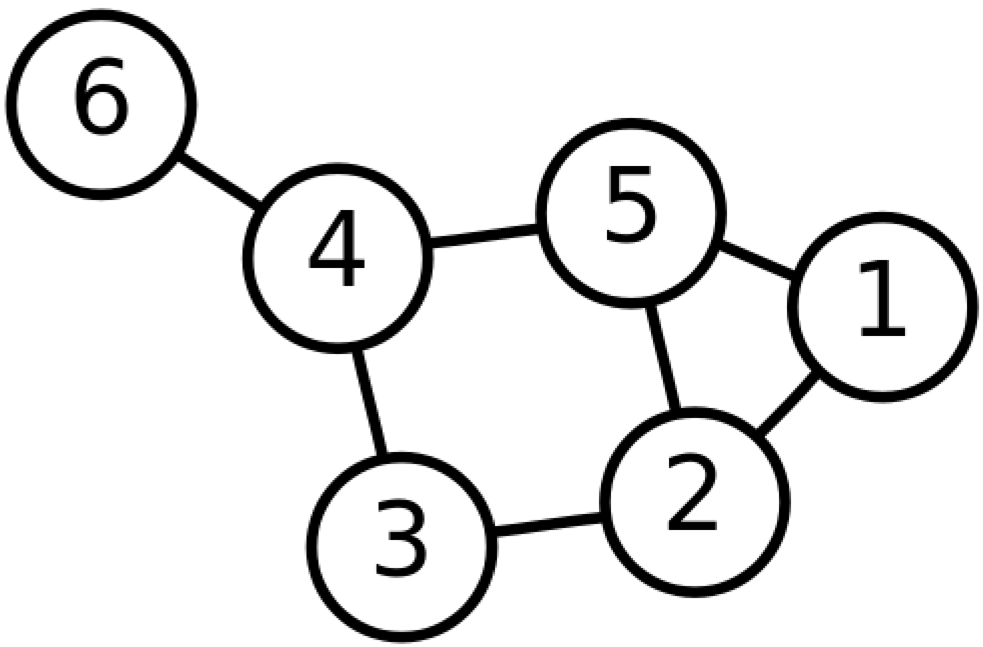
Cloud Computing

Link Analysis in the Cloud

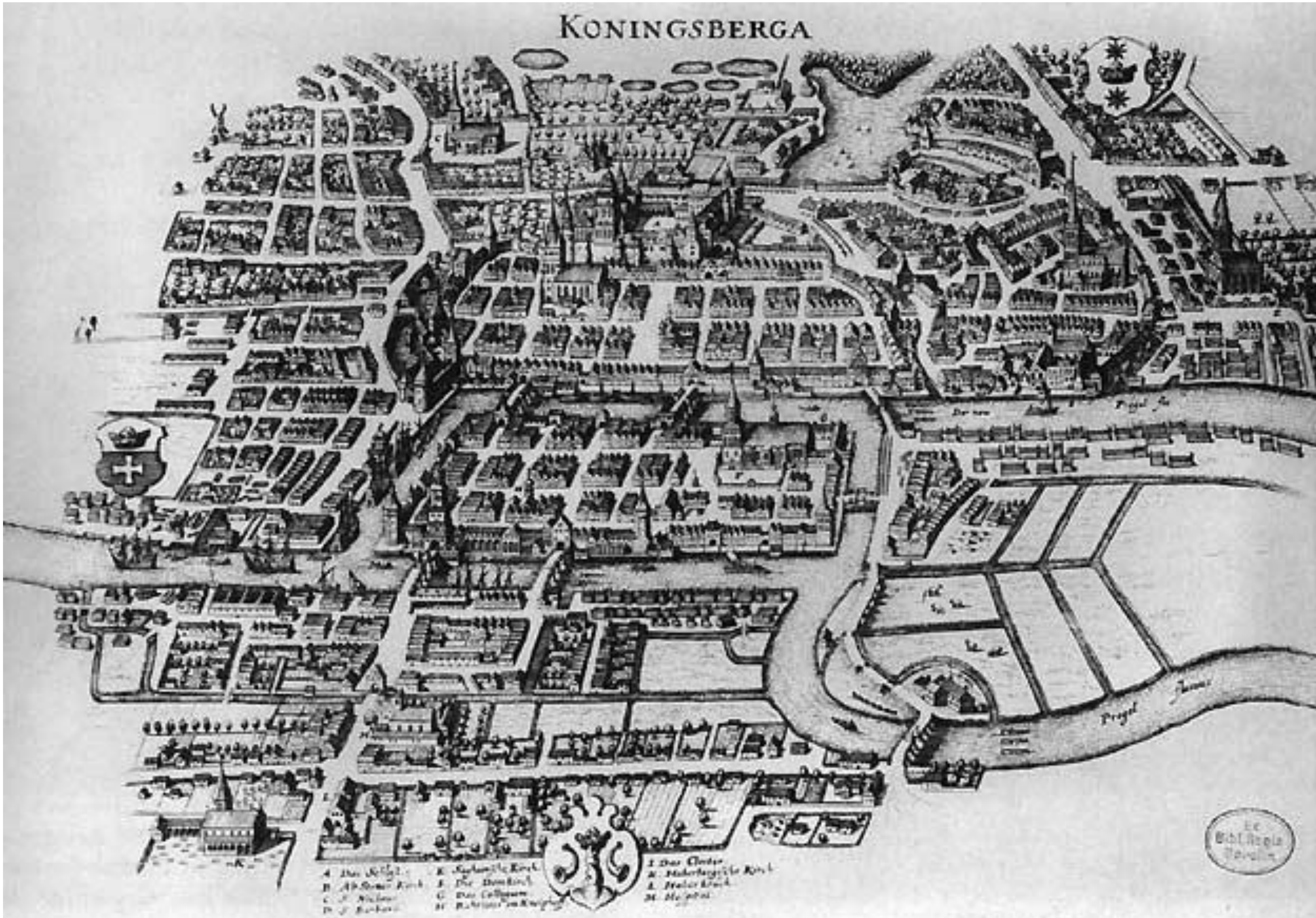
Dell Zhang
Birkbeck, University of London
2018/19

Graph Problems & Representations



What is a Graph?

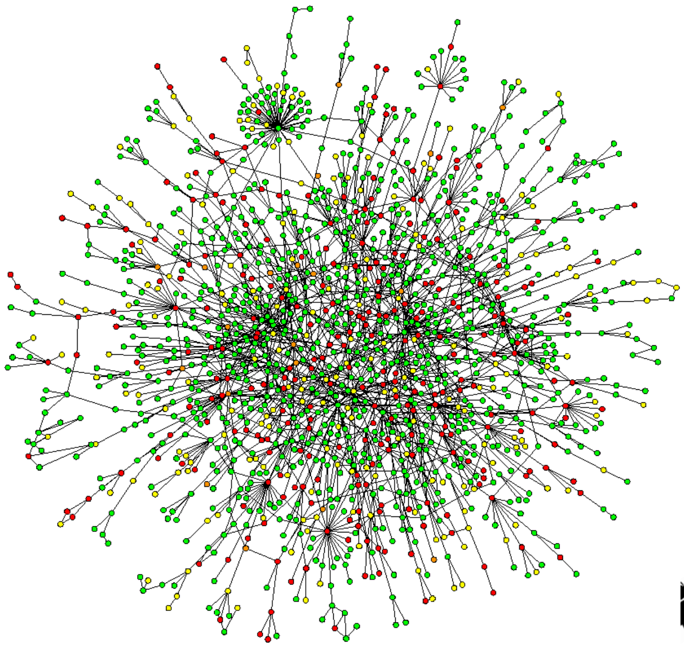
- $G = (V,E)$, where
  - V represents the set of vertices (nodes)
  - E represents the set of edges (links)
  - Both vertices and edges may contain additional information (e.g., edge weights)
- Different types of graphs:
  - directed vs. undirected edges
  - presence or absence of cycles

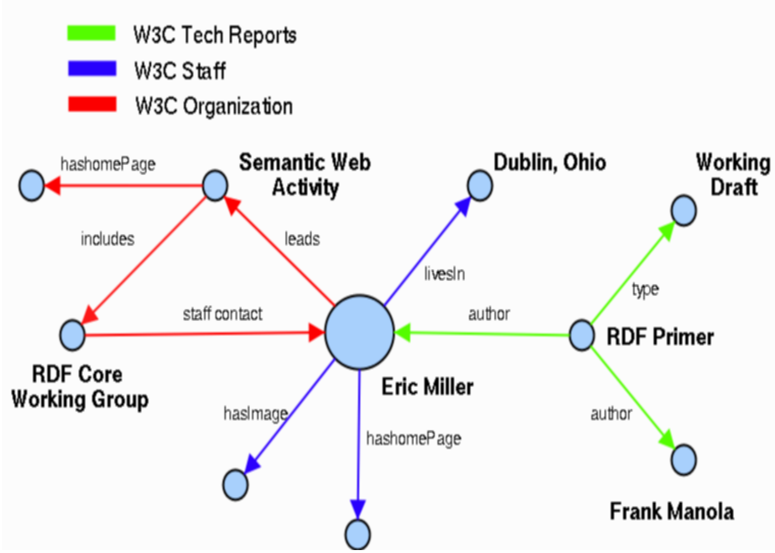


Source: Wikipedia (Königsberg)

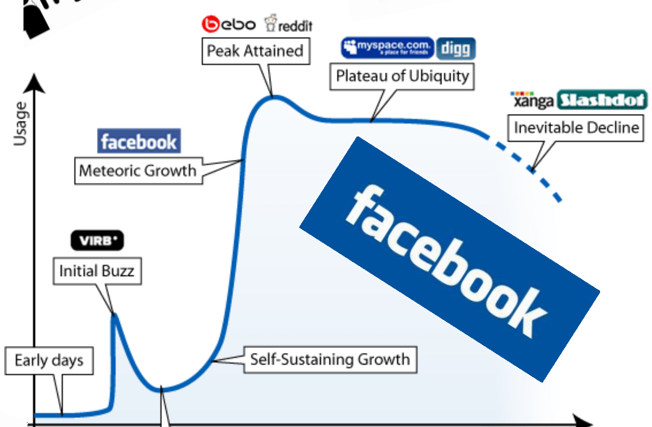
We See Graphs Everywhere

- Ubiquitous network (graph) data
  - Technological Network
    - Internet
  - Information Network
    - WWW, Sematic Web/Ontologies, XML/RD
  - – Social network
  - – Biological Network
  - – Financial Network
  - – Transportation Network

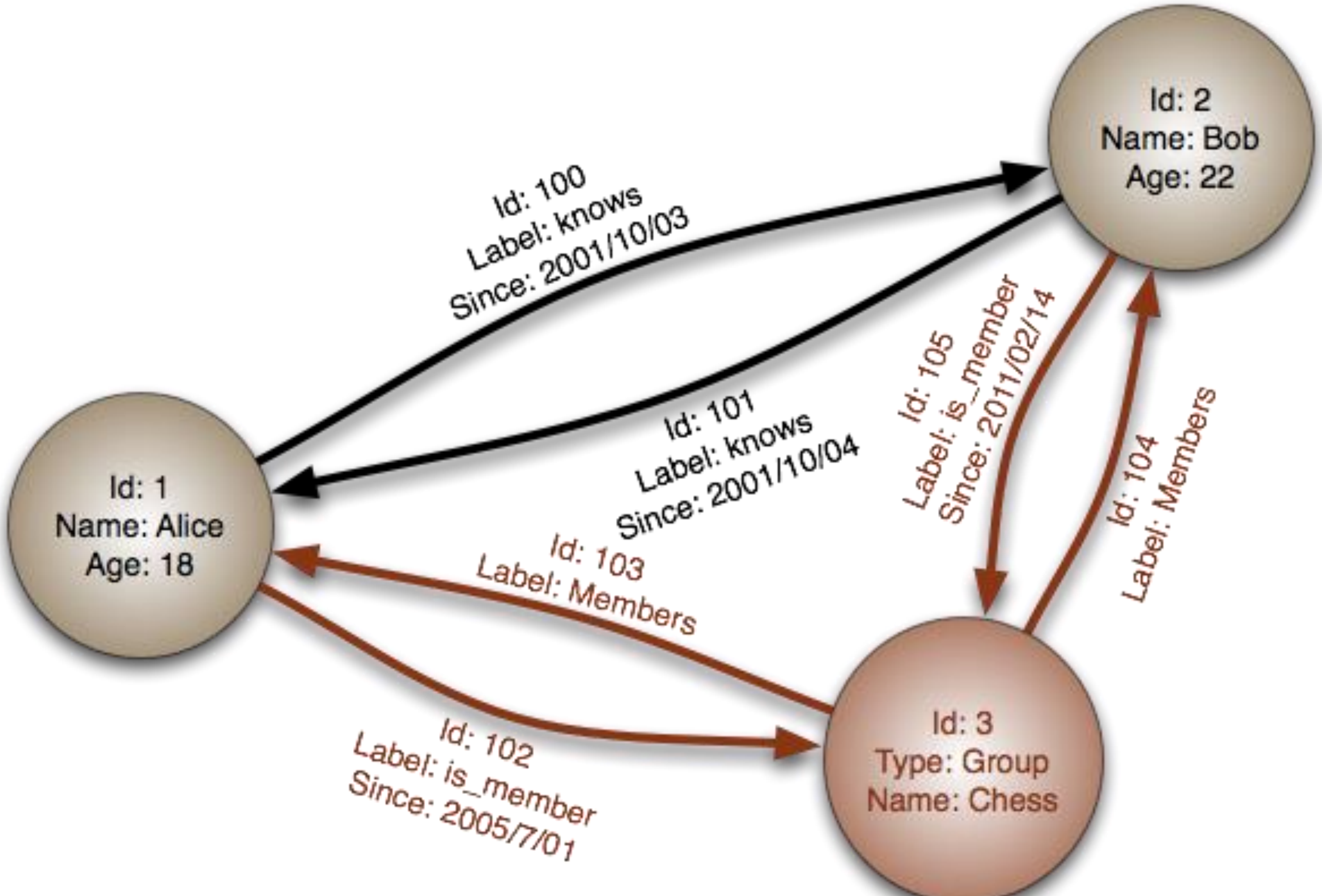




Semantic Search, Guha et. al., WWW'03



http://belanger.wordpress.com/2007/06/28/the-ebb-and-flow-of-social-networking/


Id: 2
Name: Bob
Age: 22
Id: 100
Label: knows
Since: 2001/10/03
Id: 1
Name: Alice
Age: 18
Id: 101
Label: knows
Since: 2001/10/04
Id: 105
Label: is_member
Since: 2011/02/14
Id: 104
Label: Members
Id: 103
Label: Members
Id: 102
Label: is_member
Since: 2005/7/01
Id: 3
Type: Group
Name: Chess

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telecommunication companies laying down fibre
- Finding max flow
  - Airline scheduling

Some Graph Problems

- Identify “special” nodes and communities
  - Breaking up terrorist cells, spread of avian flu
- Bipartite matching
  - Monster.com, Match.com
- And of course... PageRank

Challenge in Dealing with Graph Data

- Flat Files
  - No query support
- RDBMS
  - Can store the graph
  - But limited support for graph query
    - Connect-By (Oracle)
    - Common Table Expressions (CTEs) (Microsoft)
    - Temporal Table

Native Graph Databases

- An Emerging Field
  - http://en.wikipedia.org/wiki/Graph_database
- Storage and Basic Operators
  - **Neo4j** (an open source graph database), InfiniteGraph, VertexDB, …
- Distributed Graph Processing (mostly in-memory-only)
  - Google's Pregel, GraphLab, …

The Graph Analytics Industry

- Status of Practice
  - Graph data in many industries
  - Graph analytics are powerful and can bring great business values/insights
  - Graph analytics not utilized enough in enterprises due to lack of available platforms/tools (except leading tech companies which have high caliber in house engineering teams and resources)

Graphs and MapReduce

- Graph algorithms typically involve:
  - Performing computations at each node: based on node features, edge features, and local link structure
  - Propagating computations: “traversing” the graph
- Key questions:
  - How do you represent graph data in MapReduce?
  - How do you traverse a graph in MapReduce?

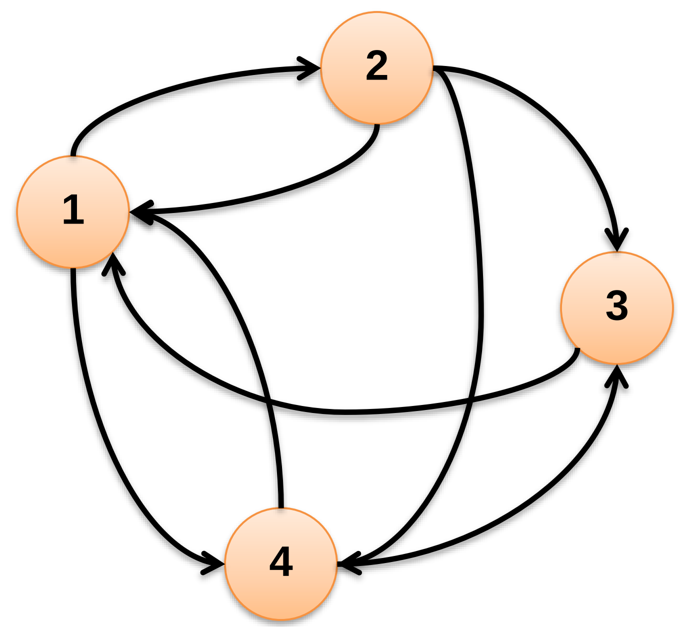
Representing Graphs

- Two common representations
  - Adjacency matrix
  - Adjacency list

Adjacency Matrices

- Represent a graph as an $n \times n$ square matrix M
  - $n = |\mathrm{V}|$
  - $M_{ij} = 1$ means a link from node i to j

	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



Adjacency Matrices: Critique

- Advantages:
  - Amenable to mathematical manipulation
  - Iteration over rows and columns corresponds to computations on out-links and in-links
- Disadvantages:
  - Lots of zeros for sparse matrices
  - Lots of wasted space

Adjacency Lists

- Take adjacency matrices… and throw away all the zeros

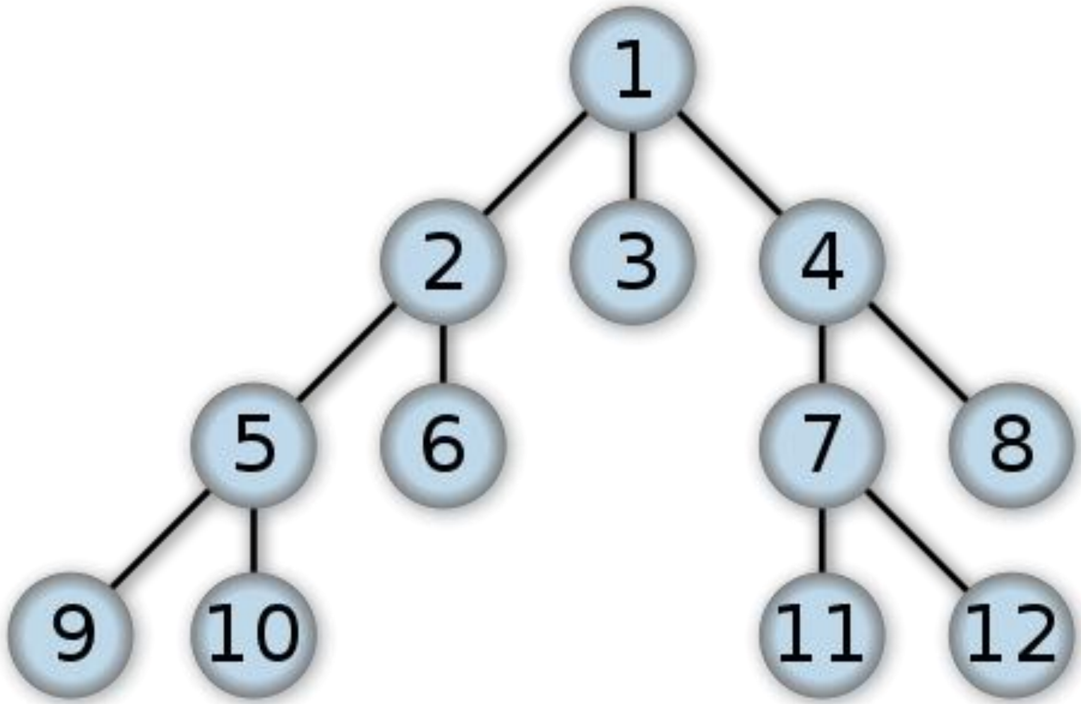
	1	**2**	**3**	**4**
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

