In [1]:
```
import numpy as np
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import eigs

edges_file = open('wisconsin_edges.csv', "r")
nodes_file = open('wisconsin_nodes.csv', "r")

# create a dictionary where nodes_dict[i] = name of wikipedia page
nodes_dict = {}
for line in nodes_file:
    nodes_dict[int(line.split(',',1)[0].strip())] = line.split(',',1)[1].strip()

node_count = len(nodes_dict)

# create adjacency matrix
A = np.zeros((node_count, node_count))
for line in edges_file:
    from_node = int(line.split(',')[0].strip())
    to_node = int(line.split(',')[1].strip())
    A[to_node, from_node] = 1.0

## Add code below to (1) prevent traps and (2) find the most important pages
# Hint -- instead of computing the entire eigen-decomposition of a matrix X using
# s, E = np.linalg.eig(A)
# you can compute just the first eigenvector with:
# s, E = eigs(csc_matrix(A), k = 1)
```

1 a)

In [2]:
```
# make a new Array to hold the normalized matrix
Anew = np.zeros((node_count, node_count))

# remove traps by adding 0.001
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        A[i, j] = A[i, j] + 0.001

# normalize
for k in range(A.shape[1]):
    norm = np.sum(A[:,k])
    Anew[:,k] = (A[:,k])/norm

# compute Eigenvectors
s, E = eigs(csc_matrix(Anew), k = 1)
E = np.abs(E)
E = E.flatten()

# sort
E_sort = np.argsort(E)
```

In [3]:
```
# print sort, take last and third last elements and find their names
print(E_sort)
```

```
[2041 4298 3874 ... 1345 2312 5089]
```

1 b)

In [4]:
```
print("5089 has page title \"Wisconsin\"")
```

```
5089 has page title "Wisconsin"
```

1 c)

In [5]:
```
print("1345 has page title \"Madison, Wisconsin\"")
```

```
1345 has page title "Madison, Wisconsin"
```

In [ ]: