1.4.1 Computing PageRank

- **Key step is matrix-vector multiplication**
  - $r^{new} = A \cdot r^{old}$

$$A = \beta \bullet M + (1-\beta)\ [1/N]_{NxN}$$

- **Easy if we have enough main memory to hold A, r^{old}, r^{new}**

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

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

- **Say N = 1 billion (十亿) pages**
  - We need 4 bytes for each entry (say)
  - r^{old}, r^{new} : 2 billion entries for vectors, approx 8GB
  - A: Matrix **A** has N^2 entries
    - 10^{18} is a large number!

【备注】观察矩阵A和M可知,
Matrix **A** dense matrix!
Matrix **M** sparse matrix

1.4.1 Rearranging the Equation

华中科技大学 计算机科学与技术学院 School of Computer Science &Technology, HUST

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$, where $A_{ji} = \beta\, M_{ji} + \frac{1-\beta}{N}$
- $r_j = \sum_{\mathrm{i}=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{\mathrm{i}=1}^{N} r_i$$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N}$$

since $\sum r_i = 1$

- **So we get:** $\boldsymbol{r} = \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$

[x]

$[x]_N$ … a vector of length N with all entries x

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

Note: Here we assumed ***M*** has no dead-ends

1.4.1 Sparse Matrix Formulation

华中科技大学 计算机科学与技术学院 School of Computer Science &Technology, HUST

- We just rearranged the **PageRank equation**

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

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

- ***M*** is a **sparse matrix!** (with no dead-ends)
  - N nodes, 10 links per node, approx 10N entries

o(2N+N*N)

o(2N+10N)

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

1.4.1 The Complete PageRank Algorithm

华中科技大学 计算机科学与技术学院 School of Computer Science &Technology, HUST

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

- **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**.
- Hence, we have to explicitly account for it by computing **S**.

1.4.2 Sparse Matrix Encoding

- **Encode sparse matrix *M* using only nonzero entries**

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

- Space proportional roughly to number of links
- Say 10N(N nodes, 10 links per node), or 4*10*1 billion = 40GB, e.g. N = 1 billion (十亿)
- ***M* still won't fit in memory, but will fit on disk**

1.4.2 Basic Update Algorithm

- **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 $\boldsymbol{i}$ (of out-degree $\boldsymbol{d_i}$):
>   Read into memory: $\boldsymbol{i}, \boldsymbol{d_i}, \boldsymbol{dest_1}, \ldots, \boldsymbol{dest_{d_i}}, \boldsymbol{r^{old}(i)}$
>   For $\mathbf{j = 1 \ldots d_i}$
> $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$

r^{new}: 0, 1, 2, 3, 4, 5, 6

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

For one iteration

r^{old}: 0, 1, 2, 3, 4, 5, 6

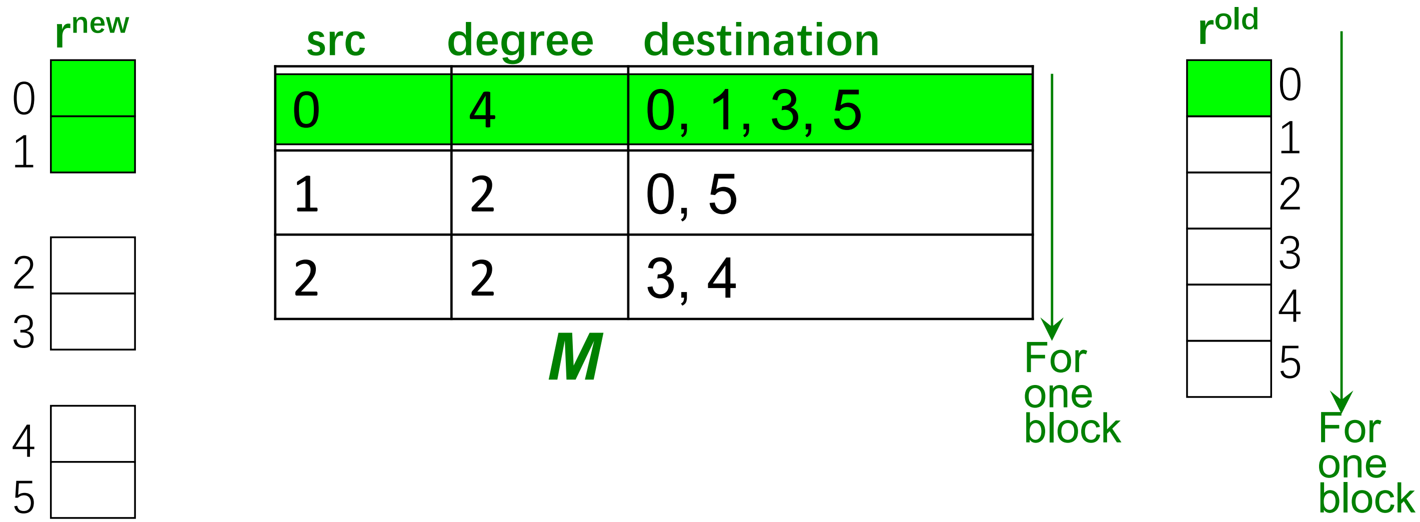
For one iteration

1.4.2 Analysis of Basic Update

- **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?

1.4.3 Block-based Update Algorithm

- Break r^{new} into k blocks that fit in memory



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

- Scan M and r^{old} once for each block
- In each iteration total k blocks, then k scans M and r^{old}

1.4.3 Analysis of Block Update

华中科技大学 计算机科学与技术学院 School of Computer Science &Technology, HUST

- **Similar to nested-loop join in databases**
  - Break $\boldsymbol{r}^{new}$ into $\boldsymbol{k}$ blocks that fit in memory
  - Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$ once for each block
- **Total cost:**
  - $\boldsymbol{k}$ scans of $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$
  - Write $\boldsymbol{r}^{new}$ back to disk ($\boldsymbol{k}$ blocks)
  - **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

1.4.4 Block-Stripe Update Algorithm

华中科技大学 计算机科学与技术学院 School of Computer Science &Technology, HUST

r^{new}: 0, 1 | 2, 3 | 4, 5

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

For one block

0	4	3
2	2	3

0	4	5
1	3	5
2	2	4

r^{old}: 0, 1, 2, 3, 4, 5

For one block

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

1.4.4 Analysis of Block-Stripe Update

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

Some Problems with PageRank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)

- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank (**next**)

- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities (**next**)