Adjacency Lists: Critique

- Advantages:
  - Much more compact representation
  - Easy to compute over out-links
- Disadvantages:
  - Much more difficult to compute over in-links

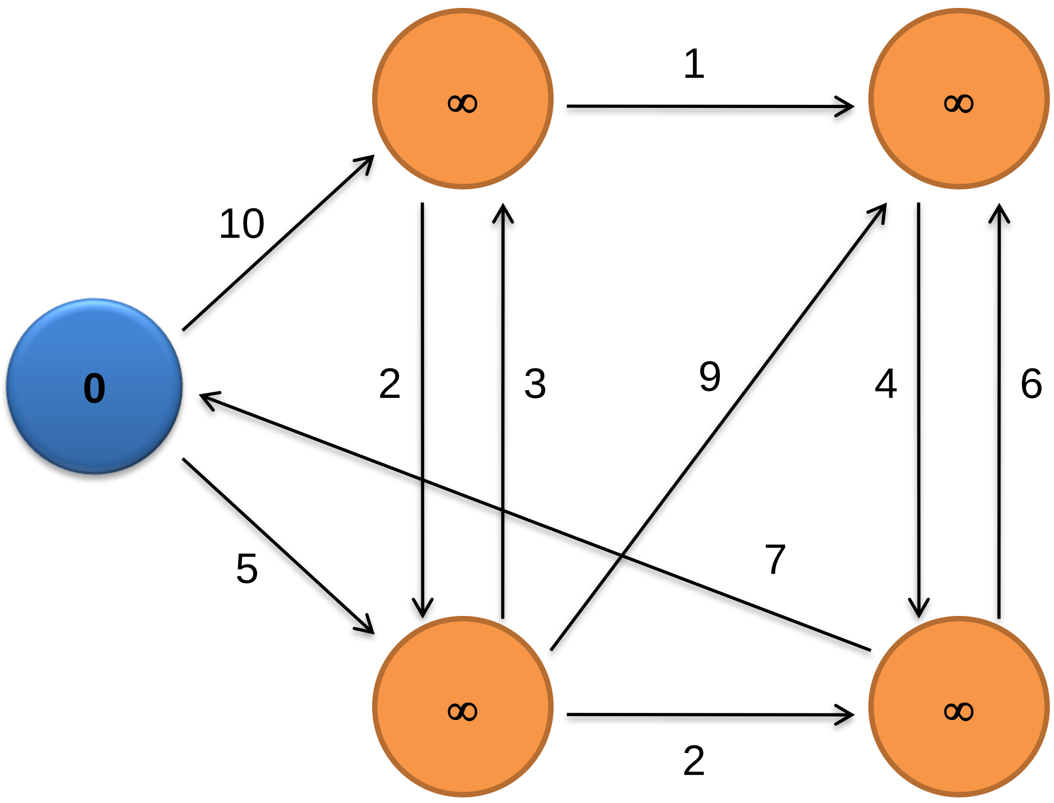
Parallel Breadth-First Search



Single Source Shortest Path

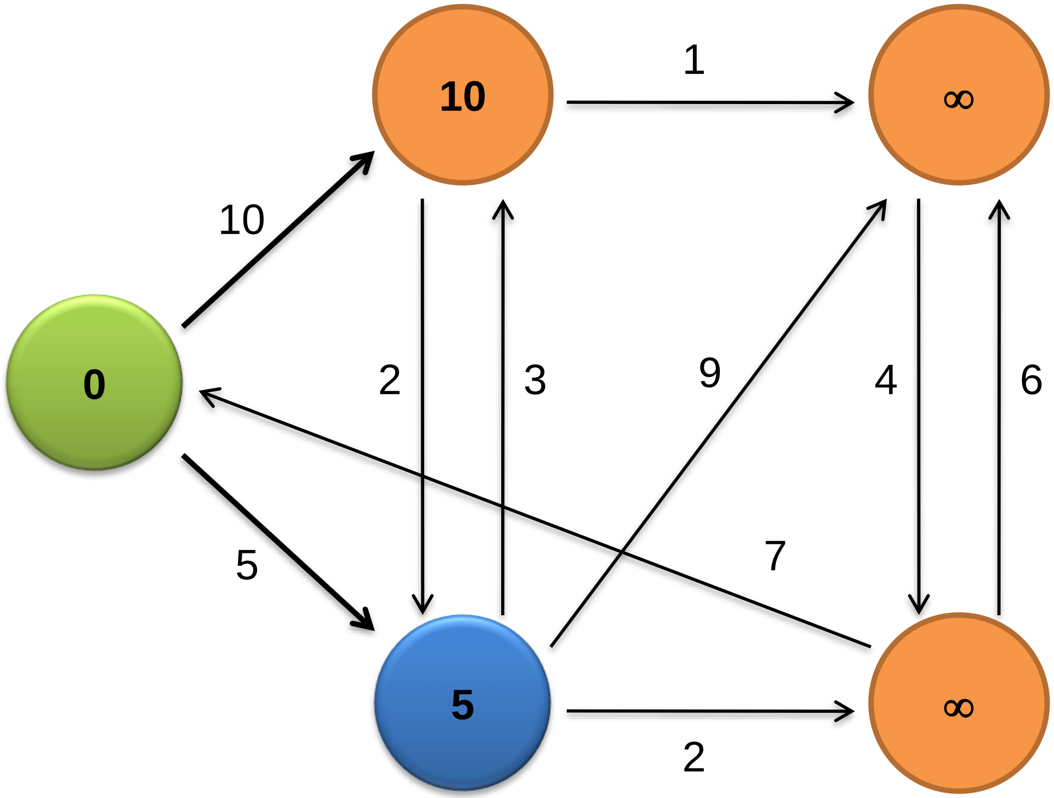
- **Problem:** find shortest path from a source node to one or more target nodes
  - "shortest" might also mean lowest weight or cost
- First, a refresher: Dijkstra's algorithm

Dijkstra's Algorithm



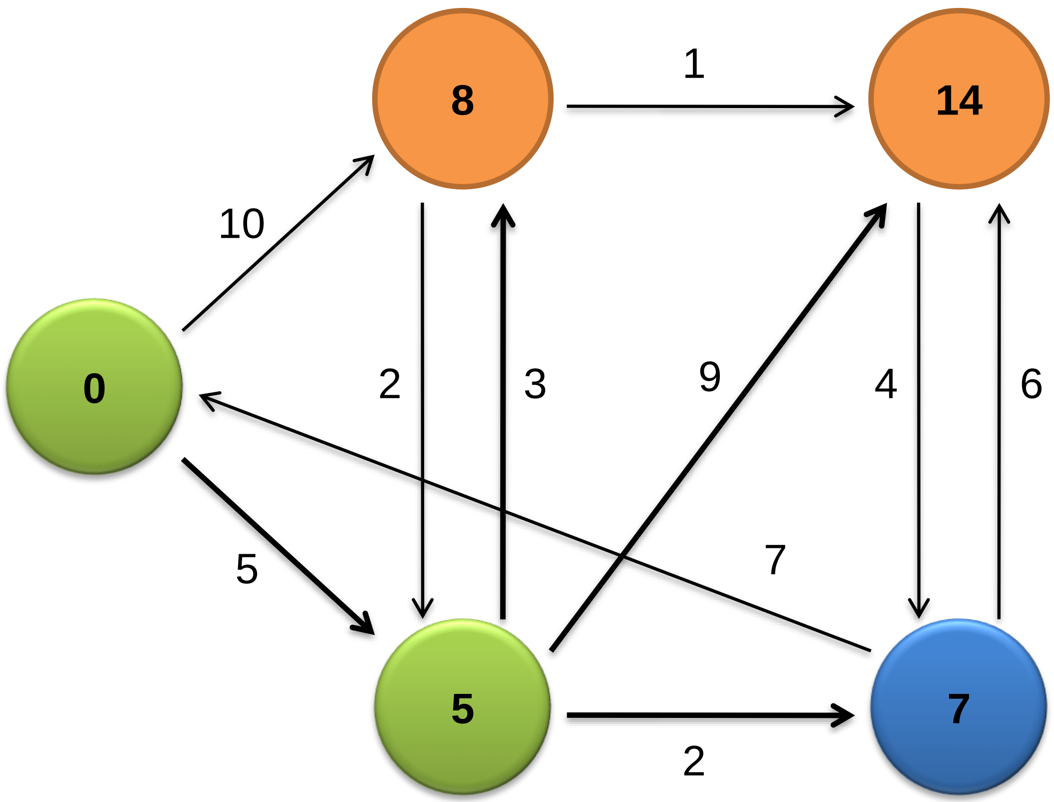
Example from CLR

Dijkstra's Algorithm



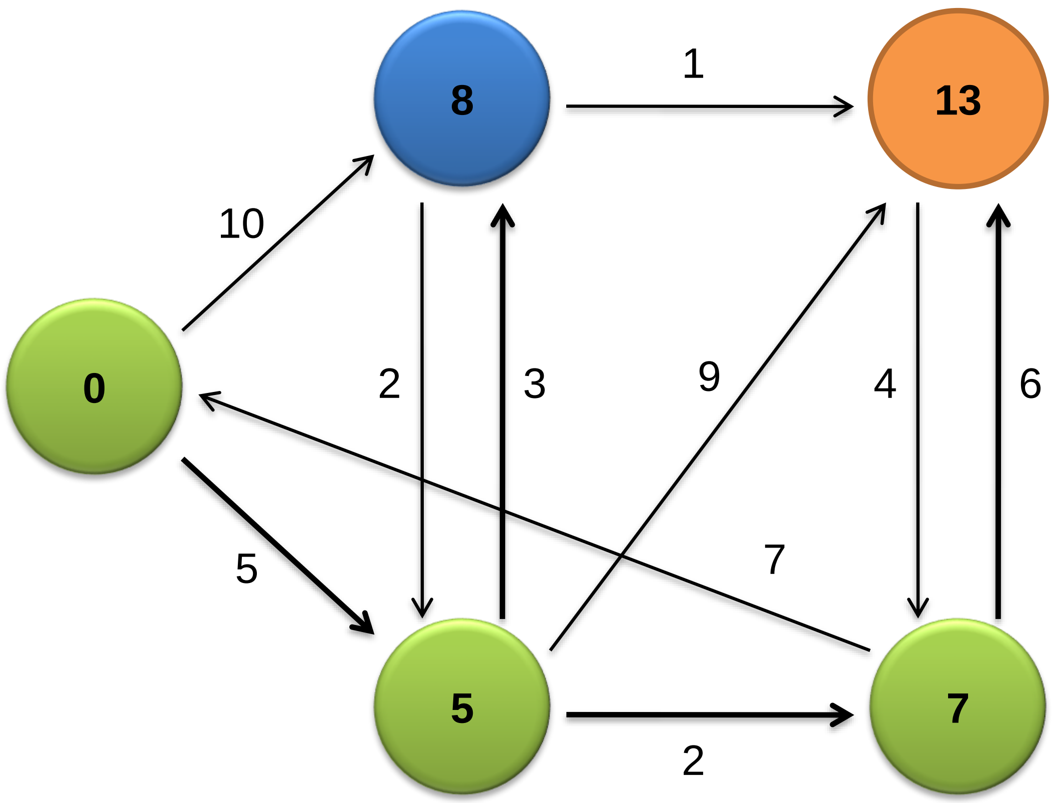
Example from CLR

Dijkstra's Algorithm



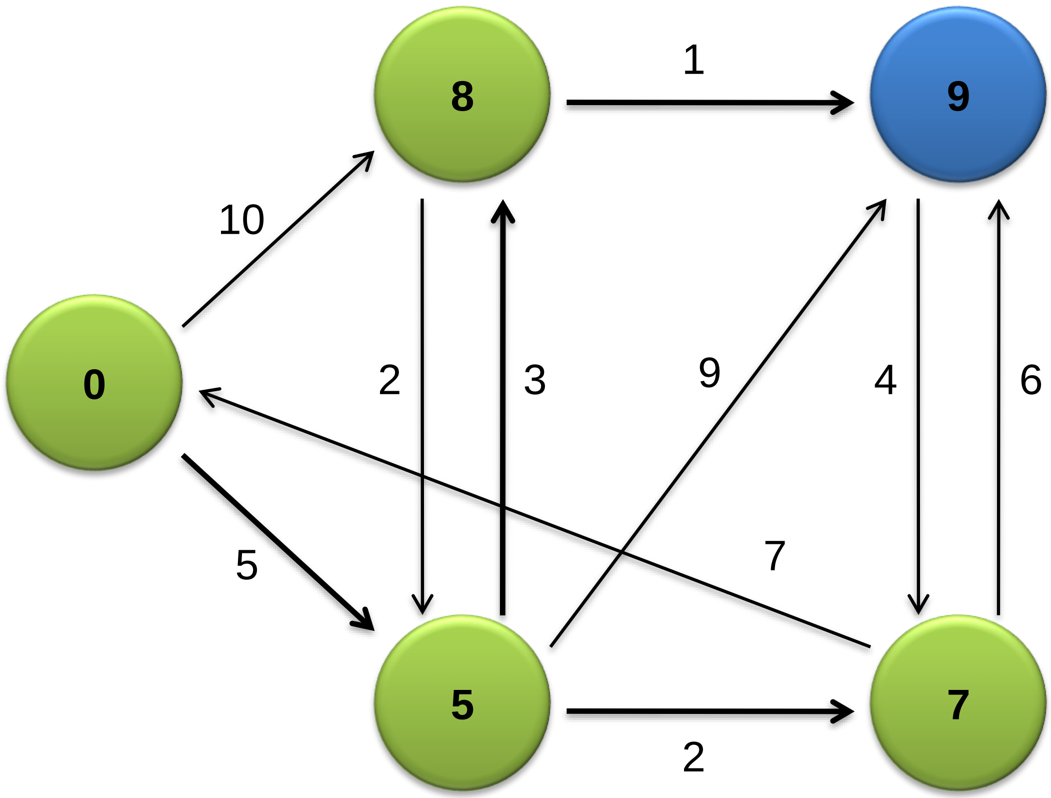
Example from CLR

Dijkstra's Algorithm



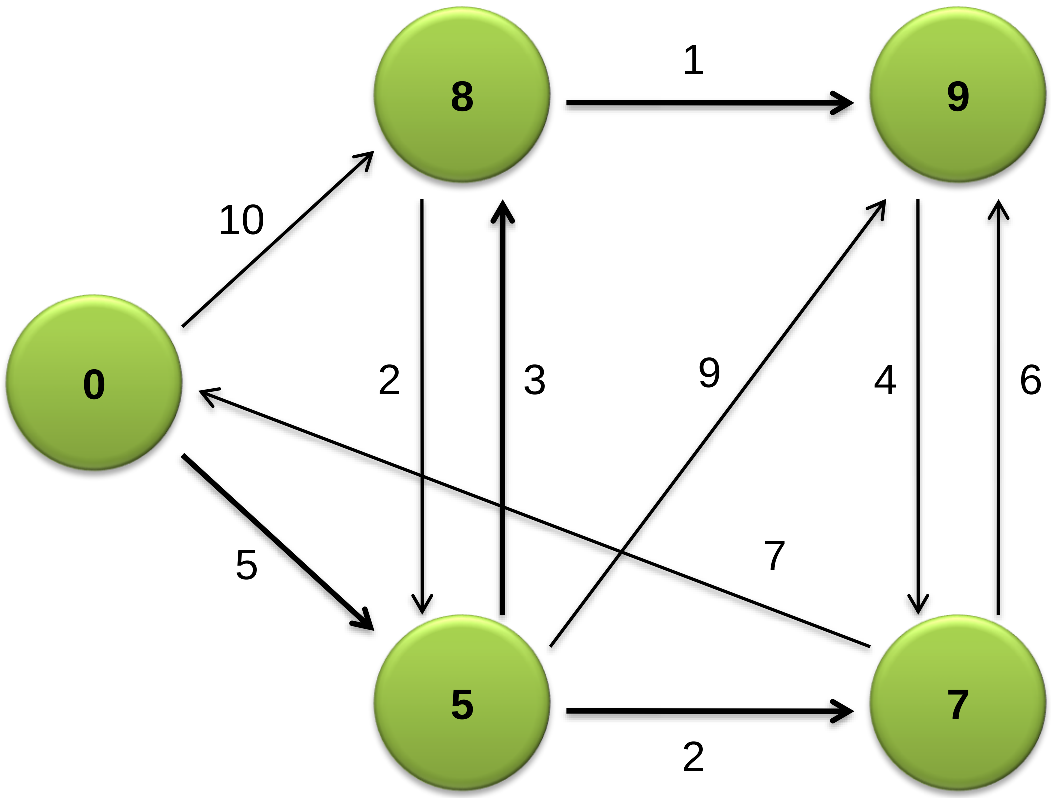
Example from CLR

Dijkstra's Algorithm



Example from CLR

Dijkstra’s Algorithm



Example from CLR

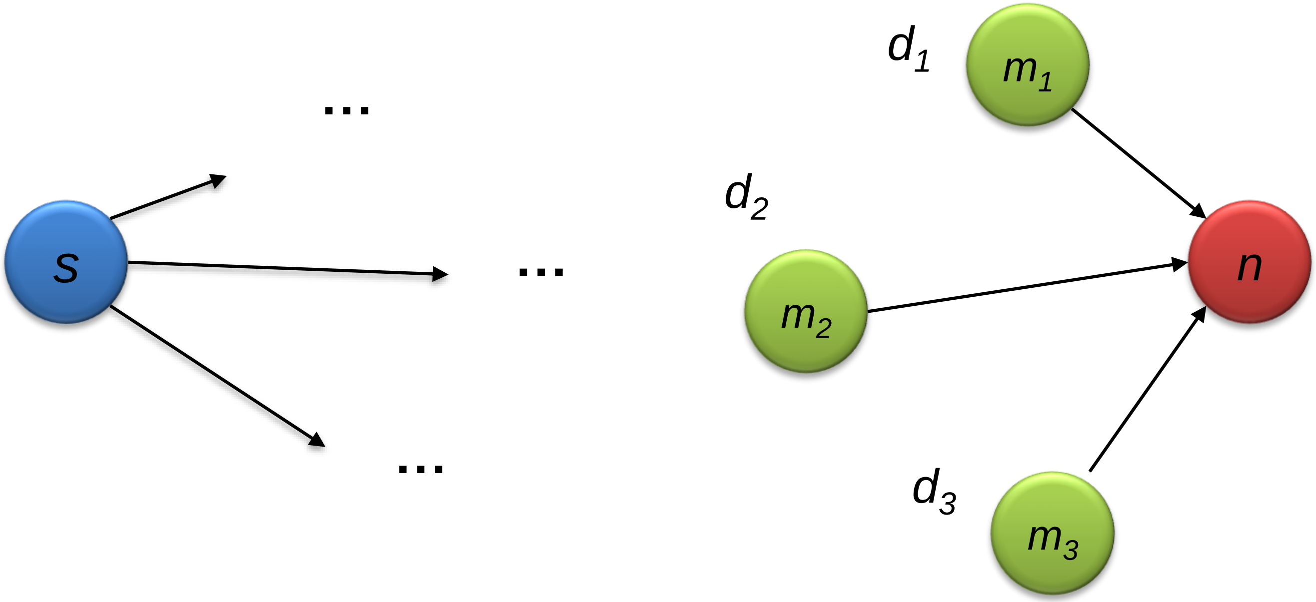
Single Source Shortest Path

- **Problem:** find shortest path from a source node to one or more target nodes
  - "shortest" might also mean lowest weight or cost
- On a single machine: Dijkstra's algorithm
- MapReduce: Parallel Breadth-First Search (BFS)
  - Consider simplest case of equal edge weights first
  - Solution to the problem can be defined inductively

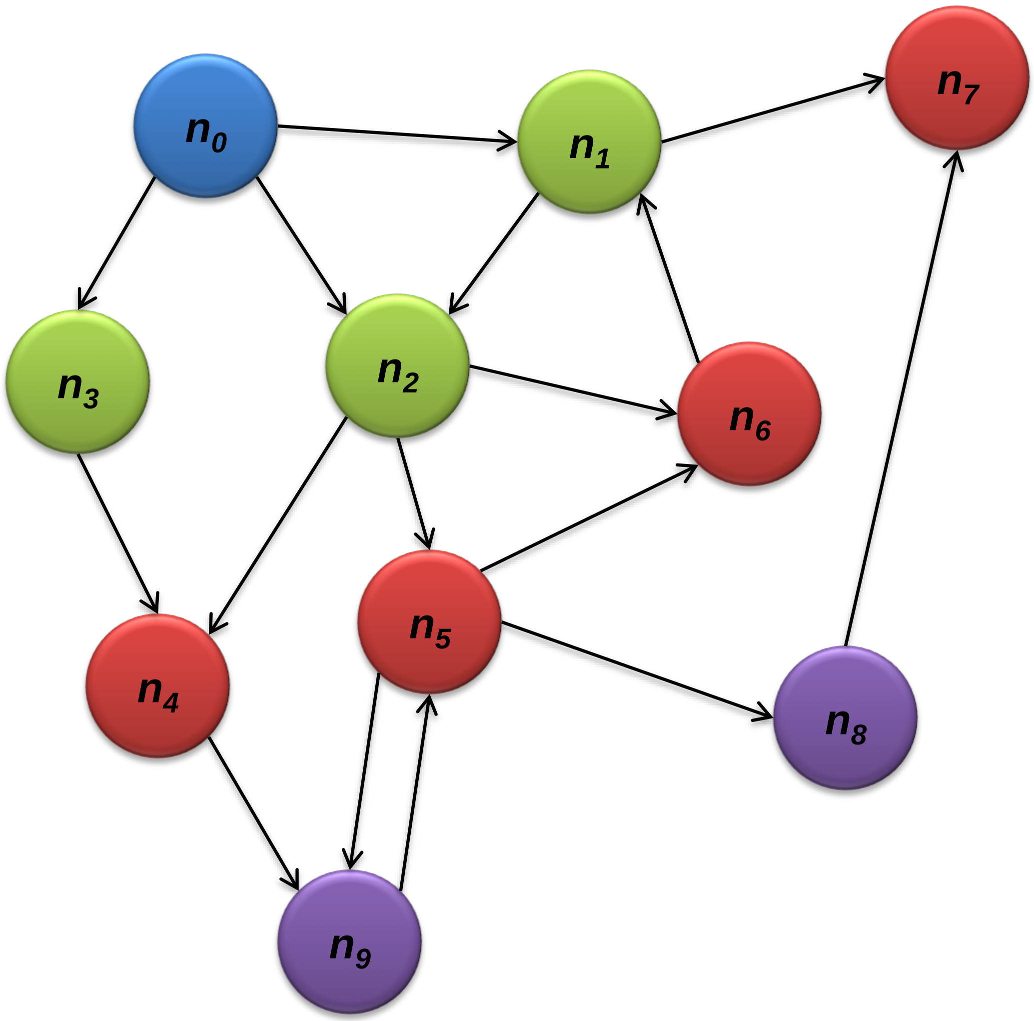
Finding the Shortest Path

- Here's the intuition:
  - Define: b is reachable from a if b is in the adjacency list of a
  - $\textsc{DistanceTo}(s) = 0$
  - For all nodes p reachable from s: $\textsc{DistanceTo}(p) = 1$
  - For all nodes n reachable from some other set of nodes M: $\textsc{DistanceTo}(n) = 1 + \min_{m \in M} \textsc{DistanceTo}(m)$

Finding the Shortest Path



Visualizing Parallel BFS



Source: Wikipedia (Wave)

From Intuition to Algorithm

- Data representation:
  - Key:
    node n
  - Value:
    d (distance from start),
    adjacency list (list of nodes reachable from n)
  - Initialization:
    for all nodes except the start node, $d = \infty$.

From Intuition to Algorithm

- Mapper:
  - $\forall m \in$ adjacency list: emit $(m, d + 1)$
- Sort/Shuffle
  - Groups distances by reachable nodes
- Reducer:
  - Selects the minimum distance path for each reachable node
  - Additional bookkeeping needed to keep track of the actual path

Multiple Iterations Needed

- Each MapReduce iteration advances the "known frontier" by one hop
  - Subsequent iterations include more and more reachable nodes as frontier expands
  - Multiple iterations are needed to explore entire graph
- Preserving graph structure:
  - Problem: Where did the adjacency list go?
  - Solution: mapper emits (n, adjacency list) as well

BFS Pseudo-Code

```
class Mapper
    method Map(nid n, node N)
        d ← N.Distance
        Emit(nid n, N)                                  ▷ Pass along graph structure
        for all nodeid m ∈ N.AdjacencyList do
            Emit(nid m, d + 1)                          ▷ Emit distances to reachable nodes

class Reducer
    method Reduce(nid m, [d1, d2, ...])
        dmin ← ∞
        M ← ∅
        for all d ∈ counts [d1, d2, ...] do
            if IsNode(d) then
                M ← d                                   ▷ Recover graph structure
            else if d < dmin then                       ▷ Look for shorter distance
                dmin ← d
        M.Distance ← dmin                               ▷ Update shortest distance
        Emit(nid m, node M)
```

Stopping Criterion

- How many iterations are needed in parallel BFS (equal edge weight case)?
- Convince yourself: when a node is first “discovered”, we’ve found the shortest path
- Now answer the question...
  - Six degrees of separation?
- Practicalities of implementation in MapReduce

Weighted Edges

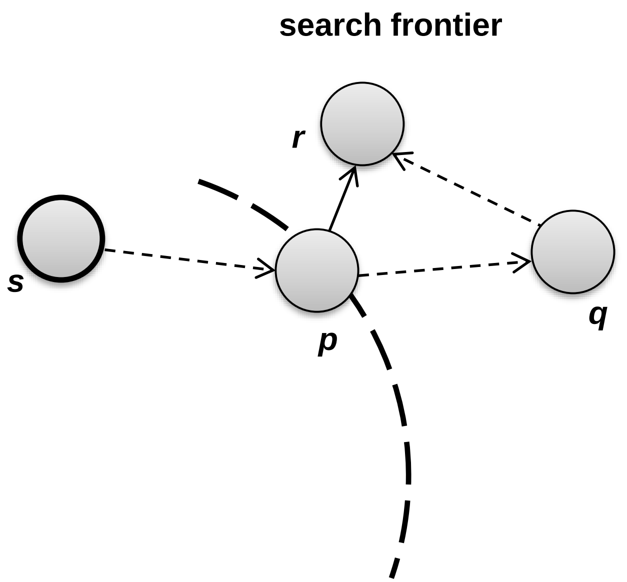
- Now add positive weights to the edges
  - Why can't edge weights be negative?
- Simple change: adjacency list now includes a weight w for each edge
  - In mapper, emit $(m, d + w_p)$ instead of $(m, d + 1)$ for each node m
- That's it?

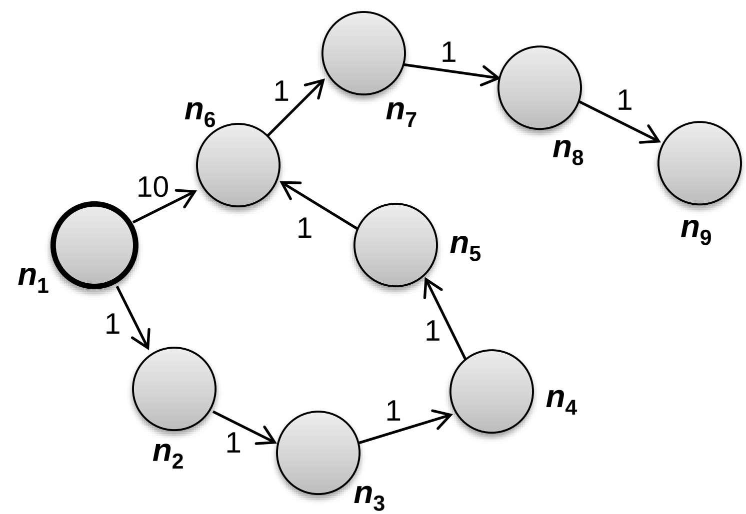
Stopping Criterion

- How many iterations are needed in parallel BFS (positive edge weight case)?
- Convince yourself: when a node is first "discovered", we've found the shortest path

Not true!

Additional Complexities





How many iterations are required to discover the shortest distances to all nodes from n_1?

Stopping Criterion

- How many iterations are needed in parallel BFS (positive edge weight case)?
- Practicalities of implementation in MapReduce

Comparison to Dijkstra

- Dijkstra’s algorithm is more efficient
  - At any step it only pursues edges from the minimum-cost path inside the frontier
- MapReduce explores all paths in parallel
  - Lots of “waste”
  - Useful work is only done at the “frontier”
- Why can’t we do better using MapReduce?

