Homework 1

Nicholas Glaze
nkg2

1.1

a)

①

$$\begin{bmatrix} 0&1&0&0&0&0&0 \\ 1&0&1&1&0&0&0 \\ 0&1&0&1&0&0&0 \\ 0&1&1&0&1&0&0 \\ 0&0&0&1&0&1&1 \\ 0&0&0&0&1&0&1 \\ 0&0&0&0&1&1&0 \end{bmatrix}$$

$$l(v) = \begin{cases} 1 & v=a \\ 2 & b \\ 3 & c \\ 4 & d \\ 5 & e \\ 6 & f \\ 7 & g \end{cases}$$

②

$$\begin{bmatrix} 0&1&1&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 1&0&0&1&0&0&0 \\ 1&0&1&0&1&0&0 \\ 0&0&0&1&0&1&1 \\ 0&0&0&0&1&0&1 \\ 0&0&0&0&1&1&0 \end{bmatrix}$$

$$l(v) = \begin{cases} 1 & v=b \\ 2 & a \\ 3 & c \\ 4 & d \\ 5 & e \\ 6 & f \\ 7 & g \end{cases}$$

b) $[A^3]_{i,j}$ = # of length-3 paths from $i \to j$; $[\mathrm{diag}(A^3)]_i$ = length-3 cycles starting + ending @ i:

Consider a triangle T:

; from node a, there are 2 length-3 cycles passing through exactly the nodes in T: (a,b,c,a) and (a,c,b,a). The same is true for nodes b + c, meaning that, among the elements of $\mathrm{diag}(A^3)$, T is counted $3 \cdot 2 = \underline{6}$ times. Further, each length-3 path described by $\mathrm{diag}(A^3)$ defines only one triangle, so to count triangles, $\frac{1}{6}\sum_i \mathrm{diag}(A^3) = \underline{\frac{1}{6}\mathrm{trace}(A^3)}$ is valid.

c) $\frac{1}{6}\mathrm{trace}(A^3) \stackrel{?}{=} \frac{1}{6}\mathrm{trace}(\overbrace{(PAP^T)^3}^{(A')^3})$

$= \frac{1}{6}\mathrm{trace}(PAP^TPAP^TPAP^T)$

$= \frac{1}{6}\mathrm{trace}(PA^3P^T)$

• We know that $\underline{P^TP = I}$, since P^T inverts the transformation applied by P

$= \frac{1}{6}\mathrm{trace}(A^3PP^T) \leftarrow \mathrm{tr}(ABC) = \mathrm{tr}(BCA) = \ldots$

$= \frac{1}{6}\mathrm{trace}(A^3 I)$

$= \frac{1}{6}\mathrm{trace}(A^3)$ □

1.2. $G=(V,E)$ $\vec{1}=\begin{bmatrix}1\\ \vdots\\ 1\end{bmatrix}$

a) $\boxed{d_{in}=A\vec{1}}$; sum of each row, since $A_{i,-}$ = edges from i to other nodes

$\boxed{d_{out}=\vec{1}^T A}$; sum of each column, since $A_{-,j}$ = edges from other nodes to i

b) $\boxed{trace(A)}$; $A_{i,i}=1$ iff the edge $(i,i)\in E$; (u,v) is a self-loop iff $u=v$

c) $\boxed{\frac{1}{2}trace(A^2)}$; $A^2_{i,i}$ = # of length-2 paths (i,v,i); length-2 path = mutual edge. Divide by 2 since same edge counted twice as (u,v,u) and (v,u,v)

d) $\boxed{\frac{1}{4}diag(A^3)(D(D-I))^{-1}}$

$\frac{1}{2}diag(A^3)_i$ = # triangles starting from i. $\frac{1}{2}$ = # edges amongst nbrhood

$\cdot\ \frac{1}{2}(D(D-I))^{-1}_i = 1/\frac{d_i(d_i-1)}{2} = 1/\binom{d_i}{2}$ = # of possible edges amongst a d_i-size nbrhood

1.3

a) $P[d]=\begin{cases}1/n & d=n-1\\ \frac{n-1}{n} & d\neq n-1\end{cases}$

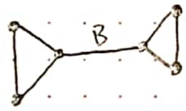
b) $\bar{d}=\frac{2n-2}{n}$

c) $C=0$ since no edges connect any node's neighboring nodes to each other

d) $Q[d]=\begin{cases}1/2 & d\in\{1,(n-1)\}\\ 0 & else\end{cases}$

• $(2n-2)$ directed edges; half lead to center $(d=n-1)$, other half lead to an outer node $(d=1)$

1.4



• Each triangle is a k-component, since 2 paths exist within them, but not through the bridge edge B.

• All nodes have degree $\geq 2 \Rightarrow$ The whole graph is a 2-core

2.1. $C_i=\frac{n}{\sum_j d(i,j)}$

• Define regions as N_1+N_2; $|N_1|=n_1$, $|N_2|=n_2$.

1) $\Rightarrow C_1=\frac{n}{\sum_{j\in N_1} d(1,j)+\sum_{j\in N_2} d(1,j)}$

2) • We know $\sum_{j\in N_1} d(2,j)=\sum_{j\in N_1} d(1,j)+1(n_1)$ ← 1 additional distance unit for each node, since $d(1,2)=1$ & all paths go thru

$\sum_{j\in N_2} d(1,j)=\sum_{j\in N_2} d(2,j)+n_2$ ← Same, vice versa

3) $\Rightarrow C_1=\frac{n}{\sum_{j\in N_1}d(1,j)+\sum_{j\in N_2}d(1,j)}$

$=\frac{n}{\sum_{j\in N_1}d(2,j)-n_1+\sum_{j\in N_2}d(2,j)+n_2}$

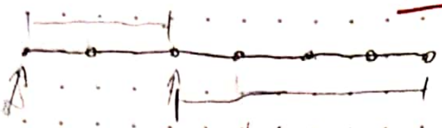
$\frac{1}{C_1}=\frac{\sum_{j\in N_1}d(2,j)}{n}-\frac{n_1}{n}+\frac{\sum_{j\in N_2}d(2,j)}{n}+\frac{n_2}{n}$

$=\frac{\sum_{j\in N_1}d(2,j)+\sum_{j\in N_2}d(2,j)}{n}-\frac{n_1}{n}+\frac{n_2}{n}=\frac{1}{C_2}-\frac{n_1}{n}+\frac{n_2}{n}=\frac{1}{C_1}$

4) $\Rightarrow \boxed{\frac{1}{C_1}+\frac{n_1}{n}=\frac{1}{C_2}+\frac{n_2}{n}}$

Homework 1 (cont.)

2.2.



between i and j, $\binom{j-i+1}{2}$ shortest paths exist

• each pair has 1 shortest path

• $\binom{n}{2}$ shortest paths, total

• For node i, paths through i can connect one of the $(i-1)$ "left" nodes and one of the $n-i$ "right" nodes

$$C_{Be}(i) = \frac{(i-1)(n-i)}{\binom{n}{2}} = \frac{-i^2+in-n+i}{\binom{n}{2}}$$

$$\boxed{C_{Be}(i) = \frac{(n+1)i - i^2 - n}{\binom{n}{2}}}$$

2.3. k-regular = all n nodes have degree k.

a). We know $\vec{1}$ is an e-vector of A, since $[A\vec{1}]_i = \sum_j A_{ij} = d_i$ $\Rightarrow A\vec{1} = k\vec{1}$ ← e-val $= k$.

• Since A is connected + undirected, we know by the Perron-Frobenius theorem that the leading e-vector of A has all same-sign elements. Also, we know that different e-vals' e-vectors for A must be orthogonal, but e-vectors orthogonal to $\vec{1}$ cannot have all same-sign elements; this means that $\vec{1}$ is the only positive-element e-vector $\Rightarrow$ k must be the largest e-val (by P-F).

