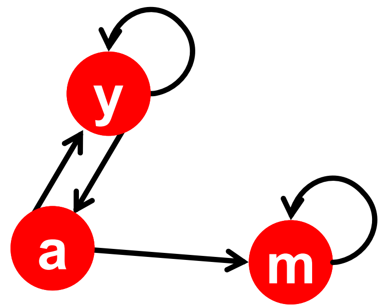
Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1/N$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$$\mathbf{r_y} = \mathbf{r_y}/2 + \mathbf{r_a}/2$$
$$\mathbf{r_a} = \mathbf{r_y}/2$$
$$\mathbf{r_m} = \mathbf{r_a}/2 + \mathbf{r_m}$$

- **Example:**

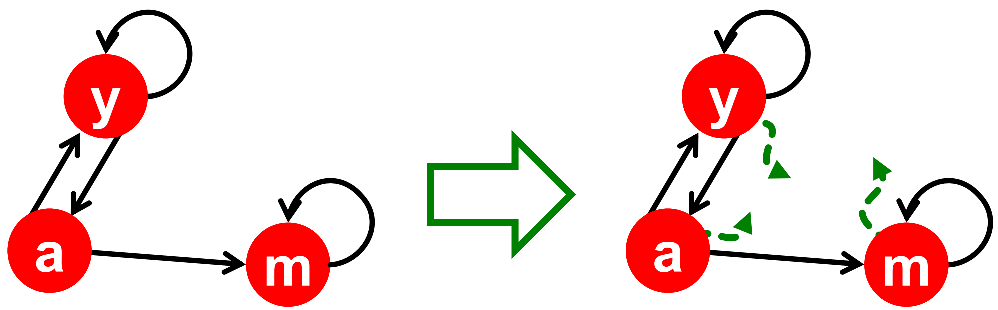
$$\begin{matrix} r_y \\ r_a \\ r_m \end{matrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{matrix} 2/6 \\ 1/6 \\ 3/6 \end{matrix} \quad \begin{matrix} 3/12 \\ 2/12 \\ 7/12 \end{matrix} \quad \begin{matrix} 5/24 \\ 3/24 \\ 16/24 \end{matrix} \quad \cdots \quad \begin{matrix} 0 \\ 0 \\ 1 \end{matrix}$$

Iteration 0, 1, 2, …

All the PageRank score gets "trapped" in node m.

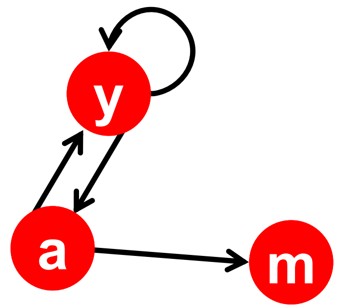
Solution: Teleports(随机跳转)!

- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1/N$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a dead end

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2$$
$$r_m = r_a/2$$

- **Example:**

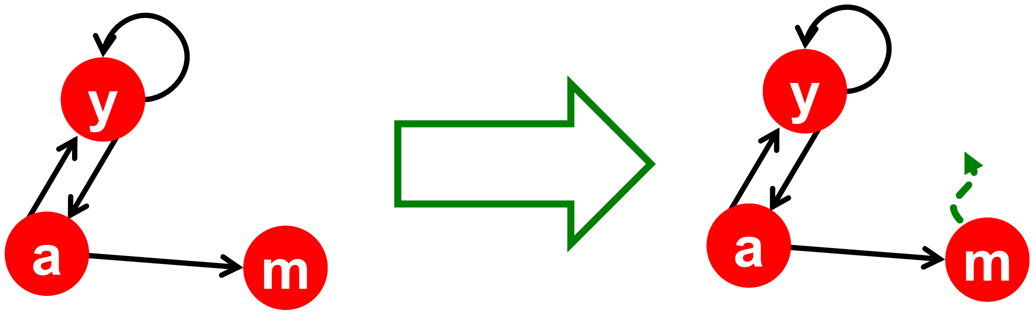
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{matrix} 2/6 \\ 1/6 \\ 1/6 \end{matrix} \quad \begin{matrix} 3/12 \\ 2/12 \\ 1/12 \end{matrix} \quad \begin{matrix} 5/24 \\ 3/24 \\ 2/24 \end{matrix} \quad \cdots \quad \begin{matrix} 0 \\ 0 \\ 0 \end{matrix}$$

Iteration 0, 1, 2, …

Here the PageRank “leaks” out since the matrix is not stochastic.

Solution: Always Teleport!

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why Teleports Solve the Problem?

$$r^{(t+1)} = Mr^t$$

Markov chains

- Set of states **x**
- Transition matrix ***p*** where $\boldsymbol{P}_{ij} = p(x_t = i | x_{t-1} = j)$
- π specifying the stationary probability of being at each state
- Goal is to find π such that π = ***p*** π

Why is This Analogy Useful?

Theory of Markov chains

- Fact: for any start vector, the power method applied to a Markov transition matrix ***p*** will converge to a unique positive stationary vector as long as ***p*** is stochastic(随机的), irreducible(不可约的) and aperiodic(非周期性的).

Why Teleports Solve the Problem?

- Why are dead-ends and spider traps a problem and why do teleports solve the problem?
- Spider-traps are not a problem, but with traps PageRank scores are not what we want
  - Solution: Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- Dead-ends are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - Solution: Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page

- **PageRank equation** [Brin-Page, 98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

d_i ... out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix

- **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

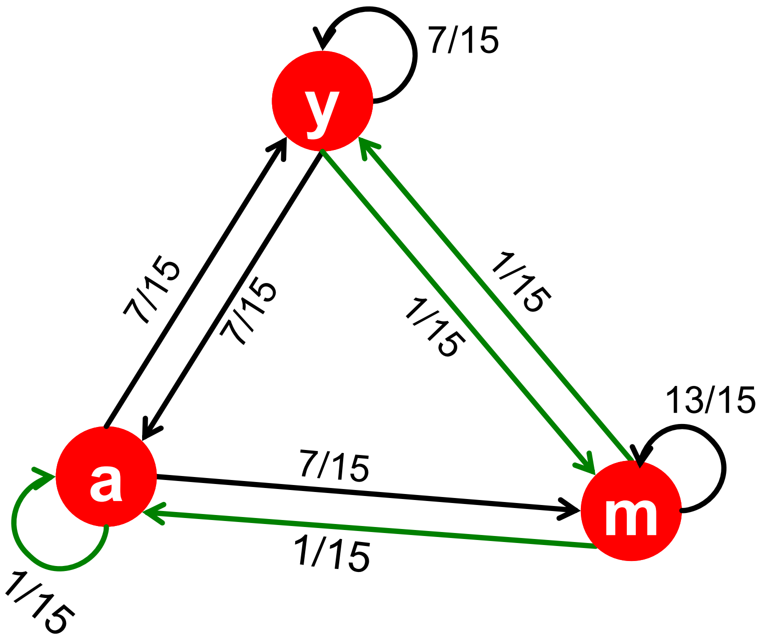
- **The Google Matrix *A*:**

$$A = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N \times N}$$

$[1/N]_{NxN}$…N by N matrix
where all entries are 1/N

- **We have a recursive problem: $\boldsymbol{r = A \cdot r}$ and the Power method still works!**
- **What is β?** In practice β =*0.8,0.9* (make *5* steps on avg., jump)

Random Teleports ($\beta = 0.8$)



$$\overset{\textbf{M}}{0.8\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}} + 0.2\overset{\textbf{[1/N]}_{\textbf{NxN}}}{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}$$

$$\underset{\textbf{A}}{\begin{array}{c} y \\ a \\ m \end{array}\begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}}$$

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{c} 1/3 \\ 1/3 \\ 1/3 \end{array} \quad \begin{array}{c} 0.33 \\ 0.20 \\ 0.46 \end{array} \quad \begin{array}{c} 0.24 \\ 0.20 \\ 0.52 \end{array} \quad \begin{array}{c} 0.26 \\ 0.18 \\ 0.56 \end{array} \quad \ldots \quad \begin{array}{c} 7/33 \\ 5/33 \\ 21/33 \end{array}$$

How do we actually compute the PageRank?

Computing Page Rank

- **Key step is matrix-vector multiplication**
  - $\boldsymbol{r}^{new} = \boldsymbol{A} \cdot \boldsymbol{r}^{old}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$
- **Say N = 1 billion (十亿) pages**
  - We need 4 bytes for each entry (say)
  - $\mathbf{r}^{old}$, $\mathbf{r}^{new}$: 2billion entries for vectors, approx 8GB
  - A: Matrix **A** has N^2 entries
    - 10^{18} is a large number!

