Cloud Programming: Lecture5 – MapReduce Algorithms

National Tsing-Hua University

2016, Spring Semester

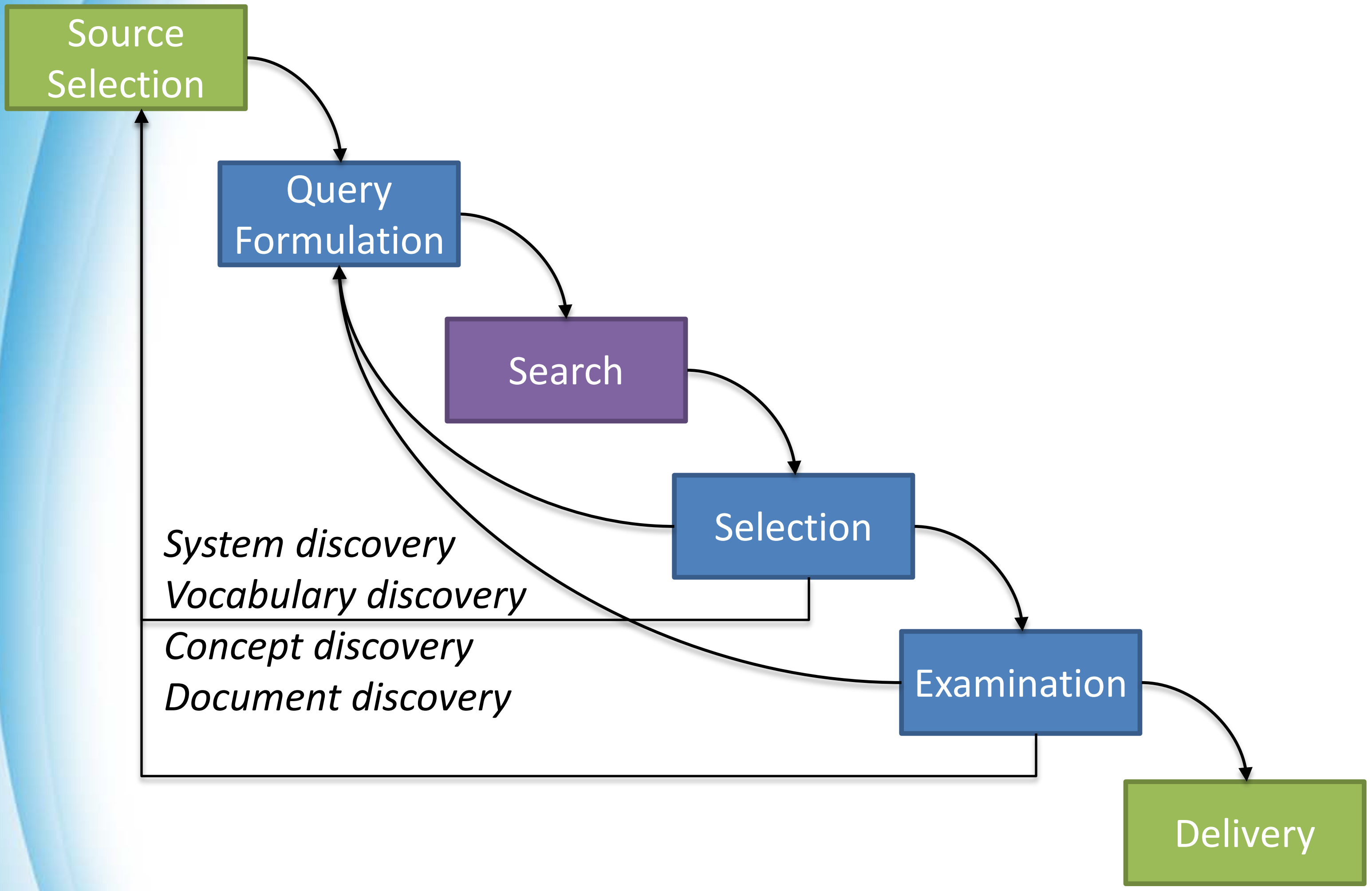
Outline

- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

Information retrieval (IR)

- Information retrieval (IR)
  - the activity of obtaining information resources relevant to an information need from a collection of information resources
  - Focus on textual information (= text/document retrieval)
  - Other possibilities include image, video, music, …
- What do we find?
  - Generically, “documents”
  - Even though we may be referring to web pages, PDFs, PowerPoint slides, paragraphs, etc.

Information Retrieval Cycle



The Central Problem in Search

Author

Concepts

↓

Query Terms

"tragic love story"

