DATASCI W261, Machine Learning at Scale

Assignement: week #7

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

Due: 2016-03-10, 8AM PST

General Description

In this assignment you will explore networks and develop MRJob code for finding shortest path graph distances. To build up to large data you will develop your code on some very simple, toy networks. After this you will take your developed code forward and modify it and apply it to two larger datasets (performing EDA along the way).

Undirected toy network dataset

In an undirected network all links are symmetric, i.e., for a pair of nodes 'A' and 'B,' both of the links:

A -> B and B -> A

will exist.

The toy data are available in a sparse (stripes) representation:

(node) \t (dictionary of links)

on AWS/Dropbox via the url:

s3://ucb-mids-mls-networks/undirected_toy.txt

On under the Data Subfolder for HW7 on Dropbox with the same file name

In the dictionary, target nodes are keys, link weights are values (here, all weights are 1, i.e., the network is unweighted).

Directed toy network dataset

In a directed network all links are not necessarily symmetric, i.e., for a pair of nodes 'A' and 'B,' it is possible for only one of:

A -> B or B -> A

to exist.

These toy data are available in a sparse (stripes) representation:

(node) \t (dictionary of links)

on AWS/Dropbox via the url:

s3://ucb-mids-mls-networks/directed_toy.txt

On under the Data Subfolder for HW7 on Dropbox with the same file name

In the dictionary, target nodes are keys, link weights are values (here, all weights are 1, i.e., the network is unweighted).

HW 7.0: Shortest path graph distances (toy networks)

In this part of your assignment you will develop the base of your code for the week.

Write MRJob classes to find shortest path graph distances, as described in the lectures. In addition to finding the distances, your code should also output a distance-minimizing path between the source and target. Work locally for this part of the assignment, and use both of the undirected and directed toy networks.

To proof you code's function, run the following jobs

- shortest path in the undirected network from node 1 to node 4

Solution: 1,5,4

- shortest path in the directed network from node 1 to node 5

Solution: 1,2,4,5

and report your output---make sure it is correct!

BFS Iteration

```
1: class Mapper
2:     method Map(nid n, node N)
3:         d ← N.Distance
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, d + 1)                  ▷ Emit distances to reachable nodes

1: class Reducer
2:     method Reduce(nid m, [d1, d2, ...])
3:         d_min ← ∞
4:         M ← ∅
5:         for all d ∈ counts [d1, d2, ...] do
6:             if IsNode(d) then
7:                 M ← d                           ▷ Recover graph structure
8:             else if d < d_min then              ▷ Look for shorter distance
9:                 d_min ← d
10:        M.Distance ← d_min                      ▷ Update shortest distance
11:        Emit(nid m, node M)
```

Two changes to improve performance for unweighted graph

- in iteration mrjob: no need to store weight value, to reduce the amount of data during processing
- in driver program: the stopping criterion for unweighted network is simpler: once the destination becomes frontier, it has the shortest path from the source

In [245]:
```
%%writefile ShortestPathIter.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class ShortestPathIter(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(ShortestPathIter, self).__init__(*args, **kwargs)


    def configure_options(self):
        super(ShortestPathIter, self).configure_options()
        self.add_passthrough_option(
            '--source', dest='source', default='1', type='string',
            help='source: source node (default 1)')
        self.add_passthrough_option(
            '--destination', dest='destination', default='1', type='string',
            help='destination: destination node (default 1)')
        self.add_passthrough_option(
            '--weighted', dest='weighted', default='1', type='string',
            help='weighted: is weighted graph (default 1)')


    def mapper_weighted(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %dic
        exec cmd
        # if the node structure is incomplete (first pass), add them
        if 'dist' not in node:
            node = {'adj':node, 'path':[]}
            node['dist'] = 0 if self.options.source==nid else -1
        # emit node
        yield nid, node
        # emit distances to reachable nodes
        if node['dist'] >= 0:
            for m in node['adj']:
                yield m, {'dd':node['adj'][m] + node['dist'], 'pp':node['path']+[nid]}

    def mapper_unweighted(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %dic
```

```
        exec cmd
        # if the node structure is incomplete (first pass), add them
        if 'dist' not in node:
            node = {'adj':node.keys(), 'path':[]}
            node['dist'] = 0 if self.options.source==nid else -1
        # emit node
        yield nid, node
        # emit distances to reachable nodes
        if node['dist'] >= 0:
            for m in node['adj']:
                yield m, {'dd':(1+node['dist']), 'pp':(node['path']+[nid])}

    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        dmin = float('inf')
        path = None
        # loop through all arrivals
        for v in value:
            if 'dist' in v:
                yield nid, v
            elif v['dd'] < dmin:
                dmin, path = v['dd'], v['pp']
        # emit the smallest distance for nid
        if path:
            yield nid, {'dd':dmin, 'pp':path}

    def reducer(self, nid, value):
        dmin = float('inf')
        path = node = None
        # loop through all arrivals
        for v in value:
            if 'dist' in v:
                node = v
            elif v['dd'] < dmin:
                dmin, path = v['dd'], v['pp']

        # handle dangling node, we only care if it's destination
        if not node:
            if nid == self.options.destination:
                node = {'adj':{}, 'dist':dmin, 'path':path}
            else:
                return
        elif (dmin < node['dist']) or (node['dist'] == -1 and path):
            node['dist'], node['path'] = dmin, path

        # emit for next iteration
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '2',
            'mapreduce.job.reduces': '2',
        }
        return [MRStep(mapper=self.mapper_weighted if self.options.weighted == 'y' else self.mapp
er_unweighted
                       , combiner=self.combiner
                       , reducer=self.reducer
                       , jobconf = jc
                      )
                ]

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

```
Overwriting ShortestPathIter.py
```

In [246]:
```
##### unit test #####
#!python ShortestPathIter.py undirected_toy.txt --source 1 --destination 4 --weighted 'n' -r 'inl
ine' # > test1
```

MrJob to retrieve path

In [107]:
```
%%writefile getPath.py
from mrjob.job import MRJob
from mrjob.step import MRStep
```

```
class getPath(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(getPath, self).__init__(*args, **kwargs)

    def configure_options(self):
        super(getPath, self).configure_options()
        self.add_passthrough_option(
            '--destination', dest='destination', default='1', type='string',
            help='destination: destination node (default 1)')

    def mapper_init(self):
        self.path = None

    def mapper(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        # emit distances to reachable nodes
        if nid.strip('"') == self.options.destination:
            cmd = 'node = %s' %dic
            exec cmd
            self.path = node['path']

    def mapper_final(self):
        if self.path:
            yield "shortest path - ", self.path+[self.options.destination]

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

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

```
Overwriting getPath.py
```

In [108]:
```
##### unit test #####
#!python getPath.py test1 --destination 4
```

MrJob to check if traverse completes for weighted graph

- compare node "distance" from two iteration
- if no node has changed "distance", traverse completed

In [314]:
```
%%writefile isTraverseCompleted.py
from mrjob.job import MRJob
from mrjob.step import MRStep


class isTraverseCompleted(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(isTraverseCompleted, self).__init__(*args, **kwargs)

    def mapper(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        # emit node ID and distance
        cmd = 'node = %s' %dic
        exec cmd
        yield nid, node['dist']

    def reducer_init(self):
        self.dist_changed = 0

    def reducer(self, nid, dist):
        pair = [d for d in dist]
        self.dist_changed += pair[0]!=pair[1]

    def reducer_final(self):
```

```
        yield self.dist_changed, 'traverse done' if self.dist_changed==0 else 'keep working'

    def steps(self):
        jc = {
            'mapreduce.job.maps': '1',
            'mapreduce.job.reduces': '1',
        }
        return [MRStep(mapper=self.mapper
                       , reducer_init=self.reducer_init
                       , reducer=self.reducer
                       , reducer_final=self.reducer_final
                       , jobconf = jc
                      )
                ]

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

