5742073671113035] "10"
[0, 1.540390989068374]  "2"
[0, 1.5221086826843588] "8"
[0, 1.4912908154538556] "7"
[0, 1.451330386178526]  "1"
[0, 1.4330922286519523] "2"
[0, 1.4282327277192604] "2"
[0, 1.420359996116097]  "8"
"====== Top 10 for topic 3 ======"      ""
[3, 3.1007775017723676] "3"
[3, 2.728338161490016]  "3"
[3, 2.630148085635196]  "3"
[3, 2.449888407509806]  "3"
[3, 2.4244890524764036] "3"
[3, 2.3737871587132]    "3"
[3, 2.2844867316425757] "3"
[3, 2.2302853241314207] "3"
[3, 1.9494422654891403] "3"
[3, 1.585222631608711]  "10"
"====== Top 10 for topic 6 ======"      ""
[6, 3.452449425532428]  "6"
[6, 3.3415010712976394] "6"
[6, 3.153677218973676]  "6"
[6, 3.1478734311978167] "6"
[6, 3.0062702584843857] "6"
[6, 2.8518406739805435] "6"
[6, 1.8353181150344853] "3"
[6, 1.7452829097188156] "1"
[6, 1.650547677607659]  "7"
[6, 1.5520807425446956] "1"
"====== Top 10 for topic 9 ======"      ""
[9, 3.060229418842161]  "9"
[9, 2.979611542186343]  "9"
[9, 2.9249475623369228] "9"
[9, 2.811445306690044]  "9"
[9, 2.753880168647277]  "9"
[9, 2.633916944665387]  "9"
[9, 2.6237843149338826] "9"
[9, 1.7082720511742717] "4"
[9, 1.6396950897219282] "8"
[9, 1.4417116318500756] "5"
"====== Top 10 for topic 1 ======"      ""
[1, 2.0277656004584634] "1"
```

```
[1, 2.0277656004584634] "1"
[1, 2.01924107206815]   "1"
[1, 2.0042618968483428] "1"
[1, 1.9840937141196724] "1"
[1, 1.9005343297098247] "1"
[1, 1.8169995237419168] "1"
[1, 1.8031939294911281] "1"
[1, 1.7643871332926433] "7"
[1, 1.697663442582149]  "1"
[1, 1.6061549426812238] "1"
"====== Top 10 for topic 10 ======"      ""
[10, 2.6403850807329095]        "10"
[10, 2.3333074490514694]        "10"
[10, 2.3242611946337393]        "10"
[10, 2.0474272607723365]        "10"
[10, 1.9832877329871108]        "10"
[10, 1.9537931823575154]        "10"
[10, 1.9089379468840644]        "10"
[10, 1.8822262817674749]        "10"
[10, 1.876481627166124] "10"
[10, 1.870432423806903] "10"
"====== Top 10 for topic 4 ======"       ""
[4, 2.658938009381804]  "4"
[4, 2.230333702606206]  "4"
[4, 2.0855709724067872] "4"
[4, 2.0479432040461996] "4"
[4, 1.939544249615961]  "4"
[4, 1.9106367376637203] "4"
[4, 1.8188861851920657] "4"
[4, 1.7276359895818532] "4"
[4, 1.7064786379523138] "4"
[4, 1.680103142772124]  "5"
"====== Top 10 for topic 7 ======"       ""
[7, 2.633474214693505]  "7"
[7, 2.6332214732046486] "7"
[7, 2.469406363261154]  "7"
[7, 2.4381961656842135] "7"
[7, 2.35101483090898]   "7"
[7, 2.3360347159692063] "7"
[7, 2.312882323078407]  "7"
[7, 2.289266817986907]  "7"
[7, 2.267395965681545]  "7"
[7, 2.026203544074973]  "7"
"====== Top 10 for topic 2 ======"       ""
[2, 3.0408533573160685] "2"
[2, 2.9527264924859447] "2"
[2, 2.94090754655984]   "2"
[2, 2.820085079982913]  "2"
[2, 2.7161250962039265] "2"
[2, 2.5739862590643385] "2"
[2, 2.437154220819449]  "2"
[2, 2.302905103183183]  "2"
[2, 1.621938299551385]  "1"
[2, 1.6007765263381522] "4"
"====== Top 10 for topic 5 ======"       ""
[5, 2.8734416137454013] "5"
[5, 2.809042600371491]  "5"
[5, 2.769751765971403]  "5"
[5, 2.6567480226145603] "5"
[5, 2.6229828497245298] "5"
[5, 2.437841665159035]  "5"
[5, 2.3264013703190445] "5"
[5, 2.317746946733371]  "5"
[5, 2.310092689539805]  "5"
[5, 1.6659469254855137] "8"
"====== Top 10 for topic 8 ======"       ""
[8, 3.3531168983697723] "8"
[8, 2.737807715019576]  "8"
[8, 2.7304141658474648] "8"
[8, 2.6779613302006835] "8"
[8, 2.5580920813747987] "8"
[8, 2.525551329273765]  "8"
[8, 2.3927393763673797] "8"
[8, 2.316853699343678]  "8"
[8, 2.0543621387363338] "8"
[8, 1.50319178717137]   "2"
cat: `>': No such file or directory
```

Comments

- for each topic, most top 10 ranked nodes are consistent with their truth topic

HW 9.5: Applying topic-specific PageRank to Wikipedia

Here you will apply your topic-specific PageRank implementation to Wikipedia, defining topics (very arbitrarily) for each page by the length (number of characters) of the name of the article mod 10, so that there are 10 topics. Once again, print out the top ten ranking nodes and their topics for each of the 11 versions, and comment on your result. Assume a teleportation factor of **0.15** in all your analyses.

Simple code to count wiki topics based on node name mod 10

In [61]:

