The Page Rank Algorithm
ML_2022: Machine Learning

https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/pagerank_lab/pagerank_lab.pdf

Like a blind judge at a beauty contest, a page rank algorithm must rank pages without reading them

The Page Rank Problem

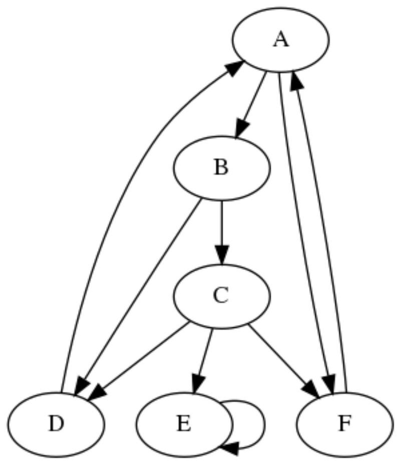
Several algorithms are available for "ranking" web pages.

- *These algorithms treat the Internet as a directed graph of web pages.*
- *The algorithm cannot read or understand the text of the page;*
- *The algorithm treats each page as a node, having "in-links" and "out-links".*
- *The "in-links" are a sort of vote for this page. the "out-links" are votes by this page for other pages.*
- *The "in-links" make a page seem important. the "out-links" transfer that importance to other pages.*
- *By imagining a sort of flow of importance around the network, it's possible to rank the web pages automatically, without reading or understanding them.*

1 Prepare for Exercise 1:

The easiest version of a page rank algorithm simply takes the network as it is, computes the incidence matrix A, then the transition matrix T, and applies the power method over and over until it converges to a ranking vector r.

Make sure you remember how to determine the incidence matrix A for a given directed graph. For this exercise, we will look at the Moler network, of 5 nodes. Notice that one of the nodes has a "self link". How do we record this in the incidence matrix?



The Moler network.

Recall the definition of the transition matrix T, and how it is formed from the incidence matrix. You will need to compute T for the Moler network.

Recall the simple form of the power method that is appropriate for our network problems:

```
For K from 0 to KMAX
  if K is 0, initialize x to a random N-vector, and set x = x/||x||
  else xnew = T * x
  print K and ||xnew - x||
  x = xnew
return x
```

2 Exercise 1:

Create a file `exercise1.py`.

1. Define the matrix A as the incidence matrix of the Moler network;
2. Compute the matrix T as the transition matrix associated with A;
3. Carry out the power method;
4. Print x, and the product $T * x$;
5. If your power method is correct, and has converged, then x and $T * x$ should be approximately equal.

Save a copy of your results, because we will want to compare them with the results from our later exercises. Your result from this exercise might be surprising. Can you explain how it makes sense?

3 Prepare for Exercise 2:

Recall the Google PageRank algorithm. This algorithm can be used as another way to rank a set of web pages which are linked together in some way. The PageRank algorithm begins with the incidence matrix, but modifies it in a simple way, which we could think of as suggesting that for every web page, not only might we consider all the official links to other pages, but there is a random chance that we will simply jump to another page because we are bored with our current search.

Maybe we are doing a better job of modeling how people use the Internet, but more importantly, now all the web pages on the Internet are connected; it is always possible to get from one page to any other. This means our results will be somewhat different from the power method results.

4 Exercise 2:

Write a code `exercise2.py` for this experiment.

1. As before, define the matrix A as the incidence matrix of the Moler network;
2. Compute the matrix G as the Google matrix associated with A; Use the "random jump" probability $p = 0.15$.
3. Carry out the Google page rank algorithm;
4. Print x, and the product $G * x$;
5. If your algorithm is correct, and has converged, then x and $G * x$ should be approximately equal.

Again, save a copy of your results. You should notice that the Google PageRank rankings are quite different from what we got with the power method. For the power method, it seemed like all the importance flowed into node E and never escaped. What real-life feature of Internet browsing is the PageRank algorithm modeling, which means that the flow doesn't simply drain into node E?

5 Prepare for Exercise 3:

Another model of Internet browsing uses the "random surfer". In this model, we imagine that a user starts at a random page; after reading it, the user goes to a new page, which is selected at random from the out-links available on the page, or with a 15 percent probability, simply jumps to a new random page. If a page doesn't have any out-links, then the user just jumps to a random new page immediately.

We want to count how many times each page is visited. We create a vector r, with an entry for each page, initialized to 0. Now, as the user visits web page i, we need to increment $r(i) \leftarrow r(i) + 1$. We want to do a lot of iterations of this procedure, so that every page has a chance to be viewed multiple times. If there are n pages, then we might start by taking $10 * n$ such steps, or maybe $100 * n$ steps.

Once our browsing is finished, we compute $rsum$, the sum of all the r values, and divide r by $rsum$ to get a final ranking for each page.

The procedure might look something like this:

```
For K from 0 to KMAX
  if K is 0
    choose 0 <= i < n at random
  else
    set L = number of outlinks of page i
    if L == 0 or random number <= 0.15
      jump to random page 0 <= i < n
    else
      jump to new page i = link(j) 0 <= j < L
  r(i) = r(i+1)
r = r / sum(r)
return r
```

6 Exercise 3:

Create a file *exercise3.py* for this experiment.