Concepts

↓

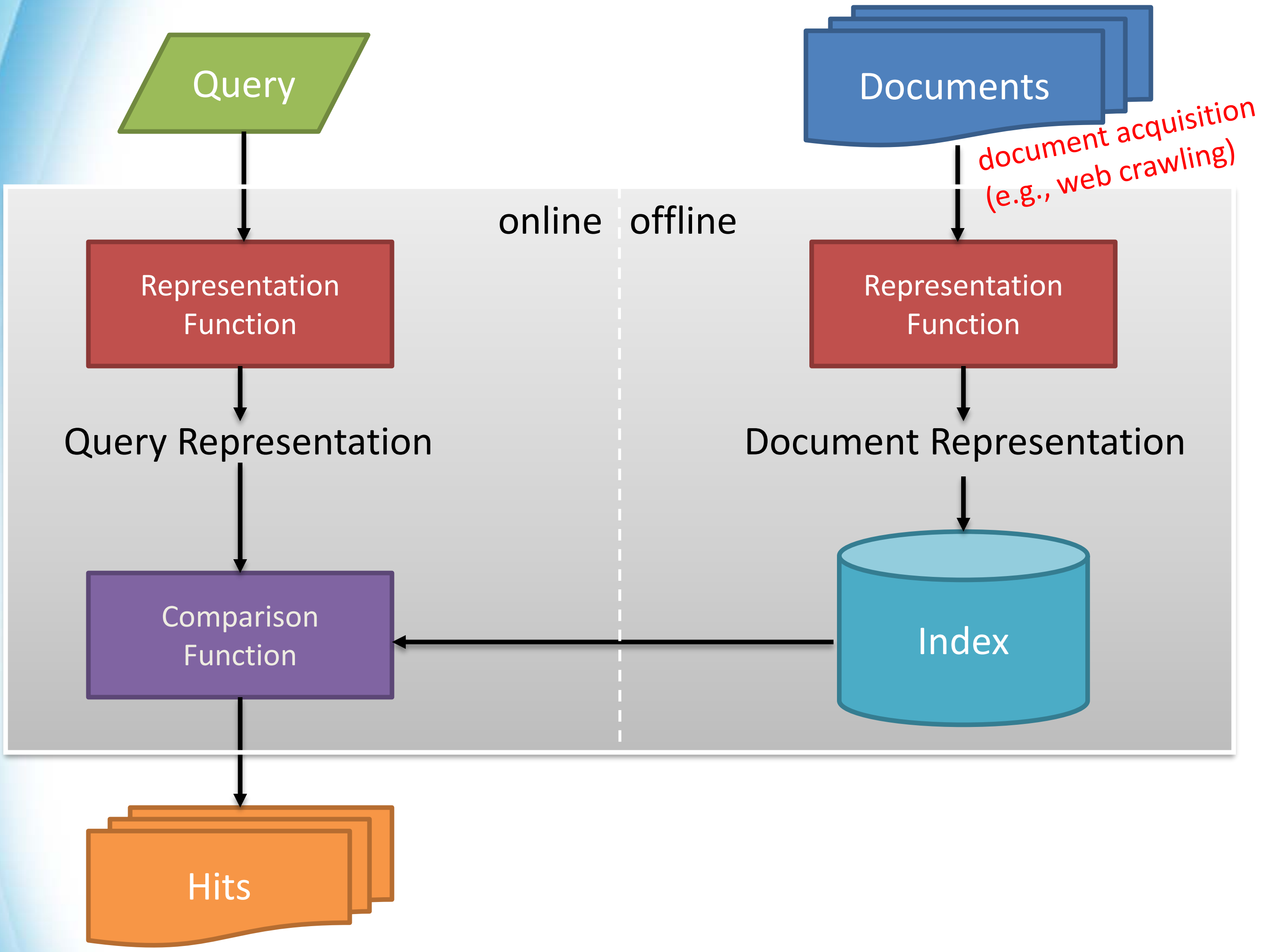
Document Terms

"fateful star-crossed romance"

↔

Do these represent the same concepts?

Abstract IR Architecture



How do we represent text?

- Remember: computers don’t “understand” anything!
- “Bag of words”
  - Treat all the words in a document as **index** terms
  - Assign a “**weight**” to each term based on “**importance**” (or, in simplest case, presence/absence of word)
  - Disregard order, structure, meaning, etc. of the words
  - Simple, yet effective!
- Assumptions
  - Term occurrence is independent
  - Document relevance is independent
  - “Words” are well-defined

What's a word?

天主教教宗若望保祿二世因感冒再度住進醫院。這是他今年第二度因同樣的病因住院。

وقال مارك ريجيف - الناطق باسم الخارجية الإسرائيلية - إن شارون قبل الدعوة وسيقوم للمرة الأولى بزيارة تونس، التي كانت لفترة طويلة المقر الرسمي لمنظمة التحرير الفلسطينية بعد خروجها من لبنان عام 1982.

Выступая в Мещанском суде Москвы экс-глава ЮКОСа заявил не совершал ничего противозаконного, в чем обвиняет его генпрокуратура России.

भारत सरकार ने आर्थिक सर्वेक्षण में वित्तीय वर्ष 2005-06 में सात फ़ीसदी विकास दर हासिल करने का आकलन किया है और कर सुधार पर ज़ोर दिया है

日米連合で台頭中国に対処…アーミテージ前副長官提言

조재영 기자= 서울시는 25일 이명박 시장이 `행정중심복합도시'' 건설안에 대해 `군대라도 동원해 막고싶은 심정''이라고 말했다는 일부 언론의 보도를 부인했다.

Sample Document

"Bag of Words"

McDonald's slims down spuds

Fast-food chain to reduce certain types of fat in its french fries with new cooking oil.

NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

But does that mean the popular shoestring fries won't taste the same? The company says no. "It's a win-win for our customers because they are getting the same great french-fry taste along with an even healthier nutrition profile," said Mike Roberts, president of McDonald's USA.

But others are not so sure. McDonald's will not specifically discuss the kind of oil it plans to use, but at least one nutrition expert says playing with the formula could mean a different taste.

Shares of Oak Brook, Ill.-based McDonald's (MCD: down $0.54 to $23.22, Research, Estimates) were lower Tuesday afternoon. It was unclear Tuesday whether competitors Burger King and Wendy's International (WEN: down $0.80 to $34.91, Research, Estimates) would follow suit. Neither company could immediately be reached for comment.

…

14 × McDonalds

12 × fat

11 × fries

8 × new

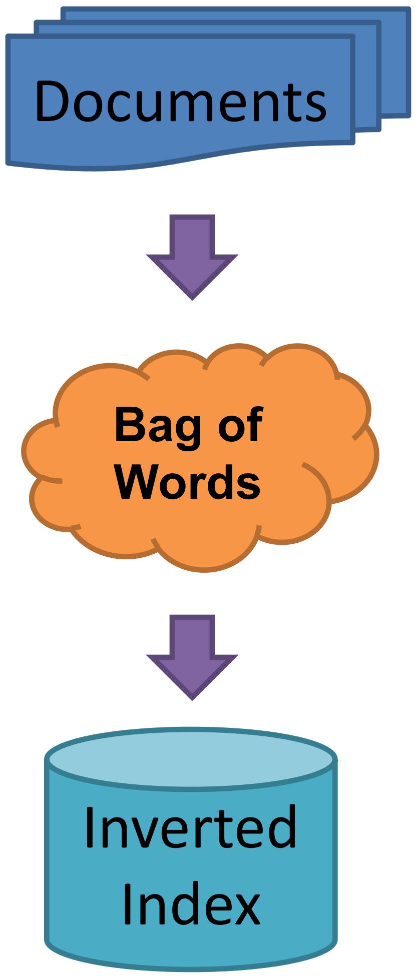
7 × french

6 × company, said, nutrition

5 × food, oil, percent, reduce, taste, Tuesday

…

Counting Words…



1. Tokenization: Aren't→ are not
2. Stopword removal: a, an, is, it, etc.
3. Normalization (equivalence classing of terms):
   - Hello → hello, windows→window, 你好→ hello

~~syntax~~, ~~semantics~~, word ~~knowledge~~, etc.