```
Overwriting isTraverseCompleted.py
```

In [110]:
```
##### unit test #####
#!python ShortestPathIter.py 'undirected_toy.txt' --source 1 --destination 5 --weighted '0' > tes
t1
#!python ShortestPathIter.py 'test1' --source 1 --destination 5 --weighted '0' > test2
#!python ShortestPathIter.py 'test2' --source 1 --destination 5 --weighted '0' > test3
#!python isTraverseCompleted.py test1 test2
```

MrJob to check if traverse completes for unweighted graph

- as soon as the destination is reached, traverse completes, shortest path found.

In [111]:
```
%%writefile isDestinationReached.py
from mrjob.job import MRJob
from mrjob.step import MRStep


class isDestinationReached(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(isDestinationReached, self).__init__(*args, **kwargs)

    def configure_options(self):
        super(isDestinationReached, self).configure_options()
        self.add_passthrough_option(
            '--destination', dest='destination', default='1', type='string',
            help='destination: destination node (default 1)')

    def mapper_init(self):
        self.path = None

    def mapper(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        # emit distances to reachable nodes
        if nid.strip('"') == self.options.destination:
            cmd = 'node = %s' %dic
            exec cmd
            if node['dist'] > 0:
                self.path = node['path']+[self.options.destination]

    def mapper_final(self):
        yield 1 if self.path else 0, self.path

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

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

```
Overwriting isDestinationReached.py
```

```
Overwriting isDestinationReached.py
```

In [112]:
```
##### unit test #####
#!python isDestinationReached.py test1 --destination 5
```

Driver Program

- init_job: take raw import, do first level traverse, output intermediate graph file
- iter_job: iteratively traverse the graph and update the shortest distance and path
- **Note:** a more elegant way to determine stopping condition would be setting a counter, however mrjob can't easily retrieve counter value when the job is running on hadoop, so we unfortunately have to run a separate job to compare node distance

In [326]:
```
%%writefile RunBFS.py
#!/usr/bin/python
from ShortestPathIter import ShortestPathIter
from isTraverseCompleted import isTraverseCompleted
from isDestinationReached import isDestinationReached
from LongestPathIter import LongestPathIter
from getLongestDistance import getLongestDistance
from getPath import getPath
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:s:d:m:w:i:l:")
    except getopt.GetoptError:
        print 'RunBFS.py -g <graph> -s <source> -d <destination> -m <mode> -w <weighted>'
        sys.exit(2)
    if len(opts) != 7:
        print 'RunBFS.py -g <graph> -s <source> -d <destination> -m <mode> -w <weighted>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'RunBFS.py -g <graph> -s <source> -d <destination> -m <mode> -w <weighted>'
            sys.exit(2)
        elif opt == '-g':
            graph = arg
        elif opt == '-s':
            source = arg
        elif opt == '-d':
            destination = arg
        elif opt == '-m':
            mode = arg
        elif opt == '-w':
            weighted = arg
        elif opt == '-i':
            index = arg
        elif opt == '-l':
            longest = arg

start = time()
FNULL = open(os.devnull, 'w')

isWeighted = weighted=='y'
isLongest = longest=='y'

if isLongest:
    print str(datetime.datetime.now()) + ': Longest distance BFS started at node %s on %s graph %
s ...' %(
        source, 'weighted' if isWeighted else 'unweighted', graph[graph.rfind('/')+1:])
else:
    print str(datetime.datetime.now()) + ': Shortest path BFS started between node %s and node %s
on %s graph %s ...' %(
        source, destination, 'weighted' if isWeighted else 'unweighted', graph[graph.rfind('/')+
1:])

# creat BFS job
if not isLongest:
    init_job = ShortestPathIter(args=[graph, '--source', source, '--destination', destination, '-
-weighted', weighted,
                                      '-r', mode, '--output-dir', 'hdfs:///user/leiyang/out'])

    iter_job = ShortestPathIter(args=['hdfs:///user/leiyang/in/part*', '--source', source, '--des
```

```
tination', destination,
                                    '--weighted', weighted, '-r', mode, '--output-dir', 'hdfs:///us
er/leiyang/out'])
else:
    init_job = LongestPathIter(args=[graph, '--source', source, '--weighted', weighted,
                                    '-r', mode, '--output-dir', 'hdfs:///user/leiyang/out'])

    iter_job = LongestPathIter(args=['hdfs:///user/leiyang/in/part*', '--source', source,
                                    '--weighted', weighted, '-r', mode, '--output-dir', 'hdfs:///us
er/leiyang/out'])

if isLongest:
    path_job = getLongestDistance(args=['hdfs:///user/leiyang/out/part*', '-r', mode])
else:
    path_job = getPath(args=['hdfs:///user/leiyang/out/part*', '--destination', destination, '-
r', mode])

if isWeighted or isLongest:
    stop_job = isTraverseCompleted(args=['hdfs:///user/leiyang/out/part*', 'hdfs:///user/leiyang/
in/part*', '-r', mode])
else:
    stop_job = isDestinationReached(args=['hdfs:///user/leiyang/out/part*', '--destination', dest
ination, '-r', mode])

# run initialization job
with init_job.make_runner() as runner:
    print str(datetime.datetime.now()) + ': starting initialization job ...'
    runner.run()
# move the result to input folder
print str(datetime.datetime.now()) + ': moving results for next iteration ...'
call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

# run BFS iteratively
i, path = 1, []
while(1):
    with iter_job.make_runner() as runner:
        print str(datetime.datetime.now()) + ': running iteration %d ...' %i
        runner.run()

    # check if traverse is completed: no node has changing distance
    with stop_job.make_runner() as runner:
        print str(datetime.datetime.now()) + ': checking stopping criterion ...'
        runner.run()
        output = []
        for line in runner.stream_output():
            n, text = stop_job.parse_output_line(line)
            output.append([n, text])

    # if traverse completed, get path and break out
    flag = sum([x[0] for x in output])
    #print 'output value: %s' %str(output)
    if (isWeighted or isLongest) and flag==0:
        print str(datetime.datetime.now()) + ': traverse has completed, retrieving path ...'
        with path_job.make_runner() as runner:
            runner.run()
            for line in runner.stream_output():
                text, ppp = path_job.parse_output_line(line)
                if len(ppp) > len(path):
                    path = ppp
        break
    elif (not isWeighted) and flag==1 and (not isLongest):
        print str(datetime.datetime.now()) + ': destination is reached, retrieving path ...'
        for x,path in output:
            if x==1:
                break
        break

    # if more iteration needed
    i += 1
    if isWeighted or isLongest:
        print str(datetime.datetime.now()) + ': %d nodes changed distance' %flag
    else:
        print str(datetime.datetime.now()) + ': destination not reached yet.'
    print str(datetime.datetime.now()) + ': moving results for next iteration ...'
    call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/in'], stdout=FNULL)
    call(['hdfs', 'dfs', '-mv', '/user/leiyang/out', '/user/leiyang/in'])

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

```
call(['hdfs', 'dfs', '-rm', '-r', '/user/leiyang/out'], stdout=FNULL)

# translate into words if index is valid
if os.path.isfile(index):
    path = [check_output(['grep', x, index]).split('\t')[0] for x in path]

print str(datetime.datetime.now()) + ": traversing completes in %.1f minutes!\n" %((time()-star
t)/60.0)
print str(datetime.datetime.now()) + ': %s path: %s\n'%('longest' if isLongest else 'shortest', '
-> '.join(path))
```

```
Overwriting RunBFS.py
```

Unit test with toy nets

In [159]:
```
!python RunBFS.py -s 1 -d 4 -m 'hadoop' -g 'hdfs:///user/leiyang/undirected_toy.txt' -w 'n' -i nu
ll -l 'n'

!python RunBFS.py -s 1 -d 5 -m 'hadoop' -g 'hdfs:///user/leiyang/directed_toy.txt' -w 'n' -i null
-l 'n'
```

```
2016-03-05 19:03:14.258764: BFS started between node 1 and node 4 on unweighted graph undirected_t
oy.txt ...
No handlers could be found for logger "mrjob.conf"
2016-03-05 19:03:14.509738: starting initialization job ...
2016-03-05 19:03:43.790088: moving results for next iteration ...
2016-03-05 19:03:45.499153: running iteration 1 ...
2016-03-05 19:04:16.218614: checking stopping criterion ...
2016-03-05 19:04:46.270971: destination is reached, retrieving path ...
2016-03-05 19:04:46.271192: clearing files ...
2016-03-05 19:04:49.349968: traversing completes in 1.6 minutes!

2016-03-05 19:04:49.350018: shortest path: 1 -> 2 -> 4

2016-03-05 19:04:49.595558: BFS started between node 1 and node 5 on unweighted graph directed_to
y.txt ...
No handlers could be found for logger "mrjob.conf"
2016-03-05 19:04:49.841229: starting initialization job ...
2016-03-05 19:05:19.919792: moving results for next iteration ...
2016-03-05 19:05:21.730136: running iteration 1 ...
2016-03-05 19:05:52.473447: checking stopping criterion ...
2016-03-05 19:06:23.077600: destination not reached yet.
2016-03-05 19:06:23.077630: moving results for next iteration ...
2016-03-05 19:06:26.333194: running iteration 2 ...
2016-03-05 19:06:55.499765: checking stopping criterion ...
2016-03-05 19:07:28.015478: destination is reached, retrieving path ...
2016-03-05 19:07:28.015511: clearing files ...
2016-03-05 19:07:31.231858: traversing completes in 2.7 minutes!