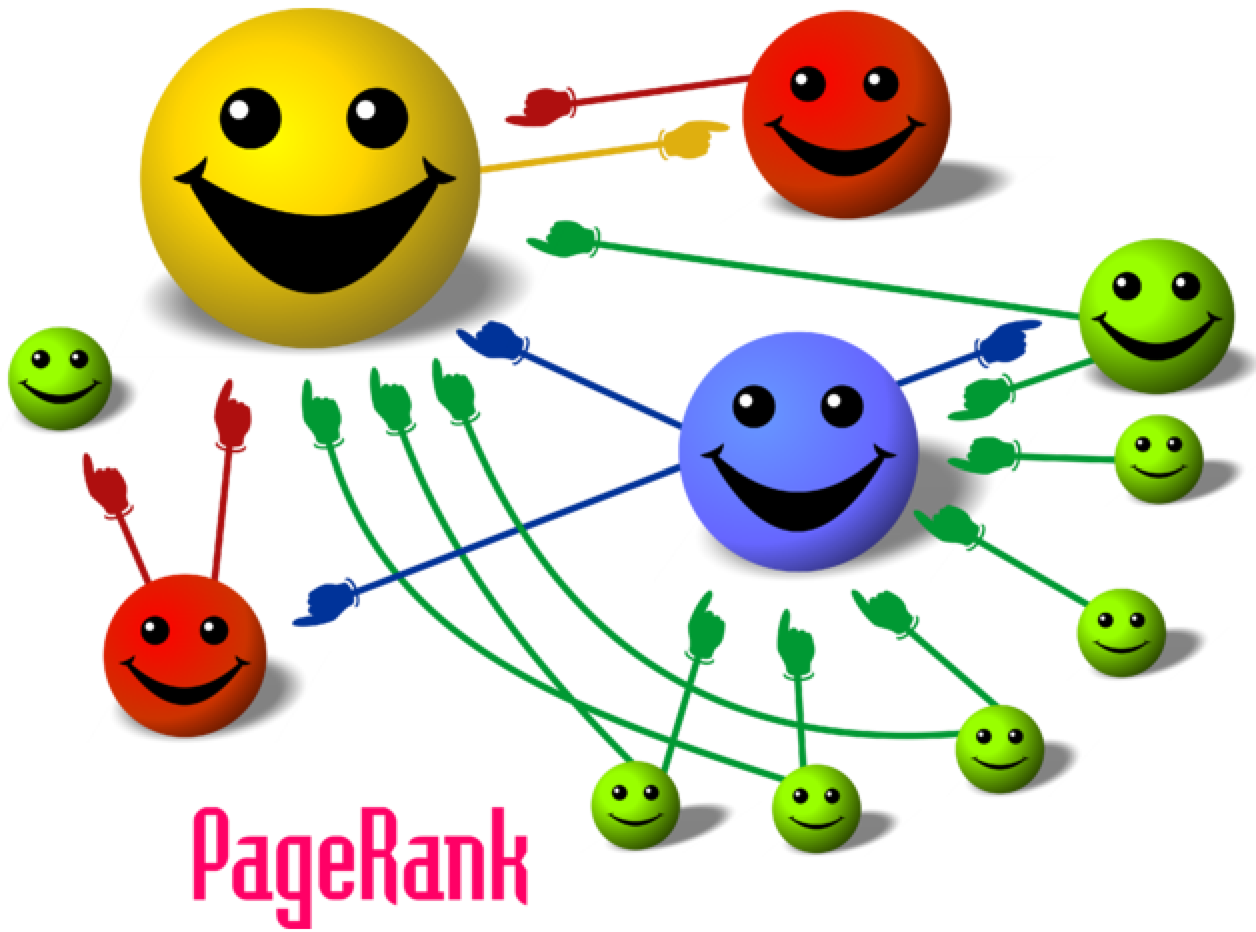
Implementation on Hadoop

http://goo.gl/TEoU4

Graphs and MapReduce

- Generic recipe:
  - Represent graphs as adjacency lists
  - Perform local computations in mapper
  - Pass along partial results via out-links, keyed by destination node
  - Perform aggregation in reducer on in-links to a node
  - Iterate until convergence: controlled by external “driver”
  - Don’t forget to pass the graph structure between iterations

PageRank



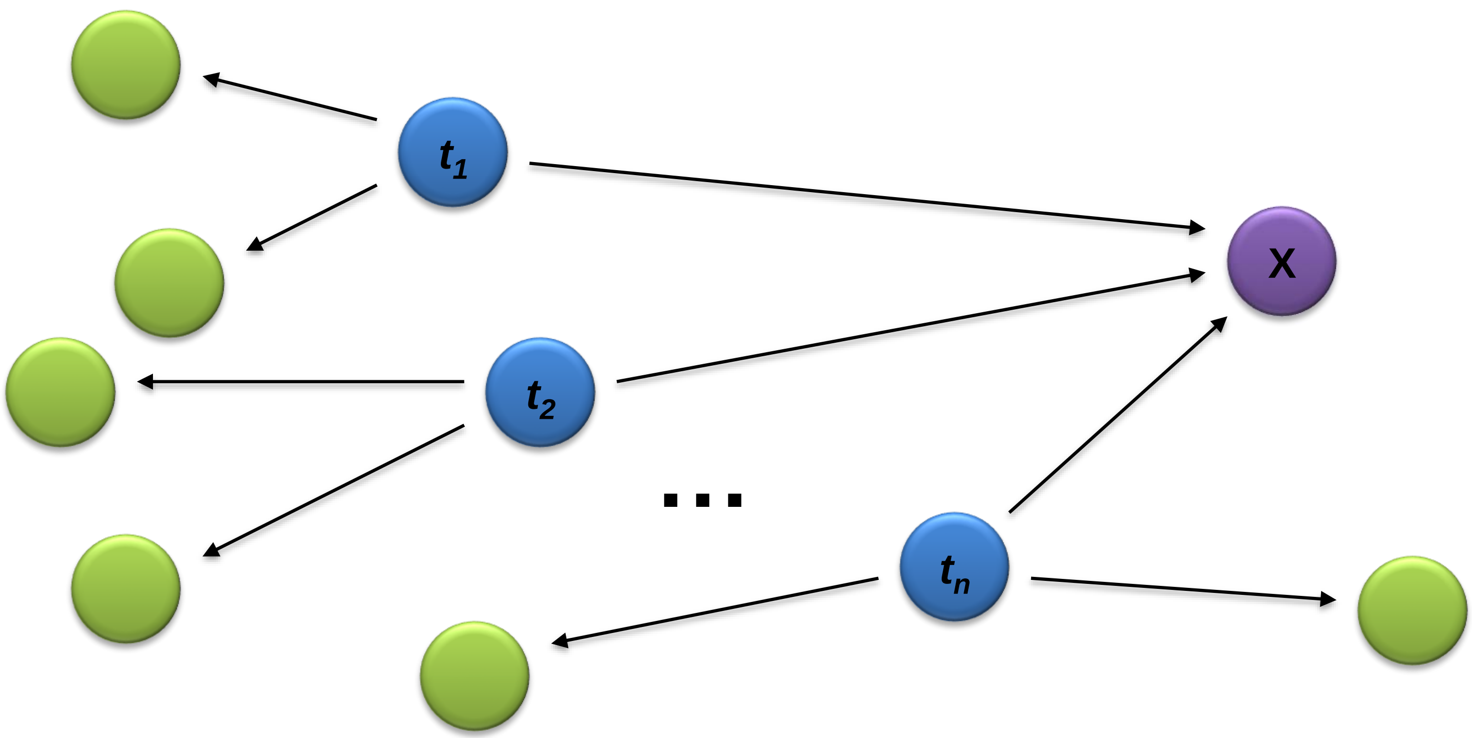
Random Walks over the Web

- Random surfer model:
  - User starts at a random Web page
  - User randomly clicks on links, surfing from page to page
- PageRank
  - Characterizes the amount of time spent on any given page
  - Mathematically, a probability distribution over pages

Random Walks over the Web

- PageRank captures the notion of page importance
  - Correspondence to human intuition?
  - One of thousands of features used in Web search
  - Note: query-independent

PageRank: Simplified

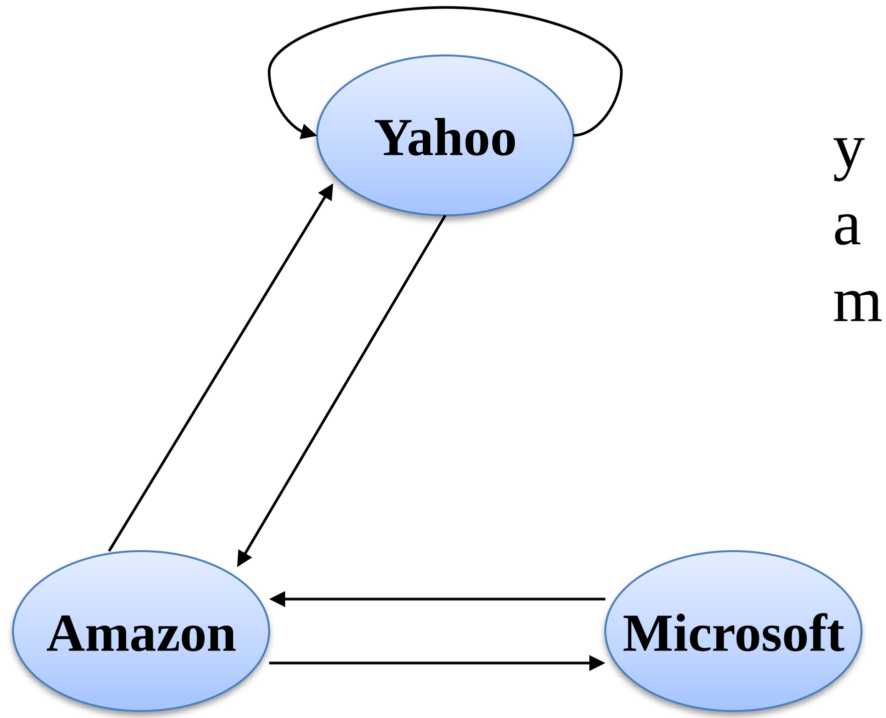


PageRank: Simplified

- Given page x with in-links $t_1 \ldots t_\mathrm{n}$, where
  - $C(t)$ is the out-degree of t

$$PR(x) = \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$

Example: the Web in 1839



$$
\begin{array}{c|ccc|}
 & y & a & m \\
\hline
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\hline
\end{array}
$$

Simulating a Random Walk

- Start with the vector $\mathbf{v} = [1,1,\ldots,1]$ representing the idea that each Web page is given one unit of *importance*.
- Repeatedly apply the matrix M to $\mathbf{v}$, allowing the importance to flow like a random walk.
- Limit exists, but about 50 iterations is sufficient to estimate final distribution.

Example: the Web in 1839

- Equations $\mathbf{v} = M\mathbf{v}$:

$$y = y/2 + a/2$$

$$a = y/2 + m$$

$$m = a/2$$

$$\begin{array}{ccccccccc} y & & 1 & 1 & 5/4 & 9/8 & & 6/5 \\ a & = & 1 & 3/2 & 1 & 11/8 & \ldots & 6/5 \\ m & & 1 & 1/2 & 3/4 & 1/2 & & 3/5 \end{array}$$

Solving the Equations

- Because there are no constant terms, these 3 equations in 3 unknowns do not have a unique solution.
- Add in the fact that $y + a + m = 3$ to solve.
- In Web-sized examples, we cannot solve by Gaussian elimination, but we need to use the power method (= iterative solution).