```
topic_index = [0]*10
with open('indices.txt') as f:
    for line in f.readlines():
        name = line.split('\t')[0]
        topic_index[len(name)%10] += 1

print topic_index
```

```
[1455304, 1536145, 1591290, 1624124, 1610195, 1550659, 1511681, 1472178, 1419076, 1421625]
```

In [62]:

```
count=0
with open('indices.txt') as f:
    for line in f.readlines():
        count += len(line.split('\t'))==2

print count
```

```
0
```

Job to attach topic to wiki file

- reducer side join to get topic

In [10]:

```
%%writefile PageRankInit.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRankInit(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankInit, self).configure_options()
        self.add_passthrough_option(
            '--n', dest='n_topic', default='0', type='int',
            help='n: number of topics (default 0)')

    def mapper(self, _, line):
        # parse line
        elem = line.strip().split('\t')
        if len(elem) == 2:
            nid, adj = elem[0].strip('"'), elem[1]
            cmd = 'adj = %s' %adj
            exec cmd
            # initialize node struct
            node = {'a':adj.keys(), 'p':[1.0]*(self.options.n_topic+1)}
            # emit node
            yield nid, node
            # emit pageRank mass
            for m in node['a']:
                yield m, {'f':0}
        else:
            yield elem[1].strip('"'), {'t':len(elem[0])%10 + 1}

    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        node, topic, adj = None, None, None
        # loop through all arrivals
        for v in value:
            if 'a' in v:
                node = v
            elif 't' in v:
                topic = v
            elif 'f' in v:
```

```
            elif 'l' in v:
                adj = v
        # emit accumulative mass for nid
        if node:
            yield nid, node
        if topic:
            yield nid, topic
        if adj:
            yield nid, adj

    # reducer for initialization pass --> need to handle dangling nodes
    def reducer(self, nid, value):
        topic, node = None, None
        # loop through all arrivals
        for v in value:
            if 't' in v:
                topic = v['t']
            elif 'a' in v:
                node = v
        # handle dangling node, create node struct and add missing mass
        if not node: # and topic:
            node = {'a':[], 'p':[1.0]*(self.options.n_topic+1)}
        node['t'] = topic if topic else 0
        # emit for next iteration
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '2',
        }
        return [MRStep(mapper=self.mapper
                       , combiner=self.combiner
                       , reducer=self.reducer
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankInit.run()
```

```
Overwriting PageRankInit.py
```

In [5]:

```
##### unit test #####
!python PageRankInit.py 'wiki_index_sample' 'wiki_sample' --n 10 -r 'hadoop' --output-dir 'hdf
s:///user/leiyang/wiki_topic' > test.t
```

```
using configs in /Users/leiyang/.mrjob.conf
Got unexpected opts from /Users/leiyang/.mrjob.conf: no_output
creating tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankInit.leiyang.20160
318.213330.310523
writing wrapper script to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankInit.leiyang.20
160318.213330.310523/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/leiyang/tmp/mrjob/PageRankInit.leiyang.20160318.213330.31052
3/files/
HADOOP: packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar2845637823127
928176/] [] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob6502102574757138418.jar tmpD
ir=null
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/leiyang/wiki_topic
removing tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankInit.leiyang.20160
318.213330.310523
deleting hdfs:///user/leiyang/tmp/mrjob/PageRankInit.leiyang.20160318.213330.310523 from HDFS
```

PageRankIter

In [11]:

```
%%writefile PageRankIter_W.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRankIter_W(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankIter_W, self).configure_options()
        self.add_passthrough_option(
            '--n', dest='n_topic', default='0', type='int',
            help='n: number of topics (default 0)')

    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # distribute rank mass
        n_adj = len(node['a'])
        if n_adj > 0:
            rankMass = [x / n_adj for x in node['p']]
            # emit pageRank mass
            for m in node['a']:
                yield m, {'m':rankMass}
        else:
            # track dangling mass for each topic with counters
            for i in range(self.options.n_topic+1):
                self.increment_counter('wiki_dangling_mass', 'topic_%d' %i, int(node['p'][i]*1e1
0))
        # reset pageRank and emit node
        node['p'] = [0]*(self.options.n_topic+1)
        yield nid, node


    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        rankMass, node = [0]*(self.options.n_topic+1), None
        # loop through all arrivals
        for v in value:
            if 'm' in v:
                rankMass = [a+b for a,b in zip(rankMass, v['m'])]
            else:
                node = v
        # emit accumulative mass for nid
        if node:
            node['p'] = [a+b for a,b in zip(rankMass, node['p'])]
            yield nid, node
        else:
            yield nid, {'m':rankMass}

    def reducer(self, nid, value):
        rankMass, node = [0]*(self.options.n_topic+1), None
        # loop through all arrivals
        for v in value:
            if 'm' in v:
                rankMass = [a+b for a,b in zip(rankMass, v['m'])]
            else:
                node = v
        # update pageRank
        if True: #node:
            node['p'] = [a+b for a,b in zip(rankMass, node['p'])]
            # emit for next iteration
            yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '2',
        }
        return [MRStep(mapper=self.mapper
                       , combiner=self.combiner
                       , reducer=self.reducer
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankIter_W.run()
```

```
Overwriting PageRankIter_W.py
```

In [6]:

```
##### unit test #####
!python PageRankIter_W.py test.t --n 10 -r 'hadoop' > test2
```

```
using configs in /Users/leiyang/.mrjob.conf
Got unexpected opts from /Users/leiyang/.mrjob.conf: no_output
creating tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankIter_W.leiyang.201
60318.213522.819781
```

```
60318.213522.819781
writing wrapper script to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankIter_W.leiyang.
20160318.213522.819781/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/leiyang/tmp/mrjob/PageRankIter_W.leiyang.20160318.213522.819
781/files/
HADOOP: packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar2327074916773
165398/] [] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob1778154114577902672.jar tmpD
ir=null
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/leiyang/tmp/mrjob/PageRankIter_W.leiyang.20160318.213522.
819781/output
removing tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankIter_W.leiyang.201
60318.213522.819781
deleting hdfs:///user/leiyang/tmp/mrjob/PageRankIter_W.leiyang.20160318.213522.819781 from HDFS
```

PageRankDist_W

In [12]:

```
%%writefile PageRankDist_W.py
from mrjob.job import MRJob
from mrjob.step import MRStep
from subprocess import Popen, PIPE

class PageRankDist_W(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def configure_options(self):
        super(PageRankDist_W, self).configure_options()
        self.add_passthrough_option(
            '--j', dest='alpha', default=0.15, type='float',
            help='jump: teleport factor (default 0.15)')
        self.add_passthrough_option(
            '--b', dest='beta', default=0.99, type='float',
            help='beta: topic bias factor (default 0.99)')
        self.add_passthrough_option(
            '--m', dest='m', default='', type='str',
            help='m: rank mass from dangling nodes')

    def mapper_init(self):
        # load topic file and count
        T_j = [1455304, 1536145, 1591290, 1624124, 1610195, 1550659, 1511681, 1472178, 1419076, 14
21625]
        N = sum(T_j)
        # prepare adjustment factors
        self.damping = 1 - self.options.alpha
        cmd = 'm = %s' %self.options.m
        exec cmd
        # assuming here -m is specified with a list syntax string
        self.p_dangling = [1.0*x / N for x in m]
        # for each topic, get topic bias
        b = self.options.beta
        self.v_ij = [[1, 1]] + [[(1-b)*N/(N-t), b*N/t] for t in T_j]

    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # get final pageRank
        for i in range(len(self.v_ij)):
            vij = self.v_ij[i][i==node['t']]
            node['p'][i] = (self.p_dangling[i]+node['p'][i])*self.damping + self.options.alpha*vij
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
        }
        return [MRStep(mapper_init=self.mapper_init
                       , mapper=self.mapper
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankDist_W.run()
```

```
Overwriting PageRankDist_W.py
```

In [7]:
```
##### unit test #####
!python PageRankDist_W.py 'hdfs:///user/leiyang/in' --m '[9410987]*11' -r 'hadoop' '--output-dir'
'hdfs:///user/leiyang/out'
```

```
using configs in /Users/leiyang/.mrjob.conf
Got unexpected opts from /Users/leiyang/.mrjob.conf: no_output
creating tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankDist_W.leiyang.201
60318.213626.447008
writing wrapper script to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankDist_W.leiyang.
20160318.213626.447008/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/leiyang/tmp/mrjob/PageRankDist_W.leiyang.20160318.213626.447
008/files/
HADOOP: packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar8191536429609
644621/] [] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob4098817486675800499.jar tmpD
ir=null
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/leiyang/tmp/mrjob/PageRankDist_W.leiyang.20160318.213626.
447008/output
removing tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankDist_W.leiyang.201
60318.213626.447008
deleting hdfs:///user/leiyang/tmp/mrjob/PageRankDist_W.leiyang.20160318.213626.447008 from HDFS
```

PageRankSort

In [13]:
```
%%writefile PageRankSort_W.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class PageRankSort_W(MRJob):
    DEFAULT_PROTOCOL = 'json'
    PARTITIONER = 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner'


    def mapper(self, _, line):
        # parse line
        nid, node = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %node
        exec cmd
        # emit (vector_ID, pageRank)~topic_id
        for i in range(len(node['p'])):
            yield (i, node['p'][i]), node['t']

    def reducer_init(self):
        self.current_v = None
        self.i = 0
        self.top = 10

    def reducer(self, key, value):
        if self.current_v != key[0]:
            self.current_v = key[0]
            self.i = 0
            yield '====== Top 10 for topic %d ======' %self.current_v, ''
        if self.i < self.top:
            self.i += 1
            for v in value:
                yield key, v
                break



    def steps(self):
        jc = {
            'mapreduce.job.maps': '3',
            'mapreduce.job.reduces': '3',
            'mapreduce.partition.keypartitioner.options': '-k1,1',
            'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.mapreduce.lib.partitio
n.KeyFieldBasedComparator',
            'mapreduce.partition.keycomparator.options': '-k1,1 -k2,2nr',
            'stream.num.map.output.key.fields': '2',
            'mapreduce.map.output.key.field.separator': ' ',
            'stream.map.output.field.separator': ' ',
        }
        return [MRStep(mapper=self.mapper,
        return [MRStep(mapper=self.mapper
                       , reducer_init=self.reducer_init
                       , reducer=self.reducer
                       , jobconf = jc
                      )
               ]

if __name__ == '__main__':
    PageRankSort_W.run()
```

```
Overwriting PageRankSort_W.py
```

In [9]:

```
##### unit test #####
!python PageRankSort_W.py 'hdfs:///user/leiyang/out/p*' -r 'hadoop' > results
```

```
using configs in /Users/leiyang/.mrjob.conf
Got unexpected opts from /Users/leiyang/.mrjob.conf: no_output
creating tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankSort_W.leiyang.201
60318.213704.233404
writing wrapper script to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankSort_W.leiyang.
20160318.213704.233404/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/leiyang/tmp/mrjob/PageRankSort_W.leiyang.20160318.213704.233
404/files/
HADOOP: packageJobJar: [/var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/hadoop-unjar8037685516670
150734/] [] /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/streamjob5144151202206446323.jar tmpD
ir=null
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/leiyang/tmp/mrjob/PageRankSort_W.leiyang.20160318.213704.
233404/output
removing tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/PageRankSort_W.leiyang.201
60318.213704.233404
deleting hdfs:///user/leiyang/tmp/mrjob/PageRankSort_W.leiyang.20160318.213704.233404 from HDFS
```

Driver

In [14]:

```
%%writefile RunPageRank_W.py
#!/usr/bin/python

from PageRankIter_W import PageRankIter_W
from PageRankDist_W import PageRankDist_W
from PageRankSort_W import PageRankSort_W
from helper import getCounter, getCounters
from subprocess import call, check_output
from time import time
import sys, getopt, datetime, os

# parse parameter
if __name__ == "__main__":

    try:
        opts, args = getopt.getopt(sys.argv[1:], "hg:j:i:")
    except getopt.GetoptError:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index> -s <size>'
        sys.exit(2)
    if len(opts) != 3:
        print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'RunBFS.py -g <graph> -j <jump> -i <iteration> -d <index>'
            sys.exit(2)
        elif opt == '-g':
            graph = arg
        elif opt == '-j':
            jump = arg
        elif opt == '-i':
            n_iter = arg


start = time()
FNULL = open(os.devnull, 'w')
n_iter = int(n_iter)
host = 'localhost'

print '%s: %s topic sensitive PageRanking on \'%s\' for %d iterations with damping factor %.2f
...' %(str(datetime.datetime.now()),
          'start', graph[graph.rfind('/')+1:], n_iter, 1-float(jump))
          score', graph[graph.find('\t')+1:], n_iter, 1-float(jump))

# clear directory
print str(datetime.datetime.now()) + ': clearing directory ...'
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/out'], stdout=FNULL)
call(['hdfs', 'dfs', '-cp', '/user/leiyang/wiki_topic', '/user/leiyang/in'])

# create iteration job
iter_job = PageRankIter_W(args=['hdfs:///user/leiyang/in/part*', '--n', '10',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])

# run pageRank iteratively
iteration = 1
while(1):
    print str(datetime.datetime.now()) + ': running iteration %d ...' %iteration
    with iter_job.make_runner() as runner:
        runner.run()

    # check counters for topic loss mass
    loss = getCounters('wiki_dangling_mass', host)
    loss_array = ['0']*11
    for k in loss:
        i = int(k.split('_')[1])
        loss_array[i] = str(loss[k]/1e10)

    # move results for next iteration
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

    # run redistribution job
    loss_param = '[%s]' %(','.join(['0']*11) if len(loss)==0 else ','.join(loss_array))
    dist_job = PageRankDist_W(args=['hdfs:///user/leiyang/in/part*', '--m', loss_param,
                                  '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/out'])

    print str(datetime.datetime.now()) + ': distributing loss mass ...'
    with dist_job.make_runner() as runner:
        runner.run()

    if iteration == n_iter:
        break

    # if more iteration needed
    iteration += 1
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'], stdout=FNULL)

# run sort job
print str(datetime.datetime.now()) + ': sorting PageRank ...'
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/rank'], stdout=FNULL)
sort_job = PageRankSort_W(args=['hdfs:///user/leiyang/out/part*',
                              '-r', 'hadoop', '--output-dir', 'hdfs:///user/leiyang/rank'])

with sort_job.make_runner() as runner:
    runner.run()

print "%s: PageRank job completes in %.1f minutes!\n" %(str(datetime.datetime.now()), (time()-star
t)/60.0)
call(['hdfs', 'dfs', '-cat', '/user/leiyang/rank/p*'])
###
```