$\Rightarrow$ since $k\vec{1} = A\vec{1}$, $\boxed{C_{E_i} = \vec{1}.}$ (or a multiple of $\vec{1}$)

b) $C_k = (I - \alpha A)^{-1}\vec{1}$

$= (VIV^T - \alpha V\Lambda V^T)^{-1}\vec{1}$ $\quad (VV^T = I)$

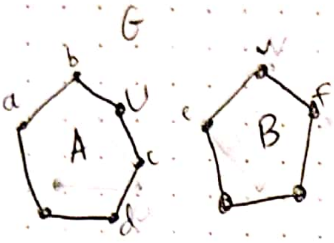
$= V(I - \alpha\Lambda)^{-1}V^T \cdot \vec{1}$ ← $(I-\alpha\lambda)^{-1}$ modifies e-vals without changing e-vectors

$\Rightarrow$ we want e-val λ s.t. $V(I-\alpha\Lambda)^{-1}V^T \cdot \vec{1} = \lambda\vec{1}$

• λ_{max} of A becomes $\frac{1}{1-\alpha\lambda_{max}} = \frac{1}{1-\alpha k}$ $\Rightarrow$ $(I - \alpha A)^{-1}\vec{1} = \frac{1}{1-\alpha k}\vec{1}$

$$\boxed{\Rightarrow C_k = \frac{1}{1-\alpha k}\vec{1}}$$

$(0.8 \le \alpha \le 0.9)$

c) 

• Betweenness centrality; assume that paths where $d(u,v)=\infty$ are excluded.

\# shortest paths =

	length-1	2	3
A	6	6	6 ← 2 per pair
B	5	5	0

$= \underline{28}$

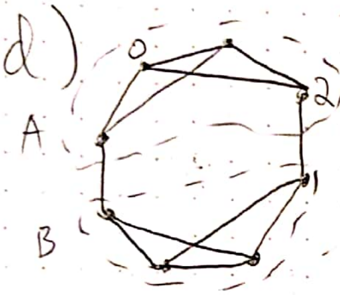
• U is contained in: a-c, b-c, b-d => $\underline{3 \text{ paths}}$

• V is contained in: e-f => $\underline{1 \text{ path}}$

$\Rightarrow C_{Be}(U) = 3/28$

$C_{Be}(V) = 1/28$

$\boxed{C_{Be}(U) \neq C_{Be}(V), \text{ and } G \text{ is 2-regular}}$

d) 

Node 0 has $C_{Be}(0) = 0.0476$ (computed w/networkx)

$C_{Be}(2) = 0.214$

This is because node 2 is part of a local bridge between A & B, while node 0 is not.

2.4

a) <u>No</u>; nodes A and B each have out-degree 2, so in evaluating the next $r(k+1)$, they will contribute 0.075 each to C, D, and E respectively, resulting in C = 0.075, D = 0.15, E = 0.075. These are different from the given values.

b) From code, $\pi = $

$$\pi = \begin{array}{c} \begin{bmatrix} .1538 \\ .1538 \\ .0769 \\ .2308 \\ .0769 \\ .3078 \end{bmatrix} \end{array} \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}$$

This is consistent w/ a). (different values)

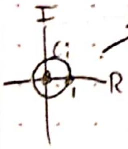
• π is an e-vector of P^T w/ e-val 1; found using numpy

2.4 (cont)

c) $[D_{out}^{-1}]_{i,j} = \begin{cases} 0 & d_i^{out}=0 \\ 1/d_i^{out} & i=j \\ 0 & \text{else} \end{cases}$; d_i^{out} is always non-negative, so $1/d_i^{out}$ must be as well. (or 0 if $d_i^{out}=0$) // D_{out} is non-negative.

- By definition, $A_{i,-}$ will have d_i^{out} "1"s in its row, and all else 0s.
  $\Rightarrow [D_{out}^{-1}A]_{i,-}$ will contain d_i^{out} non-0 values, all equal to $1/d_i^{out}$, since the non-0 value in row i of (D_{out}^{-1}) is the only one that will be multiplied against the "1"s in row i of A. Row-wise sums of P are thus $d_i^{out} \cdot 1/d_i^{out} = 1$.
- Since D_{out}^{-1} is non-negative (& A is by definition), P is non-negative, since adds & multiplications of non-negative numbers $\in \mathbb{R}$ produce only non-negative results.
  $\Rightarrow$ P is Markov

- The Gershgorin circle theorem states that for a square matrix, each e-val must lie within one of the Gershgorin discs C_i.
  C_i is a circle with $P_{i,i}$ as its center and the L_1 norm of all other values in $P_{i,-}$ as its radius.
- Since G has no self-loops, $\text{diag}(A)=\vec{0}$; since D_{out} is a diagonal matrix, we know $\text{diag}(D_{out}A)=\vec{0} \Rightarrow$ all Gershgorin discs are centered at 0. Since all rows in P sum to 1, we then know the radii of all discs $=1$ (since the other row elements must sum to 1).



$\Rightarrow$ all e-vals must be within radius 1 of the origin $\Rightarrow |\lambda| \leq 1, \forall \lambda$.

---

d) - We know P is stochastic (non-negative + rows sum to 1)
  $\Rightarrow P\vec{1} = 1 \cdot \vec{1}$ (e-val 1, e-vector $\vec{1}$), by P-F theorem
- Also, P is irreducible + aperiodic $\Rightarrow$ leading e-val is simple
- $\left(\lim_{k\to\infty} x^T P^k\right)^T = \lim_{k\to\infty} (P^T)^k x = c V_{max}^T$
- The largest e-val $\lambda_{max}=1$ of P^T has $\boxed{V_{max}=\pi}$ (from b)), which is equal to V_{max} here.
  Because of this, we know that the above limit converges to cV_{max}, (since cols of $(P^T)^k$ converge to π)
  and through inspection, let $\boxed{c = \underbrace{n\bar{x}}_{n\,\text{mean}(x)}}$

3.2

a) $Q(G, P_1) = \frac{1}{2|E|} \sum_s \sum_{i,j \in s} \left(A_{ij} - \frac{d_i d_j}{2|E|}\right) = \frac{1}{n^2} \sum_s \sum_{i,j \in s} \left(A_{ij} - \frac{1}{2}\right)$

- $|E| = 2\binom{n}{2} + n = \frac{n(n-1)}{2} \cdot 2 + n = \underline{n^2}$
- $d_i = ((n-1)+1) = \underline{n} \ \forall i$
- $\frac{d_i d_j}{2|E|} = \underline{\frac{1}{2}} \ \forall i,j$

- To maximize Q, we must minimize the # of $-\frac{1}{2}$ terms + maximize the # of $A_{ij} = 1$ terms. P_1 has each node connecting to $(n-1)$ others in its group, resulting in $\underset{\uparrow \text{ 2 groups}}{2}\underset{\uparrow \text{ edges/group}}{\binom{n}{2}}$ "$A_{ij}=1$" terms.

