In [4]:
```
#imports libraries
import pandas as pd
```

In [77]:
```
def indexValuePick(indexes, max_index):
    table = []
    for index, item in enumerate(indexes[::-1]):
            #print "index: "+str(index)+" item: "+str(item)
        if item<max_index-index:
            table.append(True)
        else:
            table.append(False)
    return table

def index_to_increment(table):
    for index,boolean in enumerate(table):
        if boolean:
            return len(table)-index -1
    return -1
#Input:
    #indexes: list of indexes or integers
    #max_value an index can take
#Output:
    #The input indexes with one of them incremented or -1 if not needed
def incrememtor(indexes, max_index):

    truth_table = indexValuePick(indexes, max_index)
    #index to increment
    index = index_to_increment(truth_table)
    if index!=-1:
        indexes[index]+=1
        for i in range(len(indexes[index:])):
            indexes[index+i]=indexes[index]+i
        return indexes
    return -1
```

In [56]:
```
my_list = [1,2,3,4,5]
d = {i**2:i for i in my_list}
range(1,3)
```

Out[56]:
```
[1, 2]
```

In [87]:
```
#Sampling latitudal points
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches
import random as rd
import numpy as np

class LatitudalPolygon:
    def __init__(self,bound1 = (42.745458, -73.265071),bound2=(42.061597, -73.527701),
                bound3 = (41.247793, -70.002255), bound4 = (42.948057, -70.589209)):
        self.bound1 = bound1
        self.bound2 = bound2
        self.bound3 = bound3
        self.bound4 = bound4
        #define plot bounds
        self.latMin = min([bound[0] for bound in [self.bound1,self.bound2, self.bound3,se
lf.bound4]])
        self.latMax = max([bound[0] for bound in [self.bound1,self.bound2, self.bound3,se
lf.bound4]])
        self.longMin = min([bound[1] for bound in [self.bound1,self.bound2, self.bound3,s
elf.bound4]])
        self.longMax = max([bound[1] for bound in [self.bound1,self.bound2, self.bound3,s
elf.bound4]])
    def plot(self,lat_offset=.125, long_offset = .125):
        verts = [
        self.bound1,self.bound2,self.bound3,self.bound4,self.bound1
        ]
        codes = [
            Path.MOVETO,
             Path.LINETO,
             Path.LINETO,
             Path.LINETO,
             Path.CLOSEPOLY,
        ]
        path = Path(verts, codes)
        fig =  plt.figure()
        ax = fig.add_subplot(111)
        patch = patches.PathPatch(path, facecolor='orange', lw=2)
        ax.add_patch(patch)
        ax.set_xlim(self.latMin-lat_offset,self.latMax+lat_offset)
        ax.set_ylim(self.longMin-long_offset,self.longMax+long_offset)
        plt.show()
    #helper function for incrementor
    #Inputs:
        #Indexes:
        #max_index:
    #Output:
        #Truth table indicating the non-maxed out indices
    def indexValuePick(self, indexes, max_index):
        table = []
        for index, item in enumerate(indexes[::-1]):
            #print "index: "+str(index)+" item: "+str(item)
            if item<max_index-index:
                table.append(True)
            else:
                table.append(False)
        return table
    #Helper function for incrementor
    #Input:
        #Table: True table indicating indices that hasn't maxed out yet
    #Output:
        #Index of the table to be incremented first
    def index_to_increment(self,table):
        for index,boolean in enumerate(table):
            if boolean:
                return len(table)-index -1
        return -1
    #Helper function for pointSets
    #Input:
        #indexes: list of indexes or integers
        #max_value an index can take
    #Output:
        #The input indexes with one of them incremented or -1 if not needed
    def incrememtor(self, indexes, max_index):

        truth_table = self.indexValuePick(indexes, max_index)
        #index to increment
        index = self.index_to_increment(truth_table)
        if index!=-1:
            indexes[index]+=1
            for i in range(len(indexes[index:])):
                indexes[index+i]=indexes[index]+i
            return indexes
        return -1
    #Helper function for pointSets
    #Input:
        #indexes: list of indexes to extract points from
        #points: List of points to extract points from
    #Output:
        #points: List of points with at given indices
    def extractPoints(self, indexes,points):
        points = [points[i] for i in indexes]
        return points
    #Input:
        #points: List of points within in the polygon
        #Numbpoints: number of points in a group(list) of points
    #Output:
        #list of lists of points each with length numPoints
    def pointSets(self, numPoint, points):

        point_set = []

        indexes = range(numPoint)
        max_index = len(points)-1

        while isinstance(indexes, list):
            #print 'running with indices: '+str(indexes)+" max_index: "+str(max_index)
            extracted_points = self.extractPoints(indexes,points)
            indexes = self.incrememtor(indexes, max_index)
            #print 'running with indices: '+str(indexes)+" max_index: "+str(max_index)
            point_set.append(extracted_points)
        return point_set
    #Helper function for point_set_min_dist
    #Input:
        #point1: latitudal point(latitude, logitude)
        #point2: latitudal point(latidue, logitude)
    #Output:
        #The euclidean distance between point1 and point2, assumes the latitude/logitude
lines are fairly straight
    def distance_calc(self,point1, point2):
        distance = np.sqrt((point1[0]-point2[0])**2+(point1[1]-point2[1])**2)
        return distance
    #Helper function for variances
    #Input:
        #point_set: a list of points within the polygon
    #Output:
        #minimum of the distances between any two points in the list
    def point_set_min_dist(self,point_set):
        dists = []
        for i in range(len(point_set)-1):
            for j in range(i+1,len(point_set)):
                dists.append(self.distance_calc(point_set[i], point_set[j]))
        return min(dists)
    #Helper function for sample_points
    #Input:
        #numPoints: number of points to taken from a population of points within the poly
gon
        #points:  The population of points
    #output:
        #map of the min distance between any two points in a list of points to the list o
f points
    def variances(self,numPoints, points):
        point_set = self.pointSets(numPoints, points)
        point_map = {self.point_set_min_dist(points): points for points in point_set}
        return point_map
    #Helper function for sample_points
    #Input:
        #point_map: map of the min distance any two points in a list of points to the lis
t of points
    #Output:
        #List of points with highest min distance
    def highest_variance(self, point_map):
        return max(point_map.iteritems(), key=operator.itemgetter(0))[1]
    #Input:
        #numPoints: Number of points to pick from the sample
        #point_population: Number of points to sample
    #Output:
        #list of length numPoints with the highest min distance between
        #any two points in the list; we're looking for a list with high variance
    def sample_points(self,numPoints = 3, point_population = 100):
        point_array = []
        point_count = point_population
        while(point_count>0):
            #print "still running with point count: "+str(point_count)
            lat = rd.uniform(self.latMin,self.latMax)
            longi = rd.uniform(self.longMin, self.longMax)
            point_count-=1
            point = (lat,longi)
            point_array.append(point)
        #print "point array: "+str(point_array)
        point_map = self.variances(numPoints, point_array)
        return self.highest_variance(point_map)
```

In [88]:
```
polygon = LatitudalPolygon()
polygon.sample_points()
```

Out[88]:
```
[(42.78276623910992, -71.70639492893704),
 (41.275431126459466, -70.08691047708709),
 (41.386247948681536, -73.44379147880029)]
```

In [ ]:
```
polygon = LatitudalPolygon()
points = [(41.31395762873526, -71.69616264513381), (42.522305534733476, -73.0655632355322
4), (41.68187346897041, -71.22668210050055), (42.867713305593455, -72.0475987629223), (42
.060067874475884, -70.80662684493723), (42.37059208765898, -70.69643744669166), (41.95872
871223908, -70.45117660303463), (41.3002269496563, -73.05528975850659), (42.4176644636605
8, -70.07823501369909), (42.64523243052771, -70.43373209323119)]
polygon.pointSets(2, points)
```

In [34]:
```
polygon = LatitudalPolygon()
point1 = (42.06912523489147, -70.29480757892502)
point2 = (42.769805655634165, -72.72405496464135)
polygon.distance_calc(point1, point2)
```

Out[34]:
```
2.5282792395267211
```

In [1]:
```
#SELENIUM STUFF
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

#open the firefox session
profile = webdriver.FirefoxProfile()
profile.set_preference('browser.download.folderList', 2) # custom location
profile.set_preference('browser.download.manager.showWhenStarting', False)
profile.set_preference('browser.download.dir', '~/Downloads/towers')
profile.set_preference('browser.helperApps.neverAsk.saveToDisk', 'csv')
driver = webdriver.Firefox(firefox_profile=profile)
#driver = webdriver.Firefox()

#getting the page
driver.get("http://www.antennasearch.com")

#filling the form
form  = driver.find_element_by_xpath("//form[@Action='sitestart.asp']")
form1 = driver.find_elements_by_tag_name('form')
#print "form: "+str(form)
#print "form1: "+str(form1)
def fill_form(driver, address = "129 Franklin Street",city = 'Cambridge', state='MA', zip
Code='02139'):
    addr = driver.find_element_by_name('AddressIn').send_keys(address)
    town = driver.find_element_by_name('CityName').send_keys(city)
    providence = driver.find_element_by_name('StateName').send_keys(state)
    post_code = driver.find_element_by_name('ZipCodeNum').send_keys(zipCode)
    form.submit()
    return driver
```

```
#input src="/images/process.png"
def process(driver):
    xpath = '//input[@src="/images/process.png"]'
    driver.find_element_by_xpath(xpath).click()
    #print driver.find_element_by_tag_name('INPUT')

    return driver


#address = driver.find_element_by_name('AddressIn')
#address.send_keys("129 Franklin Street")
driver = fill_form(driver)

def latitudal_sampler(latitudal_bounds):
    bound1 = '42.745458, -73.265071'
    bound2 = '42.061597, -73.527701'
    bound3 = '41.247793, -70.002255'
    bound4 = '42.948057, -70.589209'

#Latitudal point set filtering

#Latitudal point to address


#address to latitudal point

#Searching AntennaSearch

#Processing AntennaSearch results

#Downloading the results

#adding the results to the database
```

In [2]:
```
driver = process(driver)
#print driver.page_source
```

In [ ]:

In [3]:
```
#<TD><CENTER><SPAN CLASS=txtlev0><A HREF=/downloads_ant_free/TowersMAF702668OAT591409.csv
>Download Records</A></CENTER></TD>

driver.find_element_by_link_text('Download Records').click()
```

In [2]:
```
#open tower files
sagamore_tower_data = pd.read_csv('SagamoreBeach.csv')
mashpee_tower_data = pd.read_csv('Mashpee.csv')
sandwich_tower_data = pd.read_csv('sandwich.csv')

lnSandwich_tower_data = pd.read_csv('lnSandwich.csv')
bourne_tower_data = pd.read_csv('bourne.csv')
buzzardsBay_tower_data = pd.read_csv('buzzardsBay.csv')

tower_data = pd.concat([sagamore_tower_data,mashpee_tower_data,sandwich_tower_data,
                    lnSandwich_tower_data,bourne_tower_data,buzzardsBay_tower_data])
#tower_data
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-c534acc126a8> in <module>()
      1 #open tower files
----> 2 sagamore_tower_data = pd.read_csv('SagamoreBeach.csv')
      3 mashpee_tower_data = pd.read_csv('Mashpee.csv')
      4 sandwich_tower_data = pd.read_csv('sandwich.csv')
      5 

NameError: name 'pd' is not defined
```

In [4]:
```
#columns names
col_names = tower_data.columns.get_values()
#Get the columns starting with rep or owner

regex_keys = ['own', 'rep']

#select columns of the data frame starting with given prefixes
def select_columns(prefix_list, columns):
    return [item for item in col_names
            if any(item.startswith(prefex) for prefex in prefix_list)]
```

In [5]:
```
#tests for select_columns
owner_rep = select_columns(regex_keys, col_names)
owner = select_columns(['owner'], col_names)
rep = select_columns(['rep'], col_names)
```

In [6]:
```
tower_data.drop_duplicates()
```

Out[6]:

	tower_type	faa_study_number	registration_number	latitude	longitude	status_code	date_constructed	structure_street_a
0	Registered	2010-ANE-51-OE	1273684.0	42.353917	-71.105000	Constructed	06/28/2010	575 Memorial Drive 4B New England
1	Non-Registered	94-ANE-453-OE	NaN	42.360917	-71.098083	NaN	NaN	NaN
2	Non-Registered	94-ANE-354-OE	NaN	42.365361	-71.092250	NaN	NaN	NaN
3	Non-Registered	2016-ANE-4034-OE	NaN	42.364556	-71.090722	NaN	NaN	NaN
4	Non-Registered	2012-ANE-929-OE	NaN	42.359361	-71.086528	NaN	NaN	NaN
5	Non-Registered	94-ANE-447-OE	NaN	42.347306	-71.097806	NaN	NaN	NaN
0	Registered	2015-ANE-334-OE	1019302.0	41.603083	-70.492861	Granted	01/01/1980	22 INDUSTRIAL WAY