Outline

- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

Boolean Retrieval

- Express queries as a **Boolean expression**
  - AND, OR, NOT
  - Can be arbitrarily nested
- Retrieval is based on the notion of sets
  - Any query divides the collection into TWO **sets**: retrieved, not-retrieved
  - Pure Boolean systems do NOT define an **ordering** of the results

Inverted Index: Boolean Retrieval

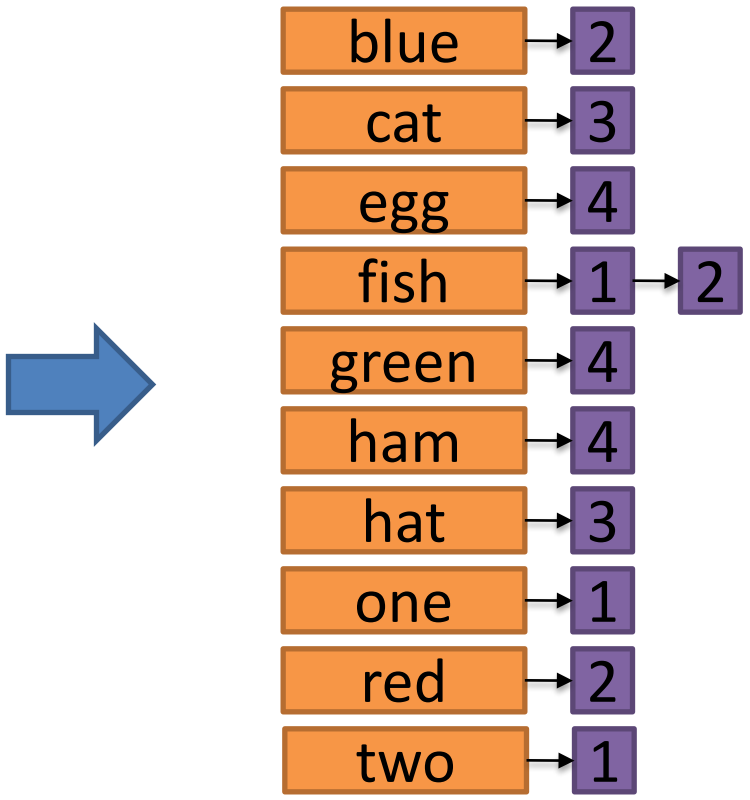
Doc 1 **one fish, two fish**

Doc 2 **red fish, blue fish**

Doc 3 **cat in the hat**

Doc 4 **green eggs and ham**

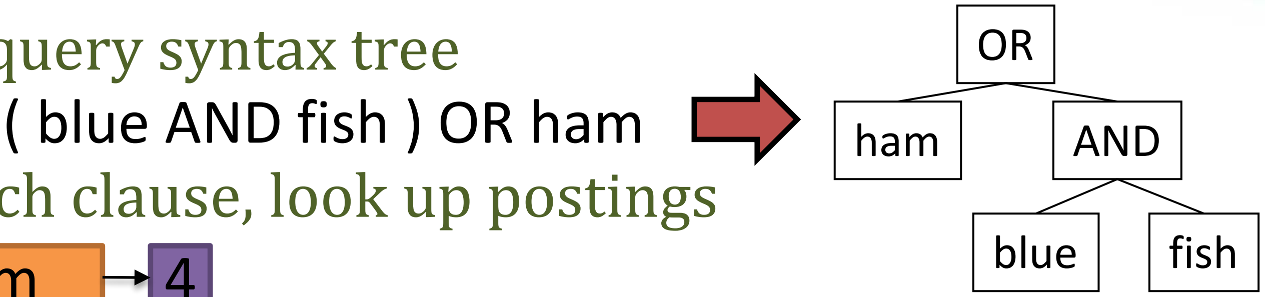
	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			



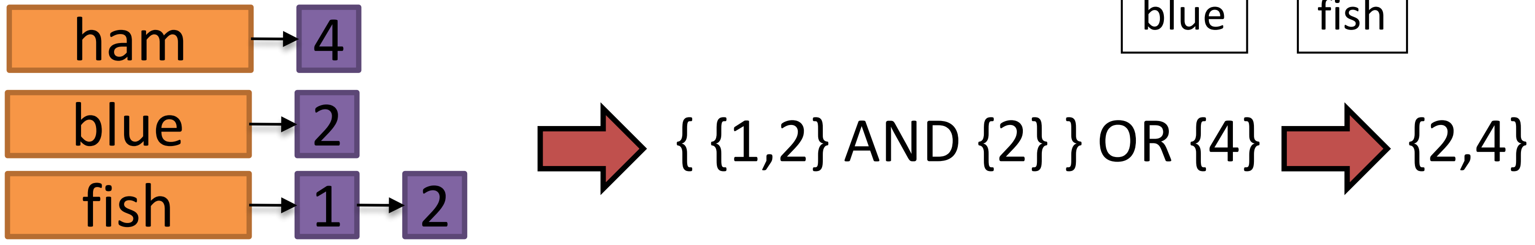
Indexed on words instead of documents

Boolean Retrieval

- To execute a Boolean query:
  - Build query syntax tree



  - For each clause, look up postings



  - Traverse postings and apply Boolean operator
- Efficiency analysis
  - Postings traversal is linear (assuming **sorted** postings)
  - Start with shortest posting first

Strengths and Weaknesses

- Strengths
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you're looking for
  - Implementations are fast and efficient
- Weaknesses
  - Users must learn Boolean logic
  - Boolean logic insufficient to capture the richness of language
  - No control over the size of result set: either too many hits or none
  - When do you stop reading? All documents in the result set are considered "equally good"
  - What about partial matches? Documents that "don't quite match" the query may be useful also