```
Overwriting RunPageRank_W.py
```

In [15]:

```
##### unit test #####
#!python PageRankInit.py 'hdfs:///user/leiyang/all-pages-indexed-out.txt' 'hdfs:///user/leiyang/in
dices.txt' --n 10 -r 'hadoop' --output-dir 'hdfs:///user/leiyang/wiki_topic'
```

```
2016-03-18 17:38:25.487866: start topic sensitive PageRanking on 'wiki_topic' for 2 iterations wit
h damping factor 0.85 ...
2016-03-18 17:38:25.487929: clearing directory ...
2016-03-18 17:38:30.482881: running iteration 1 ...
No handlers could be found for logger "mrjob.conf"
2016-03-18 17:39:11.994335: distributing loss mass ...
2016-03-18 17:39:42.283399: running iteration 2 ...
2016-03-18 17:40:22.785699: distributing loss mass ...
2016-03-18 17:40:49.335183: sorting PageRank ...
2016-03-18 17:41:28.649036: PageRank job completes in 3.1 minutes!

"====== Top 10 for topic 0 ======"      ""
[0, 0.415152627627402]  0
[0, 0.40930942137065895]        0
```

```
[0, 0.40930942137063895]        0
[0, 0.37349099859512114]        0
[0, 0.3395550264825884] 0
[0, 0.32312547832835326]        0
[0, 0.3199925262450779] 0
[0, 0.3018223204242624] 0
[0, 0.3014296013586007] 0
[0, 0.2840296303889138] 0
[0, 0.28382341757596924]        0
"====== Top 10 for topic 3 ======"      ""
[3, 1.4179382306289952] 3
[3, 1.2089068480984022] 5
[3, 0.06556011692643458]        0
[3, 0.06546354673712068]        0
[3, 0.06462157871002236]        0
[3, 0.0624105432575417] 0
[3, 0.06238474192878037]        0
[3, 0.06221833133549339]        0
[3, 0.062214846844126405]       0
[3, 0.04538148176667206]        0
"====== Top 10 for topic 6 ======"      ""
[6, 1.4550584277219532] 6
[6, 1.2404497808808812] 0
[6, 0.31279721462058324]        0
[6, 0.3114844578751008] 0
[6, 0.06552633801976636]        0
[6, 0.06542988902587364]        0
[6, 0.0453439538816413] 0
[6, 0.037218505368632374]       8
[6, 0.03639486043442384]        0
[6, 0.03146917940841542]        0
"====== Top 10 for topic 9 ======"      ""
[9, 1.5899276472060748] 9
[9, 1.3550673329253282] 0
[9, 0.09355534599092745]        0
[9, 0.09308838079607308]        0
[9, 0.09293555582321163]        0
[9, 0.09274612588720692]        0
[9, 0.09272582305264804]        0
[9, 0.0927096406459382] 0
[9, 0.09229818879592665]        0
[9, 0.09220731989314417]        0
"====== Top 10 for topic 1 ======"      ""
[1, 1.5503279388536626] 1
[1, 1.3214084209563755] 0
[1, 0.0697729621351361] 0
[1, 0.06969492103373486]        0
[1, 0.06803948620310993]        0
[1, 0.06793900618566649]        0
[1, 0.06785102304356486]        0
[1, 0.06780683719091259]        0
[1, 0.06777958870538443]        0
[1, 0.06777241203511307]        0
"====== Top 10 for topic 10 ======"     ""
[10, 1.587154155812098] 10
[10, 1.5870580869627688]        10
[10, 1.3526256612372696]        0
[10, 0.10719578472116759]       0
[10, 0.10672878337026095]       0
[10, 0.10657594656450971]       0
[10, 0.10638650196137618]       0
[10, 0.10636619755481502]       0
[10, 0.10635001389513825]       0
[10, 0.1060642710292506]        0
"====== Top 10 for topic 4 ======"      ""
[4, 1.3891842978320106] 4
[4, 1.1844562511644956] 0
[4, 0.06547251432166346]        0
[4, 0.06537584566377594]        0
[4, 0.0452969251231107] 0
[4, 0.036340509542733435]       0
[4, 0.03141614738189926]        0
[4, 0.031021432018330446]       0
[4, 0.030978358691095175]       0
[4, 0.03090129784779581]        0
"====== Top 10 for topic 7 ======"      ""
[7, 1.492860051094673]  7
[7, 1.4927580510181022] 7
[7, 1.492661717612452]  7
[7, 1.272421223662592]  0
```