- Swapping k ~~nodes~~ $(k \leq \frac{n}{2})$ pairs of nodes between the groups of P_1 to produce P_2, such that k of the "bridge" edges are contained within each group, produces the following change in each ~~node's with~~ cluster's total degree:

bridge edge

$\Delta = \left(-(\underbrace{k(n-k)}_{\text{each swapped node loses an edge to the (n-k) not-swapped nodes in its } P_1 \text{ group}}) + \underbrace{k}_{\text{k bridge edges contained per group}}\right) \cdot \underbrace{2}_{\text{symmetric groups}} = (-kn + k^2 + k) \cdot 2 = \underline{2(k^2 + k - kn)}$

upwards-facing parabola, bounds maximize

$\underline{k=1}$

$\Delta = 2(1+1-n)$

$n \geq 3 \Rightarrow \underline{\Delta < 0}$

$\underline{k = n/2}$

$\Delta = 2\left(\frac{n^2}{4} + \frac{n}{2} - \frac{n^2}{2}\right)$

$= \frac{n^2}{2} + n - n^2 = -\frac{n^2}{2} + n$

$= n\left(\frac{1}{2}n + 1\right)$

$0 \overset{?}{>} \Delta = n(n+2)$

true for $n > 2$, we have $\underline{n \geq 3}$, so $\underline{\Delta < 0}$

$\underline{\Delta < 0} \ \forall n \geq 3, \ \forall 1 \leq k \leq \frac{n}{2}$,

so swapping from P_1 decreases modularity

$\Rightarrow P_1$ is best for 2 equal-size groups. $\square$

3.2

b) From a), $Q(G, P_1) = \frac{1}{4} - \frac{1}{4n}$, $Q(G, P_2) = \frac{1}{4n^2}(n^2 - n + k^2 + k - nk)$

1) Start with P_1; let P_3 be P_1, but with ℓ $(\ell \le n)$ nodes shifted from one partition to the other. The change in modularity is as follows:

$$\Delta = \underbrace{\underbrace{(-\ell(n-1)}_{\substack{\sum A_{ij} \text{ term};\\ i,j\\ d_i = n-1 \\ \text{within group}}} + \underbrace{\tfrac{1}{2}((n-1)\cdot\ell + \tbinom{\ell}{2}))}_{\substack{\frac{d_i d_j}{2N_e} \text{ term} \\ ((n-1)\cdot 2 + \binom{\ell}{2}) \\ \text{pairs lost})}}}_{\Delta_{old} = \text{old group}} + \underbrace{\underbrace{(+(\ell + \tbinom{\ell}{2}))}_{\substack{\ell \text{ bridge edges, } \binom{\ell}{2} \\ \text{within new nodes}}} - \underbrace{(\tfrac{1}{2}(n\ell + \tbinom{\ell}{2}))}_{\substack{n\ell \cdot (\text{old-new}) \text{ pairs,} \\ \binom{\ell}{2} \text{ within new nodes}}}}_{\Delta_{new} \text{ new group}}$$

$$\Delta = -n\ell + \cancel{\ell} + \tfrac{1}{2}\cancel{n\ell} - \tfrac{1}{2}\cancel{\ell}^2 + \cancel{\ell} + \frac{\ell!}{2!(\ell-2)!} - \tfrac{1}{2}\cancel{n\ell}$$
$$= -n\ell - \tfrac{1}{2}\ell^2 + 2\ell + \tfrac{1}{2}\ell(\ell-1)$$
$$= -n\ell - \cancel{\tfrac{1}{2}\ell^2} + 2\ell + \cancel{\tfrac{1}{2}\ell^2} - \tfrac{1}{2}\ell = 2\ell - \tfrac{1}{2}\ell - n\ell = \underline{\tfrac{3}{2}\ell - n\ell}$$

; $n > \frac{3}{2}$, so $\underline{\Delta < 0 \Rightarrow \text{mod decreases}}$

2) Let P_4 be P_1, with ℓ $(\ell \le n)$ nodes moved from the same group to a new group.

No 2-partition groups work (nor does 1 partition)

$$\Delta = \underbrace{\Delta_{old}}_{\text{same}} + (\underbrace{\tbinom{\ell}{2}}_{\substack{\text{new partition is} \\ \text{fully connected}}} - \underbrace{\tfrac{1}{2}\tbinom{\ell}{2}}_{\binom{\ell}{2} \text{ pairs of nodes}}) = -\cancel{n\ell} + \cancel{\ell} + \tfrac{1}{2}\cancel{n\ell} - \tfrac{1}{2}\ell^2 + \tbinom{\ell}{2} + \tbinom{\ell}{2} - \tfrac{1}{2}\tbinom{\ell}{2}$$
$$= -\frac{n\ell}{2} + \ell - \tfrac{1}{2}\ell^2 + \tfrac{3}{4}\ell(\ell-1)$$
$$= -\tfrac{1}{2}n\ell - \tfrac{1}{2}\ell^2 + \ell + \tfrac{3}{4}\ell^2 - \tfrac{3}{4}\ell$$
$$= \underline{-\tfrac{1}{2}n\ell + \tfrac{1}{4}\ell^2 + \tfrac{1}{4}\ell}$$

← also upwards parabola

$\underline{\ell = 1}$

$\Delta = -\frac{1}{2}n + \frac{1}{4} + \frac{1}{4}$

$\Delta = -\frac{1}{2}n + \frac{1}{2} < 0$ for $\underline{n > 1}$, which is $\underline{\text{true}}$.

$\underline{\ell = n}$

$\Delta = -\frac{1}{2}n^2 + \frac{1}{4}n^2 + \frac{1}{4}n$

$\Delta = -\frac{1}{4}n^2 + \frac{1}{4}n < 0$, for $\underline{n > 1}$, which is $\underline{\text{true}}$.

$\Rightarrow$ $\underline{P_4 \text{ decreases modality}}$

3) Let P_5 be P_1, but with ℓ pairs of nodes moved to a new group, with one node in each pair coming from each group.

$$\Delta = (-2\ell(n-1) + \cancel{\tfrac{1}{2}}\cancel{2}((n-\ell)\cdot\ell + \tbinom{\ell}{2}))) + (\ell + 2\tbinom{\ell}{2} - \tfrac{1}{2}\tbinom{2\ell}{2})$$
$$= -2\ell n + 2\ell + n\ell - \ell^2 + \frac{\ell(\ell-1)}{2} + \ell + \ell(\ell-1) - \frac{2\ell(2\ell-1)}{4}$$
$$= -n\ell + 2\ell - \ell^2 + \tfrac{1}{2}\ell^2 - \tfrac{1}{2}\ell + \ell + \ell^2 - \ell - \ell^2 - \tfrac{1}{2}\ell$$
$$= \tfrac{1}{2}\ell^2 - n\ell + \ell$$

$\underline{\ell = 1}$ ✓

$\underline{\frac{1}{2} - n + 1 < 0 \text{ for } \underline{n \ge 3}}$

$\underline{\ell = n}$

$\frac{1}{2}n^2 - n^2 + n$

$= -\frac{1}{2}n^2 + n$

$0 > n(1 - \frac{1}{2}n)$

$0 > n(2 - n)$ true for $n \ge 3$, which we have

✓

$\boxed{\text{Since } P_2 \text{ (a)}, P_3, P_4, \text{ and } P_5 \text{ all decrease modularity, and can be used to construct all partitions from } P_1,}$ $\underline{P_1 \text{ maximizes modularity.}}$ □

In [ ]:

```
import networkx as nx
import csv
import matplotlib.pyplot as plt
import numpy as np
from itertools import product
from scipy.cluster.hierarchy import dendrogram, linkage
```

Problem 1.2.4: Limiting PageRank values

b)

In [ ]:

```
# Define the graph, using alphabet-order node enumeration
A = np.array([
    [0, 0, 1, 1, 0, 0],
    [0, 0, 0, 1, 1, 0],
    [0, 0, 0, 1, 0, 1],
    [0, 0, 0, 0, 0, 1],
    [0, 0, 0, 1, 0, 1],
    [1, 1, 0, 0, 0, 0]
])

# compute P
D_out = np.diag(
    [2, 2, 2, 1, 2, 2]
)

P = np.linalg.inv(D_out) @ A
```