Computing PageRank

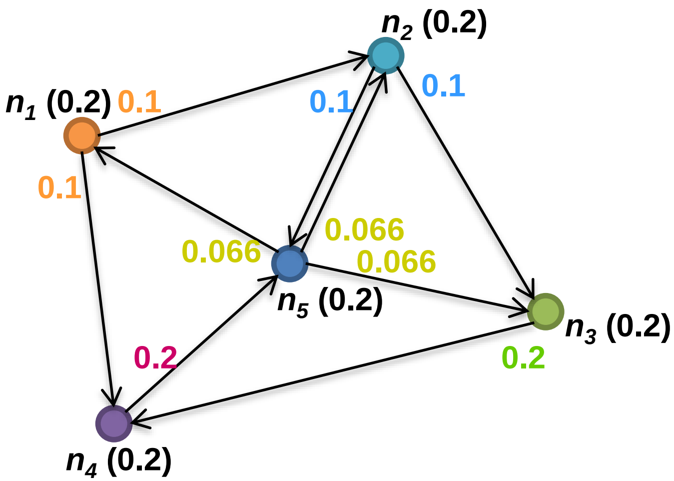
- Properties of PageRank
  - Can be computed iteratively
  - Effects at each iteration are local

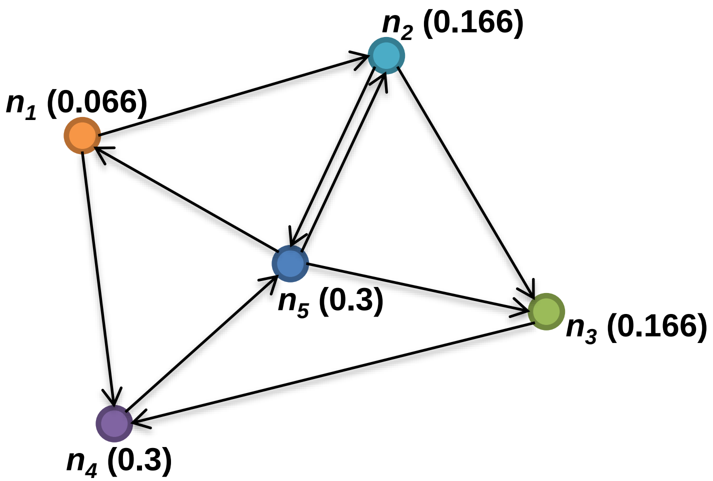
Computing PageRank

- Sketch of algorithm:
  - Start with seed PR_i values
  - Each page distributes its PR_i “credit” to all of its out-links
  - Each page adds up the “credits” from all of its in-links to compute PR_{i+1}
  - Iterate until the values converge

Sample PageRank Iterations

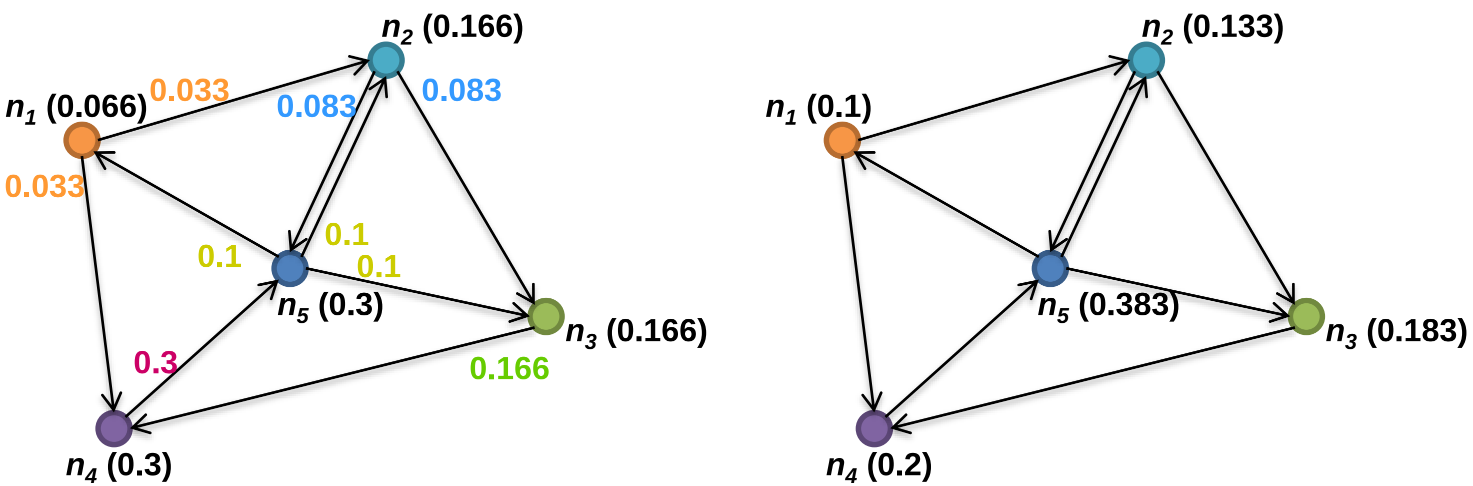
Iteration 1



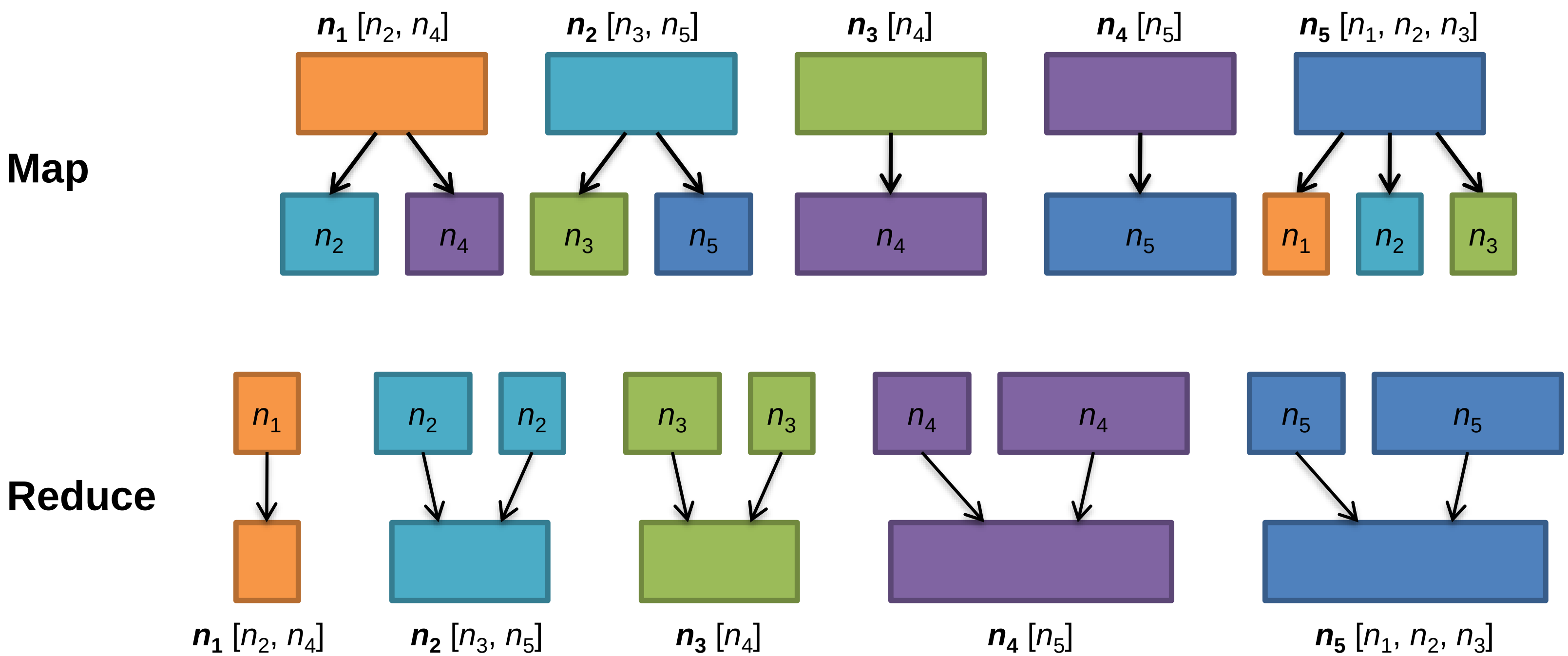


Sample PageRank Iterations

Iteration 2



PageRank in MapReduce



PageRank Pseudo-Code

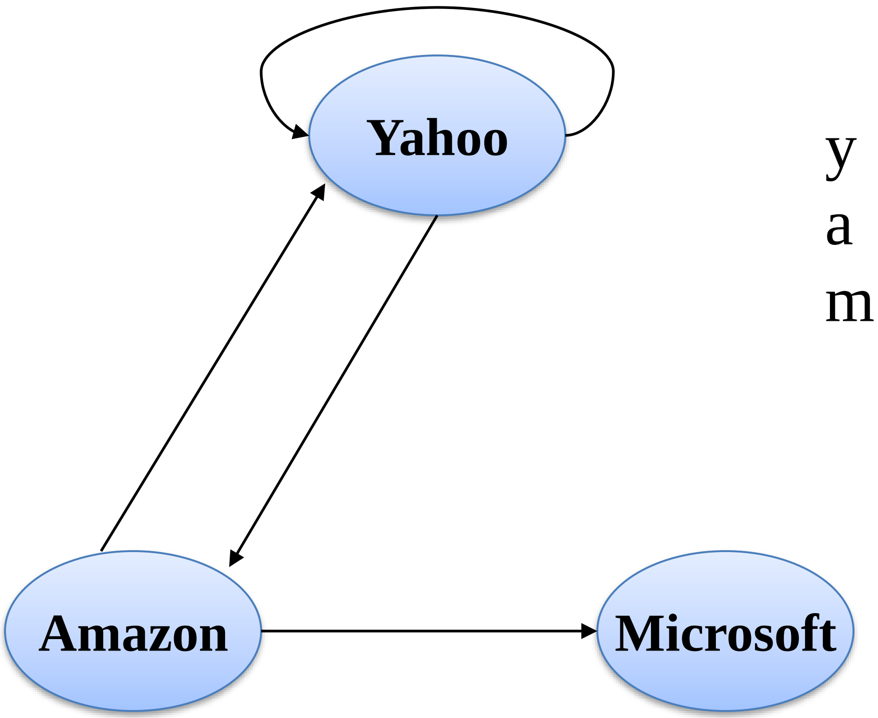
```
class Mapper
    method Map(nid n, node N)
        p ← N.PageRank/|N.AdjacencyList|
        Emit(nid n, N)                              ▷ Pass along graph structure
        for all nodeid m ∈ N.AdjacencyList do
            Emit(nid m, p)                          ▷ Pass PageRank mass to neighbors

class Reducer
    method Reduce(nid m, [p1, p2, ...])
        M ← ∅
        for all p ∈ counts [p1, p2, ...] do
            if IsNode(p) then
                M ← p                               ▷ Recover graph structure
            else
                s ← s + p                           ▷ Sum incoming PageRank contributions
        M.PageRank ← s
        Emit(nid m, node M)
```

Real-World Problems

- Some pages are "dead ends" (no out-links).
  - Such a page causes importance to leak out.
- Some other (groups of) pages are *spider traps* (all out-links are within the group).
  - Eventually spider traps absorb all importance.

