Exercise A

Document D taken from a collection C.

Document D: "The University of California, Riverside is one of 10 universities within the prestigious University of California system, and the only UC located in Inland Southern California. Widely recognized as one of the most ethnically diverse research universities in the nation."

Collection C statistics:

- $|C| = 10,000$
- $|c \in C : \text{"Riverside"}| = 100$
- $|c \in C : \text{"university OR universities"}| = 500$
- $|c \in C : \text{"diverse"}| = 250$

Q_1: university Riverside

(1) BM25 score:

$$
\begin{aligned}
\text{BM25}(Q_1, D) &= \log \frac{(r_1 + 0.5)/(R - r_1 + 0.5)}{(n_1 - r_1 + 0.5)/(N - n_1 - R + r_1 + 0.5)} \frac{(k_1+1)f_1}{K + f_1} \frac{(k_2+1)qf_1}{k_2 + qf_1} \\
&+ \log \frac{(r_2 + 0.5)/(R - r_2 + 0.5)}{(n_2 - r_2 + 0.5)/(N - n_2 - R + r_2 + 0.5)} \frac{(k_1+1)f_2}{K + f_2} \frac{(k_2+1)qf_2}{k_2 + qf_2} \\
&= \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(500 - 0 + 0.5)/(10,000 - 500 - 0 + 0 + 0.5)} \frac{(1.2+1) * 4}{1.11 + 4} \frac{(100+1) * 1}{100 + 1} \\
&+ \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(100 - 0 + 0.5)/(10,000 - 100 - 0 + 0 + 0.5)} \frac{(1.2+1) * 1}{1.11 + 1} \frac{(100+1) * 1}{100 + 1} \\
&= \log \frac{1}{500.5/9,500.5} \frac{8.8}{5.11} \frac{101}{101} + \log \frac{1}{100.5/9,900.5} \frac{2.2}{2.11} \frac{101}{101} \\
&\approx 5.069 + 4.786 \\
&\approx 9.855
\end{aligned} \tag{1}
$$

Assumptions: $R = r_1 = r_2 = 0$, document length is 90% of average document length, $k_1 = 1.2, b = 0.75, k_2 = 100$ and $K = 1.2(0.25 + 0.75 * 9) = 1.11$.

(2) ULM score: $|D| = 40$ with $|\text{university} \in D| = 4$ and $|\text{Riverside} \in D| = 1$. We use Jelinek-Mercer smoothing with $\lambda = 0.1$. Then:

$$P(Q_1|D) = ((1-\lambda)\frac{f_{q_1}}{|D|} + \lambda\frac{c_{q_1}}{|C|})((1-\lambda)\frac{f_{q_2}}{|D|} + \lambda\frac{c_{q_2}}{|C|})$$

$$= (\frac{9}{10}\frac{4}{40} + \frac{1}{10}\frac{500}{10,000})(\frac{9}{10}\frac{1}{40} + \frac{1}{10}\frac{100}{10,000})$$

$$= (\frac{9}{100} + \frac{1}{200})(\frac{9}{400} + \frac{1}{1,000})$$

$$= (\frac{1900}{20,000})(\frac{9,400}{400,000}) = \frac{17,860,000}{8,000,000,000} = \frac{893}{400,000} = .0022325$$

Q_2: diverse university

(1) BM25 score:

$$
\begin{aligned}
\text{BM25}(Q_2, D) &= \log \frac{(r_1 + 0.5)/(R - r_1 + 0.5)}{(n_1 - r_1 + 0.5)/(N - n_1 - R + r_1 + 0.5)} \frac{(k_1+1)f_1}{K + f_1} \frac{(k_2+1)qf_1}{k_2 + qf_1} \\
&+ \log \frac{(r_2 + 0.5)/(R - r_2 + 0.5)}{(n_2 - r_2 + 0.5)/(N - n_2 - R + r_2 + 0.5)} \frac{(k_1+1)f_2}{K + f_2} \frac{(k_2+1)qf_2}{k_2 + qf_2} \\
&= \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(250 - 0 + 0.5)/(10,000 - 250 - 0 + 0 + 0.5)} \frac{(1.2+1)*1}{1.11 + 1} \frac{(100+1)*1}{100+1} \\
&+ \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(500 - 0 + 0.5)/(10,000 - 500 - 0 + 0 + 0.5)} \frac{(1.2+1)*4}{1.11 + 4} \frac{(100+1)*1}{100+1} \\
&= \log \frac{1}{250.5/9,750.5} \frac{2.2}{2.11} \frac{101}{101} + \log \frac{1}{500.5/9,500.5} \frac{8.8}{5.11} \frac{101}{101} \\
&\approx 3.818 + 5.069 \\
&\approx 8.887
\end{aligned}
$$

(2)

Assumptions: $R = r_1 = r_2 = 0$, document length is 90% of average document length, $k_1 = 1.2, b = 0.75, k_2 = 100$ and $K = 1.2(0.25 + 0.75 * 9) = 1.11$.