1	Registered	2011-ANE-1775-OE	1216584.0	41.581778	-70.539167	Constructed	12/14/2011	996 E Falmouth Hwy 0
2	Registered	2014-ANE-626-OE	1296443.0	41.594472	-70.566333	Constructed	05/23/2016	off Old Meeting House
3	Registered	2004-ANE-661-OE	1234487.0	41.656694	-70.497500	Constructed	05/12/2004	54 Echo Road
4	Registered	2008-ANE-981-OE	1268285.0	41.671083	-70.516417	Constructed	06/29/2010	23 Falmouth Sandwich
5	Registered	2004-ANE-1195-OE	1059742.0	41.603778	-70.488722	Constructed	04/30/1998	LOT 38 INDUSTRIAL D MASHPEE INDUSTRIA
6	Registered	2006-ANE-1512-OE	1261973.0	41.615361	-70.538750	Constructed	11/21/2008	0 Nathan Ellis Highway
7	Registered	99-ANE-0706-OE	1209840.0	41.652028	-70.520583	Granted	NaN	131 Otis Air Force Bas
8	Non-Registered	2010-ANE-793-OE	NaN	41.650000	-70.535833	NaN	NaN	NaN
9	Non-Registered	2009-WTE-4440-OE	NaN	41.617500	-70.531361	NaN	NaN	NaN
10	Non-Registered	2017-ANE-964-OE	NaN	41.651389	-70.538944	NaN	NaN	NaN
11	Non-Registered	2009-ANE-490-NRA	NaN	41.648611	-70.541389	NaN	NaN	NaN
12	Non-Registered	2017-ANE-957-OE	NaN	41.657083	-70.530833	NaN	NaN	NaN
13	Non-Registered	2011-ANE-343-NRA	NaN	41.654139	-70.508444	NaN	NaN	NaN
14	Non-Registered	2009-ANE-278-NRA	NaN	41.660833	-70.522139	NaN	NaN	NaN
15	Non-Registered	2016-ANE-3914-OE	NaN	41.653611	-70.541111	NaN	NaN	NaN
16	Non-Registered	2016-ANE-540-OE	NaN	41.662417	-70.522361	NaN	NaN	NaN
17	Non-Registered	2011-ANE-404-NRA	NaN	41.660000	-70.535278	NaN	NaN	NaN
18	Non-Registered	2015-ANE-1090-OE	NaN	41.659917	-70.535833	NaN	NaN	NaN
19	Non-Registered	2005-ANE-798-OE	NaN	41.657056	-70.505639	NaN	NaN	NaN
20	Non-Registered	2014-ANE-1904-OE	NaN	41.658278	-70.541250	NaN	NaN	NaN
21	Non-Registered	2009-ANE-438-NRA	NaN	41.664972	-70.519472	NaN	NaN	NaN
22	Non-Registered	2016-ANE-1272-NRA	NaN	41.665278	-70.521667	NaN	NaN	NaN
23	Non-Registered	2011-ANE-340-NRA	NaN	41.662556	-70.535694	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
16	Non-Registered	2014-ANE-648-OE	NaN	41.744306	-70.456361	NaN	NaN	NaN
22	Non-Registered	99-ANE-307-OE	NaN	41.712833	-70.496167	NaN	NaN	NaN
25	Non-Registered	2006-ANE-1344-OE	NaN	41.708611	-70.427222	NaN	NaN	NaN
29	Non-Registered	2010-ANE-1-OE	NaN	41.715944	-70.498944	NaN	NaN	NaN
30	Non-Registered	2006-ANE-106-OE	NaN	41.717917	-70.503250	NaN	NaN	NaN
31	Non-Registered	2011-ANE-360-OE	NaN	41.730278	-70.490972	NaN	NaN	NaN
11	Non-Registered	2006-ANE-764-OE	NaN	41.649667	-70.424361	NaN	NaN	NaN

12	Non-Registered	2011-ANE-1657-OE	NaN	41.629056	-70.441639	NaN	NaN	NaN
18	Non-Registered	2015-ANE-416-NRA	NaN	41.650500	-70.517306	NaN	NaN	NaN
32	Non-Registered	98-ANE-208-OE	NaN	41.713139	-70.423056	NaN	NaN	NaN
34	Non-Registered	00-ANE-0142-OE	NaN	41.654806	-70.404444	NaN	NaN	NaN
36	Non-Registered	98-ANE-0494-OE	NaN	41.713417	-70.422500	NaN	NaN	NaN
0	Registered	97-ANE-0227-OE	1013714.0	41.678333	-70.598056	Constructed	01/01/1960	OFF OLD COUNTY R
1	Registered	2009-ANE-239-OE	1004089.0	41.700861	-70.588083	Constructed	02/10/2009	50 PORTSIDE DR 010
2	Registered	2014-ANE-670-OE	1292176.0	41.741806	-70.582750	Constructed	01/01/1998	Macarthur Boulevard
3	Registered	01-ANE-0704-OE	1230487.0	41.744056	-70.587778	Granted	12/07/2001	Bourne
6	Non-Registered	2010-WTE-4899-OE	NaN	41.754222	-70.595250	NaN	NaN	NaN
7	Non-Registered	2009-ANE-114-OE	NaN	41.741333	-70.623833	NaN	NaN	NaN
8	Non-Registered	97-ANE-0581-OE	NaN	41.761472	-70.575861	NaN	NaN	NaN
9	Non-Registered	2004-ANE-788-OE	NaN	41.665444	-70.604611	NaN	NaN	NaN
15	Non-Registered	2009-ANE-651-OE	NaN	41.769111	-70.564361	NaN	NaN	NaN
16	Non-Registered	2014-ANE-1946-OE	NaN	41.770167	-70.567639	NaN	NaN	NaN
17	Non-Registered	00-ANE-0469-OE	NaN	41.754444	-70.539722	NaN	NaN	NaN
18	Non-Registered	94-ANE-329-OE	NaN	41.772861	-70.569750	NaN	NaN	NaN
19	Non-Registered	96-ANE-215-OE	NaN	41.773972	-70.570306	NaN	NaN	NaN
20	Non-Registered	2003-ANE-378-OE	NaN	41.697028	-70.587250	NaN	NaN	NaN
21	Non-Registered	2015-ANE-1092-OE	NaN	41.707722	-70.564417	NaN	NaN	NaN
22	Non-Registered	92-ANE-347-OE	NaN	41.683139	-70.593083	NaN	NaN	NaN
8	Non-Registered	2006-ANE-441-OE	NaN	41.783167	-70.601778	NaN	NaN	NaN
10	Non-Registered	2014-ANE-486-OE	NaN	41.791944	-70.603111	NaN	NaN	NaN

```
104 rows × 51 columns
```

```
In [7]: #select tower owner names
        owner_names=tower_data[['owner_entity_name']].drop_duplicates()
        #owner_names
```

In [8]:
```
#Validation fields
exceptions = ['COLONIAL GAS COMPANY']
probably_cell_towers = ['T-Mobile Northeast LLC','']
wireless_or_towers = ['American Towers LLC']

lte = ['Adams Networks','AT&T',('Big River', 'Broad'),
        'BIT Broad',('Bluegrass','Cellular'),('Bug Tussel','Wireless'),'C Spire',
        'ClearTalk','Colorado Valley','ETC',('Evolve', 'Broadband'),
        ('Infrastructure','Networks'),('iWireless'),('Limitless','Mobile'),
       'Mobile Nation',('Mosaic','Telecom'),('Nex-Tech','Wireless'),
        'Nortex','PTCI','PTC',('Redzone','Wireless'),'Rise Broadband',
        'Rock Wireless','Silver Star',('Space Data','Corporation'),
        'Speed Connect','Sprint',('Syringa','Wireless'),'T-Mobile',
        'U.S. Cellular','US Cellular','United Wireless','Verizon',
       ('VTel','Wireless'),'United States Cellular']
```

In [9]:
```
#rtf file processing utility functions

#read from the file
def read_rtf(rtf_file):
    _file = open(rtf_file, 'r')
    rtf_data = _file.read().strip()
    _file.close()
    return rtf_data
import re

#pick the links in the file
def rtf_links(rtf_data):
    return re.findall('\s+(.*?)}}', rtf_data)
#split and clean strings
def split_clean_str(string, split, index):
    return string.split(split)[index].strip()
#clean all the links in the rtf file
def link_text(rtf_links):
    return [split_clean_str(item, '3', 1) for item in rtf_links if not 'http' in item]
#Get the text from all the links in the rtf file
def get_rtf_linkTexts(rtf_file):
    twr_comps = read_rtf(rtf_file)
    twr_comp_links = rtf_links(twr_comps)
    return link_text(twr_comp_links)

link_texts = get_rtf_linkTexts('tower_companies.rtf')#link_text(twr_comp_links)
```

In [10]:
```
tower_companies = pd.DataFrame(link_texts, columns=['tower companies'])
tower_companies
```

Out[10]:

	tower companies
0	Crown Castle
1	American Tower
2	SBA Communications
3	United States Cellular Co.
4	Vertical Bridge