2016-03-05 19:07:31.231916: shortest path: 1 -> 2 -> 4 -> 5
```

Main dataset 1: NLTK synonyms

- In the next part of this assignment you will explore a network derived from the NLTK synonym database used for evaluation in HW 5.
- At a high level, this network is undirected, defined so that there exists link between two nodes/words if the pair or words are a synonym.

These data may be found at the location:

- s3://ucb-mids-mls-networks/synNet/synNet.txt
- s3://ucb-mids-mls-networks/synNet/indices.txt

On under the Data Subfolder for HW7 on Dropbox with the same file names

where synNet.txt contains a sparse representation of the network:

(index) \t (dictionary of links)

in indexed form, and indices.txt contains a lookup list

(word) \t (index)

of indices and words. This network is small enough for you to explore and run scripts locally, but will also be good for a systems test (for later) on AWS.

In the dictionary, target nodes are keys, link weights are values (here, all weights are 1, i.e., the network is unweighted).

HW 7.1: Exploratory data analysis (NLTK synonyms)

Using MRJob, explore the synonyms network data. Consider plotting the degree distribution (does it follow a power law?), and determine some of the key features, like:

- number of nodes,
- number links,
- or the average degree (i.e., the average number of links per node),
- etc...

As you develop your code, please be sure to run it locally first (though on the whole dataset). Once you have gotten you code to run locally, deploy it on AWS as a systems test in preparation for our next dataset (which will require AWS).

MrJob to explore the graph

In [43]:
```
%%writefile ExploreGraph.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class ExploreGraph(MRJob):

    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(ExploreGraph, self).__init__(*args, **kwargs)

    # assuming we are dealing with directed graph
    # this job can handle undirected graph as well,
    # but it can be done more efficiently
    def mapper(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        cmd = 'adj = %s' %dic
        exec cmd
        # we need to emit node ID as key, in order to count dangling nodes
        # the value emitted here is (out, in) degree for the node
        yield nid, (len(adj), 0)
        # emit in degree from adjacency list
        for n in adj:
            yield n, (0, 1)

    def combiner(self, nid, deg):
        din = dout = 0
        for d in deg:
            dout += d[0]
            din += d[1]
        yield nid, (dout, din)

    def reducer_init(self):
        self.node_cnt = 0
        self.out_deg = {}
        self.in_deg = {}

    def reducer(self, node, deg):
        # add node by 1
        self.node_cnt += 1
        # aggregate in/out degree
        _out = _in = 0
        for d in deg:
            _out += d[0]
            _in += d[1]
        # accumulate degree distribution
        if _out not in self.out_deg:
            self.out_deg[_out] = 1
        else:
            self.out_deg[_out] += 1
        if _in not in self.in_deg:
            self.in_deg[_in] = 1
        else:
            self.in_deg[_in] += 1


    def reducer_final(self):
        # final aggregation
        tot_degree = sum([d*self.out_deg[d] for d in self.out_deg])
        yield 'total nodes: ', self.node_cnt
```

```
        yield 'total links: ', tot_degree
        yield 'average in degree: ', 1.0*tot_degree/sum(self.in_deg.values())
        yield 'average out degree: ', 1.0*tot_degree/sum(self.out_deg.values())
        yield 'dist of in-degree: ', self.in_deg
        yield 'dist of out-degree: ', self.out_deg

    def steps(self):
        jc = {
            'mapreduce.job.maps': '10',
            'mapreduce.job.reduces': '1',
        }
        return [MRStep(mapper=self.mapper
                       , combiner=self.combiner
                       , reducer_init=self.reducer_init
                       , reducer=self.reducer
                       , reducer_final=self.reducer_final
                       #, jobconf = jc
                      )
                ]

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

```
Overwriting ExploreGraph.py
```

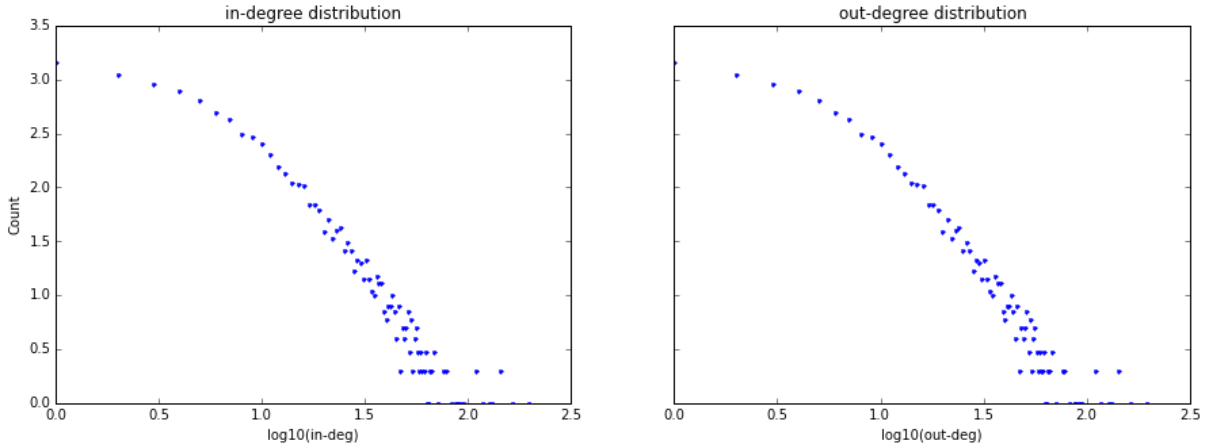
Explore NLTK Synonyms Net

In [44]:
```
!python ExploreGraph.py synNet.txt -r 'inline' > sumstats
!echo '\nSummary stats of the graph:'
!cat sumstats
```

```
using configs in /Users/leiyang/.mrjob.conf
creating tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160
305.194841.544439
writing to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160305.194841.5
44439/step-0-mapper_part-00000
Counters from step 1:
  (no counters found)
writing to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160305.194841.5
44439/step-0-mapper-sorted
> sort /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160305.194841.54443
9/step-0-mapper_part-00000
writing to /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160305.194841.5
44439/step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160305.194841.54443
9/step-0-reducer_part-00000 -> /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiya
ng.20160305.194841.544439/output/part-00000
Streaming final output from /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.
20160305.194841.544439/output
removing tmp directory /var/folders/tx/5ldq67q511q8wqwqkvptnxd00000gn/T/ExploreGraph.leiyang.20160
305.194841.544439

Summary stats of the graph:
"total nodes: " 8271
"total links: " 61134
"average in degree: "   7.391367428364164
"average out degree: "  7.391367428364164
"dist of in-degree: "   {"1": 1421, "2": 1127, "3": 906, "4": 783, "5": 637, "6": 488, "7": 429,
"8": 309, "9": 296, "10": 254, "11": 200, "12": 158, "13": 135, "14": 111, "15": 108, "16": 104,
"17": 70, "18": 70, "19": 62, "20": 39, "21": 51, "22": 34, "23": 40, "24": 42, "25": 26, "26": 3
1, "27": 26, "28": 17, "29": 21, "30": 20, "31": 14, "32": 21, "33": 14, "34": 11, "35": 10, "36":
15, "37": 13, "38": 13, "39": 7, "40": 6, "41": 8, "42": 8, "43": 10, "44": 7, "45": 4, "46": 8,
"47": 2, "48": 5, "49": 4, "50": 5, "51": 7, "52": 3, "53": 6, "54": 2, "55": 4, "56": 5, "57": 3,
"58": 2, "59": 3, "60": 2, "61": 2, "62": 3, "63": 1, "64": 1, "65": 2, "66": 2, "68": 3, "71": 1,
"76": 2, "78": 2, "83": 1, "87": 1, "88": 1, "164": 1, "90": 1, "143": 2, "93": 1, "95": 1, "131":
1, "196": 1, "110": 2, "117": 1, "127": 1}
"dist of out-degree: "  {"1": 1421, "2": 1127, "3": 906, "4": 783, "5": 637, "6": 488, "7": 429,
"8": 309, "9": 296, "10": 254, "11": 200, "12": 158, "13": 135, "14": 111, "15": 108, "16": 104,
"17": 70, "18": 70, "19": 62, "20": 39, "21": 51, "22": 34, "23": 40, "24": 42, "25": 26, "26": 3
1, "27": 26, "28": 17, "29": 21, "30": 20, "31": 14, "32": 21, "33": 14, "34": 11, "35": 10, "36":
15, "37": 13, "38": 13, "39": 7, "40": 6, "41": 8, "42": 8, "43": 10, "44": 7, "45": 4, "46": 8,
"47": 2, "48": 5, "49": 4, "50": 5, "51": 7, "52": 3, "53": 6, "54": 2, "55": 4, "56": 5, "57": 3,
"58": 2, "59": 3, "60": 2, "61": 2, "62": 3, "63": 1, "64": 1, "65": 2, "66": 2, "68": 3, "71": 1,
"76": 2, "78": 2, "83": 1, "87": 1, "88": 1, "164": 1, "90": 1, "143": 2, "93": 1, "95": 1, "131":
1, "196": 1, "110": 2, "117": 1, "127": 1}
```

Node degree distribution

In [52]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from math import log10

in_deg = {"1": 1421, "2": 1127, "3": 906, "4": 783, "5": 637, "6": 488, "7": 429, "8": 309, "9": 2
96, "10": 254, "11": 200, "12": 158, "13": 135, "14": 111, "15": 108, "16": 104, "17": 70, "18": 7
0, "19": 62, "20": 39, "21": 51, "22": 34, "23": 40, "24": 42, "25": 26, "26": 31, "27": 26, "28":
17, "29": 21, "30": 20, "31": 14, "32": 21, "33": 14, "34": 11, "35": 10, "36": 15, "37": 13, "3
8": 13, "39": 7, "40": 6, "41": 8, "42": 8, "43": 10, "44": 7, "45": 4, "46": 8, "47": 2, "48": 5,
"49": 4, "50": 5, "51": 7, "52": 3, "53": 6, "54": 2, "55": 4, "56": 5, "57": 3, "58": 2, "59": 3,
"60": 2, "61": 2, "62": 3, "63": 1, "64": 1, "65": 2, "66": 2, "68": 3, "71": 1, "76": 2, "78": 2,
"83": 1, "87": 1, "88": 1, "164": 1, "90": 1, "143": 2, "93": 1, "95": 1, "131": 1, "196": 1, "11
0": 2, "117": 1, "127": 1}
out_deg = {"1": 1421, "2": 1127, "3": 906, "4": 783, "5": 637, "6": 488, "7": 429, "8": 309, "9":
296, "10": 254, "11": 200, "12": 158, "13": 135, "14": 111, "15": 108, "16": 104, "17": 70, "18":
70, "19": 62, "20": 39, "21": 51, "22": 34, "23": 40, "24": 42, "25": 26, "26": 31, "27": 26, "2
8": 17, "29": 21, "30": 20, "31": 14, "32": 21, "33": 14, "34": 11, "35": 10, "36": 15, "37": 13,
"38": 13, "39": 7, "40": 6, "41": 8, "42": 8, "43": 10, "44": 7, "45": 4, "46": 8, "47": 2, "48":
5, "49": 4, "50": 5, "51": 7, "52": 3, "53": 6, "54": 2, "55": 4, "56": 5, "57": 3, "58": 2, "59":
3, "60": 2, "61": 2, "62": 3, "63": 1, "64": 1, "65": 2, "66": 2, "68": 3, "71": 1, "76": 2, "78":
2, "83": 1, "87": 1, "88": 1, "164": 1, "90": 1, "143": 2, "93": 1, "95": 1, "131": 1, "196": 1,
"110": 2, "117": 1, "127": 1}
# plot the in/out distribution histogram
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
f.set_size_inches([18,6])
ax1.plot([log10(int(k)) for k in in_deg.keys()], [log10(x) for x in in_deg.values()], 'r.')
ax1.set_title('in-degree distribution')
ax1.set_ylabel('log10(Count)')
ax1.set_xlabel('log10(in-deg)')
ax2.plot([log10(int(k)) for k in out_deg.keys()], [log10(x) for x in out_deg.values()], 'b.')
ax2.set_title('out-degree distribution')
ax2.set_xlabel('log10(out-deg)')
plt.show()
```



HW 7.2: Shortest path graph distances (NLTK synonyms)

Write (reuse your code from 7.0) an MRJob class to find shortest path graph distances, and apply it to the NLTK synonyms network dataset.

Proof your code's function by running the job:

- shortest path starting at "walk" (index=7827) and ending at "make" (index=536),

and showing you code's output. Once again, your output should include the path and the distance.

As you develop your code, please be sure to run it locally first (though on the whole dataset). Once you have gotten you code to run locally, deploy it on AWS as a systems test in preparation for our next dataset (which will require AWS).

In [160]:
```
##### execute RunBFS.py from HW 7.0 #####
!python RunBFS.py -s 7827 -d 536 -m 'hadoop' -g 'hdfs:///user/leiyang/synNet.txt' -w 'n' -i 'synN
et_indices.txt' -l 'n'
```

```
2016-03-05 19:08:13.838490: BFS started between node 7827 and node 536 on unweighted graph synNet.
txt ...
No handlers could be found for logger "mrjob.conf"
2016-03-05 19:08:14.107295: starting initialization job ...
```

```
2016-03-05 19:08:44.818035: moving results for next iteration ...
2016-03-05 19:08:46.541777: running iteration 1 ...
2016-03-05 19:09:16.725373: checking stopping criterion ...
2016-03-05 19:09:45.118749: destination not reached yet.
2016-03-05 19:09:45.118792: moving results for next iteration ...
2016-03-05 19:09:48.578630: running iteration 2 ...
2016-03-05 19:10:19.467043: checking stopping criterion ...
2016-03-05 19:10:49.079862: destination is reached, retrieving path ...
2016-03-05 19:10:49.079904: clearing files ...
2016-03-05 19:10:52.373540: traversing completes in 2.6 minutes!

2016-03-05 19:10:52.373580: shortest path: walk -> pass -> cleared -> make
```

Main dataset 2: English Wikipedia

For the remainder of this assignment you will explore the English Wikipedia hyperlink network. The dataset is built from the Sept. 2015 XML snapshot of English Wikipedia. For this directed network, a link between articles:

A -> B

is defined by the existence of a hyperlink in A pointing to B. This network also exists in the indexed format:

- s3://ucb-mids-mls-networks/wikipedia/all-pages-indexed-out.txt
- s3://ucb-mids-mls-networks/wikipedia/all-pages-indexed-in.txt
- s3://ucb-mids-mls-networks/wikipedia/indices.txt

On under the Data Subfolder for HW7 on Dropbox with the same file names

but has an index with more detailed data:

(article name) \t (index) \t (in degree) \t (out degree)

In the dictionary, target nodes are keys, link weights are values . Here, a weight indicates the number of time a page links to another. However, for the sake of this assignment, treat this an unweighted network, and set all weights to 1 upon data input.

HW 7.3: Exploratory data analysis (Wikipedia)

Using MRJob, explore the Wikipedia network data on the AWS cloud. Reuse your code from HW 7.1---does is scale well? Be cautioned that Wikipedia is a directed network, where links are not symmetric. So, even though a node may be linked to, it will not appear as a primary record itself if it has no out-links. This means that you may have to ADJUST your code (depending on its design). To be sure of your code's functionality in this context, run a systems test on the directed_toy.txt network.

In [1]:
```
#!python ExploreGraph.py 'all-pages-indexed-out.txt' -r 'hadoop' --cleanup 'NONE' > wiki_sumstats
!cat wiki_explore_emr.txt
```