In [ ]:

```
# pi is the e-vector of P.T with e-val 1
# find e-vals w and e-vectors v
w, v = np.linalg.eig(P.T)
w
```

Out[ ]:

```
array([ 1.00000000e+00+0.j        , -3.40275770e-01+0.81658512j,
       -3.40275770e-01-0.81658512j,  9.11172734e-09+0.j        ,
       -9.11172727e-09+0.j        , -3.19448460e-01+0.j        ])
```

In [ ]:

```
# w_1 = 1, so get the first eigenvector
pi = np.real(v[:, 0])

# normalize to sum to 1
pi /= pi.sum()
pi
```

Out[ ]:

```
array([0.15384615, 0.15384615, 0.07692308, 0.23076923, 0.07692308,
       0.30769231])
```

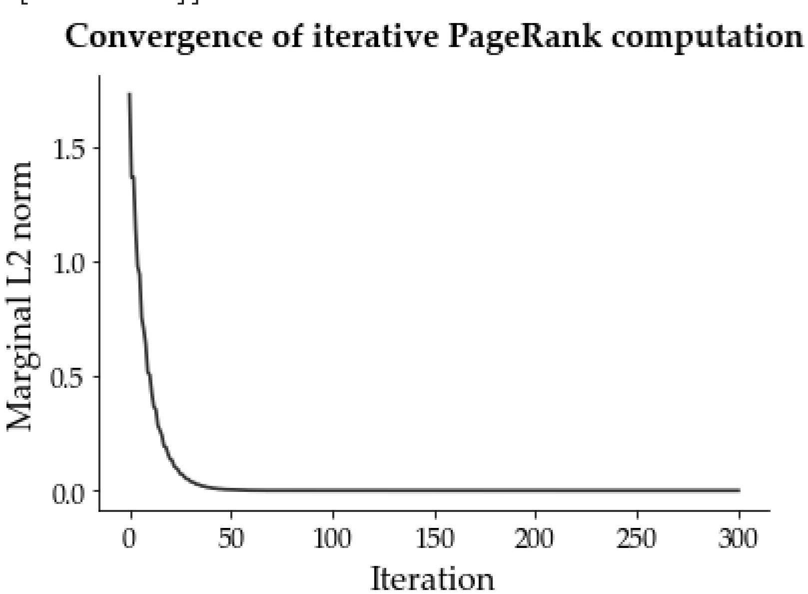
d)

In [ ]:

```
# solve for pi iteratively

# let x = ones vector
```

```
x = np.ones((A.shape[0], 1))

# track convergence
norms = []
prev = x

# compute limit of ((P^T)**k)x iteratively
while True:
    x = P.T @ x

    norms.append(np.linalg.norm(x - prev, ord=2))
    if norms[-1] < 1e-100:
        break

    prev = x

# since x is the ones vector, c = 1/n = 1/6
c = 1 / len(x)

pi = c * x

print("pi:")
print(pi)

plt.plot(range(len(norms)), norms)
plt.title("Convergence of iterative PageRank computation")
plt.xlabel("Iteration")
plt.ylabel("Marginal L2 norm");

# Converges very quickly!
```

```
pi:
[[0.15384615]
 [0.15384615]
 [0.07692308]
 [0.23076923]
 [0.07692308]
 [0.30769231]]
```



Problem 1.3.1: Modularity and spectral clustering

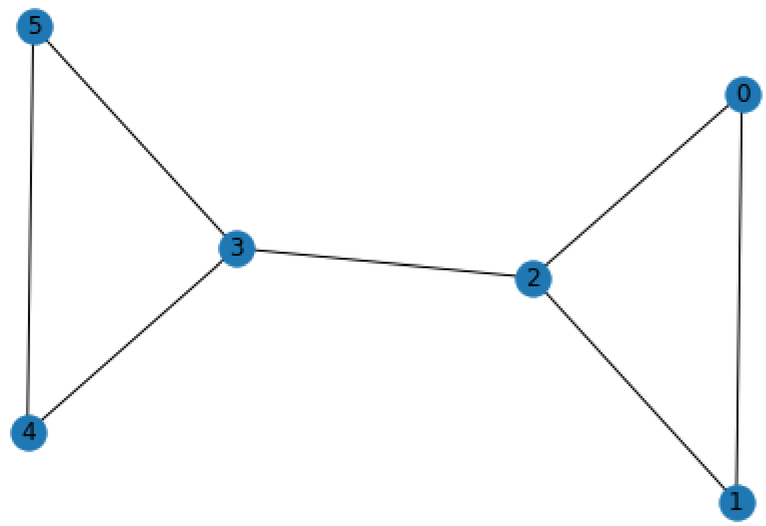
a)

In [ ]:

```
# clustering methods

def mod_max_clustering(G, A):

    # compute modularity matrix
    d = np.array([d_i for (i, d_i) in G.degree()])

    n_edges = len(G.edges)

    B = A - np.outer(d, d) / (2 * n_edges)

    # get leading e-vector
    w, v = np.linalg.eig(B)
    w_max_idx = np.argsort(w)[-1]
    v_max = v[:, w_max_idx]

    return v_max > 0 # binary

def spectral_clustering(G, A):

    # compute Laplacian
    d = np.array([d_i for (i, d_i) in G.degree()])
    D = np.diag(d)
    L = D - A

    # get second-smallest e-vector
    w, v = np.linalg.eig(L)
    w2_idx = np.argsort(w)[1]
    v2 = v[:, w2_idx]

    return v2 > 0 # binary

def draw_clusters(G, A, cluster_func):
    clusters = ["lightblue" if x else "red" for x in cluster_func(G, A)]
    nx.draw(G, node_color=clusters, with_labels=True)
```

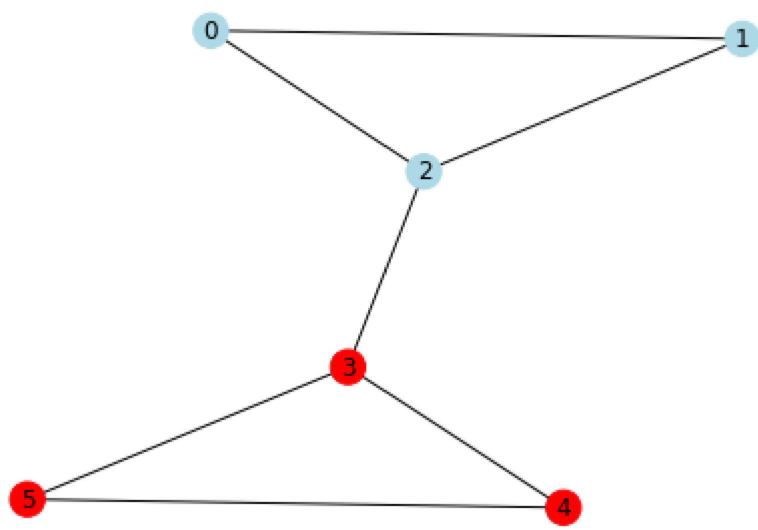
In [ ]:

```
# create graph
A = np.array(
    [
        [0, 1, 1, 0, 0, 0],
        [1, 0, 1, 0, 0, 0],
        [1, 1, 0, 1, 0, 0],
        [0, 0, 1, 0, 1, 1],
        [0, 0, 0, 1, 0, 1],
        [0, 0, 0, 1, 1, 0]
    ])

G = nx.from_numpy_matrix(A)

nx.draw(G, with_labels=True)
```



In [ ]:

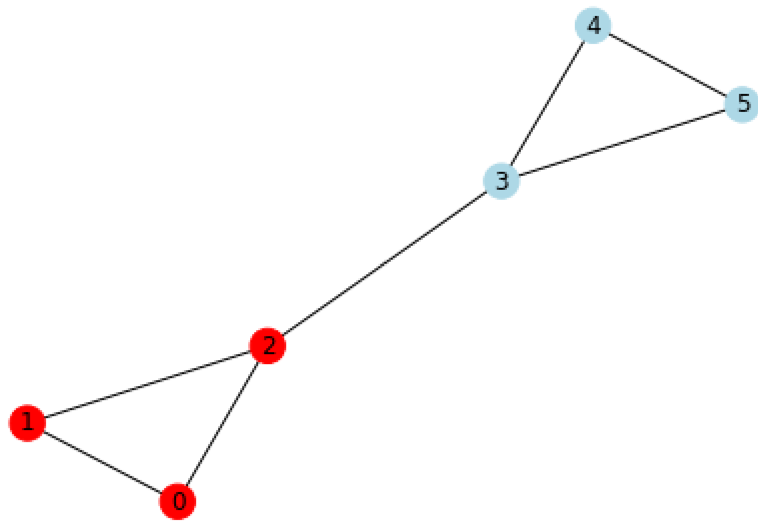
```
# Modularity maximization clustering
draw_clusters(G, mod_max_clustering)
```



These clusters make sense intuitively -- the 2-3 edge is a local bridge connecting the clusters

In [ ]:

```
# Spectral clustering
draw_clusters(G, spectral_clustering)
```

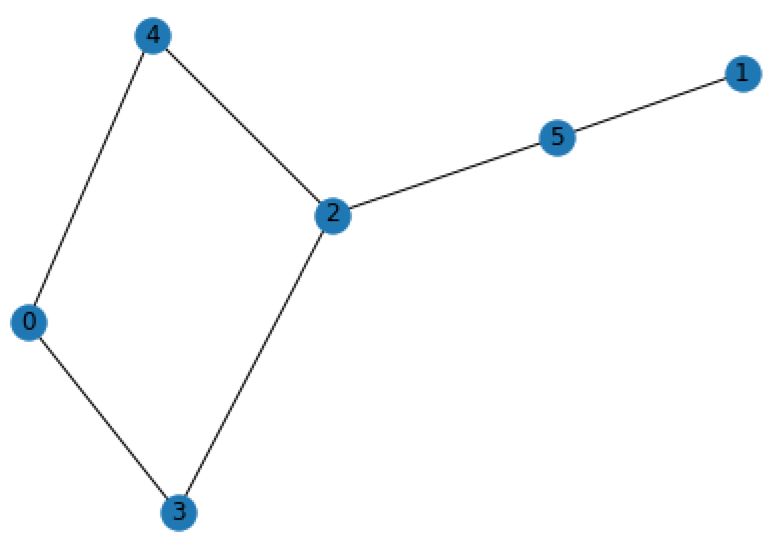


Same clusters as above! the two methods agree here.

b)

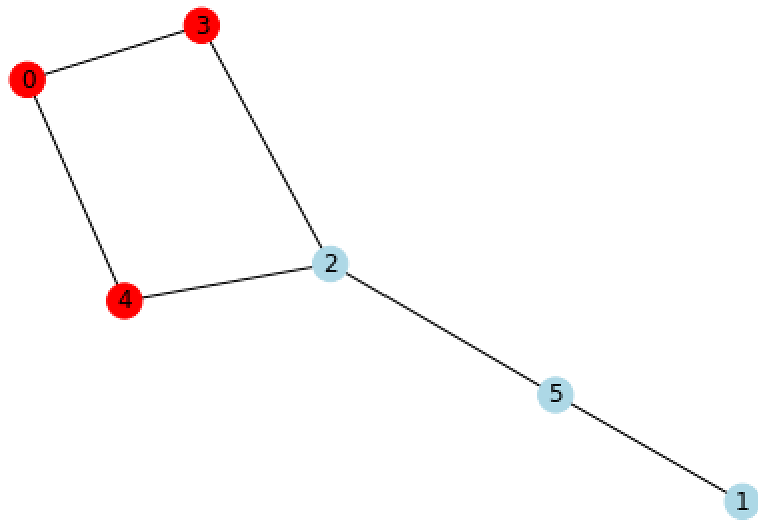
In [ ]:

```
A = np.array([
    [0, 0, 0, 1, 1, 0,],
    [0, 0, 0, 0, 0, 1,],
    [0, 0, 0, 1, 1, 1,],
    [1, 0, 1, 0, 0, 0,],
    [1, 0, 1, 0, 0, 0,],
    [0, 1, 1, 0, 0, 0,],
    ])

G = nx.from_numpy_matrix(A)

nx.draw(G, with_labels=True)
```



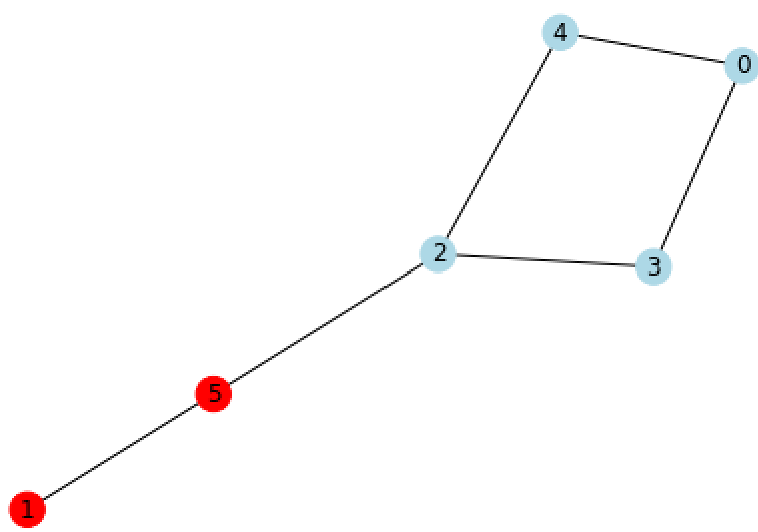
In [ ]:

```
draw_clusters(G, A, mod_max_clustering)
```



Modularity maximization clustering clusters node 2 with the "tail" of the graph.

In [ ]:
```
draw_clusters(G, A, spectral_clustering)
```



Spectral clustering clusters node 2 with the "body" of the graph.

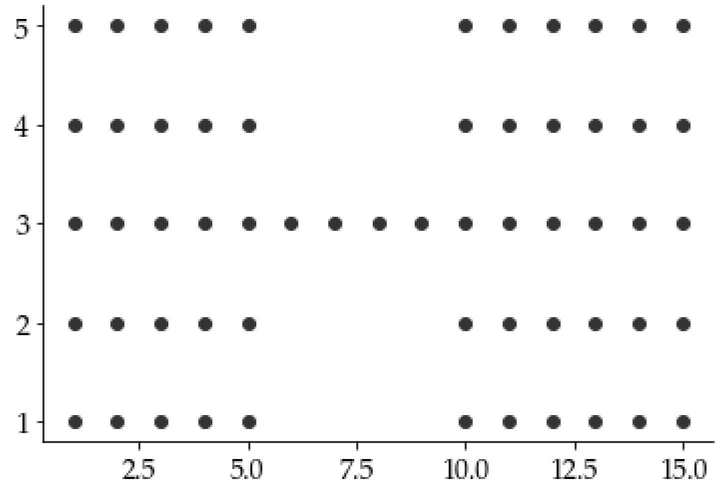
1.3.3: Hierarchical clustering and the chaining effect

a)

In [ ]:
```
# Define the point cloud
grid_x = np.linspace(1, 15, num=15)
grid_y = np.linspace(1, 5, num=5)
grid_2d = np.array(list(product(grid_y, grid_x)))

allowed_region = (grid_2d[:, 1] <= 5) | (grid_2d[:, 1] >= 10) | (grid_2d[:, 0] == 3)

point_cloud = grid_2d[allowed_region, :]
```