5	Insite Towers
6	BNSF Railroad
7	Time Warner
8	Diamond Communications
9	Phoenix Tower International
10	Tower Ventures
11	Union Pacific Railroad Company
12	C Spire Wireless
13	Industrial Communications
14	Grain Management
15	Comcast
16	Mediacom Communications
17	T-Mobile Towers
18	Lightower
19	Cumulus Broadcasting
20	Subcarrier Communications
21	Century Link
22	Charter Communications
23	Central States Tower
24	SkyWay Towers
25	Townsquare Media
26	General Communication, Inc.
27	Sprint Sites USA
28	Horvath Communications
29	Southern Company
...	...
93	MidAmerica Towers
94	Bay Communications
95	Com Site West
96	1 Source Towers
97	Great Plains Tower Properties
98	Towers of Texas
99	Bluegrass Wireless
100	Tower Sites
101	Prime Tower
102	Allcomm Wireless
103	Community Wireless Structures
104	ERS Tower Services
105	Shared Towers
106	Louis Dreyfus Pipeline
107	Centre Communications
108	Minnesota Towers, Inc.
109	JNS Tower
110	Gulf States Towers
111	Blue Ridge Telecom
112	Morris Communications Corp.
113	Datapath Tower
114	Highpoint Tower Technology
115	Clear Signal Towers

116	Dragon Communications
117	NexTower, LLC
118	Communication & Control, Inc.
119	Hemphill Corporation
120	Horizon Tower
121	ClearShot Communications
122	Arcadia Towers

```
123 rows × 1 columns
```

```
In [11]: tower_companies
```

Out[11]:

	tower companies
0	Crown Castle
1	American Tower
2	SBA Communications
3	United States Cellular Co.
4	Vertical Bridge
5	Insite Towers
6	BNSF Railroad
7	Time Warner
8	Diamond Communications
9	Phoenix Tower International
10	Tower Ventures
11	Union Pacific Railroad Company
12	C Spire Wireless
13	Industrial Communications
14	Grain Management
15	Comcast
16	Mediacom Communications
17	T-Mobile Towers
18	Lightower
19	Cumulus Broadcasting
20	Subcarrier Communications
21	Century Link
22	Charter Communications
23	Central States Tower
24	SkyWay Towers
25	Townsquare Media
26	General Communication, Inc.
27	Sprint Sites USA
28	Horvath Communications
29	Southern Company
...	...
93	MidAmerica Towers
94	Bay Communications
95	Com Site West
96	1 Source Towers
97	Great Plains Tower Properties
98	Towers of Texas

99	Bluegrass Wireless
100	Tower Sites
101	Prime Tower
102	Allcomm Wireless
103	Community Wireless Structures
104	ERS Tower Services
105	Shared Towers
106	Louis Dreyfus Pipeline
107	Centre Communications
108	Minnesota Towers, Inc.
109	JNS Tower
110	Gulf States Towers
111	Blue Ridge Telecom
112	Morris Communications Corp.
113	Datapath Tower
114	Highpoint Tower Technology
115	Clear Signal Towers
116	Dragon Communications
117	NexTower, LLC
118	Communication & Control, Inc.
119	Hemphill Corporation
120	Horizon Tower
121	ClearShot Communications
122	Arcadia Towers

```
123 rows × 1 columns
```

```
In [1]:
#Input:
    #name1: Name of a company
    #name2: Name of a company
    #lte_name: Name of a known LTE provider
#output:
    #Checks if name of the lte company is in name1 and name2
    #calculates score based on the how much of the lte provider's name
    #is in both name1 and name2
def commonGrade(name1,name2, lte_name):
    name1 = str(name1).lower()
    name2 = str(name2).lower()

    whole = 30.0
    first = 18.0
    dist = 0
    if any([isinstance(nam,float) for nam in [name1, name2, lte_name]]):
            return dist
    if isinstance(lte_name, str):
        lte_name = lte_name.lower()


        if lte_name in name1 and lte_name in name2:
            return whole;
        return dist
    if isinstance(lte_name, tuple):
        main = lte_name[0].lower()
        extension = lte_name[1].lower()
        if main in name1 and main in name2:
            dist+=first
            if extension in name1 and extension in name2:
                dist+=whole-first
        return dist
    raise ValueError("Found something lte name that's neither string nor tuple!")

def both_LTE(name1,name2):
    #print [commonGrade(name1,name2, lte_name) for lte_name in lte]
    return max([commonGrade(name1,name2, lte_name) for lte_name in lte])
'''
Calculates the distance between the names of two companies based on the edit distance and
the number of similar works present
'''
import editdistance as editd
def distance(name1, name2):
    #print "name 1: "+name1
    #print "name 2: "+name2
    _nm1 = str(name1).lower().split(',')[0]#.split(' ')
    _nm2 = str(name2).lower().split(',')[0]#.split(' ')
    nm1 = _nm1.split(' ')
    nm2 = _nm2.split(' ')

    bound = min(len(nm1),len(nm2))
    dist = 20.0
    both_lte = both_LTE(name1, name2)
    for i in range(bound):
        if nm1[i]==nm2[i]:
            dist+=dist/((i+1)**2)
        else:
            dist-=dist/((i+1)**2)
    return dist - editd.eval(_nm1, _nm2)+both_lte
```

In [13]:

```
#distance('Communication & Control, Inc.','ClearShot Communications')
#lte_name = ('C Spire','Corporation')
name1 = 'C Spire Corporation'
name2 = 'C Spire'
both_LTE(name1,name2)
```

Out[13]:

```
30.0
```

In [14]:
```
#distances array between a names and list of names
def one_to_many(one, many):
    return [(one,item, distance(one, item)) for item in many]
'''
input:
    owners: Owner names of found towers
    towerComp: Known tower companies
output:
    data frame of (owner, tower companies, names distance)
'''
def owner_to_company_name_dist(owners, towerComp):
    distances = [one_to_many(name[0], towerComp) for name in owners]
    comparision_frame = pd.concat([pd.DataFrame(item, columns = ('tower owner', 'tower co
mpany','name distance'))
                                  for item in distances])
    return comparision_frame
comparision_frame = owner_to_company_name_dist(owner_names.get_values(), tower_companies)
```

In [15]:
```
comparision_frame.to_csv(path_or_buf = 'tower_owner_tower_comp_dist.csv')
```

In [16]:
```
owner_names.get_values()
```

Out[16]:
```
array([['T-Mobile Northeast LLC'],
       ['DISPATCH COMMUNICATIONS OF NEW ENGLAND I'],
       ['NYNEX MOBILE COMMUNICATIONS CO'],
       ['DRAPER LABS'],
       ['CELLCO PARTNERSHIP D'],
       ['NYNEX MOBILE COMMS CO'],
       ['Comcast of  Massachusetts I Inc'],
       ['American Towers LLC'],
       ['Industrial Tower and Wireless LLC'],
       ['Industrial Communications  Electronics Inc'],
       ['Global Tower LLC through American Towers LLC'],
       ['SBA Properties LLC'],
       ['WinStar Wireless Fiber  Corp'],
       ['JBECC'],
       ['ALTERIS RENEWABLES INC'],
       ['CHRISTOPHER ALLEN'],
       ['ENDICOTT CONSTRUCTION CORP'],
       ['12 RADAR SQUADRON'],
       ['US COAST GUARD'],
       ['VOLPE NATIONAL TRANSPORTATION SYSTEMS CENTERUS DOT'],
       ['FAA - BOSTON TERMINAL CONSTRUCTION CENTER'],
       ['UNICON INC'],
       ['AVWATCH INC'],
       ['RAYTHEON'],
       ['CUTTER ENTERPRISES'],
       ['F CO 3126TH AVN'],
       ['SEACOAST LP'],
       ['JODY BUILDERS'],
       ['US COAST GUARD C3CEN'],
       ['EASTERN TOWERS INC'],
       ['VANGUARD CONST CO INC'],
       ['LIBERTY CORNER ASSOCIATES INC'],
       ['NEXTEL COMMUNICATIONS OF THE MID-ATLANTIC INC'],
       ['THE NUTMEG COMPANIES INC'],
       [nan],
       ['FALMOUTH AIRPARK'],
       ['BARNSTABLE COUNTY SHERIFFS OFFICE'],
       ['NYNEX MOBILE COMMUNICATIONS COMPANY'],
       ['SPRINT SPECTRUM LP'],
       ['ECO-SITE INC IT'],
       ['JAMES BROWN'],
       ['ENDICOTT CONSTRUCTORS'],
       ['AJW-E11A'],
       ['MAJOR S DEMIANCZYK'],
       ['Seacoast LP'],
       ['Mirant Canal LLC'],
       ['ATT Mobility Spectrum LLC'],
       ['VICTORY SAND  GRAVEL COMPANY'],
       ['COMMANDER FIRST COAST GUARD DISTRICT'],
       ['POLICE DEPARTMENT'],
       ['US COAST GUARD MLCLANT TP-2'],
       ['BLANK'],
       ['MR W BURKE'],
       ['SOUTHWESTERN BELL MOBILE SYSTEMS'],
       ['AMERICAN TOWERS LLC'],
       ['OMNIPOINT COMMUNICATIONS INC NEW ENGLAND'],
       ['AMERICAN TOWER CORPORATION'],
       ['CINGULAR WIRELESS-DALLAS'],
       ['COTUIT WATER DISTRICT'],
       ['SEACOAST INC'],
       ['INDUSTRIAL COMM  ELECTRONICS INC'],
       ['COLONIAL GAS COMPANY'],
       ['IWG Towers Assets I LLC'],
       ['Commonwealth of Massachusetts Department of State Police'],
       ['AERONAUTICA WINDPOWER LLC'],
       ['MASSACHUSETTS MARITIME ACADEMY'],
       ['ATT WIRELESS SERVICES'],
       ['AMERICAN TOWER - BOSTON'],
       ['NSTAR ELECTRIC AND GAS'],
       ['NSTAR'],
       ['CIVIL AIR PATROL MASS WING'],
       ['INDUSTRIAL COMMUNICATIONS  ELECTRONICS'],
       ['AMERICAN TOWER'],
       ['OMNIPOINT COMMUNICATIONS INC'],
       ['VARSITY WIRELESS LLC']], dtype=object)
```

In [17]:

```python
#input:
    #html_file: html file
#output:
    #Tables: list of raw tables extracted from the file
import re
def wekipediaScraper(html_file):
    open_file = open(html_file, 'r')
    file_data = open_file.read()
    open_file.close()
    opening_table = '<table class=[.*?] border=[.*?] cellpadding=[.*?] style=[.*?]>'
    tables= re.findall('(?s)(?<=<table)(.+?)(?=</table>)', file_data)
    #print file_data
    return tables
```

In [18]:

```python
#input:
    #tables: list of raw html table contents
#output:
    #matrix of html table row raw contents
def tableRow(tables):
    return [re.findall('(?s)(?<=<tr)(.+?)(?=</tr>)', table) for table in tables]
```

In [19]:

```python
tables = wekipediaScraper('weki.html')
#tables
#tableRow(tables)
```

In [20]:
```
#input:
    #tr: raw html row content
#output:
    #list of clean text extracted each column of the row
def separateCols(tr):
    cols = re.findall('(?s)(?<=<td)(.+?)(?=/td>)|(?s)(?<=<th)(.+?)(?=/th>)', tr)
    clean_cols = [re.findall('(?s)(?<=>)(.+?)(?=<)',max(col, key = len)) for col in cols]
    cleaned_cols = [max(col,key=len).split('>')[-1] if col else 'NaN' for col in clean_co
ls]


    return cleaned_cols
import pandas as pd
#input:
    #table: raw html table content
#output:
    #Data frame of the html table cell text
def table_to_dataframe(table):
    matrix = [separateCols(tr) for tr in table]
    return pd.DataFrame(matrix[1:], columns=matrix[0])
#input:
    #html_file: html file
    #index: index location of a table in the file
#output:
    #dataframe of the cell content of the table with given index
def html_table_to_dataframe(html_file, index):
    tables = wekipediaScraper(html_file)
    row_separated = tableRow(tables)
    return table_to_dataframe(table[index])
```

In [21]:
```
separateCols('<tr><td><a href="/wiki/Big_River_Telephone" title="Big River Telephone">Big
River Broadband</a></td><td></td><td><a href="/wiki/3GPP_Long_Term_Evolution" class="mw-r
edirect" title="3GPP Long Term Evolution">LTE</a></td><td>Unknown</td><td>MO</td><td><a h
ref="/wiki/Big_River_Telephone" title="Big River Telephone">Big River Telephone</a></td><
/tr>')
```

Out[21]:
```
['Big River Broadband', 'NaN', 'LTE', 'Unknown', 'MO', 'Big River Telephone']
```

In [22]:
```
head = '<tr style="background:skyblue;"><th>Operator</th><th>Voice&#160;technology</th><t
h>Data&#160;technology</th><th data-sort-type="number">Subscribers<br /><small>(in millio
ns)</small></th><th>Coverage<br /><small>(excluding&#160;roaming)</small></th><th>Ownersh
ip</th></tr>'
separateCols(head)
```

Out[22]:
```
['Operator',
 'Voice&#160;technology',
 'Data&#160;technology',
 '(in millions)',
 '(excluding&#160;roaming)',
 'Ownership']
```

In [23]:
```
#tables = wekipediaScraper('weki.html')
#tables
table = tableRow(tables)
nation_wide = table_to_dataframe(table[1])
nation_wide
```

Out[23]:

	Operator	Voice&#160;technology	Data&#160;technology	(in millions)	(excluding&#160;roaming)	Ownership
0	Appalachian Wireless	CDMA2000	EV-DO	[16]	KY, WV	East Kentucky Network, LLC
1	AT&amp;T Mobility	UMTS	HSPA+	[17]	Contiguous US; AK, HI, PR, VI	AT&amp;T
2	Big River Broadband	NaN	LTE	Unknown	MO	Big River Telephone
3	Big Sky Mobile	GSM	EDGE	Unknown	MT	iSmart Mobile, LLC
4	Blaze Wireless	GSM	EDGE	Unknown	NE	Pinpoint Communications
5	Blue Wireless	CDMA2000	EV-DO	Unknown	NY, PA	Buffalo-Lake Erie Wireless
6	Bluegrass Cellular	CDMA2000	EV-DO	[18]	KY	Bluegrass Cellular, Inc.
7	Bravado Wireless	CDMA2000	EV-DO	Unknown	OK	Cross Communications
8	Breakaway Wireless	CDMA2000	EV-DO	Unknown	UT	Manti Tele Communications
9	Broadpoint	GSM	EDGE	Unknown	Gulf of Mexico	Alta Communications
10	Bug Tussel Wireless	NaN	HSPA+	Unknown	WI	Bug Tussel Wireless, LLC
11	C Spire Wireless	CDMA2000	EV-DO	[19]	MS, TN, FL, AL,	Telapex, Inc.
12	Carolina West Wireless	CDMA2000	EV-DO	[20]	NC	Carolina West Wireless, Inc.
13	Cellcom	CDMA2000	EV-DO	[21]	WI	NSight Telservices
14	Cellular One of East Arizona	GSM	EDGE	[16]	AZ, NM	Smith Bagley Inc.
15	Cellular One		EDGE	Unknown	TX, LA	Alta Communications
16	Chariton Valley Wireless	CDMA2000	EV-DO	Unknown	MO	Chariton Valley Telephone Company
17	Chat Mobility	CDMA2000	EV-DO	Unknown	IA	RSA 1 and Iowa RSA No. 2 Limited Partnership
18	Choice Wireless	CDMA2000	EV-DO	Unknown	NV, CO, AZ, NM, VI	Atlantic Tele-Network
19	ClearTalk Wireless	CDMA2000	EV-DO	Unknown	AZ, CA, NM, TX	Flat Wireless
20	Colorado Valley Communications	NaN	LTE	Unknown	TX	Colorado Valley Telephone Coop
21	Commnet Wireless (wholesale)	CDMA2000	EV-DO	Unknown	AZ, CO, MT, NM, NV, TX, UT, WY	Atlantic Tele-Network
22	CTC Wireless	CDMA2000	EV-DO	Unknown	ID	Cambridge Telephone Company
23	Custer Telephone Cooperative	CDMA2000	EV-DO	Unknown	ID	Custer Telephone Cooperative, Inc
24	DTC Wireless	GSM	EDGE	Unknown	TN	Advantage Cellular Systems, Inc.
25	ETC	CDMA2000	EV-DO	Unknown	IN	Miles Communications
26	Evolve Broadband	NaN	LTE	Unknown	TX	Worldcall Interconnect Inc.
27	Farmers Mutual Telephone Company	CDMA2000	EV-DO	Unknown	ID	Farmers Mutual Telephone Company
28	FTC Wireless	UMTS	HSPA+	Unknown	SC	Farmers Telephone Cooperative Inc.

29	iWireless	GSM	EDGE	Unknown	IA, WI, IL	Iowa Wireless Services
...	...	...	...	...	...	...
39	NMobile	CDMA2000	EV-DO	Unknown	NM	Leaco Wireless, Inc.
40	NNTC Wireless	CDMA2000	EV-DO	Unknown	CO	Nucla Naturita Telephone Company
41	NorthwestCell	CDMA2000	EV-DO	Unknown	MO	NorthwestCell
42	NVC	CDMA2000	EV-DO	Unknown	SD	James Valley Telecommunications
43	Phoenix Communications	GSM	EDGE	Unknown	OK	Oklahoma Western Telephone Company
44	Pine Belt Wireless	CDMA2000	EV-DO	Unknown	AL	Pine Belt Communications
45	Pine Cellular	GSM	EDGE	Unknown	OK	Pine Cellular, Inc
46	Pioneer Cellular	CDMA2000	EV-DO	[16]	OK, KS	Pioneer Cellular
47	PTCI	GSM	EDGE	Unknown	OK	Panhandle Telecommunication Systems, Inc
48	Redzone Wireless	NaN	LTE	Unknown	ME	Redzone Wireless, LLC
49	Rock Wireless	UMTS	HSPA+	Unknown	ND, SD	Standing Rock Telecom (Tribally Owned)
50	S and R Communications	VoLTE	LTE	Unknown	IN	S and R Communications, LLC
51	Shentel	CDMA2000	EV-DO	[29]	KY, MD, OH, NC, PA, VA, WV	Shenandoah Telecommunications Company
52	Silver Star Communications	CDMA2000	EV-DO	Unknown	WY	Silver Star Telephone Company, Inc
53	Snake River PCS	CDMA2000	EV-DO	Unknown	OR	Eagle Telephone System
54	SouthernLINC	iDEN	LTE	[30]	AL, GA, MS, FL	Southern Company
55	Sprint Corporation	CDMA2000	EV-DO	[31]	Contiguous US; HI, PR, VI	SoftBank Corporation
56	SRT Communications	CDMA2000	EV-DO	Unknown	ND	North Dakota Network Co.
57	STRATA Networks	CDMA2000	EV-DO	Unknown	UT, WY, CO	Uintah Basin Electronics Telecommunications, Inc.
58	T-Mobile US	Wi-Fi calling	EDGE	[3]	Contiguous US; HI, PR, VI	Deutsche Telekom
59	Thumb Cellular	CDMA2000	EV-DO	0.037	MI	Agri-Valley Communications
60	Triangle Mobile	CDMA2000	EV-DO	Unknown	MT	Triangle Telephone Cooperative Association, Inc.
61	Union Wireless	GSM	EDGE	0.041	WY, CO	Union Telephone
62	United Wireless	CDMA2000	EV-DO	[33]	KS	United Wireless Communications, Inc.
63	U.S. Cellular	CDMA2000	EV-DO	[34]	23 states	Telephone and Data Systems
64	Verizon Wireless	CDMA2000	EV-DO	[2]	Contiguous US; AK, HI	Verizon
65	Viaero Wireless	GSM	EDGE	[35]	NE, CO, KS	Northeast Colorado Cellular, Inc.
66	VTel	NaN	LTE	Unknown	VT	Vermont Telephone Company, Inc.
67	West Central Wireless	GSM	EDGE	[36]	TX	Central Texas Telephone

						Cooperative, Inc.
68	WUE	CDMA2000	EV-DO	Unknown	NV	Wue, Inc.

```
69 rows × 6 columns
```

In [24]:
```
merges = table_to_dataframe(table[-3])
merges
```

Out[24]:

	Operator	Voice&#160;technology	Data&#160;technology	(in millions)	End&#160;date	Notes
0	Airfire Mobile	GSM	EDGE	0.02	000000002014-06-01-0000	[46]
1	Alaska Communications		EDGE	[47]	000000002015-02-01-0000	
2	Alltel	CDMA	CDMA2000	13 (at peak)	000000002013-09-01-0000	[49]
3	Cellular One of East Central Illinois	GSM	EDGE	Unknown	000000002016-01-01-0000	[50]
4	Cellular One of Northeast Pennsylvania	CDMA	CDMA2000	Unknown	000000002012-03-01-0000	[51]
5	Choice Wireless	GSM	EDGE	Unknown	000000002016-01-01-0000	T-Mobile US
6	Cincinnati Bell Wireless		EDGE	[53]	000000002015-02-01-0000	, service to any customers remaining on the ne...
7	affiliate of Clear Talk Wireless	CDMA2000	EV-DO	Unknown	000000002016-01-01-0000	WGH Communications sold South Carolina license...
8	Clearwire	NaN	WiMAX	[55]	000000002013-12-01-0000	[56]
9	Corr Wireless	GSM	EDGE	[57]	000000002013-01-01-0000	[57]
10	Element Mobile	CDMA2000	EV-DO	[58]	000000002013-01-01-0000	
11	Epic PCS	GSM	EDGE	Unknown	000000002015-01-01-0000	Acquired by United Wireless and PTCI.
12	Fuego Wireless	NaN	LTE	Unknown	000000002016-01-01-0000	Sold network and spectrum licenses to AT&amp;T...
13	Golden State Cellular	CDMA2000	EV-DO	[63]	000000002014-01-01-0000	[64]
14	KTC PACE	GSM	EDGE	Unknown	000000002015-01-01-0000	[65]
15	Leap Wireless International	CDMA	EDGE	[66]	000000002014-01-01-0000	[66]
16	Long Lines Wireless	GSM	EDGE	[67]	000000002013-12-01-0000	
17	MetroPCS Communications, Inc.	CDMA	EV-DO	9.5	000000002013-05-01-0000	[69]
18	miSpot	NaN	LTE	Unknown	000000002014-11-01-0000	NaN
19	Mobi PCS	CDMA2000	EV-DO	[70]	000000002014-01-01-0000	[64]
20	Mosaic Telecom	UMTS	HSPA+	Unknown	000000002016-01-01-0000	Discontinued cellular service
21	NEP Wireless	GSM	EDGE	[72]	000000002015-09-01-0000	
22	Nextel	iDEN	WiDEN	15 (Apr 2004)	000000002004-01-01-0000	[74]
23	nTelos	CDMA2000	EV-DO	0.300 (September 2015)	000000002016-05-01-0000	[75]
24	Peoples Wireless	CDMA2000	EV-DO	Unknown	000000002015-01-01-0000	

25	Plateau Wireless	GSM	EDGE	[77]	000000002014-01-01-0000	[78]
26	Pocket Communications	CDMA2000	1xRTT	Unknown	000000002010-10-01-0000	[79]
27	PrimeCo Wireless	CDMA	Unknown	Unknown	000000001999-01-01-0000	[80]
28	Revol Wireless	CDMA	CDMA2000	Unknown	000000002014-01-01-0000	Sold spectrum licenses to Sprint.
29	Standing Rock Telecom	CDMA2000	EV-DO	Unknown	000000002014-11-01-0000	Shut down CDMA for UMTS/LTE and rebranded to R...
30	Stelera Wireless	NaN	HSPA	Unknown	000000002013-04-01-0000	[83]
31	Syringa Wireless	CDMA2000	EV-DO	0.00135	000000002015-12-01-0000	Discontinued cellular service
32	TerreStar	GSM	EDGE	Unknown	000000002010-01-01-0000	
33	Westlink Wireless	GSM	EDGE	Unknown	000000002013-01-01-0000	Acquired by United Wireless.

In [25]:
```
operators = nation_wide['Operator'].append(merges['Operator'])
```

In [26]:
```
tel_companies = operators.append(nation_wide['Ownership'])
```

In [27]:
```
all_companies = tel_companies.append(tower_companies['tower companies']).drop_duplicates(
)
#original_shape = (295,)
```

In [28]:
```
comparison_df = owner_to_company_name_dist(owner_names.get_values(), all_companies)
comparison_df.sort(['name distance'])[21200:]
```

```
/Users/abdisamad/anaconda2/lib/python2.7/site-packages/ipykernel/__main__.py:2: FutureWarning: sort(columns=....) is deprecated, use
sort_values(by=.....)
  from ipykernel import kernelapp as app
```

Out[28]:

	tower owner	tower company	name distance
159	RAYTHEON	Verizon	-6.000000
159	NaN	Verizon	-6.000000
262	AMERICAN TOWER	Prime Tower	-6.000000
178	BLANK	Comcast	-6.000000
195	AJW-E11A	TowerCo	-6.000000
86	NaN	miSpot	-6.000000
204	NSTAR	Entercom	-6.000000
90	RAYTHEON	Nextel	-6.000000
113	NSTAR	Telapex, Inc.	-6.000000
91	RAYTHEON	nTelos	-6.000000
193	NaN	Entergy	-6.000000
163	AJW-E11A	Wue, Inc.	-6.000000
218	AMERICAN TOWER	Heartland Tower	-6.000000
71	NaN	Alltel	-6.000000
39	NaN	NMobile	-6.000000
135	NSTAR	Nex-Tech	-6.000000
236	AMERICAN TOWER	Atlas Tower	-6.000000
204	JBECC	Entercom	-6.000000
178	NaN	Comcast	-6.000000
68	AJW-E11A	WUE	-6.000000

51	NaN	Shentel	-6.000000
71	AJW-E11A	Alltel	-6.000000
66	RAYTHEON	VTel	-6.000000
178	JBECC	Comcast	-6.000000
113	NaN	Telapex, Inc.	-6.000000
39	NSTAR	NMobile	-6.000000
193	JBECC	Entergy	-6.000000
71	RAYTHEON	Alltel	-6.000000
51	BLANK	Shentel	-6.000000
100	NSTAR	TerreStar	-5.000000
...	...	...	...
163	NaN	Wue, Inc.	-3.000000
25	NaN	ETC	-3.000000
42	NaN	NVC	-2.000000
90	NEXTEL COMMUNICATIONS OF THE MID-ATLANTIC INC	Nextel	1.000000
138	JAMES BROWN	James Valley Telecommunications	7.000000
224	American Towers LLC	American Family Association	10.666667
224	AMERICAN TOWERS LLC	American Family Association	10.666667
176	Industrial Tower and Wireless LLC	Industrial Communications	11.000000
243	AMERICAN TOWER CORPORATION	American Electric Power	12.666667
224	AMERICAN TOWER	American Family Association	13.000000
224	AMERICAN TOWER - BOSTON	American Family Association	13.666667
243	American Towers LLC	American Electric Power	14.666667
243	AMERICAN TOWERS LLC	American Electric Power	14.666667
243	AMERICAN TOWER - BOSTON	American Electric Power	14.666667
224	AMERICAN TOWER CORPORATION	American Family Association	14.666667
178	Comcast of Massachusetts I Inc	Comcast	16.000000
166	SBA Properties LLC	SBA Communications	17.000000
176	INDUSTRIAL COMM ELECTRONICS INC	Industrial Communications	17.000000
243	AMERICAN TOWER	American Electric Power	21.000000
165	AMERICAN TOWERS LLC	American Tower	25.000000
165	American Towers LLC	American Tower	25.000000
176	Industrial Communications Electronics Inc	Industrial Communications	33.000000
176	INDUSTRIAL COMMUNICATIONS ELECTRONICS	Industrial Communications	37.000000
165	AMERICAN TOWER CORPORATION	American Tower	38.000000
165	AMERICAN TOWER - BOSTON	American Tower	41.000000
190	SPRINT SPECTRUM LP	Sprint Sites USA	47.666667
58	T-Mobile Northeast LLC	T-Mobile US	48.000000
180	T-Mobile Northeast LLC	T-Mobile Towers	50.000000
165	AMERICAN TOWER	American Tower	50.000000
55	SPRINT SPECTRUM LP	Sprint Corporation	50.000000

```
100 rows × 3 columns
```

In [11]:
```
import sys

import mechanize

if len(sys.argv) == 1:
    uri = "http://wwwsearch.sourceforge.net/"
else:
    uri = sys.argv[1]

request = mechanize.Request(mechanize.urljoin(uri, "http://www.antennasearch.com"))
response = mechanize.urlopen(request)
print response
response.close()
```

```
<closeable_response at 0x10d69d1b8 whose fp = <socket._fileobject object at 0x10b5681d0>>
```

