

LINK ANALYSIS

Note: slides adapted from the Mining of
Massive Datasets Textbook Ch 5.

Outline

- We will cover the link-analysis approaches:
 - PageRank
 - PageRank with Taxation
 - Computing PageRank with MapReduce

Readings

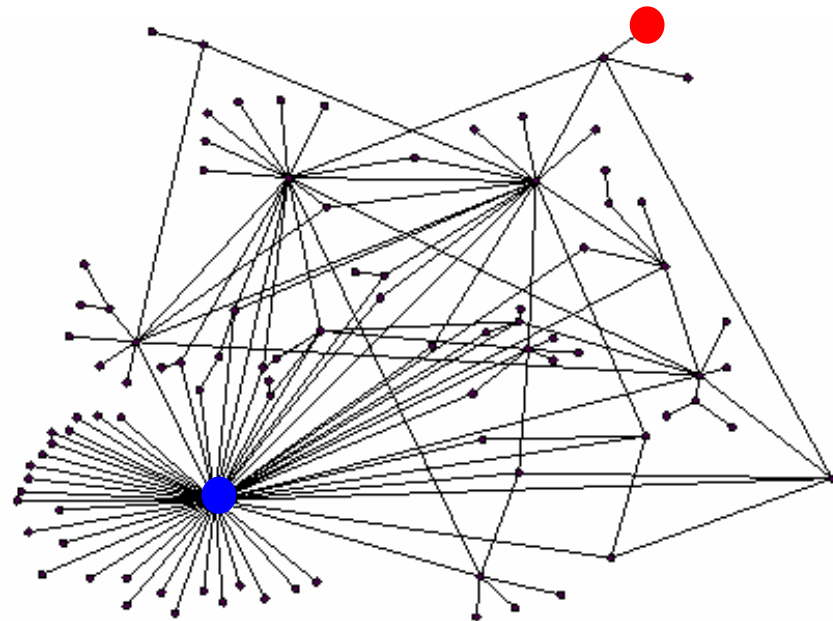
- Mining of massive DataSets Textbook (CH 5)
 - PageRank 5.1.2
 - PageRank Using MapReduce 5.2.2

How to search the Web?

- Early *search engines* crawled the Web and extracted *terms* to build an *inverted index*, which maps a term to all relevant Web pages.
- When a *search query* is issued, the query is decomposed into search terms, and the pages with those terms are look-up from the inverted index and ranked based on importance.
 - TF-IDF, cosine, etc.
- In 1996, it became content similarity (term-frequency) alone was no longer sufficient.
 - Number of pages grew rapidly in the mid-late 1990s:
 - A search query can result in 10M relevant pages, how to rank suitably?
 - Content similarity is easily spammed.
 - A page can repeat 'term' and add many relevant terms to boost ranking of the page, where in-fact the page may not be related to the search query.

Ranking Pages

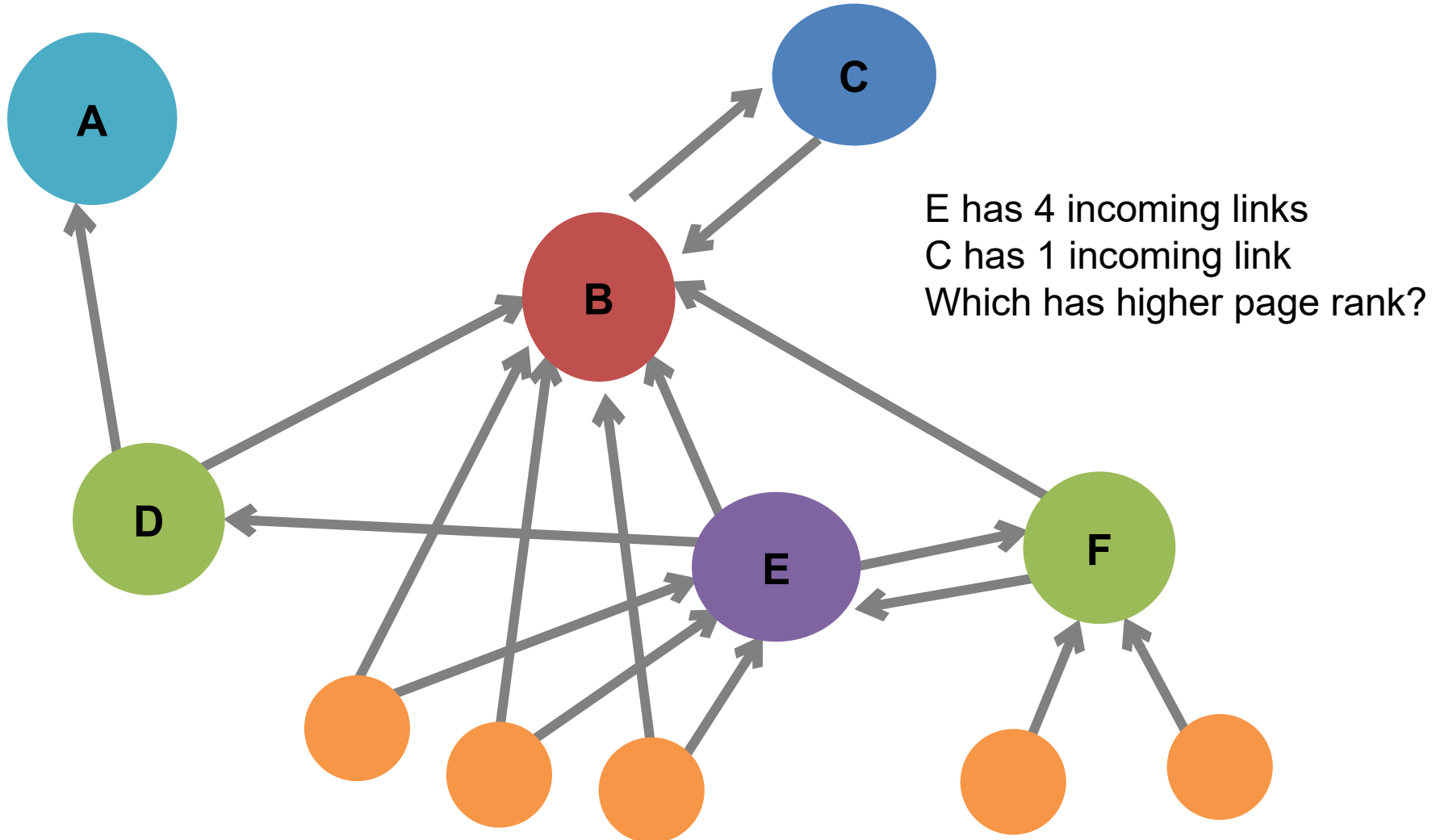
- The web can be represented as a directed graph.
- A link from one page to another is a directed edge in the graph.
- Google decided to rank the pages by the link structure.
- **Idea** – nodes with higher number of incoming links, is more likely to be important.



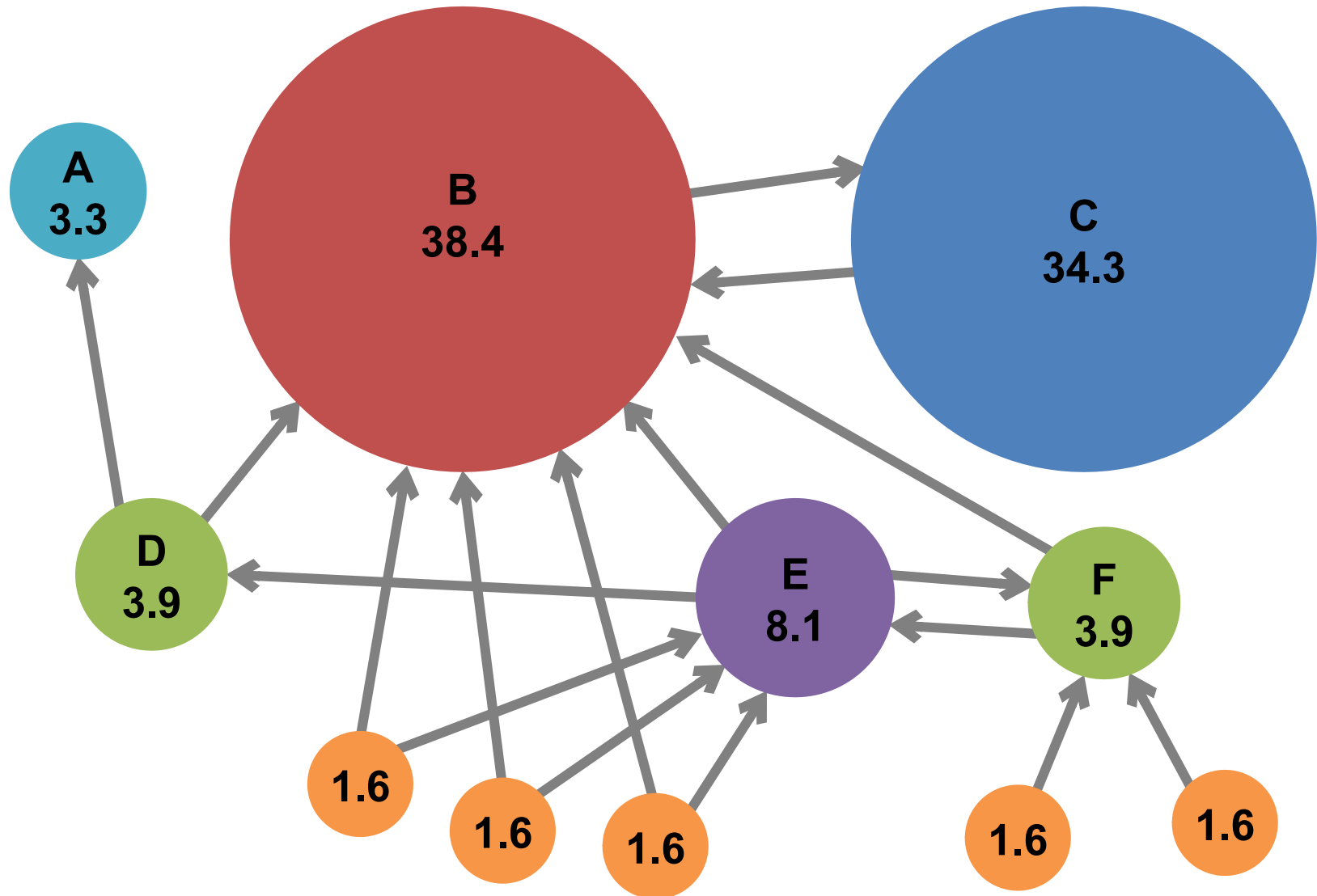
Ranking Pages

- Page Rank(PR) was used to simulate where a Web surfers, starting at a random pages, could congregate by following links from one page to another.
- **Idea: Links as votes**
 - PR is essentially a 'vote' by all other pages on the Web about the important another page.
 - Page is more important if its has *more in-coming* links (those are the ones the page cannot control).
 - Are all incoming links equal? Nah – *links from more important pages count more*

Example: PageRank Scores



Example: PageRank Scores



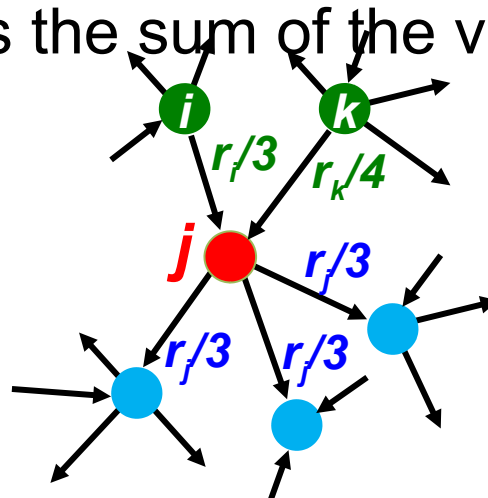
Link-based Page Ranking Background

- During 1997-1998 two most influential link-based page ranking approaches were proposed:
- Both proposed algorithms exploit the link-structure of the Web to rank pages according to 'authority' or 'importance':
 - **HITS**: Jon Kleinberg (Cornel University), at *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1998.
 - **PageRank**: Sergey Brin and Larry Page, PhD students from Stanford University, at *Seventh International World Wide Web Conference (WWW7)* in April, 1998.

PageRank Formulation

- Each link's vote is proportional to the importance of its source page
- If page j with importance r_j has n out-links, each link gets r_j / n votes
- Page j 's own importance is the sum of the votes on its in-links

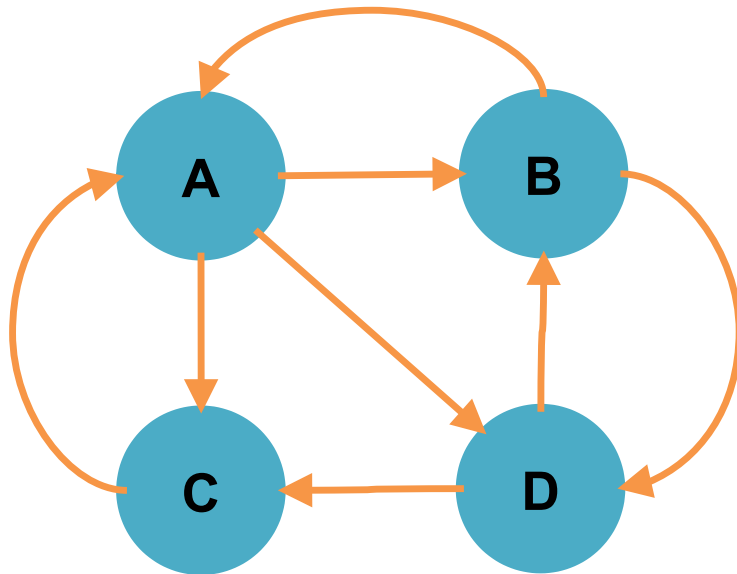
$$r_j = r_i / 3 + r_k / 4$$



PageRank Formulation

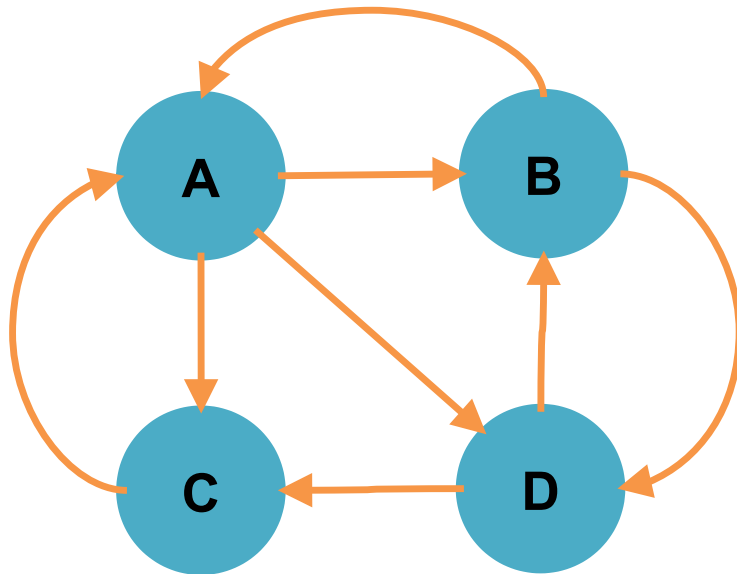
- Suppose that a page P_j has L_j links.
- If one of those links is to page P_i , then P_j will pass on $1/L_j$ of its importance to P_i .
- The importance ranking of P_i is then the sum of all the contributions made by pages linking to it.
- That is, if we denote the set of pages linking to P_i by B_i , then :

$$PR(P_i) = \sum_{P_j \in B_i} \frac{PR(P_j)}{L_j}$$



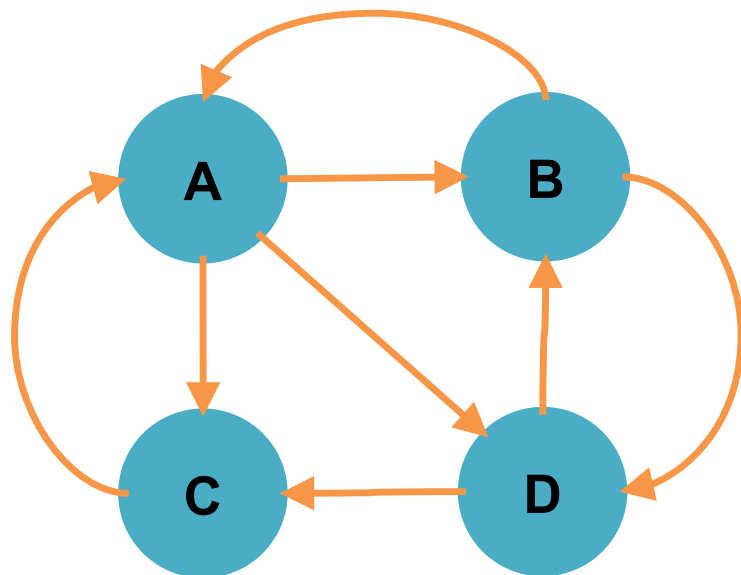
- **Page A has links to B, C, D**
- **Page B has links to A and D**
- **Page C has link to A**
- **Page D has links to B and C**

- **Suppose that a random surfer starts at page A, then**
- **Pages B, C, and D can be visited next each with probability $1/3$**
- **Since A does not link to itself, Page A has zero probability to be visited next**



- **Page A has links to B, C, D**
- **Page B has links to A and D**
- **Page C has link to A**
- **Page D has links to B and C**

- **Now, suppose the surfer is at Page B**
- **The surfer has probability of $\frac{1}{2}$ of ending up at A, $\frac{1}{2}$ of ending up at D and zero probability of ending up at B or C in the next step.**
- **Use Transition Matrix M**
 - **Describes what happened to random surfer after one step**
 - **M has n rows and columns (where n = number of pages or nodes)**



- **Page A has links to B, C, D**
- **Page B has links to A and D**
- **Page C has link to A**
- **Page D has links to B and C**

$$M = \begin{bmatrix} \text{A} & \text{B} & \text{C} & \text{D} \\ \begin{matrix} 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{matrix} & \begin{matrix} 1/2 \\ 0 \\ 0 \\ 1/2 \end{matrix} & \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{matrix} \\ \text{A} & \text{B} & \text{C} & \text{D} \end{bmatrix}$$

How to interpret?

- Consider 1st column – surfer at A has a $1/3$ probability of being at each of the other pages in the next step
- Consider 2nd column – surfer at B has $1/2$ probability of being at A next and the same for being at D next

What does this matrix mean?

- The probability distribution for the location of a random surfer
 - A column vector whose j^{th} component is the probability that the surfer is at page j
- If we surf any of the n pages of the Web with equal probability
 - The initial vector \mathbf{v}_0 will have $1/n$ for each component
 - If M is the transition matrix of the Web
 - After one step, the distribution of the surfer will be $M\mathbf{v}_0$
 - After two steps, $M(M\mathbf{v}_0) = M^2\mathbf{v}_0$ and so on
 - Multiplying the initial vector \mathbf{v}_0 by M a total of i times gives the distribution of the surfer after the i^{th} steps

What does this matrix mean?

- The probability x_i that a random surfer will be at node i at the next step

$$x_i = \sum_j m_{ij} v_j$$

- M_{ij} is the probability that a surfer at node j will move to node i at the next step
- V_j is the probability that the surfer was at node j at the previous step

What does this matrix mean?

- The distribution of the surfer approaches a limiting distribution \mathbf{v} that satisfies $\mathbf{v} = M\mathbf{v}$ provided two conditions are met:
- The graph is strongly connected
 - It is possible to get from any node to any other node
- There are no dead ends
 - Dead ends = nodes that have no outgoing links

What does this matrix mean?

- The limit is reached when multiplying the distribution by M another time does not change the distribution
 - The limiting v is an eigenvector of M
 - Since M is stochastic (its columns each add up to 1), v is the principle eigenvector
 - Its associated eigenvalue is the largest of all eigenvalues
- The principle eigenvector of M
 - Where the surfer is most likely to be after a long time
- For the Web, 50-75 iterations are sufficient to converge to within the error limits of double-precision arithmetic

$$M = \begin{bmatrix} 0 & \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

- Suppose we apply this process to the matrix M
- The initial vector v_0 and v_1 after multiplying M

$$v_1 = Mv_0 = \begin{bmatrix} 0 & \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{9}{24} \\ \frac{5}{24} \\ \frac{5}{24} \\ \frac{5}{24} \end{bmatrix}$$

- The sequence of approximations to the limit we get by multiplying at each step by M is :

$$v_1 = Mv_0 = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} \quad v_2 = Mv_1 = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix} = \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}$$

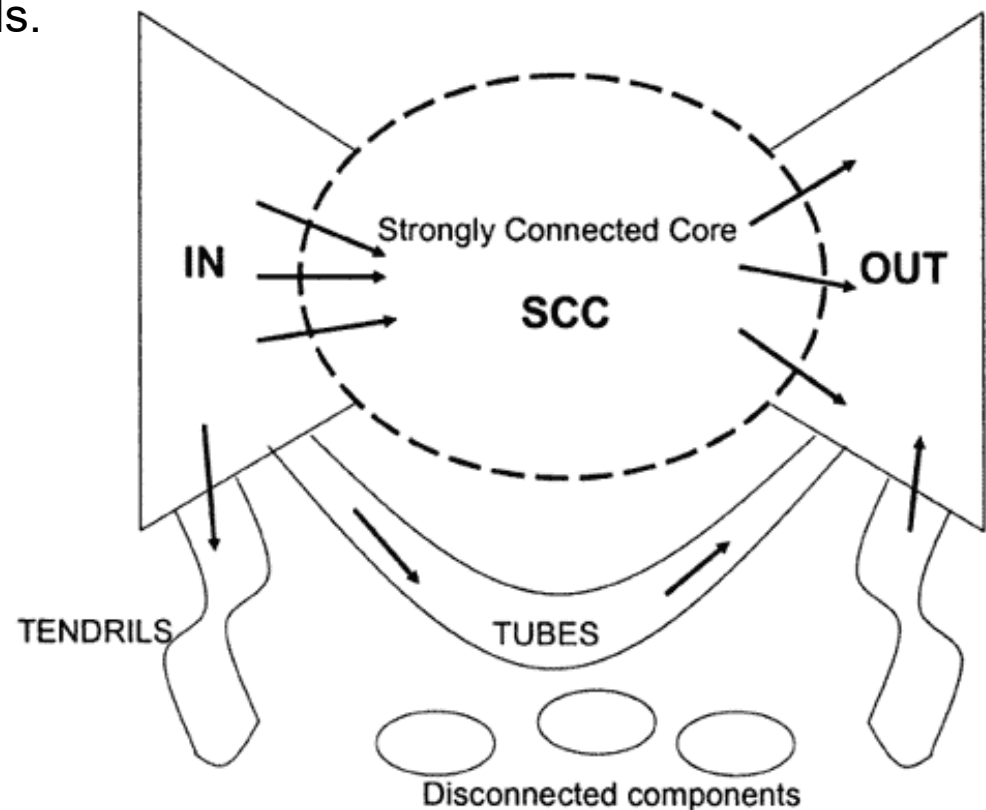
$$v_2 = Mv_1 = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix} = \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix} \dots \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

- The difference in probability is not noticeable
- In the real web there are billions of nodes greatly varying in importance

However the web does not satisfy those conditions:

- The in-component, consisting of pages that could reach the SCC by following links, but were not reachable from the SCC.
- The out-component, consisting of pages reachable from the SCC but unable to reach the SCC.
- Tendrils, which result in dead ends.

We had assumed the graph is strongly connected and has no dead ends.



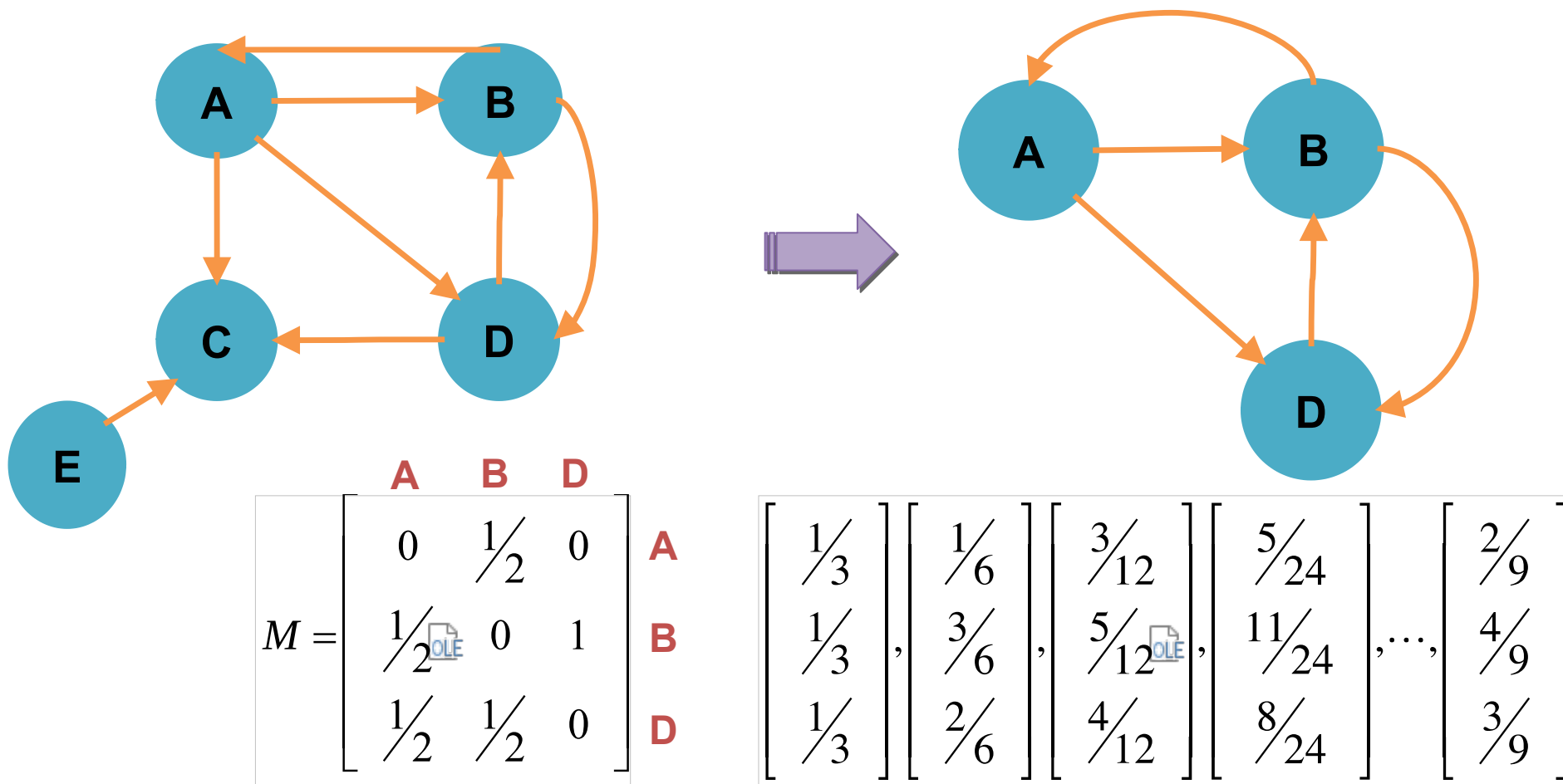
Avoiding Dead Ends

- If we allow dead ends
 - The transition matrix of the Web is no longer stochastic
 - Some of the column will sum to 0 rather than 1
- If we compute $M^i v$ for increasing powers of a sub stochastic matrix
 - Some of all the components of the vector go to 0
 - Sub stochastic matrix
 - A matrix whose column sums are at most 1
 - Importance “drains out” of the Web
 - No information about the relative importance of pages

Approaches to dealing with dead ends

- Recursive deletion
 - Drop dead ends from the graph
 - Drop their incoming arcs as well
 - Doing so may create more dead ends
 - Drop those new dead ends
- Taxation
 - Modify the process by which random surfers are assumed to move about the Web

EX. Graph with two level of dead ends

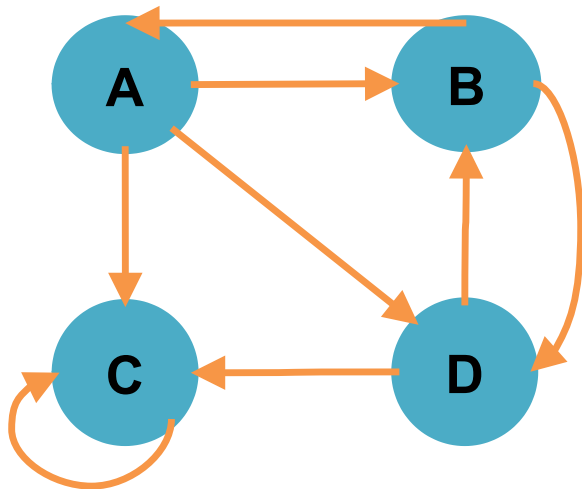


- Then compute PageRank C and then E (in opposite order of deletion) :

$$PR(C) = PR(A)/3 + (3/9)/2 = 13/54$$

$$PR(E) = PR(C)$$

Spider Traps



$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix} = \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix} = \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix} = \dots \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- Dead ends are not the only problem
- Spider traps cause PageRank to place all the importance to pages within the spider trap.

PageRank using Taxation

- To avoid dead ends, we modify the calculation of PageRank
 - Allow each random surfer a small probability of **teleporting** to a random page
 - Rather than following an out-link from their current page
- The iterative step, where we compute a new vector estimate of PageRanks v' from the current PageRank estimate v and the transition matrix M is
$$v' = \beta Mv + e(1-\beta)/n$$
- Where β is a chosen constant
 - Usually in the range 0.8 to 0.85
- e is a vector with 1's for the appropriate number of components
- n is the number of nodes in the Web graph

PageRank using Taxation

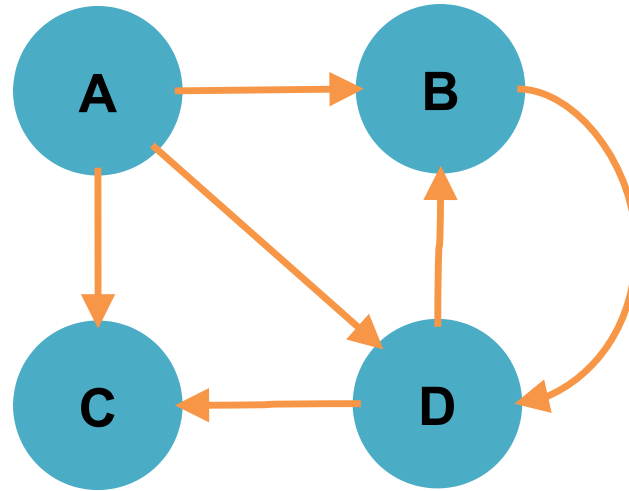
- The term βMv represents the case where:
 - With probability β , the random surfer decides to follow an out-link from their present page
- The term $(1-\beta)/n$ is a vector
 - Each of whose components has value $(1-\beta)/n$
 - Represents the introduction, with probability $1-\beta$, of a new random surfer at a random page

$$v' = \beta Mv + (1-\beta)/n$$

PageRank using Taxation

- If the graph has no dead ends
 - The probability of introducing a new random surfer is exactly equal to the probability that the random surfer will decide not to follow a link from their current page
 - Surfer decides either to follow a link or teleport to a random page
- If the graph has dead ends
 - The surfer goes nowhere
 - The term $(1-\beta)/n$ does not depend on the sum of the components of the vector \mathbf{v} , there will be some fraction of a surfer operating on Web
 - When there are dead ends, the sum of the components of \mathbf{v} may be less than 1
 - But it will never reach 0

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$



$$v' = \beta Mv + e(1-\beta)/n$$

$$\beta = 0.8$$

Factor β * each
element of M

$$v' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} v + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

The components of
the vector are
 $(1-\beta)/n$ since
 $1-\beta=1/5$ and $n=4$,
hence $= 1/20$

Sparse matrix formulation

- We can rearrange the page rank equation:
 - $\mathbf{v} = \beta \mathbf{M} \mathbf{v} + [(1-\beta)/N]_N$ *# vector of N components*
 - $[(1-\beta)/N]_N$ is an N -vector with all entries $(1-\beta)/N$
- \mathbf{M} is a sparse matrix!
 - 10 links per node, approx $10N$ entries
- So in each iteration, we need to:
 - Compute $\mathbf{v}^{\text{new}} = \beta \mathbf{M} \mathbf{v}^{\text{old}}$
 - Add a constant value $(1-\beta)/N$ to each entry in \mathbf{v}^{new}

PageRank using MapReduce

- PageRank is essentially a matrix by vector multiplication problem
- Just need to do Matrix by Vector multiplication using MapReduce
- Assumptions:
 - The transition Matrix M and even the Vector v can be large
 - The transition Matrix M and the vector V are sparse , so we are dealing with sparse matrices.

Sparse Matrix Multiplication

- Sparse matrices are matrices in which most elements are zero, and hence they are usually represented in a succinct way.
- Sparse matrices are common in scientific applications and graph representations (ex. webpage connectivity).
- Usually adjacency matrices of a graph or directed graphy are very sparse and also very large.

Sparse matrix encoding

- Encode sparse matrix using only nonzero entries
 - Space proportional roughly to number of links
 - say $10N$, or 4×10^1 billion = 40GB
 - still won't fit in memory, but will fit on disk

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

Matrix-Vector multiplication using MapReduce

- Suppose we have an $n \times k$ matrix M , whose elements in row i and column j will be denoted m_{ij}
 - M represents the transition matrix
- V is an $k \times 1$ column vector
 - V represents the 'current' rank of pages
- Then the matrix-vector product produces a vector v' of length n , whose element i^{th} element x_i is given by :

$$x_i = \sum_{j=1}^k m_{ij} v_j$$

M is a n x k matrix

$$M = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}$$

V is a k vector

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

Take dot product of v with each row of matrix M .

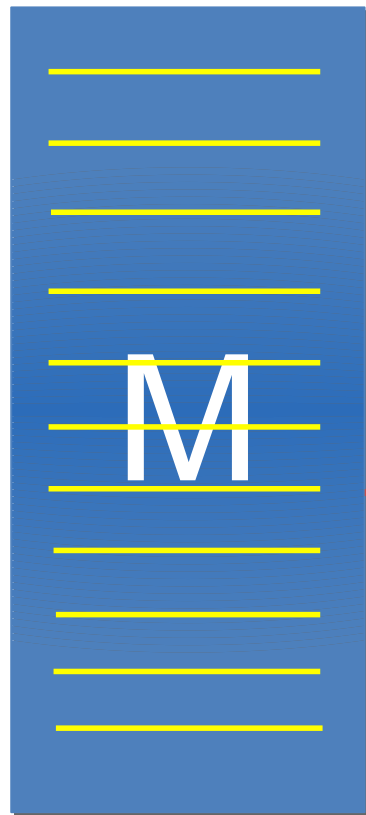
$$Mv = \begin{bmatrix} a_{11}v_1 + a_{12}v_2 + \cdots + a_{1k}v_k \\ a_{21}v_1 + a_{22}v_2 + \cdots + a_{2k}v_k \\ \vdots \\ a_{n1}v_1 + a_{n2}v_2 + \cdots + a_{nk}v_k \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^k a_{1j}v_j \\ \sum_{j=1}^k a_{2j}v_j \\ \vdots \\ \sum_{j=1}^k a_{nj}v_j \end{bmatrix}$$

Matrix-Vector multiplication using MapReduce

- If M is not very large (i.e. n is < 100) then no need for MapReduce or parallel processing.
- However, that's usually not the case when the calculation is part of ranking WebPages .
- Solution – need to divide matrix M into blocks

Map / Reduce Function

- Assumption
 - Matrix M is too large to fit in memory, but V fits in memory
- Map Function
 - Each Map task will operate on a element of the matrix M
 - For each matrix element m_{ij} , the Map function emits the key-value pair : (**key** = i , **value** = $m_{ij} * v_j$)
 - **Reasoning?** All the terms that make up component x_i in the result (the sum of those terms), will get the same key i .
- Reduce Function
 - Each Reduce task will sum up the values associated with a given key i
 - The result will be a pair (i, x_i)



input

// assume V in memory

map(key, value):

// value is a tuple ("M", i, k, a_{ik})

for j = 1 to k:

emit(i , a_{ij} * v_j))

reduce(key, values):

sum = 0

for v in values:

sum += v

emit(i , sum (v))

$$M = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nk} \end{bmatrix}$$

// assume V in memory

map(key, value):

// value is a tuple ("M", i, k, a_{ik})

for j = 1 to k:

emit(i , a_{ij} * v_j))

(1, a₁₁ * v₁)
 (1, a₁₂ * v₂)
 (1, a₁₃ * v₃)
 ...
 (1, a_{1k} * v_k)

reduce(key, values):

sum = 0

for v in values:

sum + = v

emit(i , sum (v))

For key = 1
 a₁₁*v₁ + a₁₂*v₂ + a₁₃*v₃ + a_{1k}*v_k

Matrix-Vector Product (Cont.)

- It is possible that the vector v is so large that it will not fit in main memory entirely.
- We can divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes of the same height.
 - The goal is to use enough stripes so that the partition of the vector in one stripe fits into main memory.

M is a n x k matrix

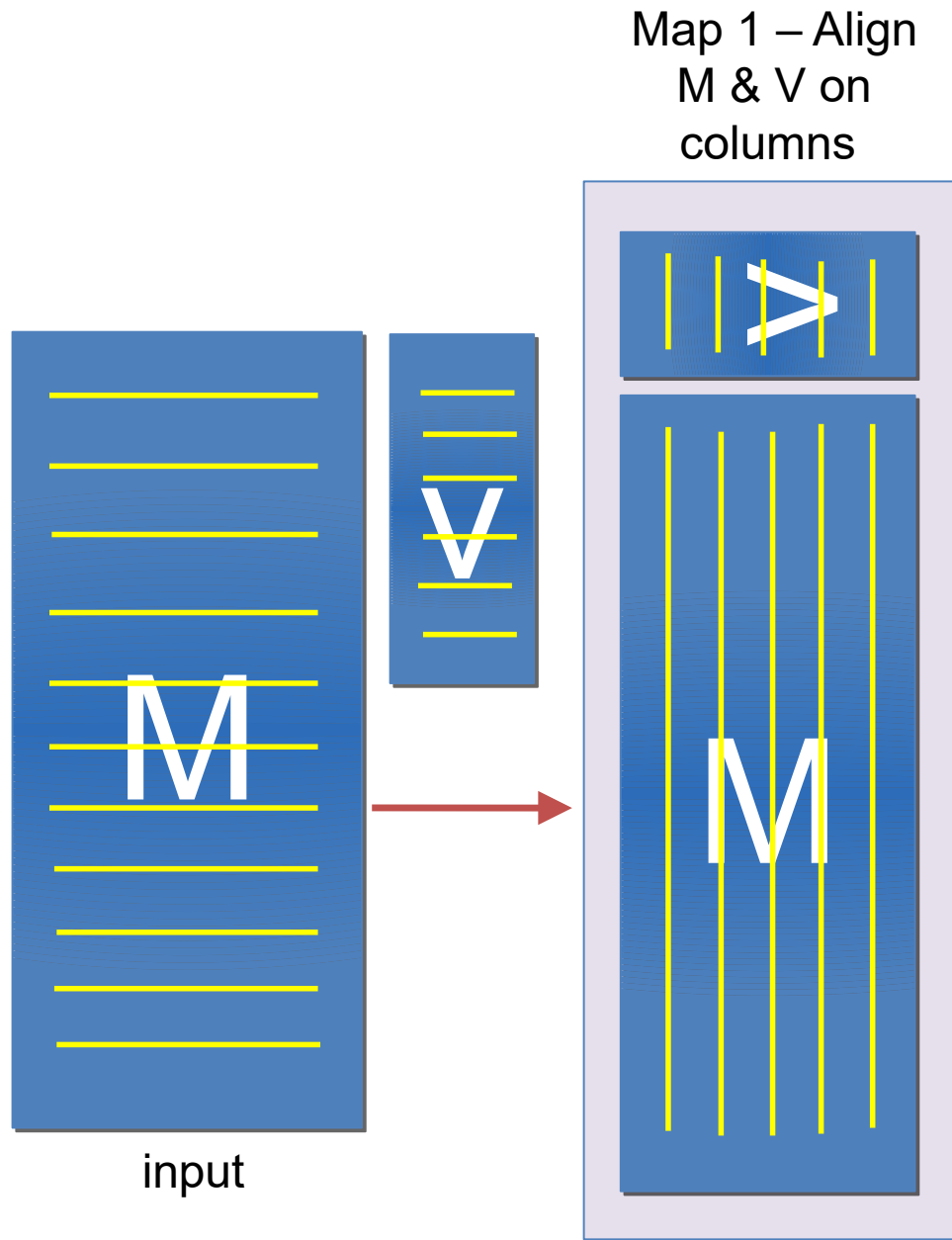
$$M = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}$$

V is a k vector

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

Take dot product of v with each row of matrix M .

$$Mv = \begin{bmatrix} a_{11}v_1 + a_{12}v_2 + \cdots + a_{1k}v_k \\ a_{21}v_1 + a_{22}v_2 + \cdots + a_{2k}v_k \\ \vdots \\ a_{n1}v_1 + a_{n2}v_2 + \cdots + a_{nk}v_k \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^k a_{1j}v_j \\ \sum_{j=1}^k a_{2j}v_j \\ \vdots \\ \sum_{j=1}^k a_{nj}v_j \end{bmatrix}$$



map(key, value):

// value is a tuple ("M", i, k, a_ik)

if value[0] == "M": # matrix

for j = 1 to k: # for each column

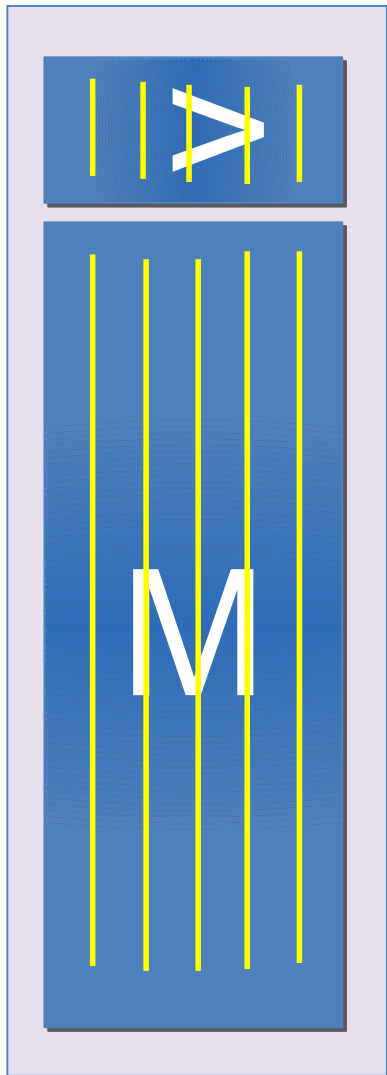
emit(j , (i, a_ij))

// value is a tuple ("V", k, v_k)

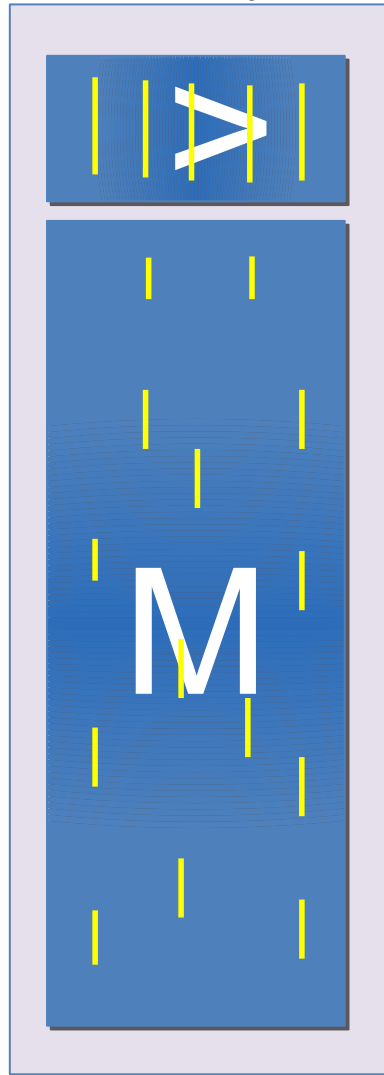
else: # vector

emit(k , (v_k))

Map 1 – Align
M & V on
columns



Reduce 1 –
Output $M_{ik} * v_k$
with key i



reduce(key, values)

for val **in** values :

if len (val) == 1: *#vector tuple (val)*

vecval = val

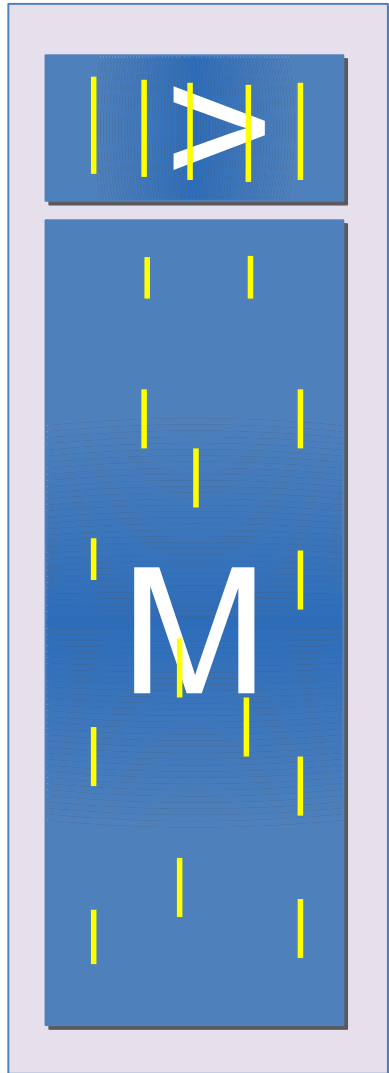
else: *# matrix tuple (i, val)*

matvals.append(val)

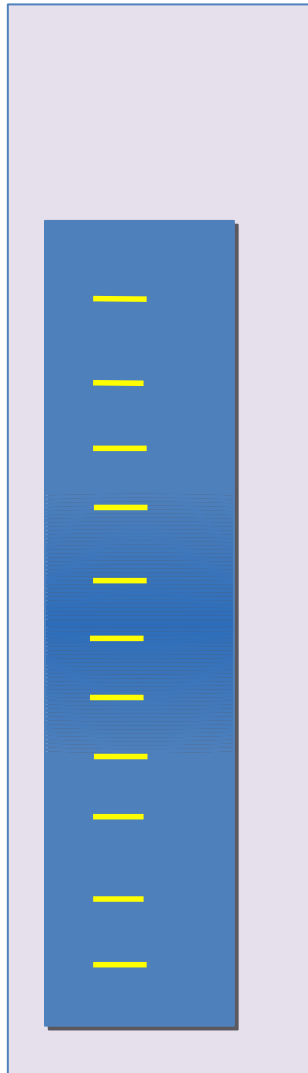
for m **in** matval :

emit(m[0] , vecval*m[1])

Reduce 1 –
Output $M_{ik} * v_k$
with key i



Reduce 2 –
Sum $M_{ik} * v_k$



reduce(key, vals)

sum = 0

for v **in** vals:

sum += v

emit (key , sum)

Basic Algorithm

- Assume we have enough RAM to fit \mathbf{v}^{new} , plus some working memory
 - Store \mathbf{v}^{old} and matrix \mathbf{M} on disk

Basic Algorithm:

- Initialize: $\mathbf{v}^{\text{old}} = [1/N]_N$, where N is the number of nodes in the graph
- Iterate:
 - **Update:** Perform a sequential scan of \mathbf{M} and \mathbf{v}^{old} to update \mathbf{v}^{new}
 - Write out \mathbf{v}^{new} to disk as \mathbf{v}^{old} for next iteration
 - Every few iterations, compute $|\mathbf{v}^{\text{new}} - \mathbf{v}^{\text{old}}|$ and stop if it is below threshold
 - Need to read in both vectors into memory

Sparse Matrix Multiplication

- Sparse matrices are matrices in which most elements are zero, and hence they are usually represented in a succinct way.
- Sparse matrices are common in scientific applications and graph representations (ex. webpage connectivity).
- Usually adjacency matrices of a graph or directed graphy are very sparse and also very large.

Sparse Matrix Product – Approach 1

- Task:**

- Compute product $A \cdot B = C$
- Assume most matrix entries are 0
- Neither matrix A or B fits on a single machine, so must figure out how to distribute the data, so must apply MapReduce.

$A(m \times n)$		$B(n \times k)$		$C(m \times k)$		$C(m \times k)$
10 20		-1		$10 \cdot -1 + 0 \cdot -2 + 20 \cdot 0$		$10 \cdot 0 + 0 \cdot -3 + 20 \cdot -4$
$\begin{bmatrix} 30 & 40 \end{bmatrix}$	\times	$\begin{bmatrix} -2 & -3 \end{bmatrix}$	$=$	$\begin{bmatrix} 0 \cdot -1 + 30 \cdot -2 + 40 \cdot 0 & 0 \cdot 0 + 30 \cdot -3 + 40 \cdot -4 \end{bmatrix}$	$=$	$\begin{bmatrix} -10 & -80 \\ -60 & -250 \end{bmatrix}$
50 60 70		-4		$50 \cdot -1 + 60 \cdot -2 + 70 \cdot 0$		$50 \cdot 0 + 60 \cdot -3 + 70 \cdot -4$
						$\begin{bmatrix} -170 & -460 \end{bmatrix}$

A is a m x n matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

B is a n x m matrix

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}$$

We want to compute the product A x B to generate C (an m x k matrix)

$$C = \begin{bmatrix} \sum_{j=1}^n a_{1j} b_{j1} & \sum_{j=1}^n a_{1j} b_{j2} & \dots & \sum_{j=1}^n a_{1j} b_{jk} \\ \sum_{j=1}^n a_{2j} b_{j1} & \sum_{j=1}^n a_{2j} b_{j2} & \dots & \sum_{j=1}^n a_{2j} b_{jk} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^n a_{mj} b_{j1} & \sum_{j=1}^n a_{mj} b_{j2} & \dots & \sum_{j=1}^n a_{mj} b_{jk} \end{bmatrix}$$

What if each entry in matrix c was computed separately by a given reducer.

Must route all relevant data to that reducer



Represent matrix as list of non-zero entries

$\langle \text{matrixID}, \text{row}, \text{col}, \text{value} \rangle$

Strategy

Phase 1: Compute all products $a_{i,j} \cdot b_{j,k}$

Phase 2: Sum products for each entry i,k
Each phase involves a Map/Reduce

$$B = \begin{bmatrix} -1 & 0 \\ -2 & -3 \\ 0 & -4 \end{bmatrix}$$

Matrix	j	k	Val
B	1	1	-1
B	2	1	-2
B	2	2	-3
B	3	2	-4

$$A = \begin{bmatrix} 10 & 0 & 20 \\ 0 & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix}$$

Matrix	i	j	Val
A	1	1	10
A	1	3	20
A	2	2	30
A	2	3	40
A	3	1	50
A	3	2	60
A	3	3	70

Consider where the data from matrix A needs to be routed

$$A = \begin{matrix} & \begin{matrix} j_1 & j_2 & j_3 \end{matrix} \\ \begin{matrix} i_1 \\ i_2 \\ i_3 \end{matrix} & \begin{bmatrix} 10 & 0 & 20 \\ 0 & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix} \end{matrix}$$

Assuming each entry in result matrix C will be computed by one reducer

$$C = \begin{bmatrix} \langle A, 10, j_1 \rangle, \langle A, 0, j_2 \rangle, \langle A, 20, j_3 \rangle & \langle A, 10, j_1 \rangle, \langle A, 0, j_2 \rangle, \langle A, 20, j_3 \rangle \\ \langle A, 0, j_1 \rangle, \langle A, 30, j_2 \rangle, \langle A, 40, j_3 \rangle & \langle A, 0, j_1 \rangle, \langle A, 30, j_2 \rangle, \langle A, 40, j_3 \rangle \\ \langle A, 50, j_1 \rangle, \langle A, 60, j_2 \rangle, \langle A, 70, j_3 \rangle & \langle A, 50, j_1 \rangle, \langle A, 60, j_2 \rangle, \langle A, 70, j_3 \rangle \end{bmatrix}$$

$$C = \begin{bmatrix} \langle A, 10, j_1 \rangle, \langle A, 0, j_2 \rangle, \langle A, 20, j_3 \rangle & \langle A, 10, j_1 \rangle, \langle A, 0, j_2 \rangle, \langle A, 20, j_3 \rangle \\ \langle A, 0, j_1 \rangle, \langle A, 30, j_2 \rangle, \langle A, 40, j_3 \rangle & \langle A, 0, j_1 \rangle, \langle A, 30, j_2 \rangle, \langle A, 40, j_3 \rangle \\ \langle A, 50, j_1 \rangle, \langle A, 60, j_2 \rangle, \langle A, 70, j_3 \rangle & \langle A, 50, j_1 \rangle, \langle A, 60, j_2 \rangle, \langle A, 70, j_3 \rangle \end{bmatrix}$$

$$A = \begin{bmatrix} \mathbf{j_1} & \mathbf{j_2} & \mathbf{j_3} \\ 10 & 0 & 20 \\ 0 & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix} \begin{matrix} \mathbf{i_1} \\ \mathbf{i_2} \\ \mathbf{i_3} \end{matrix}$$

for each non-zero entry **in** A (i, j, val) :

for k_index = 1 **to** k:

key = k_index

value = (A , j , val)

emit (key, value)

Key (i,k)

Value (matrix, j, Val)

(1,1)

(A,1,10)

(1,1)

(A,3,20)

(1,2)

(A,1,10)

(1,2)

(A,3,20)

(2,1)

(A,2,30)

(2,1)

(A,3,40)

(2,2)

(A,2,30)

(2,2)

(A,3,40)

(3,1)

(A,1,50)

(3,1)

(A,2,60)

(3,1)

(A,3,70)

(3,2)

(A,1,50)

(3,2)

(A,2,60)

(3,2)

(A,3,70)

$$B = \begin{bmatrix} -1 & 0 \\ -2 & -3 \\ 0 & -4 \end{bmatrix}$$

k_1 k_2
 j_1 j_2 j_3

for each non-zero entry **in** B (j, k, val) :

for i_index = 0 **to** i:

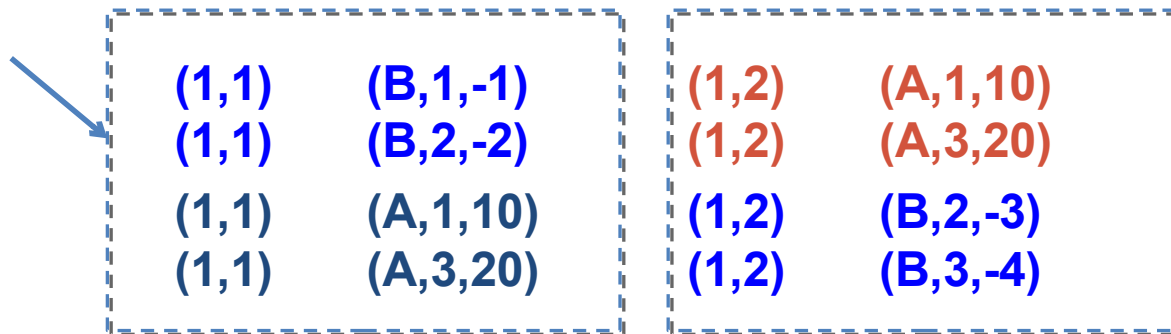
key = (i , i_index)

value = (A , j , val)

emit (key, value)

Key(i,k)	Value(matrix, j, Val)
(1,1)	(B,1,-1)
(1,1)	(B,2,-2)
(1,2)	(B,2,-3)
(1,2)	(B,3,-4)
(2,1)	(B,1,-1)
(2,1)	(B,2,-2)
(2,2)	(B,2,-3)
(2,2)	(B,3,-4)
(3,1)	(B,1,-1)
(3,1)	(B,2,-2)
(3,2)	(B,2,-3)
(3,2)	(B,3,-4)

Each
Reducer will
get relevant
keys



(1,1)	(B,1,-1)	(1,2)	(A,1,10)
(1,1)	(B,2,-2)	(1,2)	(A,3,20)
(1,1)	(A,1,10)	(1,2)	(B,2,-3)
(1,1)	(A,3,20)	(1,2)	(B,3,-4)

To compute a
given cell, need
to examine 'j'
index of each
tuple

(2,1)	(A,2,30)	(2,2)	(A,2,30)
(2,1)	(A,3,40)	(2,2)	(A,3,40)
(2,1)	(B,1,-1)	(2,2)	(B,2,-3)
(2,1)	(B,2,-2)	(2,2)	(B,3,-4)

Then compute
the product tuple
A and tuple B if
they have same j
component

(3,1)	(A,1,50)	(3,2)	(A,1,50)
(3,1)	(A,2,60)	(3,2)	(A,2,60)
(3,1)	(A,3,70)	(3,2)	(A,3,70)
(3,1)	(B,1,-1)	(3,2)	(B,2,-3)
(3,1)	(B,2,-2)	(3,2)	(B,3,-4)

$$-1 * 10 = -10$$



(1,1)	(B,1,-1)
(1,1)	(B,2,-2)
(1,1)	(A,1,10)
(1,1)	(A,3,20)

(1,2)	(A,1,10)
(1,2)	(A,3,20)
(1,2)	(B,2,-3)
(1,2)	(B,3,-4)

$$30 * -2 = -60$$



(2,1)	(A,2,30)
(2,1)	(A,3,40)
(2,1)	(B,1,-1)
(2,1)	(B,2,-2)

(2,2)	(A,2,30)
(2,2)	(A,3,40)
(2,2)	(B,2,-3)
(2,2)	(B,3,-4)

(3,1)	(A,1,50)
(3,1)	(A,2,60)
(3,1)	(A,3,70)
(3,1)	(B,1,-1)
(3,1)	(B,2,-2)

(3,2)	(A,1,50)
(3,2)	(A,2,60)
(3,2)	(A,3,70)
(3,2)	(B,2,-3)
(3,2)	(B,3,-4)

map(key, value):

// value is a tuple ("A", i, j, a_ij) or ("B", j, k, b_jk)

if value[0] == "A":

 i = value[1]

 j = value[2]

 a_ij = value[3]

for k = 1 **to** p:

emit((i, k), (A, j, a_ij)) *// key = (i , k) value = tuple of 3 vals*

else:

 j = value[1]

 k = value[2]

 b_jk = value[3]

for i = 1 **to** m:

emit((i, k), (B, j, b_jk)) *// key = (i , k) value = tuple of 3 vals*

```
reduce(key, values) : // key is (i, k)  
    // values is a list of ("A", j, a_ij) and ("B", j, b_jk)  
    hash_A = {j: a_ij for (x, j, a_ij) in values if x == A} // partition entries based on j  
    hash_B = {j: b_jk for (x, j, b_jk) in values if x == B} // partition entries based on j  
    result = 0  
    for j = 1 to n:  
        result += hash_A[j] * hash_B[j]  
    emit(key, result)
```