(2) ULM score: $|D| = 40$ with $|\text{diverse} \in D| = 1$ and $|\text{university} \in D| = 4$. We use Jelinek-Mercer smoothing with $\lambda = 0.1$. Then:

$$P(Q_2|D) = ((1-\lambda)\frac{f_{q_1}}{|D|} + \lambda\frac{c_{q_1}}{|C|})((1-\lambda)\frac{f_{q_2}}{|D|} + \lambda\frac{c_{q_2}}{|C|})$$

$$= (\frac{9}{10}\frac{1}{40} + \frac{1}{10}\frac{250}{10,000})(\frac{9}{10}\frac{4}{40} + \frac{1}{10}\frac{500}{10,000})$$

$$= (\frac{9}{400} + \frac{1}{400})(\frac{9}{100} + \frac{1}{200})$$

$$= (\frac{1}{40})(\frac{1,900}{20,000}) = \frac{1,900}{800,000} = \frac{19}{8,000} = .002375$$

Exercise B: Write a program to compute the pagerank of a graph.

In [436]:
```
import numpy as np
import pandas as pd
```

In [344]:
```
# The input for the program will be the adjacency matrix M of the graph.
# We will iterate the equation r=(1-d)/N +dMr, where r is the column of page ranks, d=0.85 is the damping factor,
# and N is the number of rows/columns of M, until the change in each component of r is < epsilon=.01.
```

In [437]:
```
# Initialize adjacency matrix M of the graph.
M=np.matrix([[0,0,0,1,0],
             [1,0,0.5,0,0],
             [0,0.5,0,0,0],
             [0,0.5,0,0,0],
             [0,0,0.5,0,0]])
# Initialize d=0.85.
d=0.85
# Initialize all pageranks as 1.
N=np.shape(M)[0]
P=np.array([1,1,1,1,1])
```

In [447]:
```
P=np.matrix([1,1,1,1,1])
def PageRank(M,T):
    P=np.matrix([1,1,1,1,1])
    thresh=1
    while thresh>T:
        Q=(1-d)/N+d*(np.inner(M,P))
        rows,cols = P.shape
        thresh=max(abs(P[0,0]-Q[0]),abs(P[0,1]-Q[1]),abs(P[0,2]-Q[2]),abs(P[0,3]-Q[3]),abs(P[0,4]-Q[4]))
        S=np.matrix([float(Q[0]),float(Q[1]),float(Q[2]),float(Q[3]),float(Q[4])])
        P=pd.DataFrame(P)
        S=pd.DataFrame(S)
        for r in range(rows):
            for c in range(cols):
                P[c][r]=S[c][r]
        P=np.matrix(P)
        S=np.matrix(S)
        print(S,thresh)
```

In [448]:
```
PageRank(M,.01)
```

```
[[0.88  1.305 0.455 0.455 0.455]] [[0.545]]
[[0.41675  0.971375 0.584625 0.584625 0.223375]] [[0.46325]]
[[0.52693125 0.63270312 0.44283438 0.44283438 0.27846562]] [[0.33867188]]
[[0.40640922 0.66609617 0.29889883 0.29889883 0.21820461]] [[0.14393555]]
[[0.284064   0.50247984 0.31309087 0.31309087 0.157032  ]] [[0.16361633]]
[[0.29612724 0.40451802 0.24355393 0.24355393 0.16306362]] [[0.09796181]]
[[0.23702084 0.38521858 0.20192016 0.20192016 0.13351042]] [[0.0591064]]
[[0.20163214 0.31728378 0.1937179  0.1937179  0.11581607]] [[0.06793479]]
[[0.19466021 0.28371742 0.16484561 0.16484561 0.11233011]] [[0.03356636]]
[[0.17011877 0.26552056 0.1505799  0.1505799  0.10005938]] [[0.02454144]]
[[0.15799292 0.23859741 0.14284624 0.14284624 0.09399646]] [[0.02692315]]
[[0.1514193  0.22500363 0.1314039  0.1314039  0.09070965]] [[0.01359378]]
[[0.14169331 0.21455307 0.12562654 0.12562654 0.08584666]] [[0.01045057]]
[[0.13678256 0.2038306  0.12118505 0.12118505 0.08339128]] [[0.01072247]]
[[0.13300729 0.19776883 0.116628   0.116628   0.08150365]] [[0.00606177]]
```

```
C:\Users\Kenny\AppData\Local\Temp\ipykernel_13120\421010696.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  P[c][r]=S[c][r]
```

