CSE 4/587
Data Intensive Computing

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Dr. Shamshad Parvin
shamsadp@buffalo.edu
313 Davis Hall
What is a Graph?

- A **graph** is a structure made up of a set of objects, where some pairs of the objects are "related".
- Mathematically, objects are represented with **vertices** (or nodes or points) and the relations between two vertices are represented with **edges** (or links or lines).
- Typically, a graph is depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges.
- Edges can be directed or undirected (a relationship can go both ways).
- Edges can be weighted to show the “strength”, distance, etc.
Graph Representations

There are two standard ways to represent a graph $G(V,E)$ [V is the set of vertices, E is the set of edges]

1. adjacency list representation
2. adjacency matrix

An adjacency matrix is 2-Dimensional Array of size $V \times V$, where $V$ is the number of vertices in the graph.

An adjacency list is an array of linked lists, where the array size is same as number of vertices in the graph. Every vertex has a linked list. Each node in this linked list represents the reference to another vertex that shares an edge with the current vertex.
Single Source Shortest Path

**Sequential solution: Dijkstra’s algorithm**

- $d[s] = 0$
- for all other vertices $d[v] = \infty$
- $Q = \{V\}$ // Q is priority queue based on distances
- while Q not empty
  - $u = \min(Q)$ // node with min d value
  - for all vertex $v$ in $u$.adjacencyList
    - if $d[v] > d[u] + w[u,v]$
      - $d[v] = d[u] + w[u,v]$
  - mark $u$ and remove from $Q$

At each iteration of while loop, the algorithm expands the node with the shortest distance and updates distances to all reachable nodes.
Sample graph: let's apply the algorithm 5.2
Example
How do we apply this algorithm if we have a graph with large number of nodes and edges between them? MapReduce?

What is the main issue here?

The algorithm is sequential, needs a global state

Global states are not possible with map reduce...
Graph Processing in MapReduce

Let's see how we can handle a graph problem in parallel breadth first with MapReduce

(The intuition behind the algorithm is this: the distance of all nodes connected directly to the source node is one; the distance of all nodes directly connected to those is two; and so on.)

Remember, the MapReduce paradigm requires a **mapper function** and a **reducer function**
Issues in Processing a Graph in MR

**Goal:** Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.
Goal: Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.

Problems to Tackle:
Goal: Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.

Problems to Tackle:

1. How do we represent our graph?
Goal: Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.

Problems to Tackle:

1. How do we represent our graph?
2. What are our <key,value> pairs?
Goal: Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.

Problems to Tackle:

1. How do we represent our graph?
2. What are our <key,value> pairs?
3. How do we iterate through various stages of processing?
Issues in Processing a Graph in MR

**Goal:** Start from a given node and label all the nodes in the graph so that we can determine the shortest distance.

Assume all the distance between edges is one.

**Problems to Tackle:**

1. How do we represent our graph?
2. What are our <key,value> pairs?
3. How do we iterate through various stages of processing?
4. When/how do we terminate execution?
Graph Representation and Input Format

Our graphs will be represented as a collection of **Node** objects

**Node:**
- nodeId,
- distanceLabel,
- adjacencyList[nodeId, distance],
- ...

Input the graph as text and parse it to build our <key, value> pairs

*So what are our <key, value> pairs?*
From mapper to reducer two types of `<key, value>` pair:

1. `<nodeId n, Node N>` // nodeId to Node object
2. `<nodeId n, distance>` // nodeId to distance so far
Iteration

- Each iteration in the algorithm is a MapReduce job
- Iterations and termination are coordinate by an external driver application (more on this in future lectures)
- The first iteration starts at the source node (with distance 0)
  - It updates and emits all distances for nodes in the adjacency list
- The next iteration takes the output from the previous and updates/emits all distances for nodes connected to this set of nodes
- Continue until termination
Termination condition also needs to be tracked in the **Node class**

Terminate when the graph has reached a steady state:
- All the nodes have been labeled with min distance
- Labels no longer change between iterations
class Mapper
    method map (nodeId n, Node N)
        d ← N.distance
        emit(n, N)          // type 1
        for nodeId m in N.adjacencyList
            emit(m, d+1)     // type 2

The method map takes in two parameters, nodeId n and Node N

The method produces two key value pairs <n, N> and the updated distance to all of the adjacent nodes <m, d+1>
Reducer Class

class Reducer
    method Reduce(nodeId n, [d1, d2, d3..])
        dmin ← ∞; // or a large #
        Node N ← null

        for all d in [d1, d2, ...]:
            if IsNode(d) then N ← d
            else if d < dmin then dmin ← d

        N.distance ← dmin  // update the shortest distance in N
        emit (n, N)
Trace with Sample Data

Input Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>adjacent outlink node</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2:3:</td>
</tr>
<tr>
<td>2</td>
<td>10000</td>
<td>3:4:</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td>2:4:5</td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
<td>5:</td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
<td>1:4</td>
</tr>
</tbody>
</table>
Trace with Sample Data

Input Graph

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10000</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
</tr>
</tbody>
</table>

Input to the first iteration:

d: 0
Intermediate Result

**Input Graph**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2:3:</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3:4:</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2:4:5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
<td></td>
<td>1:4</td>
</tr>
</tbody>
</table>

\(<\text{nodeId 2, distance 1}\> \\
\(<\text{nodeId 3, distance 1}\> \)
Intermediate Result

Input Graph

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>2:3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3:4:</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2:4:5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
</tr>
</tbody>
</table>

<nodeId 2, distance 1>
<nodeId 3, distance 1>

Input to the next iteration
Input Graph

1 0 2:3:
2 1 3:4:
3 1 2:4:5
4 2 5:
5 2 1:4

<nodeld 4, distance 2>
<nodeld 5, distance 2>
PageRank

- Last time we looked at PageRank
  - Algorithm for ranking the importance of pages on the internet
  - Internet represented as a graph
    - Pages are vertices
    - Links are edges
- The internet is huge, graph requires parallel processing...

*How can we do PageRank in MapReduce?*
Page Rank: The Flow Model

A link from an *important page* (higher ranking page) is worth more.

A page is *important* if it is pointed to by other important pages.

Define a “rank” $r_j$ for page $j$ as:

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

“Flow” equations:

$$r_y = \frac{r_y}{2} + \frac{r_a}{2}$$
$$r_a = \frac{r_y}{2} + r_m$$
$$r_m = \frac{r_a}{2}$$
Solving the Flow Equation

3 equations, 3 unknowns, no constants

No unique solution: All solutions equivalent modulo the scale factor

Adding an additional constraint forces uniqueness:

\[ r_y + r_a + r_m = 1 \]

Gaussian Elimination can be used to find the solution.

This method will work for small graphs, but won’t scale for larger graphs

```
"Flow" equations:
r_y = r_y / 2 + r_a / 2
r_a = r_y / 2 + r_m
r_m = r_a / 2
```
Page Rank: Matrix Formulation

Stochastic Adjacency matrix $M$

$M_{ji} = 1/(d_i)$ if there is a link from $i$ to $j$, else value is 0

Let $r$ be a vector with initial importance of a page and

$$\sum_i r_i = 1$$

Then the flow equation can be written as

$$r = M \cdot r$$
Solving with Power Iteration

\[ r_y = \frac{r_y}{2} + \frac{r_a}{2} \]
\[ r_a = \frac{r_y}{2} + r_m \]
\[ r_m = \frac{r_a}{2} \]

\[
\begin{bmatrix}
  y \\
  a \\
  m 
\end{bmatrix}
= \begin{bmatrix}
  \frac{1}{2} & \frac{1}{2} & 0 \\
  \frac{1}{2} & 0 & 1 \\
  0 & \frac{1}{2} & 0 
\end{bmatrix}
\begin{bmatrix}
  y \\
  a \\
  m 
\end{bmatrix}
\]

\[ r = M \cdot r \]
Solving with Power Iteration

Given a web graph with $n$ nodes, where the vertices are pages and edges are hyperlinks.

**Power iteration**: a simple iterative scheme

Suppose there are $N$ web pages

1. **Initialize**: $r(0) = [1/N, \ldots, 1/N]^T$
2. **Iterate**: $r(t+1) = M \cdot r(t)$
3. **Stop when**: $\|r(t+1) - r(t)\|_1 < \varepsilon$
Google Formulation

\[ r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i} \]

or equivalently

\[ r = Mr \]

Does this value converge?

Does it converge to the results that we want?

Are the results reasonable?
Page Rank: Two Problems

1. Dead ends:
   - Some pages are dead ends (have no out-link)
   - Such pages cause important information to leak

2. Spider traps
   - All out-links are within the group
   - Random walk gets stuck in a trap
   - And eventually spider traps absorbs all importance
**Spider Traps**

- **Power Iteration:**
  - Set $r_j = 1/3$
  - $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$
    - And iterate

- **Example:**
  
  $r_y \quad \small 1/3 \quad 2/6 \quad 3/12 \quad 5/24 \quad 0$
  $r_a \quad \small 1/3 \quad 1/6 \quad 2/12 \quad 3/24 \quad \ldots \quad 0$
  $r_m \quad \small 1/3 \quad 3/6 \quad 7/12 \quad 16/24 \quad 1$

  All the PageRank score gets “trapped” in node $m$.

- $m$ is a spider trap

### Transition Matrix

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>½</td>
<td>½</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>½</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>½</td>
<td>1</td>
</tr>
</tbody>
</table>

- $r_y = r_y/2 + r_a/2$
- $r_a = r_y/2$
- $r_m = r_a/2 + r_m$
Solution: Teleports!

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

Power Iteration:
Set $r_j = 1/3$

\[ r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i} \]

And iterate

\[
\begin{pmatrix}
    r_y \\
    r_a \\
    r_m
\end{pmatrix} =
\begin{pmatrix}
    1/3 & 2/6 & 3/12 & 5/24 & 0 \\
    1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\
    1/3 & 1/6 & 1/12 & 2/24 & 0 \\
\end{pmatrix}
\]

Iteration 0, 1, 2, ...