```
[hadoop@ip-172-31-1-180 lei_wiki]$ ls -l
total 2041476
-rw-rw-r-- 1 hadoop hadoop 2090459616 Oct 22 02:29 all-pages-indexed-out.txt
-rw-rw-r-- 1 hadoop hadoop       2655 Mar  1 01:57 ExploreGraph.py
[hadoop@ip-172-31-1-180 lei_wiki]$ python ExploreGraph.py 'all-pages-indexed-out.txt' -r 'hadoop'
--cleanup 'NONE' > wiki_sumstats
using configs in /home/hadoop/.mrjob.conf
creating tmp directory /tmp/ExploreGraph.hadoop.20160301.015847.095270
writing wrapper script to /tmp/ExploreGraph.hadoop.20160301.015847.095270/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/hadoop/tmp/mrjob/ExploreGraph.hadoop.20160301.015847.095270/
files/
HADOOP: packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-2.7.1-amzn-0.jar] /tmp/streamjob232147
8639392700904.jar tmpDir=null
HADOOP: Connecting to ResourceManager at ip-172-31-1-180.ec2.internal/172.31.1.180:8032
HADOOP: Connecting to ResourceManager at ip-172-31-1-180.ec2.internal/172.31.1.180:8032
HADOOP: MetricsConfigRecord disabledInCluster: false instanceEngineCycleSec: 60 clusterEngineCycle
Sec: 60 disableClusterEngine: false maxMemoryMb: 3072 maxInstanceCount: 500 lastModified: 14567964
97690
HADOOP: Created MetricsSaver j-273F2CEVES2O8:i-2bfc5aaf:RunJar:10200 period:60 /mnt/var/em/raw/i-2
bfc5aaf_20160301_RunJar_10200_raw.bin
HADOOP: Loaded native gpl library
HADOOP: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 02f444f0932ea7710dcc4
bcdc1aa7ca55adf48c9]
HADOOP: Total input paths to process : 1
HADOOP: number of splits:16
HADOOP: Submitting tokens for job: job_1456796483811_0001
HADOOP: Submitted application application_1456796483811_0001
HADOOP: The url to track the job: http://ip-172-31-1-180.ec2.internal:20888/proxy/application_1456
796483811_0001/
```

```
796483811_0001/
HADOOP: Running job: job_1456796483811_0001
HADOOP: Job job_1456796483811_0001 running in uber mode : false
HADOOP:  map 0% reduce 0%
HADOOP:  map 1% reduce 0%
HADOOP:  map 2% reduce 0%
HADOOP:  map 3% reduce 0%
HADOOP:  map 4% reduce 0%
HADOOP:  map 5% reduce 0%
HADOOP:  map 6% reduce 0%
HADOOP:  map 7% reduce 0%
HADOOP:  map 8% reduce 0%
HADOOP:  map 9% reduce 0%
HADOOP:  map 10% reduce 0%
HADOOP:  map 11% reduce 0%
HADOOP:  map 12% reduce 0%
HADOOP:  map 13% reduce 0%
HADOOP:  map 14% reduce 0%
HADOOP:  map 15% reduce 0%
HADOOP:  map 16% reduce 0%
HADOOP:  map 17% reduce 0%
HADOOP:  map 18% reduce 0%
HADOOP:  map 19% reduce 0%
HADOOP:  map 20% reduce 0%
HADOOP:  map 21% reduce 0%
HADOOP:  map 22% reduce 0%
HADOOP:  map 23% reduce 0%
HADOOP:  map 24% reduce 0%
HADOOP:  map 25% reduce 0%
HADOOP:  map 26% reduce 0%
HADOOP:  map 27% reduce 0%
HADOOP:  map 28% reduce 0%
HADOOP:  map 29% reduce 0%
HADOOP:  map 30% reduce 0%
HADOOP:  map 31% reduce 0%
HADOOP:  map 32% reduce 0%
HADOOP:  map 34% reduce 0%
HADOOP:  map 35% reduce 0%
HADOOP:  map 36% reduce 0%
HADOOP:  map 37% reduce 0%
HADOOP:  map 38% reduce 0%
HADOOP:  map 39% reduce 0%
HADOOP:  map 40% reduce 0%
HADOOP:  map 41% reduce 0%
HADOOP:  map 42% reduce 0%
HADOOP:  map 43% reduce 0%
HADOOP:  map 44% reduce 0%
HADOOP:  map 45% reduce 0%
HADOOP:  map 46% reduce 0%
HADOOP:  map 47% reduce 0%
HADOOP:  map 48% reduce 0%
HADOOP:  map 49% reduce 0%
HADOOP:  map 50% reduce 0%
HADOOP:  map 51% reduce 0%
HADOOP:  map 52% reduce 0%
HADOOP:  map 54% reduce 0%
HADOOP:  map 55% reduce 0%
HADOOP:  map 56% reduce 0%
HADOOP:  map 58% reduce 0%
HADOOP:  map 59% reduce 0%
HADOOP:  map 61% reduce 0%
HADOOP:  map 63% reduce 0%
HADOOP:  map 65% reduce 0%
HADOOP:  map 67% reduce 0%
HADOOP:  map 69% reduce 0%
HADOOP:  map 71% reduce 0%
HADOOP:  map 72% reduce 0%
HADOOP:  map 73% reduce 0%
HADOOP:  map 73% reduce 4%
HADOOP:  map 75% reduce 4%
HADOOP:  map 75% reduce 17%
HADOOP:  map 76% reduce 17%
HADOOP:  map 77% reduce 17%
HADOOP:  map 78% reduce 17%
HADOOP:  map 79% reduce 17%
HADOOP:  map 80% reduce 17%
HADOOP:  map 81% reduce 17%
HADOOP:  map 82% reduce 17%
HADOOP:  map 83% reduce 17%
HADOOP:  map 84% reduce 17%
```

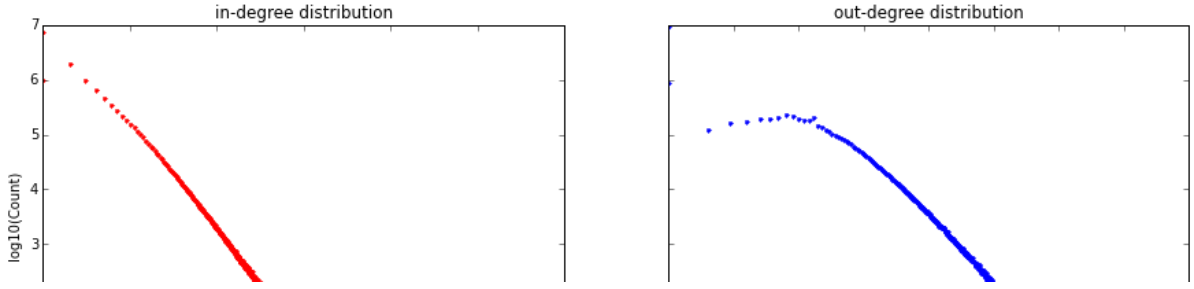
```
HADOOP:  map 85% reduce 17%
HADOOP:  map 86% reduce 17%
HADOOP:  map 87% reduce 17%
HADOOP:  map 88% reduce 17%
HADOOP:  map 89% reduce 17%
HADOOP:  map 90% reduce 17%
HADOOP:  map 92% reduce 17%
HADOOP:  map 94% reduce 17%
HADOOP:  map 95% reduce 17%
HADOOP:  map 97% reduce 17%
HADOOP:  map 98% reduce 17%
HADOOP:  map 100% reduce 17%
HADOOP:  map 100% reduce 27%
HADOOP:  map 100% reduce 33%
HADOOP:  map 100% reduce 67%
HADOOP:  map 100% reduce 68%
HADOOP:  map 100% reduce 69%
HADOOP:  map 100% reduce 70%
HADOOP:  map 100% reduce 71%
HADOOP:  map 100% reduce 72%
HADOOP:  map 100% reduce 73%
HADOOP:  map 100% reduce 74%
HADOOP:  map 100% reduce 75%
HADOOP:  map 100% reduce 76%
HADOOP:  map 100% reduce 77%
HADOOP:  map 100% reduce 78%
HADOOP:  map 100% reduce 79%
HADOOP:  map 100% reduce 80%
HADOOP:  map 100% reduce 81%
HADOOP:  map 100% reduce 82%
HADOOP:  map 100% reduce 83%
HADOOP:  map 100% reduce 84%
HADOOP:  map 100% reduce 85%
HADOOP:  map 100% reduce 86%
HADOOP:  map 100% reduce 87%
HADOOP:  map 100% reduce 88%
HADOOP:  map 100% reduce 89%
HADOOP:  map 100% reduce 90%
HADOOP:  map 100% reduce 91%
HADOOP:  map 100% reduce 92%
HADOOP:  map 100% reduce 93%
HADOOP:  map 100% reduce 94%
HADOOP:  map 100% reduce 95%
HADOOP:  map 100% reduce 96%
HADOOP:  map 100% reduce 97%
HADOOP:  map 100% reduce 98%
HADOOP:  map 100% reduce 99%
HADOOP:  map 100% reduce 100%
HADOOP: Job job_1456796483811_0001 completed successfully
HADOOP: Counters: 51
HADOOP:         File System Counters
HADOOP:                 FILE: Number of bytes read=551813329
HADOOP:                 FILE: Number of bytes written=901418328
HADOOP:                 FILE: Number of read operations=0
HADOOP:                 FILE: Number of large read operations=0
HADOOP:                 FILE: Number of write operations=0
HADOOP:                 HDFS: Number of bytes read=2091445696
HADOOP:                 HDFS: Number of bytes written=76840
HADOOP:                 HDFS: Number of read operations=51
HADOOP:                 HDFS: Number of large read operations=0
HADOOP:                 HDFS: Number of write operations=2
HADOOP:         Job Counters
HADOOP:                 Killed map tasks=3
HADOOP:                 Launched map tasks=19
HADOOP:                 Launched reduce tasks=1
HADOOP:                 Data-local map tasks=14
HADOOP:                 Rack-local map tasks=5
HADOOP:                 Total time spent by all maps in occupied slots (ms)=415795488
HADOOP:                 Total time spent by all reduces in occupied slots (ms)=85978592
HADOOP:                 Total time spent by all map tasks (ms)=17324812
HADOOP:                 Total time spent by all reduce tasks (ms)=2686831
HADOOP:                 Total vcore-seconds taken by all map tasks=17324812
HADOOP:                 Total vcore-seconds taken by all reduce tasks=2686831
HADOOP:                 Total megabyte-seconds taken by all map tasks=13305455616
HADOOP:                 Total megabyte-seconds taken by all reduce tasks=2751314944
HADOOP:         Map-Reduce Framework
HADOOP:                 Map input records=5781290
HADOOP:                 Map output records=147895347
HADOOP:                 Map output bytes=2566288353
```

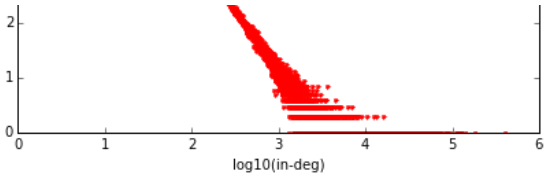
```
HADOOP:                 Map output materialized bytes=347440319
HADOOP:                 Input split bytes=3040
HADOOP:                 Combine input records=210009639
HADOOP:                 Combine output records=88093113
HADOOP:                 Reduce input groups=15192277
HADOOP:                 Reduce shuffle bytes=347440319
HADOOP:                 Reduce input records=25978821
HADOOP:                 Reduce output records=6
HADOOP:                 Spilled Records=150207405
HADOOP:                 Shuffled Maps =16
HADOOP:                 Failed Shuffles=0
HADOOP:                 Merged Map outputs=16
HADOOP:                 GC time elapsed (ms)=22362
HADOOP:                 CPU time spent (ms)=10309990
HADOOP:                 Physical memory (bytes) snapshot=6520324096
HADOOP:                 Virtual memory (bytes) snapshot=22371188736
HADOOP:                 Total committed heap usage (bytes)=5087166464
HADOOP:         Shuffle Errors
HADOOP:                 BAD_ID=0
HADOOP:                 CONNECTION=0
HADOOP:                 IO_ERROR=0
HADOOP:                 WRONG_LENGTH=0
HADOOP:                 WRONG_MAP=0
HADOOP:                 WRONG_REDUCE=0
HADOOP:         File Input Format Counters
HADOOP:                 Bytes Read=2091442656
HADOOP:         File Output Format Counters
HADOOP:                 Bytes Written=76840
HADOOP: Output directory: hdfs:///user/hadoop/tmp/mrjob/ExploreGraph.hadoop.20160301.015847.09527
0/output
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/hadoop/tmp/mrjob/ExploreGraph.hadoop.20160301.015847.0952
70/output
[hadoop@ip-172-31-1-180 lei_wiki]$
```

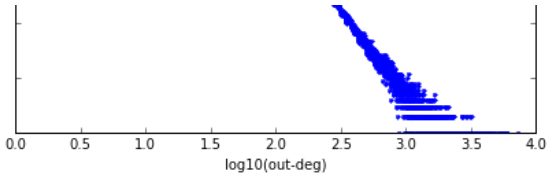
In [18]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from math import log10

with open('wiki_sumstats_emr.txt', 'r') as f:
    lines = f.readlines()
for line in lines:
    if 'dist of' not in line:
        print line.strip()
    else:
        dc, dic = line.split('\t')
        cmd = '%s = %s' %('in_deg' if 'in' in dc else 'out_deg', dic)
        exec cmd

# plot the in/out distribution histogram
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
f.set_size_inches([18,6])
ax1.plot([0 if k=='0' else log10(int(k)) for k in in_deg.keys()],
         [0 if i=='0' else log10(i) for i in in_deg.values()], 'r.')
ax1.set_title('in-degree distribution')
ax1.set_ylabel('log10(Count)')
ax1.set_xlabel('log10(in-deg)')
ax2.plot([0 if k=='0' else log10(int(k)) for k in out_deg.keys()],
         [0 if k=='0' else log10(k) for k in out_deg.values()], 'b.')
ax2.set_title('out-degree distribution')
ax2.set_xlabel('log10(out-deg)')
plt.show()
```

```
"total nodes: " 15192277
"total links: " 142114057
"average in degree: "   9.354361890584276
"average out degree: "  9.354361890584276
```