15 iterations to get epsilon <.01

Exercise C Thought process:

```
In order to compute pointwise mutual information for words a and b, need to calculate:
P(a)
P(b)
P(a,b)

In order to calculate P(a) and/or P(b), need to know:
Total number of words in collection
Number of occurances of words a and/or b in collection

In order to calculate P(a,b), need to know:
Total number of bigrams in collection
Number of occurances of bigram (a,b) in collection

So, from each document, we need:
Individual count of each distinct word in document
Individual count of each distinct bigram in document

Each reducer takes:
A SINGLE bigram (a,b)
Total number of words in each document read by every mapper
Individual count of words a and b (so now at this stage the reducer can compute P(a) and P(b))
Total number of bigrams in each document read by every mapper
Number of occurances of bigram (a,b) in each document read by every mapper (so now at this stage the reducer and compute P(a,b))
Compute P=P(a,b)/(P(a)P(b))
If P>T, emit ((a,b),P)
```

Pseudocode:

```
procedureObtainWordAndBigramCountsInDocuments(input)
        while not input.done() do
                document <-- input.next()
                tokens <-- Parse(document)                    # obtain number of each distinct word in document as well as number of each distinct bigram in document
                for every distinct bigram (a,b) in document:
                        |a|=number of occurances of "a" in document
                        |b|=number of occurances of "b" in document
                          n=number of words in document
                        |(a,b)|=number of occurances of bigram (a,b) in document
                          m=number of bigrams in document
                         emit((a,b),(|a|,|b|,n,|(a,b)|,m))    # these are our (key,value) pairs, the bigram (a,b) is the key
                end for
        end while
end procedure

procedureCheckForUnseenWords(input)
        for all bigrams/reducers (a,b):
                if there exists another bigram/reducer (c,d) != (a,b) such that c=a or b OR d= a or b   # bigrams (a,b) and (c,d) have at least one word in common
                        compare inputs of reducer(a,b) and reducer(c,d)
                        for any input read by reducer(c,d) and NOT read by reducer(a,b)
                                combine inputs by
                                        |a|=|a1|+|a2| if a is the common word
                                        |b|=|b1|+|b2| if b is the common word
                                        n=n1+n2
                                        m=m1+m2
                                emit((a,b),(|a|,|b|,n,|(a,b)|,m))
                        end for
                end if
        end for
end procedure
```

Assuming we have a reducer for every distinct bigram (a,b) in the input collection.

```
procedureCalculatePointwiseMutualInformation(input)
        while not input.done() do
                A = sum of all |a|s from every mapper
                B = sum of all |b|s from every mapper
                N = sum of all ns from every mapper
                P(a)=A/N
                P(b)=B/N
                AB = sum of all |(a,b)|s from every mapper
                M = sum of all ms from every mapper
                P(a,b)=AB/M
                P=P(a,b)/(P(a)P(b))                          # calculate pointwise mutual information of bigram (a,b)
                if P>T
                        emit((a,b),P)
                end if
        end while
end procedure
```

If multiple mappers are connected to a combiner, the combiner can save the reducer the work of having to sum up the |a|s, |b|s, ns, |(a,b)|s and ms.

Exercise D

7 relevant results in a collection. Search engine returns:

$$r\ r\ x\ x\ x\ x\ r\ x\ x\ r$$

where r is relevant and x is not relevant.

Precision-at-5: 2/5=.4

Recall-at-5: $2/7 \approx .29$

F1-at-5:

$$\frac{(\beta^2+1)RP}{R+\beta^2 P} = \frac{(1^2+1)(2/7)(2/5)}{(2/7)+1^2(2/5)} = \frac{8/35}{24/35} = \frac{1}{3} \approx 0.33$$

Average Precision:

$$(1/7)\sum_{k=1}^{7}(\text{precision at } k) = (1/7)(1+1+(3/7)+(4/10)+0+0+0) = 198/490 \approx 0.40$$

DCG-at-5: $1 + 1/(\log_2 2) = 2$

2. In testing patients for cancer, higher recall is more important than higher precision because a false positive diagnosis is not nearly as bad as a false negative diagnosis. Or, for a search-engine-related result, I will steal the innovation patent example from lecture: higher recall is more important here because you need to check every result to see if somebody has already patented your innovation. It only takes one positive result for the patent to be dismissed.

For an example where you would want higher precision, and I apologize if this is too morbid, but the certainty of guilt of inmates on Death Row. A false positive in this instance is executing someone who should not be put to death. Or, for a search-engine-related result, again stealing from lecture, precision is more important for query searches because most people will not dig through hundreds or more of results to find what they are looking for. If it's not near the top, they will give up and try a new query.