Crawl data with requests and bs4

```
#get raw page markup with requests
resp = requests.get(url)

#parse raw to html with BeautifulSoup
soup = BeautifulSoup(resp.content, "html.parser")

#select element with selector
links = soup.select('a[href^="/"], a[href^="https://demo.org"]')
for link in links:
  successor = link['href'] #value of href attribute
```

Analyze

OutDegree and DeadEnd

OutDegree is the number of out-link from a page

DeadEnd is a page that has no out-link

Google ‘98 PageRank Algo

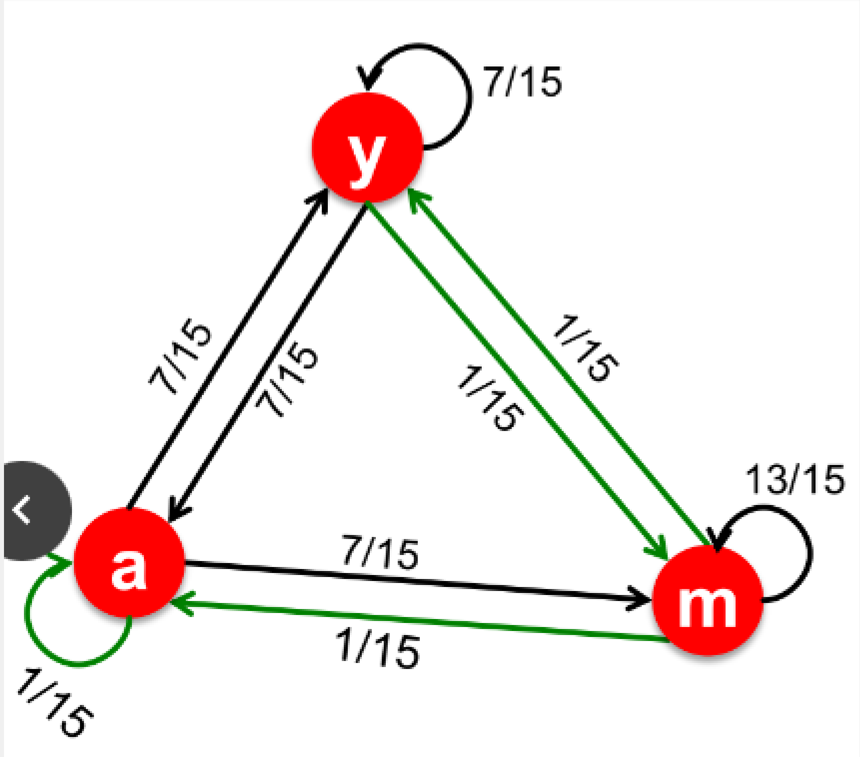
Introduce and how to implement the algorithm

Calculate OutDegree and DeadEnd

```
data = sqlc.sql("\
  SELECT pr.*, od.OutDegree, de.DeadEnds \
  FROM PageRank pr \
  INNER JOIN ( \
    SELECT Page, COUNT(*) as OutDegree \
    FROM PageRank GROUP BY Page\
  ) od ON od.Page = pr.Page \
  INNER JOIN ( \
    SELECT DISTINCT s.Successor, (CASE WHEN p.Page IS NULL THEN 1 ELSE 0 END) AS DeadEnds \
    FROM PageRank s \
    LEFT JOIN PageRank as p ON p.Page = s.Successor \
  ) de ON de.Successor = pr.Successor \
")

data.show()
```

Calculate OutDegree and DeadEnd



Page	Successor
y	y
y	m
y	a
a	a
a	y
a	m
m	m
m	y
m	a

Calculate OutDegree

Page	Successor
y	y
y	m
y	a
a	a
a	y
a	m
m	m
m	y
m	a

```
select Page, count(Successor) as
OutDegree
from df
group by Page
```

Page	OutDegree
y	3
a	3
m	3

Check DeadEnd

Page	Successor
y	y
y	m
y	a
a	a
a	y
a	m
m	m
m	y
m	a

Ideas:

For each distinct element in Successor:
    if(element exists in Page):
        element is not deadend
    else:
        element is deadend

Google '98 PageRank Algo

What is PageRank

Simply pagerank is the likelihood (probability) that one user click on that page

Google '98 Algo

Introduce and how to implement the algorithm

Introduce the algo

- We just rearranged the **PageRank equation**

$$\boldsymbol{r} = \boldsymbol{\beta M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$$

  - where **$[(1-\beta)/N]_N$** is a vector with all ***N*** entries **$(1-\beta)/N$**

- ***M*** is a **sparse matrix!** (with no dead-ends)
  - 10 links per node, approx $10N$ entries
- So in each iteration, we need to:
  - Compute $\boldsymbol{r}^{new} = \beta \boldsymbol{M} \cdot \boldsymbol{r}^{old}$
  - Add a constant value **$(1-\beta)/N$** to each entry in $\boldsymbol{r}^{new}$
    - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

Introduce the algo

- **Input: Graph G and parameter β**
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
    - $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$

      $r'^{new}_j = 0$ if in-degree of j is $\mathbf{0}$
    - **Now re-insert the leaked PageRank:**

      $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-S}{N}$ **where:** $S = \sum_j r'^{new}_j$
    - $r^{old} = r^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is **1-β**. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing **S**.

Implement

Convert data to a graph

Main functions

Create graph - adjacency matrix

Page	Succ
y	y
y	m
y	a
a	a
a	y
a	m
m	m
m	y
m	a

Dict =
{'y': 0,
'a': 1,
'm': 2}

Page	Succ
0	0
0	2
0	1
1	1
1	0
1	2
2	2
2	0
2	1

M[p,s]=1

m[0,0]=1
...
m[1,1]=1

Col j **Row i**	**0**	**1**	**2**
0	1	1	1
1	1	1	1
2	1	1	1

Remember to transpose

Col i **Row j**	**0**	**1**	**2**
0	1	1	1
1	1	1	1
2	1	1	1

Main Functions

- Compute $\boldsymbol{r}^{\text{new}} = \beta \boxed{\boldsymbol{M}} \cdot \boldsymbol{r}^{\text{old}}$
- Add a constant value **(1-β)/N** to each entry in $\boldsymbol{r}^{\text{new}}$
  - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

- **Set:** $r_j^{old} = \frac{1}{N}$
- **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
  - $\forall j$: $\boldsymbol{r'}_j^{new} = \sum_{i \to j} \boldsymbol{\beta} \frac{r_i^{old}}{d_i}$

    $\boldsymbol{r'}_j^{new} = \boldsymbol{0}$ if in-degree of ***j*** is **0**
  - **Now re-insert the leaked PageRank:**

    $\forall \boldsymbol{j}$: $\boldsymbol{r}_j^{new} = \boldsymbol{r'}_j^{new} + \frac{\boldsymbol{1-S}}{\boldsymbol{N}}$ **where:** $S = \sum_j r'^{new}_j$
  - $\boldsymbol{r}^{old} = \boldsymbol{r}^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is 1-β. But since we have

Find M

Col i Row j	0	1	2
0	1	1	1
1	1	1	1
2	1	1	1

3 values != 0

Col i Row j	0	1	2
0	1/3	1/3	1/3
1	1/3	1/3	1/3
2	1/3	1/3	1/3

Col i Row j	0	1	2
0	1	1	1
1	1	1	1
2	0	1	1

2 values != 0

Col i Row j	0	1	2
0	1/2	1/3	1/3
1	1/2	1/3	1/3
2	0	1/3	1/3

Find Matrix M

```
def findM(G, N):
  tmp = [] # tmp === r_tmp_new_j
  for col_i in range(N):

      col = G[:, col_i]

      if(np.sum(col) > 0): #no deadend
        #divide r_i to number of out degree (number of values != 0)
        out_dgs = (col > 0).sum() #total values != 0 in each col
        tmp.append(col/out_dgs)
      else: #deadend
        tmp.append(np.array([0]*N))

  return np.array(tmp).T #stochastic matrix with prob.
```

Main Functions

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

- Compute $r^{new} = \beta M \cdot r^{old}$
- Add a constant value **(1-β)/N** to each entry in r^{new}
  - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

?

Output: PageRank vector r

- **Set:** $r_j^{old} = \frac{1}{N}$
- **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
  - $\forall j$: $\boldsymbol{r'^{new}_j} = \sum_{i \to j} \boldsymbol{\beta} \frac{\boldsymbol{r_i^{old}}}{\boldsymbol{d_i}}$

    $\boldsymbol{r'^{new}_j} = \boldsymbol{0}$ if in-degree of $\boldsymbol{j}$ is **0**
  - **Now re-insert the leaked PageRank:**

    $\forall \boldsymbol{j}$: $\boldsymbol{r_j^{new}} = \boldsymbol{r'^{new}_j} + \frac{\boldsymbol{1-S}}{\boldsymbol{N}}$ **where:** $S = \sum_j r'^{new}_j$
  - $\boldsymbol{r^{old}} = \boldsymbol{r^{new}}$

If the graph has no dead-ends then the amount of leaked PageRank is **1-β**. But since we have c

PageRank Function

```python
def gg_pagerank(G, b, N):
  r_j_old = np.array([ 1/N ]*N).T
  r_j_new = np.array([ 0 ]*N).T

  thresh_hold = 10**-8

  ### stochastic matrix with prob in gg algo
  M = findM(G,N)*b

  ### leaked
  leaked = (1-b)/N

  ### begin iteration
  while np.sum((np.absolute(r_j_new - r_j_old))) >= thresh_hold:

    ###update to exit while
    r_j_old = r_j_new
    r_j_new = M.dot(r_j_old) + leaked

    ###normalized
    if(np.sum(r_j_new.T) < 1):
      tmp = [r_j_new.T[i]/np.sum(r_j_new.T) for i in range(N)]
      r_j_new = np.array(tmp).T

  return r_j_new
```

[a, b, c] -> a + b + c != 1

[a/(a+ b + c), b/(a+b+c), c/(a+b+c)]
-> a/(a+ b + c) + b/(a+b+c) + c/(a+b+c) = 1

[i/sum(arr) for i in arr]

Thanks!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon and infographics & images by Freepik