1. As before, define the matrix A as the incidence matrix of the Moler network;
2. Apply the random surfer algorithm.
3. If you are at page i, and you need to count the number of outlinks, this is simply

```
    links = np.sum ( A[i,:] )
```

4. If you are at page i, and you need to randomly select a linked page:

```
    v = np.random.rand ( )
    w = 0.0
    for j in range ( 0, n ):
      w = w + A[i,j] / links
      if ( v <= w ):
        i = j
        break
```

If your algorithm is correct, and you have taken a large enough number of steps, then your ranking vector r should be close to the result from Google PageRank.

7 Prepare for Exercise 4:

An alternate method of ranking web pages is known as the "HITS algorithm" The HITS algorithm for the nodes in a directed network computes two vectors of node ratings:

- **a**, the authority index, the degree to which a node is pointed to by important (authoritative) nodes;
- **h**, the hub index, the degree to which a node points to important (authoritative) nodes;

If A is the incidence matrix of the network, then the two vectors will satisfy the relations:

$$a = \frac{A' * h}{||A' * h||}$$
$$h = \frac{A * a}{||A * a||}$$

The HITS algorithm estimates the vectors a and h for a directed network of n nodes with adjacency matrix A:

```
for 0 <= k < KMAX

  if ( k == 0 )
    Initialize n-vector a to 1's
    Initialize n-vector h to 1's
  else
    for 0 <= j < n
      a(j) = sum of h(i)'s for which node j is pointed to by node i
    for 0 <= i < n
      h(i) = sum of a(j)'s for which node i points to node j

  a = a / ||a||
  h = h / ||h||
```

To be clear, node j is pointed to by node i if $A[i,j] = 1$.

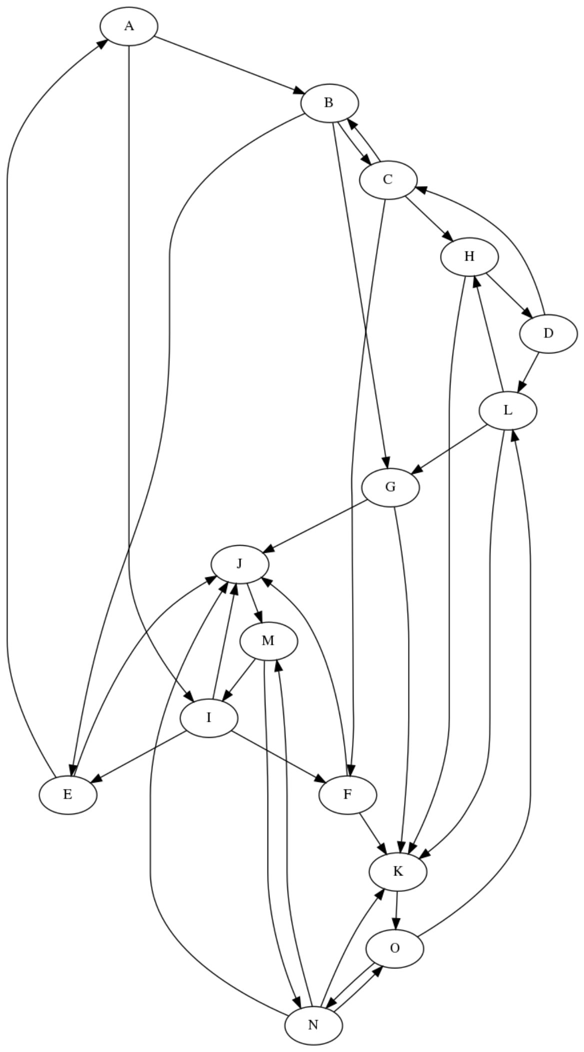
An alternative version of the HITS algorithm uses the SVD factorization of the incidence matrix, which Python reports as $A = U * S * V$. You can compute this factorization with the command

```
U, svec, V = np.linalg.svd ( A )
```

(Since the matrix S is diagonal, Python only returns the diagonal vector `svec`.)

Once we compute this factorization, it turns out that the HITS vector a is the first column of V, and h is the first column of U.

For this exercise, we will **not** use the 6 node Moler incidence matrix. Because it has a "sink" node, the HITS algorithm doesn't provide interesting information. Instead, we will need the 15 node Sauer incidence matrix. Download a copy of the function `sauer_incidence()` before beginning the next exercise.



The Sauer network.

8 Exercise 4:

Create a file *exercise4.py* for this experiment.

1. Define your network by the command `A = sauer_incidence();`
2. Apply the HITS algorithm to get the authority and hub vectors for A.
3. Report your results as $a1$ and $h1$.

4. Apply the SVD version of the HITS algorithm to A.
5. Report your results as $a2$ and $h2$.

You should expect that the two sets of HITS vectors correspond closely.