```
[7, 0.06561048104938284]        0
[7, 0.0655141476437325] 0
[7, 0.04542452138162684]        0
[7, 0.036479280875822036]       0
[7, 0.03155290585666481]        0
[7, 0.03115955938774681]        0
"====== Top 10 for topic 2 ======"      ""
[2, 1.4687586460764956] 2
[2, 1.2520862687746388] 0
[2, 0.06547823882097754]        0
[2, 0.065381832945078]  0
[2, 0.055699724790925755]       0
[2, 0.05393465138145588]        0
[2, 0.05387746145507479]        5
[2, 0.04529452089959706]        0
[2, 0.03634686471881865]        0
[2, 0.03142092481187879]        0
"====== Top 10 for topic 5 ======"      ""
[5, 1.4045822632446798] 5
[5, 1.4013412384249397] 5
[5, 1.4012003243023423] 5
[5, 1.194668106684581]  0
[5, 0.08129898011159387]        0
[5, 0.06547032492845133]        0
[5, 0.06537369810152739]        0
[5, 0.045293441758758456]       0
[5, 0.0363384205438339] 0
[5, 0.03141380722940887]        0
"====== Top 10 for topic 8 ======"      ""
[8, 1.5328261386245454] 8
[8, 1.5327242618744756] 8
[8, 1.532628044943854]  8
[8, 1.306375857982865]  0
[8, 0.32864527887368467]        0
[8, 0.32796610053988534]        0
[8, 0.19459938053768946]        0
[8, 0.1926328255320875] 0
[8, 0.18895781257783495]        0
[8, 0.18830553113375628]        0
```

Topic-sensitive pagerank on wikipedia dataset

- **only one iteration, not enough to have a decent result**

```
In [1]: #!python RunPageRank_W.py -g 'hdfs:///user/leiyang/wiki_topic' -j 0.15 -i 1
        !cat wiki_topic
```

```
"====== Top 10 for topic 7 ======"      ""
[7, 18168.2859148241]   4
[7, 13311.059756407594] 7
[7, 10588.6805693981]   10
[7, 10189.495447501646] 7
[7, 9153.828542626043]  7
[7, 8180.532910231774]  8
[7, 7302.697713785382]  2
[7, 6898.382134812705]  6
[7, 6599.769135912638]  7
[7, 6410.850248318637]  8
"====== Top 10 for topic 3 ======"      ""
[3, 18168.28592457402]  4
[3, 13309.569018403192] 7
[3, 10588.680579148015] 10
[3, 10188.004709497243] 7
[3, 9152.33780462164]   7
[3, 8180.53291998169]   8
[3, 7302.697723535298]  2
[3, 6898.382144562621]  6
[3, 6598.278397908237]  7
[3, 6410.850258068553]  8
"====== Top 10 for topic 10 ======"     ""
[10, 18168.285903930606]        4
[10, 13309.568997759781]        7
[10, 10590.265857407618]        10
[10, 10188.004688853833]        7
[10, 9152.33778397823]  7
[10, 8180.532899338278] 8
[10, 7302.697702891886] 2
[10, 6898.382123919209] 6
```