Microsoft becomes a dead end



$$\begin{array}{c|ccc|} & y & a & m \\ \hline y & 1/2 & 1/2 & 0 \\ a & 1/2 & 0 & 0 \\ m & 0 & 1/2 & 0 \\ \hline \end{array}$$

Microsoft becomes a dead end

- Equations $\mathbf{v} = M\mathbf{v}$:

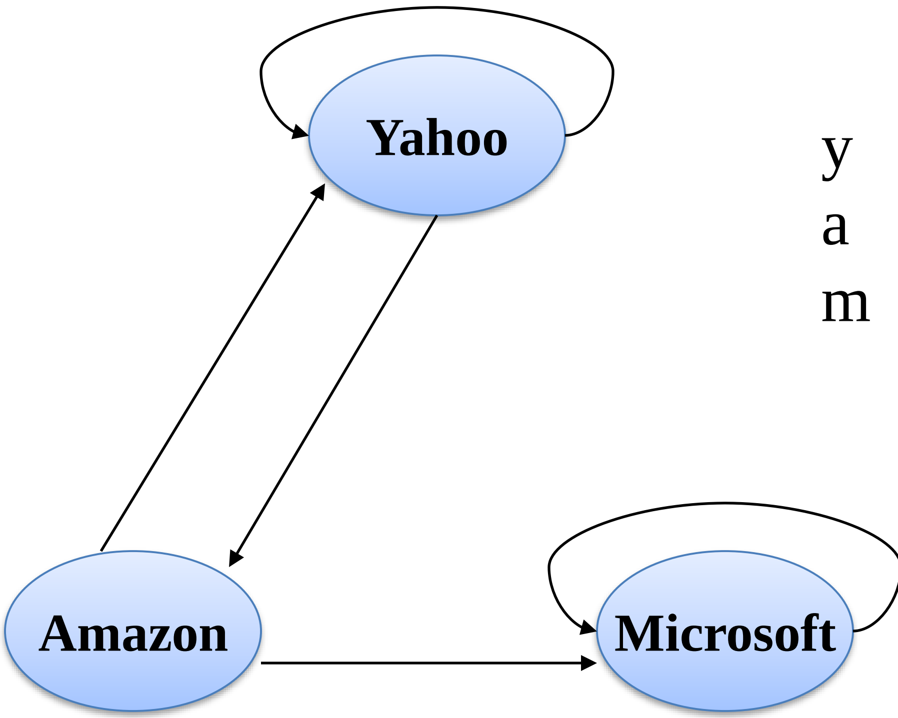
$$y = y/2 + a/2$$

$$a = y/2$$

$$m = a/2$$

$$\begin{array}{l} y \\ a \\ m \end{array} = \begin{array}{c} 1 \\ 1 \\ 1 \end{array} \quad \begin{array}{c} 1 \\ 1/2 \\ 1/2 \end{array} \quad \begin{array}{c} 3/4 \\ 1/2 \\ 1/4 \end{array} \quad \begin{array}{c} 5/8 \\ 3/8 \\ 1/4 \end{array} \quad \cdots \quad \begin{array}{c} 0 \\ 0 \\ 0 \end{array}$$

Microsoft becomes a spider trap



$$\begin{array}{c|ccc|} & y & a & m \\ \hline y & 1/2 & 1/2 & 0 \\ a & 1/2 & 0 & 0 \\ m & 0 & 1/2 & 1 \\ \hline \end{array}$$

Microsoft becomes a spider trap

- Equations $\mathbf{v} = M\,\mathbf{v}$:

$$y = y/2 + a/2$$

$$a = \mathrm{y}/2$$

$$m = a/2 + m$$

$$\begin{array}{llllllll}
\mathrm{y} & & 1 & 1 & 3/4 & 5/8 & & 0 \\
\mathrm{a} & = & 1 & 1/2 & 1/2 & 3/8 & \ldots & 0 \\
\mathrm{m} & & 1 & 3/2 & 7/4 & 2 & & 3
\end{array}$$

Google's Solution

- "Tax" each page a fixed percentage at each iteration.
- Add the same constant to all pages.
- Models a random walk with a fixed probability of going to a random place next.

Example: with 20% Tax

- Equations $\mathbf{v} = 0.8(M\,\mathbf{v}) + 0.2$:

$$y = 0.8(y/2 + a/2) + 0.2$$

$$a = 0.8(y/2) + 0.2$$

$$m = 0.8(a/2 + m) + 0.2$$

$$\begin{array}{ccccccccc} y & & 1 & 1.00 & 0.84 & 0.776 & & 7/11 \\ a & = & 1 & 0.60 & 0.60 & 0.536 & \ldots & 5/11 \\ m & & 1 & 1.40 & 1.56 & 1.688 & & 21/11 \end{array}$$

PageRank: Complete

- Two additional complexities
  - What is the proper treatment of dangling nodes (i.e., nodes with no out-links)?
  - How do we factor in the random jump factor?

PageRank: Complete

- Solution:
  - Second pass to redistribute "missing PageRank mass" and account for random jumps

$$p' = \alpha\left(\frac{1}{N}\right) + (1-\alpha)\left(\frac{m}{N} + p\right)$$

    - p is PageRank value from before,
      p' is updated PageRank value
    - N is the total number of nodes in the graph
    - m is the missing PageRank mass

PageRank Convergence

- Alternative convergence criteria
  - Iterate until PageRank values don't change
  - Iterate until PageRank rankings don't change
  - Fixed number of iterations
- Convergence for web graphs?

Beyond PageRank

- Link structure is important for web search
  - PageRank is one of many link analysis algorithms: HITS, SALSA, etc.
  - Used with thousands of other features in ranking…
- Adversarial nature of web search
  - Link spamming
  - Spider traps
  - Keyword stuffing
  - …

Efficient Graph Algorithms

- Sparse vs. Dense Graphs
- Graph Topologies

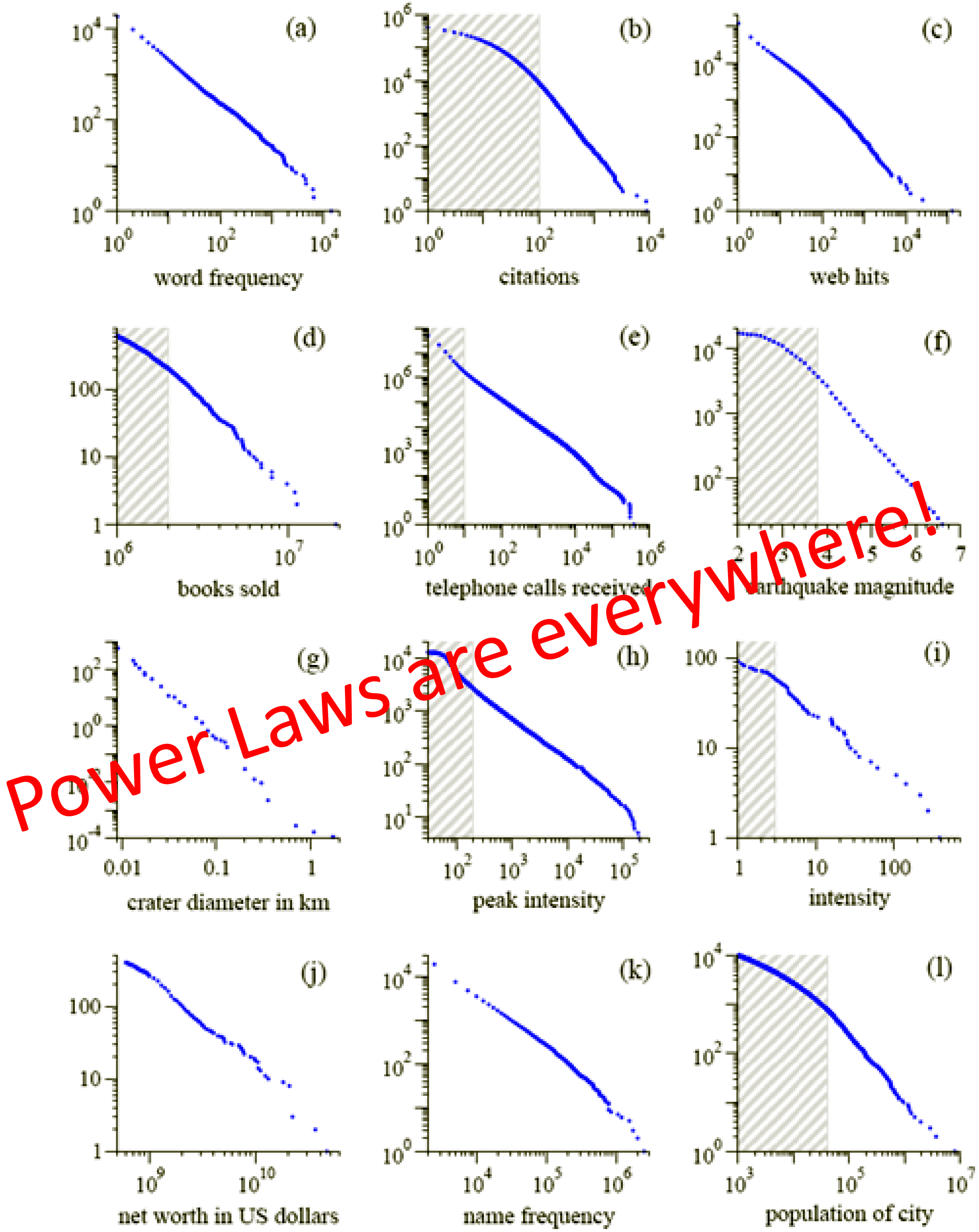


Figure from: Newman, M. E. J. (2005) “Power laws, Pareto distributions and Zipf's law.” Contemporary Physics 46:323–351.

Local Aggregation

- Use combiners!
  - In-mapper combining design pattern also applicable
- Maximize opportunities for local aggregation
  - Simple tricks: sorting the dataset in specific ways

Take Home Messages

- Graph Problems and Representations
- Parallel Breadth-First Search
- PageRank: Simplified and Complete