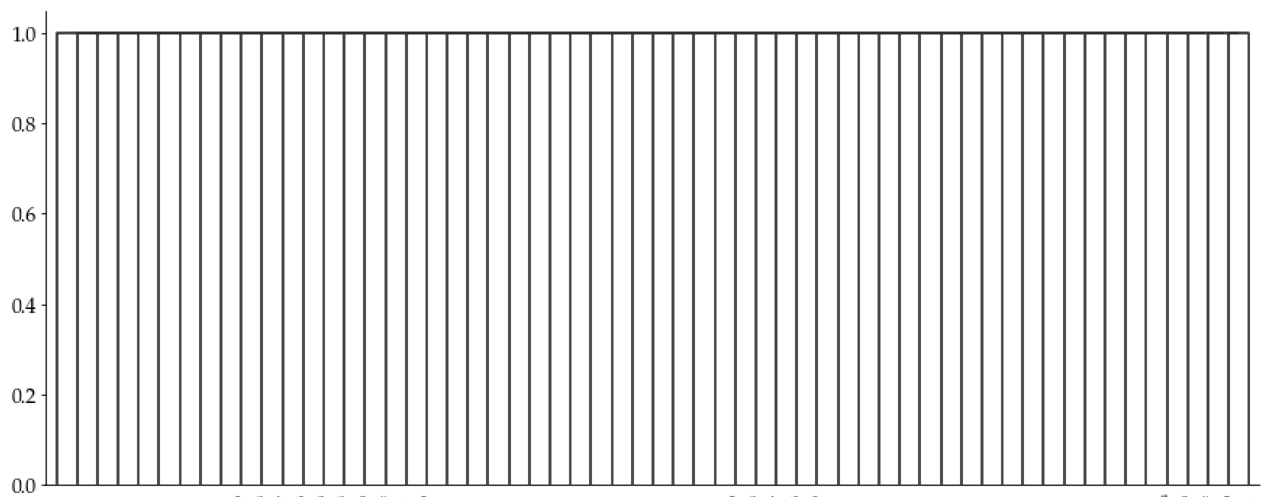
```
plt.scatter(point_cloud[:, 1], point_cloud[:, 0]);
```

Out[ ]: <matplotlib.collections.PathCollection at 0x1f4cf1ee190>



In [ ]:
```
def clust_dendrogram(X, linkage_metric):
    hier_clust = linkage(X, linkage_metric)
    fig = plt.figure(figsize=(15, 6))
    dn = dendrogram(hier_clust)
    plt.show()
```

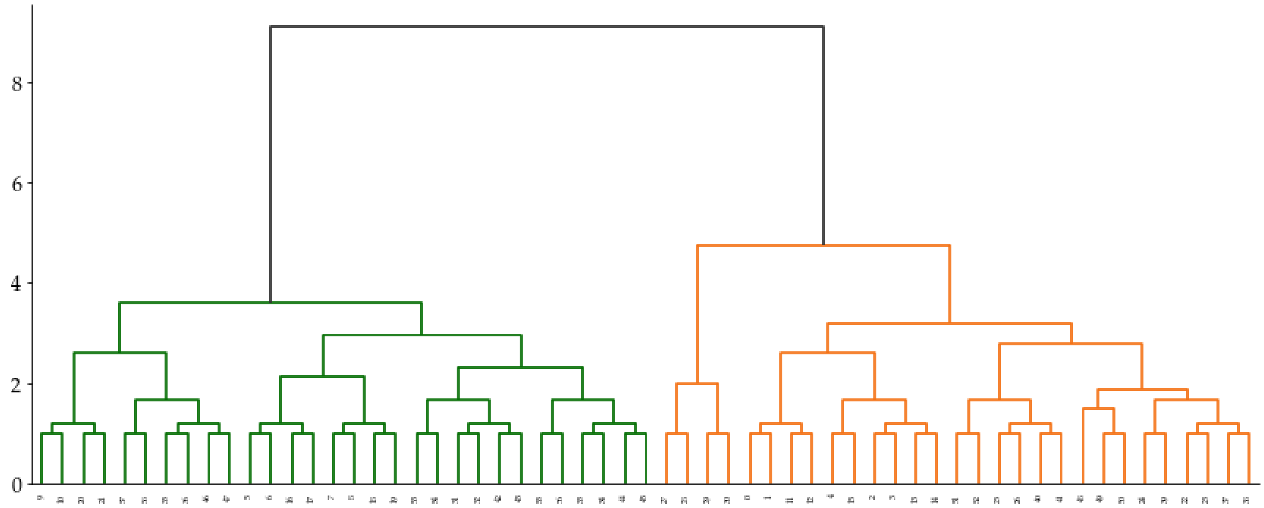
In [ ]:
```
# Apply single-linkage hierarchical clustering to the point cloud
clust_dendrogram(point_cloud, 'single')
```



Single linkage causes individual points to be added to larger clusters one-by-one, instead of points being combined into clusters (which are then combined with other clusters). This is called the chaining effect, and produces degenerate trees such as the one above. Most other linkage metrics have the (more desirable) inherent property of considering clusters of several points as farther away than neighboring individual points, since some of these clusters' points are generally not immediate neighbors of "this" point.

In [ ]:
```
# Apply average-linkage hierarchical clustering to the point cloud
```

```
clust_dendrogram(point_cloud, 'average')
```



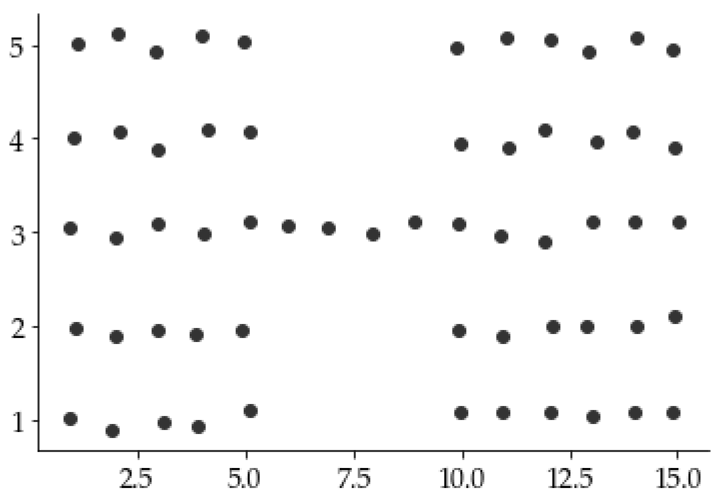
In this case, the two regions of points are clearly identified; the largest vertical distance in the dendrogram exists in the top-most level, meaning that 2 clusters is (as expected) the best choice for this data.

b)

In [ ]:

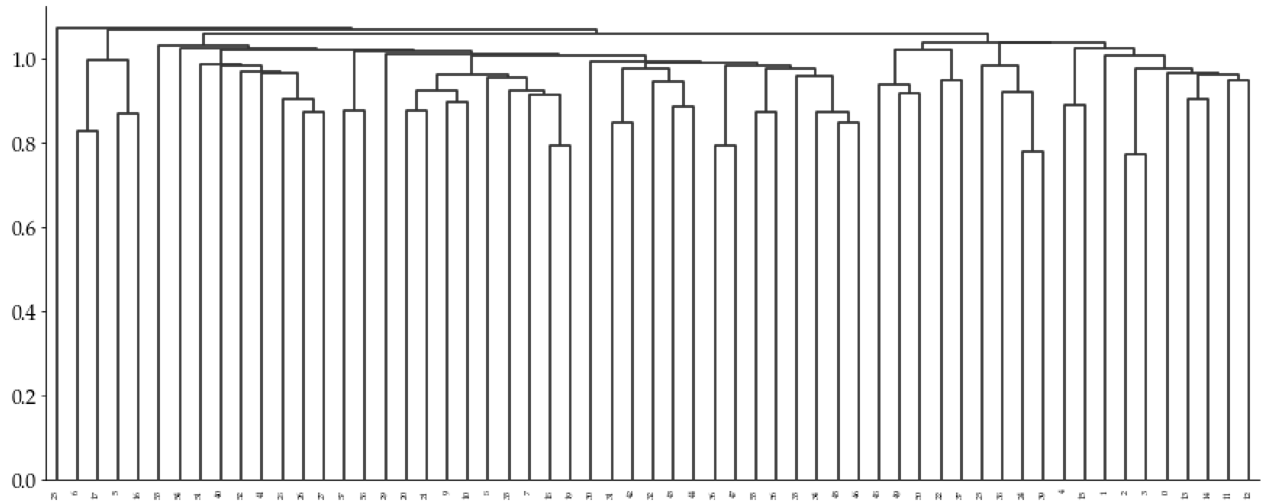
```
# Add uniform noise to the point cloud in the range [-0.125, 0.125) along both axes
point_cloud_perturbed = point_cloud + ((np.random.rand(*point_cloud.shape) - 0.5) / 4)

plt.scatter(point_cloud_perturbed[:, 1], point_cloud_perturbed[:, 0])
```

Out[ ]: <matplotlib.collections.PathCollection at 0x1f4d085c460>



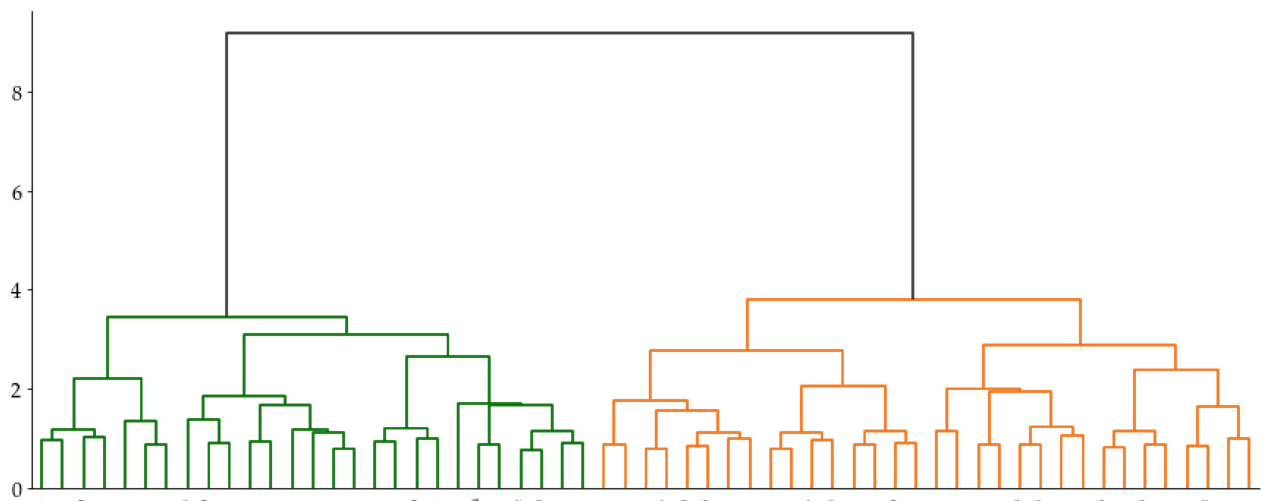
In [ ]:

```
# Single linkage on the perturbed points
clust_dendrogram(point_cloud_perturbed, 'single')
```



Now that the points have been perturbed, the single linkage dengrogram much better resembles the expected tree structure. At the lowest level of the tree, it appears that each point is being combined with the neighbor it is farthest from after perturbation, and the "odd nodes out" (e.g. 52) are combined into clusters later. However, the largest vertical distance exists at the lowest level of the tree; that is, where pairs of neighboring points are combined into 2-point clusters. This implies that the best number of clusters is equal to the number of points; that is, each point should be in its own cluster. This is not a useful result at all, showing that even after perturbation, single-linkage hierarchical clustering is not useful for this problem.

In [ ]:
```
# Average linkage on the perturbed points
clust_dendrogram(point_cloud_perturbed, 'average')
```



The average linkage dendrogram looks almost exactly the same after the points are perturbed. From this example, average linkage appears to be much more robust than single linkage.

Problem 4

1.4.1: Generating the graph

In [ ]:
```
# create an empty undirected graph
```

```
G = nx.Graph()

# read in table
with open('primaryschool.csv') as f:
    reader = csv.reader(f, delimiter='\t')

    for row in reader:
        # interpret each data point
        t, i, j, c_i, c_j = row
        t, i, j = map(int, [t, i, j])

        if not G.has_node(i):
            G.add_node(i, c=c_i)

        if not G.has_node(j):
            G.add_node(j, c=c_j)

        # add the edge described by this row to the graph,
        #   or modify its weight if already present
        if G.has_edge(i, j):
            G[i][j]['weight'] += 1
        else:
            # weight = count of this pair's interactions
            G.add_edge(i, j, weight=1)

print('Nodes: {}; Edges: {}'.format(len(G.nodes), len(G.edges)))
```

```
Nodes: 242; Edges: 8317
```

1.4.2: Descriptive analysis

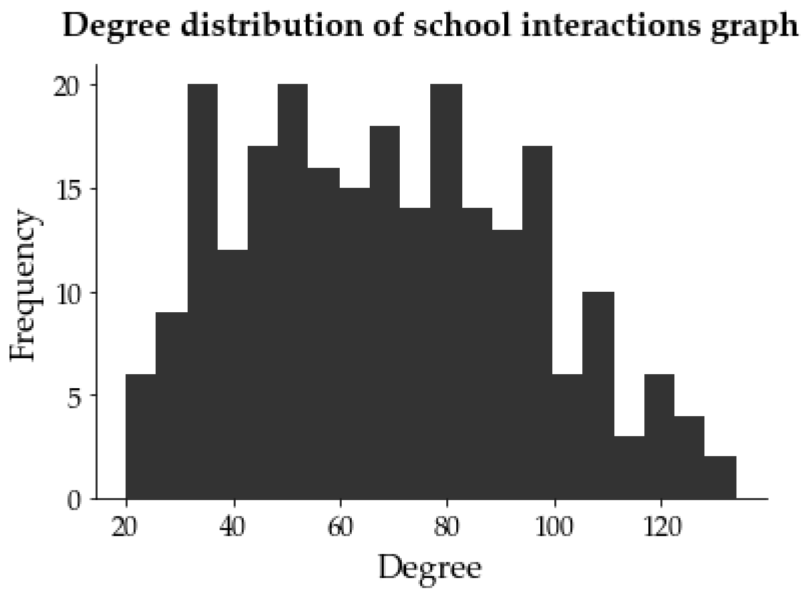
In [ ]:

```
# Generic function for plotting a distribution histogram
def analyze(data, name):
    plt.hist(data, bins=20)
    plt.title('{} distribution of school interactions graph'.format(name.capitalize()))
    plt.xlabel(name.capitalize())
    plt.ylabel('Frequency')
    print('Average {} of G: {}'.format(name, np.mean(data)))
```

In [ ]:

```
# Plot degree distribution
degrees = [G.degree(u) for u in G.nodes]
analyze(degrees, 'degree')
```