HW 7.4: Shortest path graph distances (Wikipedia)

Using MRJob, find shortest path graph distances in the Wikipedia network on the AWS cloud. Reuse your code from 7.2, but once again be warned of Wikipedia being a directed network. To be sure of your code's functionality in this context, run a systems test on the directed_toy.txt network.

When running your code on the Wikipedia network, proof its function by running the job:

- shortest path from "Ireland" (index=6176135) to "University of California, Berkeley" (index=13466359),

and show your code's output.

Once your code is running, find some other shortest paths and report your results.

BFS on Wikipedia dataset

- List of machine learning concepts 7861634 1 91 --> University of Toronto Schools 13472800 114 144
- Ireland 6176135 --> University of California, Berkeley 13466359

In [328]:
```
#!python RunBFS.py -s 6176135 -d 13466359 -m 'hadoop' -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -w 'n' -i 'wiki_index.txt' -l 'n'
#!python RunBFS.py -s 7861634 -d 13472800 -m 'hadoop' -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -w 'n' -i 'wiki_index.txt' -l 'n'
!cat wiki_BFS_log
```

```
[hadoop@ip-172-31-59-125 lei]$ python RunBFS.py -s 6176135 -d 13466359 -m 'hadoop' -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -w 0 -i 'wiki_index.txt'
2016-03-06 01:33:26.554194: BFS started between node 6176135 and node 13466359 on unweighted graph all-pages-indexed-out.txt ...
2016-03-06 01:33:26.675776: starting initialization job ...
2016-03-06 01:46:18.214446: moving results for next iteration ...
2016-03-06 01:46:24.341754: running iteration 1 ...
2016-03-06 01:58:11.857995: checking stopping criterion ...
2016-03-06 02:03:00.944833: destination is reached, retrieving path ...
2016-03-06 02:03:00.944968: clearing files ...
2016-03-06 02:03:27.535561: traversing completes in 30.0 minutes!

2016-03-06 02:03:27.535736: shortest path: Ireland -> Seamus Heaney -> University of California, Berkeley

[hadoop@ip-172-31-59-125 lei]$ python RunBFS.py -s 7861634 -d 13472800 -m 'hadoop' -g 'hdfs:///user/leiyang/all-pages-indexed-out.txt' -w '0' -i 'wiki_index.txt'
2016-03-06 04:04:26.412718: BFS started between node 7861634 and node 13472800 on unweighted graph all-pages-indexed-out.txt ...
2016-03-06 04:04:26.560733: starting initialization job ...
2016-03-06 04:15:57.179658: moving results for next iteration ...
2016-03-06 04:16:02.978906: running iteration 1 ...
2016-03-06 04:27:10.875095: checking stopping criterion ...
2016-03-06 04:31:45.328014: destination not reached yet.
2016-03-06 04:31:45.328144: moving results for next iteration ...
2016-03-06 04:32:01.208318: running iteration 2 ...
2016-03-06 04:42:34.688983: checking stopping criterion ...
2016-03-06 04:47:10.074148: destination is reached, retrieving path ...
2016-03-06 04:47:10.074306: clearing files ...
2016-03-06 04:47:24.722793: traversing completes in 43.0 minutes!

2016-03-06 04:47:24.722975: shortest path: List of machine learning concepts -> Artificial neural network -> University of Toronto -> University of Toronto Schools
```

HW 7.5: Conceptual exercise: Largest single-source network distances

Suppose you wanted to find the largest network distance from a single source, i.e., a node that is the furthest (but still reachable) from a single source.