In [5]:
```
import sys

import mechanize

if len(sys.argv) == 1:
    uri = "http://wwwsearch.sourceforge.net/"
else:
    uri = sys.argv[1]

request = mechanize.Request(mechanize.urljoin(uri, "mechanize/example.html"))
response = mechanize.urlopen(request)
forms = mechanize.ParseResponse(response, backwards_compat=False)
response.close()
## f = open("example.html")
## forms = mechanize.ParseFile(f, "http://example.com/example.html",
##                             backwards_compat=False)
## f.close()
form = forms[0]
print form  # very useful!

# A 'control' is a graphical HTML form widget: a text entry box, a
# dropdown 'select' list, a checkbox, etc.

# Indexing allows setting and retrieval of control values
original_text = form["comments"]  # a string, NOT a Control instance
form["comments"] = "Blah."

# Controls that represent lists (checkbox, select and radio lists) are
# ListControl instances.  Their values are sequences of list item names.
# They come in two flavours: single- and multiple-selection:
form["favorite_cheese"] = ["brie"]  # single
form["cheeses"] = ["parmesan", "leicester", "cheddar"]  # multi
#  equivalent, but more flexible:
form.set_value(["parmesan", "leicester", "cheddar"], name="cheeses")

# Add files to FILE controls with .add_file().  Only call this multiple
# times if the server is expecting multiple files.
#  add a file, default value for MIME type, no filename sent to server
form.add_file(open("data.dat"))
#  add a second file, explicitly giving MIME type, and telling the server
#   what the filename is
form.add_file(open("data.txt"), "text/plain", "data.txt")

# All Controls may be disabled (equivalent of greyed-out in browser)...
control = form.find_control("comments")
print control.disabled
#  ...or readonly
print control.readonly
#  readonly and disabled attributes can be assigned to
control.disabled = False
#  convenience method, used here to make all controls writable (unless
#   they're disabled):
form.set_all_readonly(False)

# A couple of notes about list controls and HTML:

# 1. List controls correspond to either a single SELECT element, or
# multiple INPUT elements.  Items correspond to either OPTION or INPUT
# elements.  For example, this is a SELECT control, named "control1":

#    <select name="control1">
#     <option>foo</option>
#     <option value="1">bar</option>
#    </select>

# and this is a CHECKBOX control, named "control2":

#    <input type="checkbox" name="control2" value="foo" id="cbe1">
#    <input type="checkbox" name="control2" value="bar" id="cbe2">

# You know the latter is a single control because all the name attributes
# are the same.

# 2. Item names are the strings that go to make up the value that should
# be returned to the server.  These strings come from various different
# pieces of text in the HTML.  The HTML standard and the mechanize
# docstrings explain in detail, but playing around with an HTML file,
# ParseFile() and 'print form' is very useful to understand this!

# You can get the Control instances from inside the form...
control = form.find_control("cheeses", type="select")
print control.name, control.value, control.type
control.value = ["mascarpone", "curd"]
# ...and the Item instances from inside the Control
item = control.get("curd")
print item.name, item.selected, item.id, item.attrs
item.selected = False

# Controls may be referred to by label:
#  find control with label that has a *substring* "Cheeses"
#  (e.g., a label "Please select a cheese" would match).
control = form.find_control(label="select a cheese")

# You can explicitly say that you're referring to a ListControl:
#  set value of "cheeses" ListControl
form.set_value(["gouda"], name="cheeses", kind="list")
#  equivalent:
form.find_control(name="cheeses", kind="list").value = ["gouda"]
#  the first example is also almost equivalent to the following (but
#  insists that the control be a ListControl -- so it will skip any
#  non-list controls that come before the control we want)
form["cheeses"] = ["gouda"]
# The kind argument can also take values "multilist", "singlelist", "text",
# "clickable" and "file":
#  find first control that will accept text, and scribble in it
form.set_value("rhubarb rhubarb", kind="text", nr=0)
#  find, and set the value of, the first single-selection list control
form.set_value(["spam"], kind="singlelist", nr=0)

# You can find controls with a general predicate function:
def control_has_caerphilly(control):
for item in control.items:
if item.name == "caerphilly": return True
form.find_control(kind="list", predicate=control_has_caerphilly)

# HTMLForm.controls is a list of all controls in the form
for control in form.controls:
if control.value == "inquisition": sys.exit()

# Control.items is a list of all Item instances in the control
for item in form.find_control("cheeses").items:
print item.name

# To remove items from a list control, remove it from .items:
cheeses = form.find_control("cheeses")
curd = cheeses.get("curd")
del cheeses.items[cheeses.items.index(curd)]
# To add items to a list container, instantiate an Item with its control
# and attributes:
# Note that you are responsible for getting the attributes correct here,
# and these are not quite identical to the original HTML, due to
# defaulting rules and a few special attributes (e.g. Items that represent
# OPTIONs have a special "contents" key in their .attrs dict).  In future
# there will be an explicitly supported way of using the parsing logic to
# add items and controls from HTML strings without knowing these details.
mechanize.Item(cheeses, {"contents": "mascarpone",
"value": "mascarpone"})

# You can specify list items by label using set/get_value_by_label() and
# the label argument of the .get() method.  Sometimes labels are easier to
# maintain than names, sometimes the other way around.
form.set_value_by_label(["Mozzarella", "Caerphilly"], "cheeses")
```

```
# Which items are present, selected, and successful?
#  is the "parmesan" item of the "cheeses" control successful (selected
#   and not disabled)?
print "parmesan" in form["cheeses"]
#  is the "parmesan" item of the "cheeses" control selected?
print "parmesan" in [
    item.name for item in form.find_control("cheeses").items if item.selected]
#  does cheeses control have a "caerphilly" item?
print "caerphilly" in [item.name for item in form.find_control("cheeses").items]

# Sometimes one wants to set or clear individual items in a list, rather
# than setting the whole .value:
#  select the item named "gorgonzola" in the first control named "cheeses"
form.find_control("cheeses").get("gorgonzola").selected = True
# You can be more specific:
#  deselect "edam" in third CHECKBOX control
form.find_control(type="checkbox", nr=2).get("edam").selected = False
#  deselect item labelled "Mozzarella" in control with id "chz"
form.find_control(id="chz").get(label="Mozzarella").selected = False

# Often, a single checkbox (a CHECKBOX control with a single item) is
# present.  In that case, the name of the single item isn't of much
# interest, so it's a good idea to check and uncheck the box without
# using the item name:
form.find_control("smelly").items[0].selected = True  # check
form.find_control("smelly").items[0].selected = False  # uncheck

# Items may be disabled (selecting or de-selecting a disabled item is
# not allowed):
control = form.find_control("cheeses")
print control.get("emmenthal").disabled
control.get("emmenthal").disabled = True
#  enable all items in control
control.set_all_items_disabled(False)

request2 = form.click()  # mechanize.Request object
try:
    response2 = mechanize.urlopen(request2)
except mechanize.HTTPError, response2:
pass

print response2.geturl()
# headers
for name, value in response2.info().items():
if name != "date":
print "%s: %s" % (name.title(), value)
print response2.read()  # body
response2.close()
```

```
  File "<ipython-input-5-4e1fa9d305ef>", line 133
    for item in control.items:
      ^
IndentationError: expected an indented block
```

```
In [ ]:
```