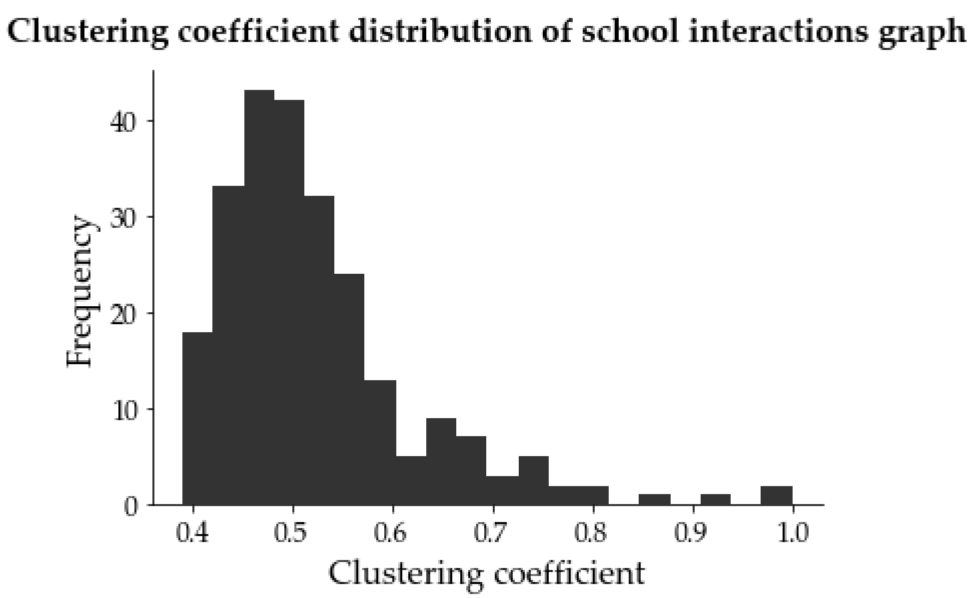
```
Average degree of G: 68.73553719008264
```



In [ ]:

```
# Plot clustering coefficient distribution
clustering_coeffs = list(nx.algorithms.cluster.clustering(G).values())
analyze(clustering_coeffs, 'clustering coefficient')
```

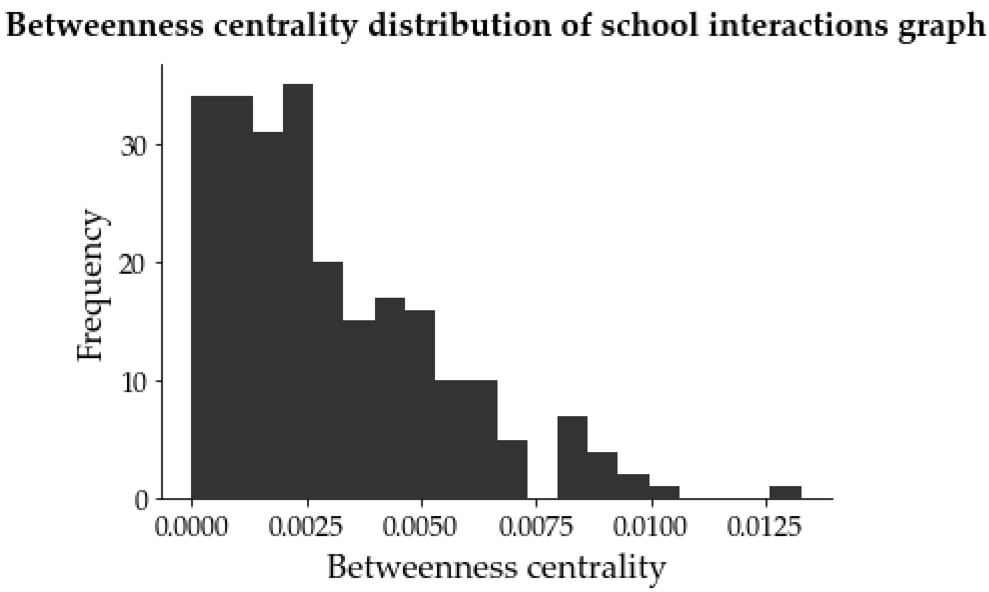
```
Average clustering coefficient of G: 0.5255415410620273
```



In [ ]:

```
# Plot betweenness distribution
betweenness = list(nx.algorithms.centrality.betweenness_centrality(G).values())
analyze(betweenness, 'betweenness centrality')
```

```
Average betweenness centrality of G: 0.00305187865070928
```



Problem 1.4.3: Plotting the graph

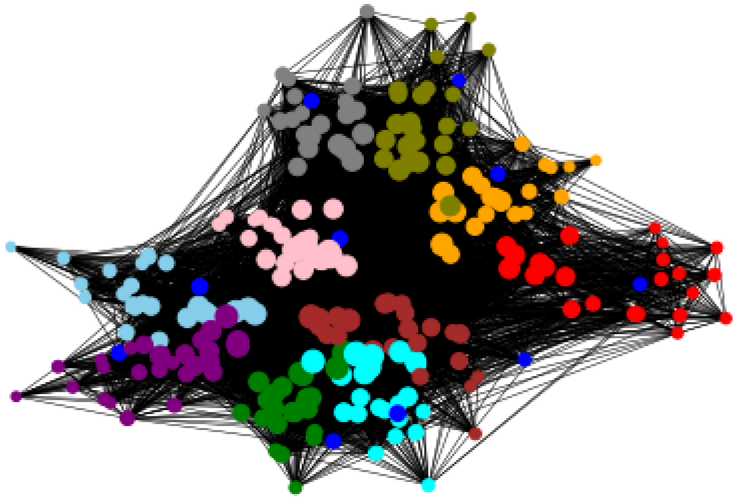
In [ ]:

```
# Get the class of each node + class sizes
node_classes = [G.nodes[n]['c'] for n in G.nodes]
_, class_counts = np.unique(node_classes, return_counts=True)

# map classes to colors for drawing
colors = ['red', 'orange', 'green', 'blue', 'purple', 'pink', 'brown', 'cyan', 'gray',
class_id_map = {cl: i for i, cl in enumerate(set(node_classes))}

node_colors = [colors[class_id_map[c]] for c in node_classes]

# size nodes by degree
node_sizes = [G.degree(u) for u in G.nodes]
```

In [ ]:

```
# Draw the graph
nx.draw(
    G,
    pos=nx.spring_layout(G),
    node_size=node_sizes,
    node_color=node_colors,
    width=0.5)

print('Color key:')
[print(cl + ': ' + colors[i] + ', {} members'.format(ct)) for (cl, i), ct in zip(sorted
```

```
Color key:
1A: brown, 23 members
1B: pink, 25 members
2A: skyblue, 23 members
2B: purple, 26 members
3A: green, 23 members
3B: cyan, 22 members
4A: orange, 21 members
4B: red, 23 members
5A: gray, 22 members
5B: olive, 24 members
Teachers: blue, 10 members
```



Typically, larger nodes (i.e. those with higher degree) seem to be closer to the center of the graph, and closer to other nodes. This makes sense, since spring layout treats more central nodes as harder to "repel" from the center of the graph, and higher-degree nodes tend to be better-connected to others in the graph. Also, the nodes generally divide themselves into communities by color (i.e. class), and each class community contains one teacher (dark blue) node nearby. Also, some classes seem to interact with others more: the brown, light blue, and green classes are positioned close together and have many large nodes, meaning that they tend to interact more (maybe their classrooms are nearby), while the red class has many small, somewhat isolated nodes, making it appear as if they interact with others less (maybe they're in a temporary building or something). I'd predict that if I were to run a community detection algorithm on this graph with k=10, it would find 10 communities that are relatively class-homogeneous, with one teacher node per community.