How would you implement this task? How is this different from finding the shortest path graph distances?

Is this task more difficult to implement than the shortest path distance?

As you respond, please comment on program structure, runtimes, iterations, general system requirements, etc...

Answer:

In theory, the below step should work to get the largest network distance

- change the distance comparison step, instead of keeping smaller distance, choose the larger distance
- stopping criterion: all distances from all nodes don't change anymore
- the challenge here is to avoid the scenario where the traversal going back-and-forth between nodes to increase distance count, the mapper needs to change accordingly: not emit nodes from the adjacency list that are on the current incoming path, so that the path won't go in loops or bounce between nodes.

However, this approach is not practical, in that

- theorectically the largest distances can be the number of nodes in the network, when all node are traversed on a single path
- for BFS strategy, each iteration will only grow the distance by one, and in the meantime increase the graph file dramatically, in order to keep track of the path.
- below code implemented this method and tested on the toy network, but for wiki file, it's not a realistic way
- in general, longest path problem (https://en.wikipedia.org/wiki/Longest_path_problem) is NP-hard, and needs different philosophy in order to effectly address.

HW 7.6 (optional): Computational exercise: Largest single-source network distances

Using MRJob, write a code to find the largest graph distance and distance-maximizing nodes from a single-source. Test your code first on the toy networks and synonyms network to proof its function.

Longest path iteration

In [294]:
```
%%writefile LongestPathIter.py
from mrjob.job import MRJob
from mrjob.step import MRStep

class LongestPathIter(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(LongestPathIter, self).__init__(*args, **kwargs)


    def configure_options(self):
        super(LongestPathIter, self).configure_options()
        self.add_passthrough_option(
            '--source', dest='source', default='1', type='string',
            help='source: source node (default 1)')
        self.add_passthrough_option(
            '--weighted', dest='weighted', default='n', type='string',
            help='weighted: is weighted graph (default n)')


    def mapper_weighted(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %dic
        exec cmd
        # if the node structure is incomplete (first pass), add them
        if 'dist' not in node:
            node = {'adj':node, 'path':[]}
            node['dist'] = 0 if self.options.source==nid else -1
        # emit node
        yield nid, node
        # emit distances to reachable nodes
        if node['dist'] >= 0:
            for m in node['adj']:
                yield m, {'dd':node['adj'][m] + node['dist'], 'pp':node['path']+[nid]}

    def mapper_unweighted(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        nid = nid.strip('"')
        cmd = 'node = %s' %dic
        exec cmd
        # if the node structure is incomplete (first pass), add them
        if 'dist' not in node:
```

```
            node = {'adj':node.keys(), 'path':[]}
            node['dist'] = 0 if self.options.source==nid else -1
        # emit node
        yield nid, node
        # emit distances to reachable nodes
        if node['dist'] >= 0:
            for m in node['adj']:
                if m not in node['path']:
                    yield m, {'dd':(1+node['dist']), 'pp':(node['path']+[nid])}

    # write a separate combiner ensure the integrity of the graph topology
    # no additional node object will be generated
    def combiner(self, nid, value):
        dmax = 0
        path = None
        # loop through all arrivals
        for v in value:
            if 'dist' in v:
                yield nid, v
            elif v['dd'] > dmax:
                dmax, path = v['dd'], v['pp']
        # emit the smallest distance for nid
        if path:
            yield nid, {'dd':dmax, 'pp':path}

    def reducer(self, nid, value):
        dmax = -2
        path = node = None
        # loop through all arrivals
        for v in value:
            if 'dist' in v:
                node = v
            elif v['dd'] > dmax:
                dmax, path = v['dd'], v['pp']

        # handle dangling node, we only care if it's destination
        if not node:
            node = {'adj':{}, 'dist':dmax, 'path':path}
        elif (dmax > node['dist']): # or (node['dist'] == -1 and path):
            node['dist'], node['path'] = dmax, path

        # emit for next iteration
        yield nid, node

    def steps(self):
        jc = {
            'mapreduce.job.maps': '1',
            'mapreduce.job.reduces': '1',
        }
        return [MRStep(mapper=self.mapper_weighted if self.options.weighted == 'y' else self.mapp
er_unweighted
                       , combiner=self.combiner
                       , reducer=self.reducer
                       , jobconf = jc
                      )
                ]

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

```
Overwriting LongestPathIter.py
```

In [323]:
```
##### unit test #####
#!python LongestPathIter.py 'synNet.txt' --source 7827 --weighted 'n'
#!python LongestPathIter.py 'undirected_toy.txt' --source 1 --weighted 'n' > test1
#!python LongestPathIter.py 'test3' --source 1 --weighted 'n' > test4
```

Get the longest distance

In [316]:
```
%%writefile getLongestDistance.py
from mrjob.job import MRJob
from mrjob.step import MRStep


class getLongestDistance(MRJob):
    DEFAULT_PROTOCOL = 'json'
```

```
    def __init__(self, *args, **kwargs):
        super(getLongestDistance, self).__init__(*args, **kwargs)


    def mapper_init(self):
        self.longest = []

    def mapper(self, _, line):
        nid, dic = line.strip().split('\t', 1)
        cmd = 'node = %s' %dic
        exec cmd
        if len(node['path']) >= len(self.longest):
            self.longest = node['path']+[nid.strip('"')]

    def mapper_final(self):
        yield "longest distance - ", self.longest

    def steps(self):
        jc = {
            'mapreduce.job.maps': '1',
        }
        return [MRStep(mapper_init=self.mapper_init
                       , mapper=self.mapper
                       , mapper_final=self.mapper_final
                       , jobconf = jc
                      )
                ]

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

```
Overwriting getLongestDistance.py
```

In [324]:
```
##### unit test #####
#!python getLongestDistance.py test4
```

In [322]:
```
##### unit test #####
!python RunBFS.py -s 1 -d 0 -m 'hadoop' -g 'hdfs:///user/leiyang/undirected_toy.txt' -w 'n' -i 'n
ull' -l 'y'
```

```
2016-03-06 00:32:38.928663: Longest distance BFS started at node 1 on unweighted graph undirected_
toy.txt ...
No handlers could be found for logger "mrjob.conf"
2016-03-06 00:32:39.211086: starting initialization job ...
2016-03-06 00:33:09.761798: moving results for next iteration ...
2016-03-06 00:33:11.815789: running iteration 1 ...
2016-03-06 00:33:40.933963: checking stopping criterion ...
2016-03-06 00:34:16.496482: 4 nodes changed distance
2016-03-06 00:34:16.496522: moving results for next iteration ...
2016-03-06 00:34:20.100232: running iteration 2 ...
2016-03-06 00:34:48.868586: checking stopping criterion ...
2016-03-06 00:35:22.096681: 3 nodes changed distance
2016-03-06 00:35:22.096717: moving results for next iteration ...
2016-03-06 00:35:25.422469: running iteration 3 ...
2016-03-06 00:35:54.214739: checking stopping criterion ...
2016-03-06 00:36:28.208354: 2 nodes changed distance
2016-03-06 00:36:28.208389: moving results for next iteration ...
2016-03-06 00:36:31.744452: running iteration 4 ...
2016-03-06 00:37:01.460318: checking stopping criterion ...
2016-03-06 00:37:36.451759: traverse has completed, retrieving path ...
2016-03-06 00:38:04.219553: clearing files ...
2016-03-06 00:38:05.861466: traversing completes in 5.4 minutes!

2016-03-06 00:38:05.861580: longest path: 1 -> 5 -> 4 -> 3 -> 2
```