$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\,[1/N]_{N \times N}$$

$$\boldsymbol{A} = 0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

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

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

- ***M*** is a **sparse matrix!** (with no dead-ends)
  - N nodes, 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 $\mathbf{(1\text{-}\beta)/N}$ 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 $\boldsymbol{r}^{new}$ so that it sums to 1**

PageRank: The Complete Algorithm

- **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 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**.

Sparse Matrix Encoding

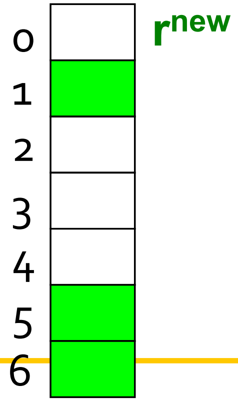
- **Encode sparse matrix using only nonzero entries**
  - Space proportional roughly to number of links
  - Say 10N(N nodes, 10 links per node), or 4*10*1 billion = 40GB
  - **Still won't fit in memory, but will fit on disk**

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

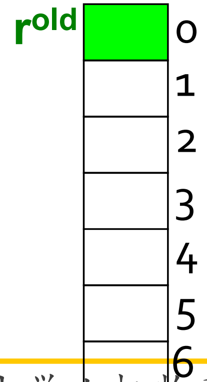
Basic Algorithm: Update Step

- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix **M** on disk
- **1 step of power-iteration is:**

Initialize all entries of $\mathbf{r^{new}} = \mathbf{(1-\beta) / N}$
For each page i (of out-degree d_i):
  Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}$, $r^{old}(i)$
  For $\mathbf{j = 1 \ldots d_i}$
    $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$



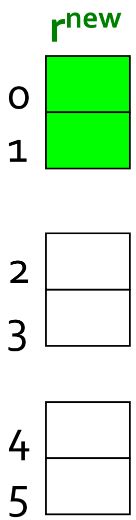
source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



Analysis

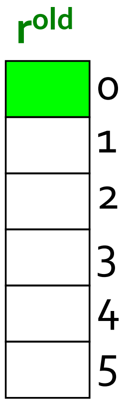
- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix M on disk
- **In each iteration, we have to:**
  - Read r^{old} and M
  - Write r^{new} back to disk
  - **Cost per iteration of Power method:**
    $= 2|r| + |M|$
- **Question:**
  - What if we could not even fit r^{new} in memory?

Block-based Update Algorithm



src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4

M

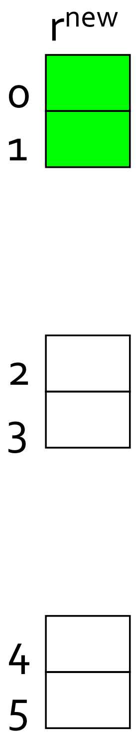


- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

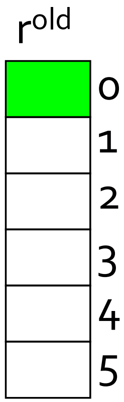
Analysis of Block Update

- **Similar to nested-loop join in databases**
  - Break $\boldsymbol{r}^{\text{new}}$ into $\boldsymbol{k}$ blocks that fit in memory
  - Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{\text{old}}$ once for each block
- **Total cost:**
  - $\boldsymbol{k}$ scans of $\boldsymbol{M}$ and $\boldsymbol{r}^{\text{old}}$
  - **Cost per iteration of Power method:**
    $k(|\boldsymbol{M}| + |\boldsymbol{r}|) + |\boldsymbol{r}| = \boldsymbol{k|M| + (k+1)|r|}$
- **Can we do better?**
  - **Hint:** $\boldsymbol{M}$ is much bigger than $\boldsymbol{r}$ (approx 10-20x), so we must avoid reading it $\boldsymbol{k}$ times per iteration

Block-Stripe Update Algorithm



src	degree	destination
0	4	0, 1
1	3	0
2	2	1
0	4	3
2	2	3
0	4	5
1	3	5
2	2	4



Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of r^{new}

Block-Stripe Analysis

- Break $\boldsymbol{M}$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\boldsymbol{r}^{\text{new}}$
- Some additional overhead per stripe
  - But it is usually worth it
- **Cost per iteration of Power method:**
  $=|\boldsymbol{M}|(1+\varepsilon) + (k+1)|\boldsymbol{r}|$

Some Problems with Page Rank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)
- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities (**next**)
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank (**next**)