```
# Using numpy for functionality convenience, i.e. matrix mult
import numpy as np

# Setting printing options to confirm data is correctly set
np.set_printoptions(precision=2, linewidth=100)
```

```
# Reading all lines for links.txt into variable 'lines'. Each line is an element of the list.
with open('/content/links.txt') as f:
    lines = f.readlines()
```

```
# Redefine 'lines', now cleaning the read strings and converting each element to an intenger
lines = [[int(k) for k in l.replace("\n","").split(" ")] for l in lines]
```

```
# Extracting the number of pages and links
num_pages, num_links = lines[0]

# Creating the A array with all zeros and (number of pages) by (number of pages) shape.
A = np.zeros((num_pages,num_pages))

# Creating the Nk array with all zeros and 1 by (number of pages) shape (to allow proper broadcasting).
# This array contains the number of outgoing links of k.
Nk = np.zeros((1,num_pages))

# For loop to fill in A with 1s on respective positions and count elements into Nk
for i,j in lines[1:]:
  A[j-1,i-1] = 1
  Nk[0,i-1] = Nk[0,i-1]+1

# Devide each column k of A by the frequency ot outgoing links of k
A = A/Nk

print(num_pages, num_links)
print(A)
print(Nk)
```

```
15 34
[[0.   0.   0.   0.   0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.5  0.   0.33 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.33 0.   0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.5  0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.33 0.   0.   0.   0.   0.   0.   0.33 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.33 0.   0.   0.   0.   0.   0.33 0.   0.   0.   0.   0.   0.  ]
 [0.   0.33 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.33 0.   0.   0.  ]
 [0.   0.   0.33 0.   0.   0.   0.   0.   0.   0.   0.   0.33 0.   0.   0.  ]
 [0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.5  0.   0.  ]
 [0.   0.   0.   0.   0.5  0.5  0.5  0.   0.33 0.   0.   0.   0.   0.25 0.  ]
 [0.   0.   0.   0.   0.   0.5  0.5  0.5  0.   0.   0.   0.33 0.   0.25 0.  ]
 [0.   0.   0.   0.5  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.5 ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.5  0.   0.5 ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.25 0.  ]]
[[2. 3. 3. 2. 2. 2. 2. 2. 3. 1. 1. 3. 2. 4. 2.]]
```

```
# Setting given value for mniu and initializing e
mniu = 0.15
e = np.ones((num_pages,1))

# Initializing xk (first iteration input) and setting stopping error threshold
xk = e/num_pages
error = 1e-8

# Initialize a counter
k = 0

# Perform power iteration loop:
# compute xk_1, calculate the norm of the difference between xk_1 and xk
# set xk to xk_1 and increment counter k
# if the norm is lower than error, then break while cycle
while(True):
  xk_1 = (1-mniu)*A@xk + (mniu/num_pages)*e
  diff = np.linalg.norm(xk_1-xk)
  xk = xk_1
  k = k+1
  if diff < error:
    break

```

After how many iterations does this happen?

```
print("Total number of iterations: ",k)
```

```
Total number of iterations:  36
```

What is the resulting ranking of the web pages?

```
# Extract the indexes to sort the rankings
# Flip the result so the first element is the highest ranked
# Broadcast 1 unit to correspond to the exercise page numbering
ranking_page_number = np.flip(np.argsort(xk.reshape(-1)))+1

# Same ideia as before but just to extract the ranks
sorted_rankings = np.flip(np.sort(xk.reshape(-1)))

# For each page/rank, print it according its ranking position
print("Page number and ranking, from highest to lowest:")
for i,(p,r) in enumerate(zip(ranking_page_number,sorted_rankings),1):
  print(f"{i} -> page {p} with rank {r:.4f}")
```

```
Page number and ranking, from highest to lowest:
1 -> page 15 with rank 0.1251
2 -> page 13 with rank 0.1251
3 -> page 14 with rank 0.1163
4 -> page 11 with rank 0.1063
5 -> page 10 with rank 0.1063
6 -> page 12 with rank 0.0746
7 -> page 9 with rank 0.0746
8 -> page 8 with rank 0.0396
9 -> page 7 with rank 0.0396
10 -> page 6 with rank 0.0396
11 -> page 5 with rank 0.0396
12 -> page 3 with rank 0.0299
13 -> page 2 with rank 0.0299
14 -> page 4 with rank 0.0268
15 -> page 1 with rank 0.0268
```

How many iterations does it take to get to this ranking?

```
# Lets set a variable for the sort indexes order, run the algorithm again and notice on which k this happens
# The setup will be the same as before, except we compare the result form the iteration and look for
# our final convergance rank setup.
rank_setup = np.argsort(xk.reshape(-1))

xk_new = e/num_pages
k = 0

# This list will save all iteration indexes where the ranking setup converges to the same as the final
setup_convergance = []

while(True):
  xk_1 = (1-mniu)*A@xk_new + (mniu/num_pages)*e
  diff = np.linalg.norm(xk_1-xk_new)
  xk_new = xk_1
  k = k+1
  if np.array_equal(np.argsort(xk_new.reshape(-1)),rank_setup):
    setup_convergance.append(k)
  if diff < error:
    break

print(setup_convergance)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
```

We can see that the rank setup converges in the 5th iteration and stays like that until the end, on iteration 36.

How does the result change if page 14 adds a links to itself?

```
# Lets repeat the same steps as before, but add an element to the list of lines
# All of the following code has been explained in previous sections. The only difference lies
# in the addition of [14,14] to our initial list
lines2 = lines
lines2.append([14,14])

num_pages, num_links = lines2[0]
A = np.zeros((num_pages,num_pages))
Nk = np.zeros((1,num_pages))

for i,j in lines2[1:]:
```

```
  A[j-1,i-1] = 1
  Nk[0,i-1] = Nk[0,i-1]+1

A = A/Nk

mniu = 0.15
e = np.ones((num_pages,1))
xk = e/num_pages
error = 1e-8
k = 0

while(True):
  xk_1 = (1-mniu)*A@xk + (mniu/num_pages)*e
  diff = np.linalg.norm(xk_1-xk)
  xk = xk_1
  k = k+1
  if diff < error:
    break
```

```
print("Total number of iterations: ",k)
```

```
Total number of iterations:  38
```

```
ranking_page_number = np.flip(np.argsort(xk.reshape(-1)))+1
sorted_rankings = np.flip(np.sort(xk.reshape(-1)))

print("Page number and ranking, from highest to lowest:")
for i,(p,r) in enumerate(zip(ranking_page_number,sorted_rankings),1):
  print(f"{i} -> page {p} with rank {r:.4f}")
```

```
Page number and ranking, from highest to lowest:
1 -> page 14 with rank 0.1361
2 -> page 15 with rank 0.1212
3 -> page 13 with rank 0.1212
4 -> page 11 with rank 0.1036
5 -> page 10 with rank 0.1036
6 -> page 12 with rank 0.0728
7 -> page 9 with rank 0.0728
8 -> page 8 with rank 0.0390
9 -> page 7 with rank 0.0390
10 -> page 6 with rank 0.0390
11 -> page 5 with rank 0.0390
12 -> page 3 with rank 0.0297
13 -> page 2 with rank 0.0297
14 -> page 4 with rank 0.0266
15 -> page 1 with rank 0.0266
```

- We can see that the algorithm takes 38 steps, oposed to 36, to reach the stopping codition.
- Page 14 gets ranked to first place, the relative order of the others remain the same
- The overall ranks decrease, except of page 14.