Find longest graph distance for synNet

In [327]:
```
!python RunBFS.py -s 7827 -d 0 -m 'hadoop' -g 'hdfs:///user/leiyang/synNet.txt' -w 'n' -i 'synNet
_indices.txt' -l 'y'
```

```
2016-03-06 00:41:53.045348: Longest distance BFS started at node 7827 on unweighted graph synNet.t
xt ...
No handlers could be found for logger "mrjob.conf"
2016-03-06 00:41:53.323959: starting initialization job ...
2016-03-06 00:42:23.496350: moving results for next iteration ...
2016-03-06 00:42:25.353791: running iteration 1 ...
2016-03-06 00:42:55.294750: checking stopping criterion ...
```

```
2016-03-06 00:43:29.828506: 111 nodes changed distance
2016-03-06 00:43:29.828548: moving results for next iteration ...
2016-03-06 00:43:33.269016: running iteration 2 ...
2016-03-06 00:44:02.945366: checking stopping criterion ...
2016-03-06 00:44:39.462602: 686 nodes changed distance
2016-03-06 00:44:39.462640: moving results for next iteration ...
2016-03-06 00:44:42.897357: running iteration 3 ...
2016-03-06 00:45:13.069223: checking stopping criterion ...
2016-03-06 00:45:47.857125: 2597 nodes changed distance
2016-03-06 00:45:47.857187: moving results for next iteration ...
2016-03-06 00:45:51.399726: running iteration 4 ...
2016-03-06 00:46:22.075929: checking stopping criterion ...
2016-03-06 00:46:56.650977: 4739 nodes changed distance
2016-03-06 00:46:56.651023: moving results for next iteration ...
2016-03-06 00:47:00.126158: running iteration 5 ...
2016-03-06 00:47:30.368259: checking stopping criterion ...
2016-03-06 00:48:04.754511: 5939 nodes changed distance
2016-03-06 00:48:04.754550: moving results for next iteration ...
2016-03-06 00:48:08.056691: running iteration 6 ...
2016-03-06 00:48:40.375529: checking stopping criterion ...
2016-03-06 00:49:13.472244: 6362 nodes changed distance
2016-03-06 00:49:13.472280: moving results for next iteration ...
2016-03-06 00:49:16.761586: running iteration 7 ...
2016-03-06 00:49:47.031885: checking stopping criterion ...
2016-03-06 00:50:21.481270: 6491 nodes changed distance
2016-03-06 00:50:21.481306: moving results for next iteration ...
2016-03-06 00:50:24.735636: running iteration 8 ...
2016-03-06 00:50:55.791621: checking stopping criterion ...
2016-03-06 00:51:29.774611: 6526 nodes changed distance
2016-03-06 00:51:29.774646: moving results for next iteration ...
2016-03-06 00:51:33.035606: running iteration 9 ...
2016-03-06 00:52:03.977695: checking stopping criterion ...
2016-03-06 00:52:37.567296: 6541 nodes changed distance
2016-03-06 00:52:37.567332: moving results for next iteration ...
2016-03-06 00:52:40.794911: running iteration 10 ...
2016-03-06 00:53:10.207203: checking stopping criterion ...
2016-03-06 00:53:44.141662: 6540 nodes changed distance
2016-03-06 00:53:44.141698: moving results for next iteration ...
2016-03-06 00:53:47.511667: running iteration 11 ...
2016-03-06 00:54:20.494333: checking stopping criterion ...
2016-03-06 00:54:54.971660: 6544 nodes changed distance
2016-03-06 00:54:54.971700: moving results for next iteration ...
2016-03-06 00:54:58.314589: running iteration 12 ...
2016-03-06 00:55:29.422773: checking stopping criterion ...
2016-03-06 00:56:03.597949: 6542 nodes changed distance
2016-03-06 00:56:03.597985: moving results for next iteration ...
2016-03-06 00:56:06.955033: running iteration 13 ...
2016-03-06 00:56:37.695321: checking stopping criterion ...
2016-03-06 00:57:11.537365: 6542 nodes changed distance
2016-03-06 00:57:11.537402: moving results for next iteration ...
2016-03-06 00:57:15.013627: running iteration 14 ...
2016-03-06 00:57:45.669783: checking stopping criterion ...
2016-03-06 00:58:19.477599: 6543 nodes changed distance
2016-03-06 00:58:19.477644: moving results for next iteration ...
2016-03-06 00:58:23.023297: running iteration 15 ...
2016-03-06 00:58:54.002759: checking stopping criterion ...
2016-03-06 00:59:28.683817: 6543 nodes changed distance
2016-03-06 00:59:28.683895: moving results for next iteration ...
2016-03-06 00:59:31.940197: running iteration 16 ...
2016-03-06 01:00:02.996792: checking stopping criterion ...
2016-03-06 01:00:36.986858: 6543 nodes changed distance
2016-03-06 01:00:36.986895: moving results for next iteration ...
2016-03-06 01:00:40.264892: running iteration 17 ...
2016-03-06 01:01:09.441314: checking stopping criterion ...
2016-03-06 01:01:43.557613: 6540 nodes changed distance
2016-03-06 01:01:43.557654: moving results for next iteration ...
2016-03-06 01:01:47.031037: running iteration 18 ...
2016-03-06 01:02:17.868729: checking stopping criterion ...
2016-03-06 01:02:51.574480: 6542 nodes changed distance
2016-03-06 01:02:51.574517: moving results for next iteration ...
2016-03-06 01:02:55.009621: running iteration 19 ...
2016-03-06 01:03:25.723651: checking stopping criterion ...
2016-03-06 01:03:59.699792: 6542 nodes changed distance
2016-03-06 01:03:59.699833: moving results for next iteration ...
2016-03-06 01:04:03.273161: running iteration 20 ...
2016-03-06 01:04:34.986758: checking stopping criterion ...
2016-03-06 01:05:10.614589: 6543 nodes changed distance
2016-03-06 01:05:10.614626: moving results for next iteration ...
2016-03-06 01:05:13.857444: running iteration 21 ...
2016-03-06 01:05:44.276611: checking stopping criterion
```

```
2016-03-06 01:06:18.512695: 6542 nodes changed distance
2016-03-06 01:06:18.512731: moving results for next iteration ...
2016-03-06 01:06:21.890786: running iteration 22 ...
2016-03-06 01:06:52.761740: checking stopping criterion ...
2016-03-06 01:07:27.391854: 6542 nodes changed distance
2016-03-06 01:07:27.391898: moving results for next iteration ...
2016-03-06 01:07:30.707943: running iteration 23 ...
2016-03-06 01:08:02.985101: checking stopping criterion ...
2016-03-06 01:08:37.014382: 6542 nodes changed distance
2016-03-06 01:08:37.014418: moving results for next iteration ...
2016-03-06 01:08:40.301410: running iteration 24 ...
2016-03-06 01:09:09.780028: checking stopping criterion ...
2016-03-06 01:09:43.449238: 6542 nodes changed distance
2016-03-06 01:09:43.449286: moving results for next iteration ...
2016-03-06 01:09:46.808870: running iteration 25 ...
2016-03-06 01:10:18.758574: checking stopping criterion ...
2016-03-06 01:10:52.409067: 6541 nodes changed distance
2016-03-06 01:10:52.409105: moving results for next iteration ...
2016-03-06 01:10:55.729808: running iteration 26 ...
2016-03-06 01:11:25.525758: checking stopping criterion ...
2016-03-06 01:11:59.177439: 6542 nodes changed distance
2016-03-06 01:11:59.177473: moving results for next iteration ...
2016-03-06 01:12:02.815666: running iteration 27 ...
^CTraceback (most recent call last):
  File "RunBFS.py", line 92, in <module>
    runner.run()
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/runner.py", line 470, in run
    self._run()
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/hadoop.py", line 233, in _run
    self._check_input_exists()
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/hadoop.py", line 247, in _check_
input_exists
    if not self.path_exists(path):
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/fs/composite.py", line 78, in pa
th_exists
    return self._do_action('path_exists', path_glob)
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/fs/composite.py", line 54, in _d
o_action
    return getattr(fs, action)(path, *args, **kwargs)
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/fs/hadoop.py", line 212, in path
_exists
    ok_stderr=[_HADOOP_LS_NO_SUCH_FILE])
  File "/Users/leiyang/anaconda/lib/python2.7/site-packages/mrjob/fs/hadoop.py", line 87, in invok
e_hadoop
    stdout, stderr = proc.communicate()
  File "/Users/leiyang/anaconda/lib/python2.7/subprocess.py", line 799, in communicate
    return self._communicate(input)
  File "/Users/leiyang/anaconda/lib/python2.7/subprocess.py", line 1409, in _communicate
    stdout, stderr = self._communicate_with_poll(input)
  File "/Users/leiyang/anaconda/lib/python2.7/subprocess.py", line 1463, in _communicate_with_poll
    ready = poller.poll()
KeyboardInterrupt
```

stop yarn, hdfs, and job history

In [329]:
```
!/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*/li
bexec/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
```

start yarn, hdfs, and job history