Outline

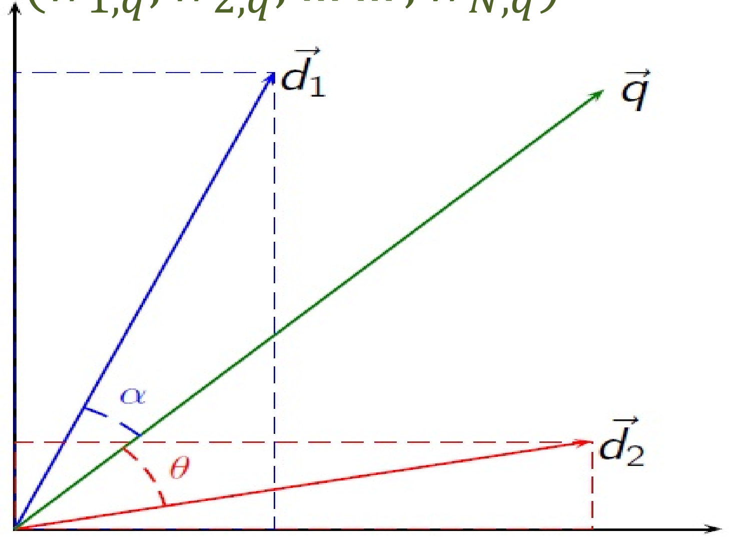
- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

Ranked Retrieval

- Order documents by how likely they are to be relevant to the information need
  - Sort documents by relevance
  - Display sorted results
- User model
  - Present hits one screen at a time, best results first
  - At any point, users can decide to stop looking
- How do we estimate relevance?
  - Assume document is relevant if it has a lot of query **terms**
  - Represent query and document in **vector**
  - Relevance means the **closeness of vectors**

Vector Space Model

- Documents and queries are represented as vectors: Each term is a dimension
  - Document: $d_j = (w_{1,j}, w_{2,j}, \ldots \ldots, w_{N,j})$
  - Query: $q = (w_{1,q}, w_{2,q}, \ldots \ldots, w_{N,q})$



retrieve documents based on how close the document is to the query (i.e., similarity ~ "closeness")

Similarity Metric

- Use "angle" between the vectors:

$$sim(d_j, q) = \cos(\theta) = \frac{\overrightarrow{d_j} \cdot \vec{q}}{\left|\overrightarrow{d_j}\right| \cdot |\vec{q}|}$$

$$sim(d_j, q) = \frac{\sum_{i=0}^{n} w_{i,j}\, w_{i,q}}{\sqrt{\sum_{i=0}^{n} (w_{i,j})^2}\, \sqrt{\sum_{i=0}^{n} (w_{i,q})^2}}$$

- Or, more generally, inner products:

$$sim(d_j, q) = \overrightarrow{d_j} \cdot \overrightarrow{q_k} = \sum_{i=0}^{n} w_{i,j}\, w_{i,q}$$

Term Weighting

- Term weights consist of two components
  - Local: how important is the term in this document?
  - Global: how important is the term in the collection?
- Here's the intuition:
  - Terms that appear often in a document should get high weights
  - Terms that appear in many documents should get low weights
- How do we capture this mathematically?
  - **Term frequency** (local)
  - Inverse **document frequency** (global)

TF.IDF Term Weighting

- Term Frequency-Inverse Document Frequency mode:
  - Terms appear often in a document should get high weights
  - Terms appear in many documents should get low weights

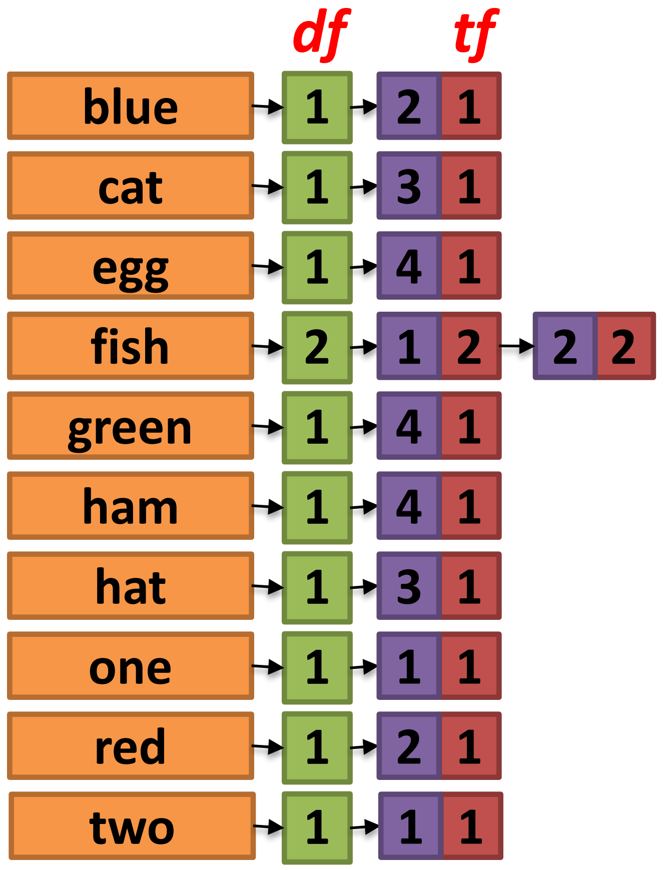
$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{df_i}$$

$w_{i,j}$	weight assigned to term ***i*** in document ***j***
$\mathrm{tf}_{i,j}$	frequency of occurrence of term ***i*** in document ***j***
N	number of documents in entire collection
df_i	number of documents with term ***i***