```
[10, 6598.278377264825] 7
[10, 6410.850237425141] 8
"====== Top 10 for topic 8 ======"      ""
[8, 18168.28591002807]  4
[8, 13309.569003857247] 7
[8, 10588.68056460207]  10
[8, 10188.004694951298] 7
[8, 9152.337790075695]  7
[8, 8182.063703965961]  8
[8, 7302.697708989352]  2
[8, 6898.382130016675]  6
[8, 6598.278383362291]  7
[8, 6412.381042052824]  8
"====== Top 10 for topic 4 ======"      ""
[4, 18169.67333829323]  4
[4, 13309.569022457781] 7
[4, 10588.680583202604] 10
[4, 10188.004713551833] 7
[4, 9152.33780867623]   7
[4, 8180.532924036278]  8
[4, 7302.697727589886]  2
[4, 6898.382148617209]  6
[4, 6598.278401962825]  7
[4, 6410.850262123141]  8
"====== Top 10 for topic 0 ======"      ""
[0, 18168.43424907683]  4
[0, 13309.717342906002] 7
[0, 10588.828903650825] 10
[0, 10188.153034000054] 7
[0, 9152.48612912445]   7
[0, 8180.681244484499]  8
[0, 7302.846048038107]  2
[0, 6898.53046906543]   6
[0, 6598.426722411046]  7
[0, 6410.998582571362]  8
"====== Top 10 for topic 9 ======"      ""
[9, 18168.285903624343] 4
[9, 13309.568997453518] 7
[9, 10588.68055819834]  10
[9, 10188.00468854757]  7
[9, 9152.337783671966]  7
[9, 8180.532899032015]  8
[9, 7302.697702585623]  2
[9, 6898.382123612946]  6
[9, 6598.278376958562]  7
[9, 6410.850237118878]  8
"====== Top 10 for topic 5 ======"      ""
[5, 18168.285926906156] 4
[5, 13309.56902073533]  7
[5, 10588.680581480152] 10
[5, 10188.00471182938]  7
[5, 9152.337806953778]  7
[5, 8180.532922313826]  8
[5, 7302.697725867434]  2
[5, 6898.382146894757]  6
[5, 6598.278400240373]  7
[5, 6410.850260400689]  8
"====== Top 10 for topic 1 ======"      ""
[1, 18168.285907987818] 4
[1, 13309.569001816993] 7
[1, 10588.680562561816] 10
[1, 10188.004692911045] 7
[1, 9152.337788035442]  7
[1, 8180.532903395491]  8
[1, 7302.6977069490995] 2
[1, 6898.382127976422]  6
[1, 6598.2783813220385] 7
[1, 6410.850241482354]  8
"====== Top 10 for topic 6 ======"      ""
[6, 18168.285919583617] 4
[6, 13309.569013412793] 7
[6, 10588.680574157615] 10
[6, 10188.004704506844] 7
[6, 9152.337799631241]  7
[6, 8180.5329149912895] 8
[6, 7302.697718544898]  2
[6, 6899.835368649737]  6
[6, 6598.278392917837]  7
[6, 6410.8502530781525] 8
```

```
"====== Top 10 for topic 2 ======"       ""
[2, 18168.28591780817]  4
[2, 13309.569011637346] 7
[2, 10588.680572382169] 10
[2, 10188.004702731398] 7
[2, 9152.337797855795]  7
[2, 8180.532913215843]  8
[2, 7304.164693989173]  2
[2, 6898.382137796774]  6
[2, 6598.27839114239]   7
[2, 6410.850251302706]  8
```

HW 9.6: TextRank (OPTIONAL)

What is TextRank. Describe the main steps in the algorithm. Why does TextRank work? Implement TextRank in MrJob for keyword phrases (not just unigrams) extraction using co-occurrence based similarity measure with with sizes of N = 2 and 3. And evaluate your code using the following example using precision, recall, and FBeta (Beta=1):

"Compatibility of systems of linear constraints over the set of natural numbers Criteria of compatibility of a system of linear Diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generating sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types of systems and systems of mixed types."

The extracted keywords should in the following set:

linear constraints, linear diophantine equations, natural numbers, non-strict inequations, strict inequations, upper bounds

stop yarn, hdfs, and job history

In [76]:

```
!/usr/local/Cellar/hadoop/2*/sbin/stop-yarn.sh
!/usr/local/Cellar/hadoop/2*/sbin/stop-dfs.sh
!/usr/local/Cellar/hadoop/2*/sbin/mr-jobhistory-daemon.sh --config /usr/local/Cellar/hadoop/2*/libexec/etc/hadoop/ stop historyserver
```

```
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
localhost: nodemanager did not stop gracefully after 5 seconds: killing with kill -9
no proxyserver to stop
Stopping namenodes on [localhost]
localhost: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
stopping historyserver
```