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

\[
\begin{array}{c}
r_y = r_y/2 + r_a/2 \\
r_a = r_y/2 \\
r_m = r_a/2
\end{array}
\]
Solution: Teleports

Teleport with probability 1.0 at dead ends
Adjust matrix accordingly
Google's Solution

Googles solution for PageRank:

\[ r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N} \]

In matrix notation:

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

- In the **map phase** we evenly divide up each node's PageRank mass and pass each piece along outgoing edges to neighbors.
- In the **reduce phase** PageRank contributions are summed up at each destination node.
- Each MapReduce job corresponds to one iteration of the algorithm.
PageRank with MapReduce

Let us assume for now that $\beta$ is 1.0 (no teleporting)

class Mapper
    method map (nodeId n, Node N)
        p ← N.pagerank / N.adajacencyList.size
        emit(n, N) // Emit the graph structure
        for all m in N.adjacencyList
            emit(m, p) // Emit the contributions from N
PageRank with MapReduce

Class Reducer

method Reduce(nodeId n, [p1, p2, p3..])
node N ← null; s ← 0;
for all p in [p1,p2, ..]:
    if IsNode(p) then: N ← p
    else: s ← s + p
N.pagerank ← s
emit(n, N)
PageRank – MapReduce
PageRank with MapReduce

How do we account for dead ends nodes?
● Simply redistribute its PageRank to all other nodes
● One iteration requires PageRank computation + redistribution of “unused” PageRank
  ○ Track total leaked PageRank during the computation, then redistribute it as a second MapReduce job in the same iteration
PageRank with MapReduce

How do we account for dead ends nodes?
- Simply redistribute its PageRank to all other nodes
- One iteration requires PageRank computation + redistribution of “unused” PageRank
  - Track total leaked PageRank during the computation, then redistribute it as a second MapReduce job in the same iteration

What about breaking out of spider traps?
- This can also be taken care of when the leaked PageRank is redistributed from dead ends
PageRank with MapReduce

How do we account for dead ends nodes?
● Simply redistribute its PageRank to all other nodes
● One iteration requires PageRank computation + redistribution of “unused” PageRank
  ○ Track total leaked PageRank during the computation, then redistribute it as a second MapReduce job in the same iteration

What about breaking out of spider traps?
● This can also be taken care of when the leaked PageRank is redistributed from dead ends

Second Phase Redistribution Formula:
\[ p' = \alpha \left( \frac{1}{|G'|} \right) + (1 - \alpha) \left( \frac{m}{|G|} + p \right) \]
PageRank with MapReduce

**How do we account for dead ends nodes?**
- Simply redistribute its PageRank to all other nodes
- One iteration requires PageRank computation + redistribution of “unused” PageRank
  - Track total leaked PageRank during the computation, then redistribute it as a second MapReduce job in the same iteration

**What about breaking out of spider traps?**
- This can also be taken care of when the leaked PageRank is redistributed from dead ends

---

**Second Phase Redistribution Formula:**
\[
p' = \alpha \left( \frac{1}{|G|} \right) + (1 - \alpha) \left( \frac{m}{|G|} + p \right)
\]

- \( p' \): New PageRank after redistribution
- \( \alpha \): Random hop probability
- \( |G| \): Number of nodes in the entire graph
- \( m \): Total leaked PageRank
- \( p \): PageRank from first phase

- Total leaked PageRank "mass"
One iteration of PageRank requires two MapReduce jobs:
- First to distribute PageRank mass along graph edges
- Second to take care of dangling nodes and the random jump factor.

At end of each iteration, we end up with exactly the same data structure as the beginning, which is a requirement for the iterative algorithm to work.

The PageRank values of all nodes sum up to one, which ensures a valid probability distribution.
PageRank with MapReduce

How do we know when convergence is reached?
- When the ranks of pages do not change (or change by less than some small epsilon value)

For large graphs, the rank of any particular node is often so small that it underflows standard floating point representations
- A very common solution to this problem is to represent ranks using their logarithms
How many iterations are necessary for PageRank to converge?

This is a difficult question to precisely answer since it depends on many factors. According to this paper [3] convergence on a graph with 322 million edges was reached in 52 iterations. On today's web, the answer is not very meaningful:

-- web is full of spam and untrusted sites that are actively trying to game PageRank

Many strategies/algorithm has developed by web search companies

-- these algorithmic modifications impacted the convergence behavior.
Summery

We covers Graph and Page rank algorithm in MapReduce. Both are instances of a large class of iterative algorithms that share the following characteristics:

- The graph structure is represented with adjacency lists.
- Algorithms map over nodes and pass results to nodes on their adjacency lists.
- Results are aggregated for each node in the reducer.
- The graph structure itself is passed from the mapper to the reducer, such that the output is in the same form as the input.
- Algorithms are iterative and under the control of a non-MapReduce program, which checks for termination at the end of each iteration.
References


[2] Chapter: 5 Lin and Dyer