Inverted Index: TF.IDF

Doc 1	Doc 2	Doc 3	Doc 4
one fish, two fish	red fish, blue fish	cat in the hat	green eggs and ham

	tf 1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1

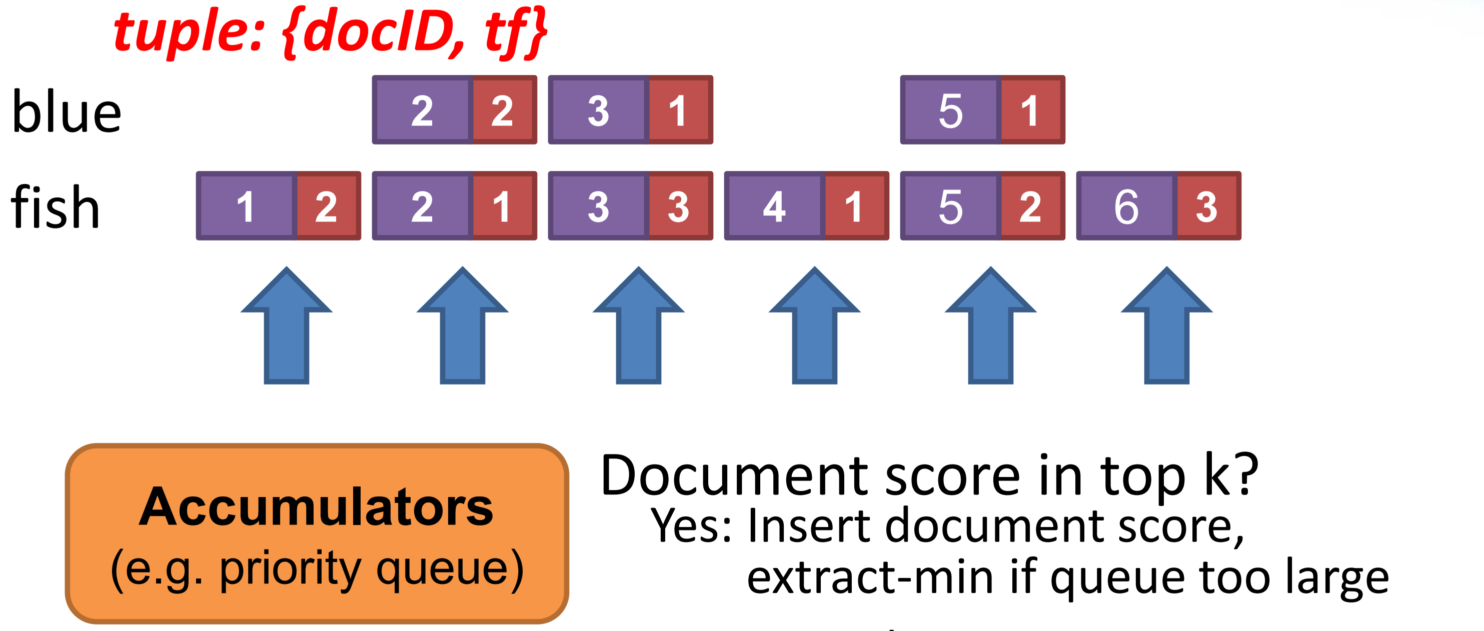


Retrieval in a Nutshell

- Look up postings lists corresponding to **query terms**
- **Traverse postings** for each query term
- **Store partial query-document scores** in accumulators
- **Select top *k* results** to return

Retrieval: Document-at-a-Time

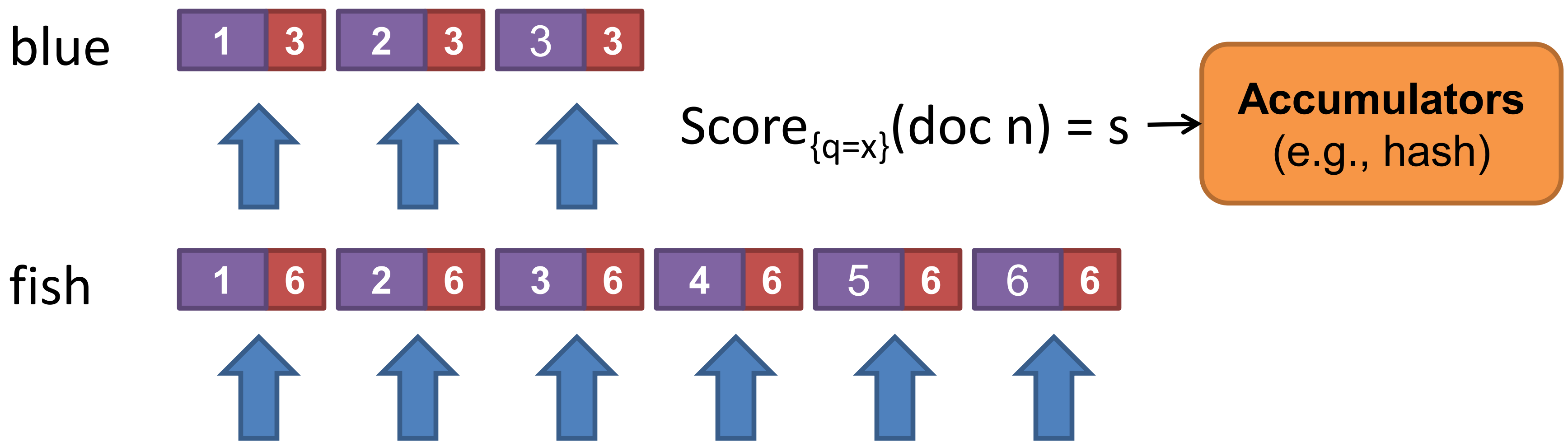
- Evaluate documents one at a time (score **all query terms**)



- Tradeoffs
  - Small memory footprint (good) ➔less query terms than posting
  - Must read through all postings (bad)➔if need to evaluate all docs
  - More disk seeks (bad)➔data normally stored in row major

Retrieval: Query-At-A-Time

- Evaluate documents **one query term** at a time
  - Usually, starting from most rare term



- Tradeoffs
  - Large memory footprint (bad) ➔ Too many Docs
  - Early termination heuristics (good) ➔ Evaluate all Docs together
  - Less disk seek (good) ➔ Line-by-line sequential Read

Outline

- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

MapReduce it?

- The indexing problem

Perfect for MapReduce!

  - **Scalability is critical**
  - Must be relatively fast, but need not be real time
  - Fundamentally a **batch operation**
  - Incremental updates may or may not be important
  - For the web, crawling is a challenge in itself
- The retrieval problem

Uh… not so good…

  - Must have sub-second **response time**
  - For the web, only need relatively **few results**

MapReduce: Index Construction

- Map over all documents
    - Emit *term* as key, (*docno, tf*) as value
    - Emit other information as necessary (e.g., term position)
- Sort/shuffle:
    - group postings by term
- Reduce
    - Gather and sort the postings (e.g., by *docno* or *tf*)
    - Write postings to disk
- MapReduce does all the heavy lifting!

Inverted Indexing with MapReduce

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Map

one	1	1
two	1	1
fish	1	2

red	2	1
blue	2	1
fish	2	2

cat	3	1
hat	3	1

Shuffle and Sort: aggregate values by keys

Reduce

cat	3	1		
fish	1	2	2	2
one	1	1		
red	2	1		

blue	2	1
hat	3	1
two	1	1

Inverted Indexing: Pseudo-Code

```
1: class Mapper
2:     procedure Map(docid $n$, doc $d$)
3:         $H \leftarrow$ new AssociativeArray
4:         for all term $t \in$ doc $d$ do
5:             $H\{t\} \leftarrow H\{t\} + 1$
6:         for all term $t \in H$ do
7:             Emit(term $t$, posting $\langle n, H\{t\} \rangle$)
```

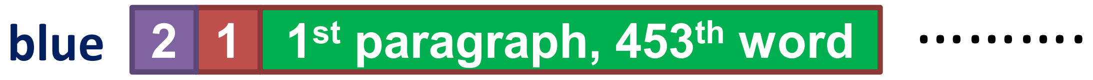
H{t}: term frequency in a file

```
1: class Reducer
2:     procedure Reduce(term $t$, postings $[\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$)
3:         $P \leftarrow$ new List
4:         for all posting $\langle a, f \rangle \in$ postings $[\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$ do
5:             Append($P$, $\langle a, f \rangle$)
6:         Sort($P$)
7:         Emit(term $t$, postings $P$)
```

Problem?

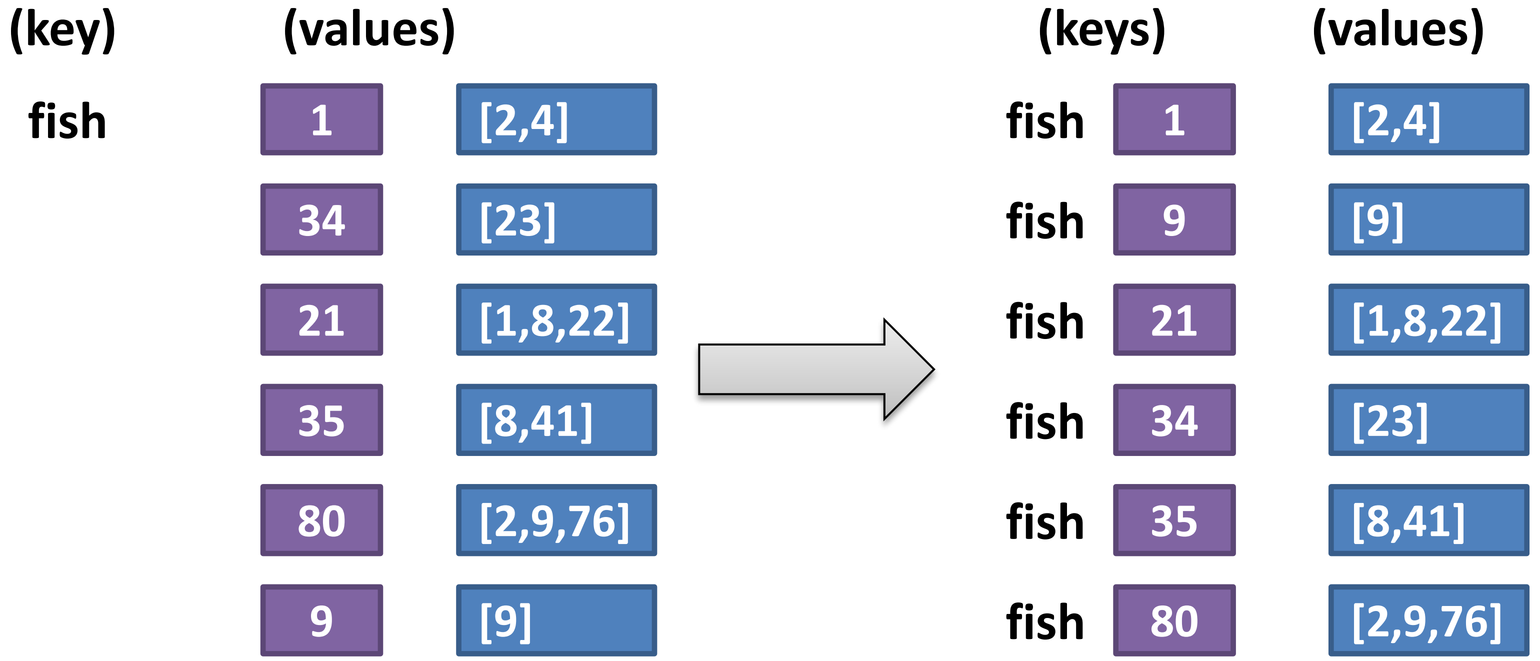
Scalability Bottleneck

- Initial ideal: terms as keys, postings as values
  - Reducers must buffer all postings associated with key (to sort)
  - What if we run **out of memory** to buffer postings?
- Think about 100million documents….
- Each posting can be long for storing additional info
  - Term position in a doc
  - Article title
  - Snippet



Another Try…

- Use <*term*><*docid*> as the key
  - Let the framework do the sorting
  - Term frequency implicitly stored
  - **Directly write postings to disk!**



Retrieval with MapReduce?

- MapReduce is fundamentally **batch-oriented**
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for **real-time** queries!
  - Initial a job takes time
  - Use separate infrastructure for retrieval…

Outline

- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

What's 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
- Different types of graphs:
  - Directed vs. undirected edges
  - Presence or absence of cycles
- Graphs are everywhere:
  - Hyperlink structure of the Web
  - Physical structure of computers on the Internet
  - Interstate highway system
  - Social networks

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telco laying down fiber
- Finding Max Flow
  - Airline scheduling
- Bipartite matching
  - Monster.com, Match.com
- And of course... PageRank

Graphs and MapReduce

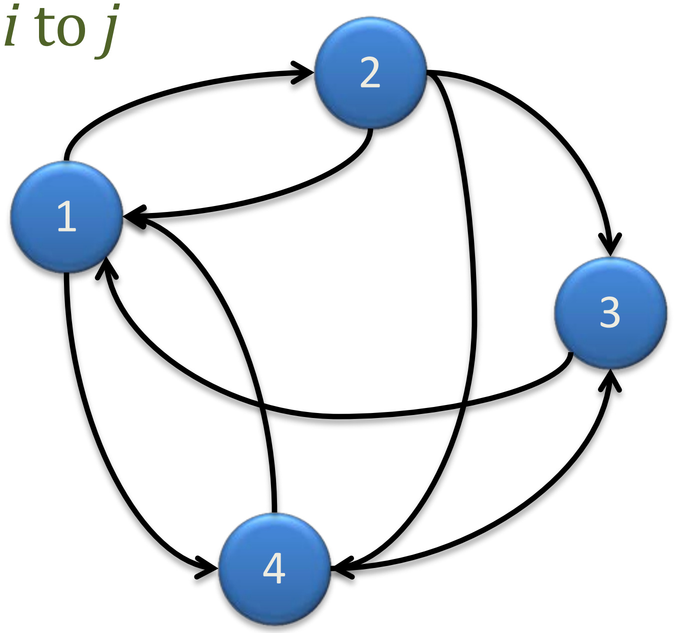
- Graph algorithms typically involve:
  - Performing computations at each node: based on node features, edge features, and local link structure
  - Propagating computations: “traversing” the graph
- Key questions:
  - How do you represent graph data in MapReduce?
  - How do you traverse a graph in MapReduce?

Adjacency Matrices

Represent a graph as an n x n square matrix M

- $n = |V|$
- $M_{ij} = 1$ means a link from node i to j

$$
\begin{array}{c|cccc}
 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\
\hline
\mathbf{1} & 0 & 1 & 0 & 1 \\
\mathbf{2} & 1 & 0 & 1 & 1 \\
\mathbf{3} & 1 & 0 & 0 & 0 \\
\mathbf{4} & 1 & 0 & 1 & 0
\end{array}
$$



Adjacency Lists

Take adjacency matrices and throw away all the zeros

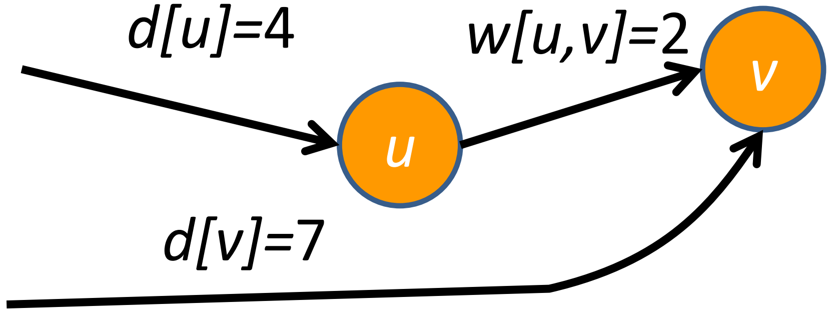
$$
\begin{array}{c|cccc}
 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\
\hline
\mathbf{1} & 0 & 1 & 0 & 1 \\
\mathbf{2} & 1 & 0 & 1 & 1 \\
\mathbf{3} & 1 & 0 & 0 & 0 \\
\mathbf{4} & 1 & 0 & 1 & 0
\end{array}
\Rightarrow
$$

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

Single Source Shortest Path

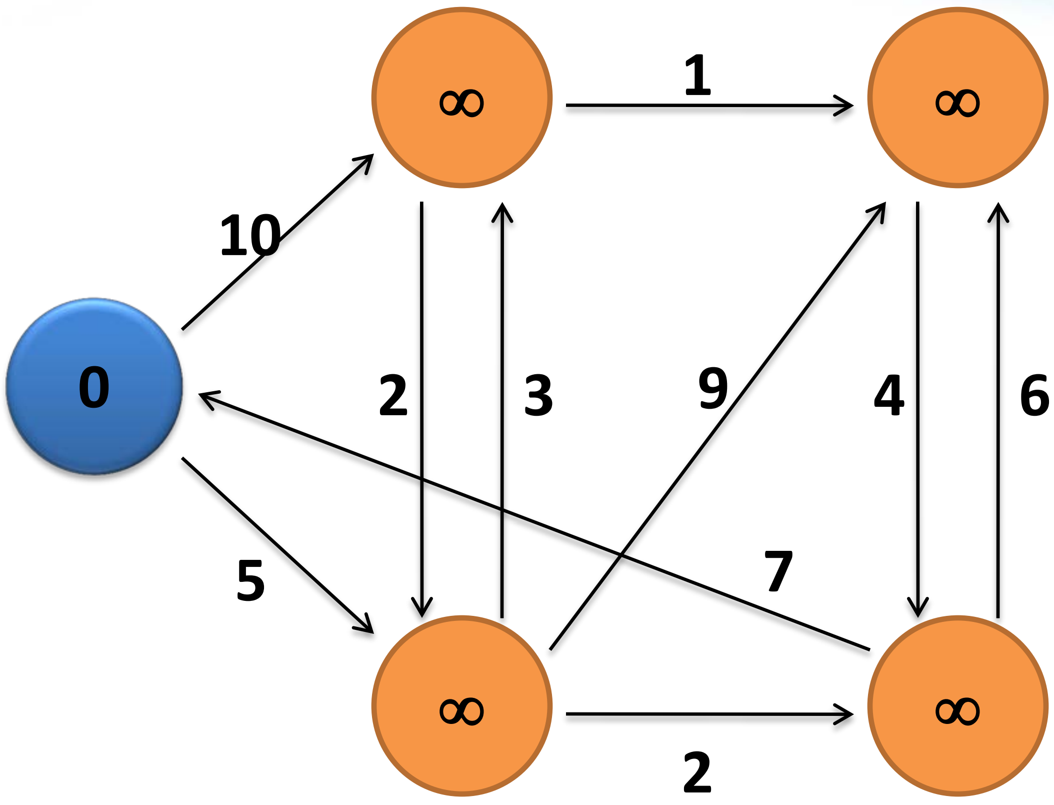
- **Problem:** find shortest path from a source node to one or more target nodes
    - Shortest might also mean lowest weight or cost
- First, a refresher: Dijkstra's Algorithm

```
1:  DIJKSTRA($G, w, s$)
2:      $d[s] \leftarrow 0$
3:      for all vertex $v \in V$ do
4:          $d[v] \leftarrow \infty$
5:      $Q \leftarrow \{V\}$
6:      while $Q \neq \emptyset$ do
7:          $u \leftarrow$ EXTRACTMIN($Q$)
8:          for all vertex $v \in u$.ADJACENCYLIST do
9:              if $d[v] > d[u] + w(u, v)$ then
10:                 $d[v] \leftarrow d[u] + w(u, v)$
```

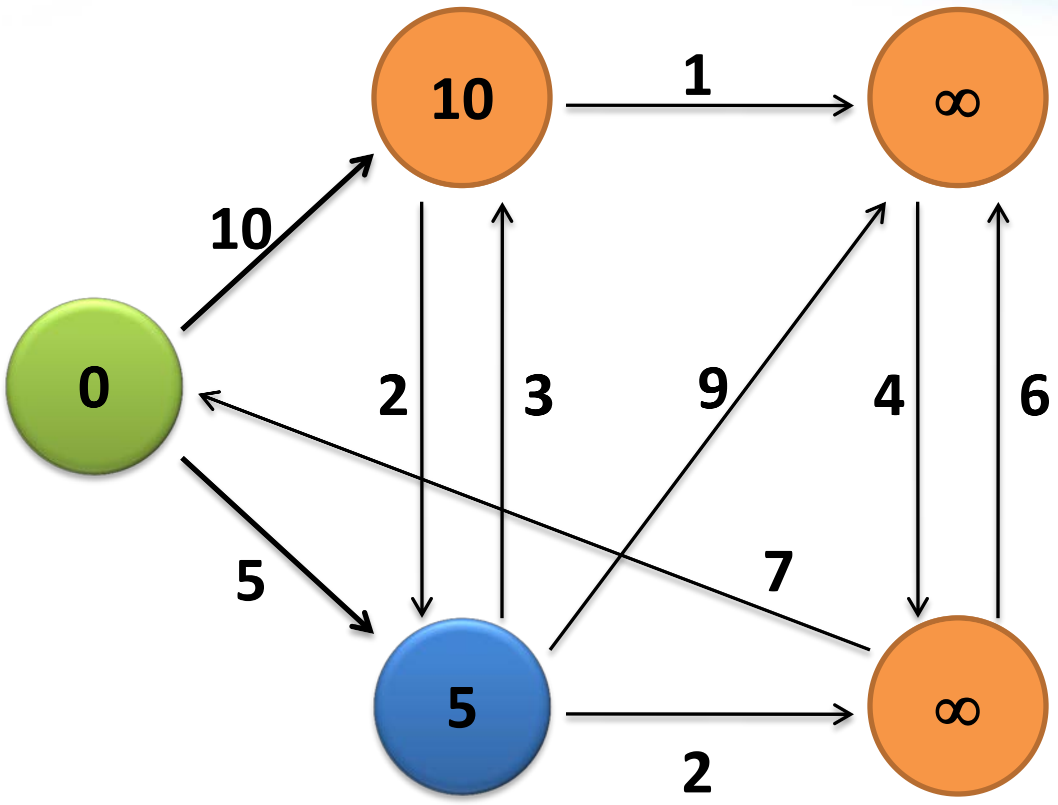


d[v]=min(d[v], d[u]+w[u,v])=6

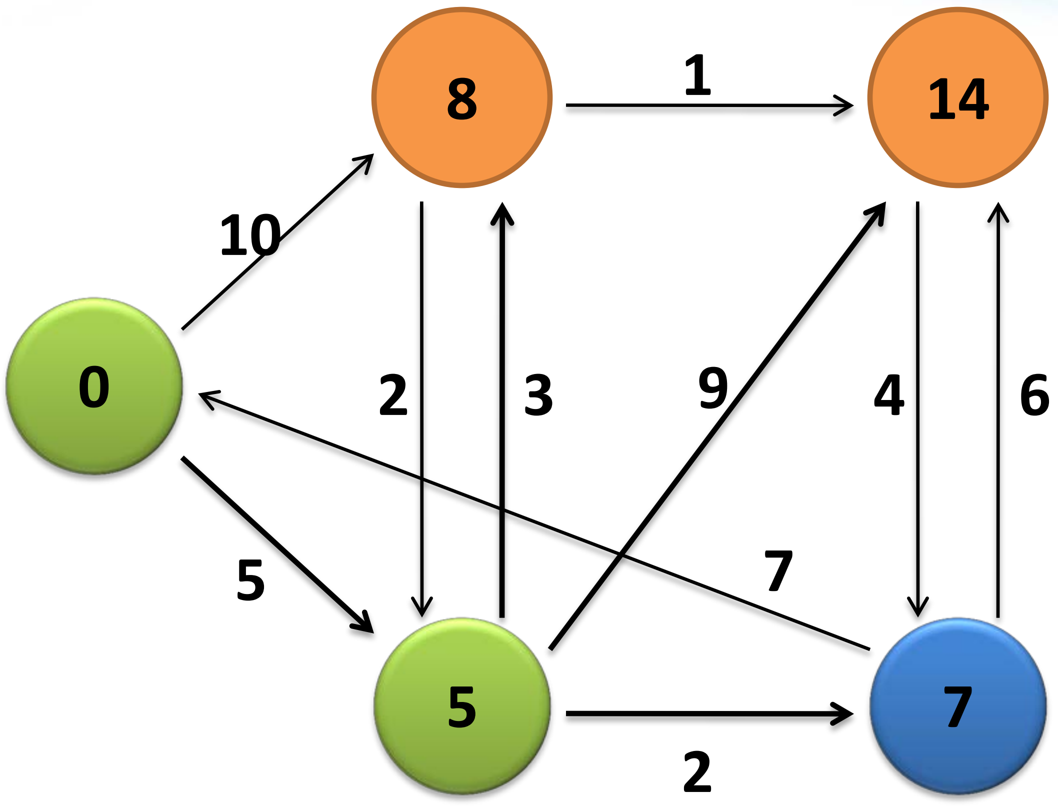
Dijkstra's Algorithm Example



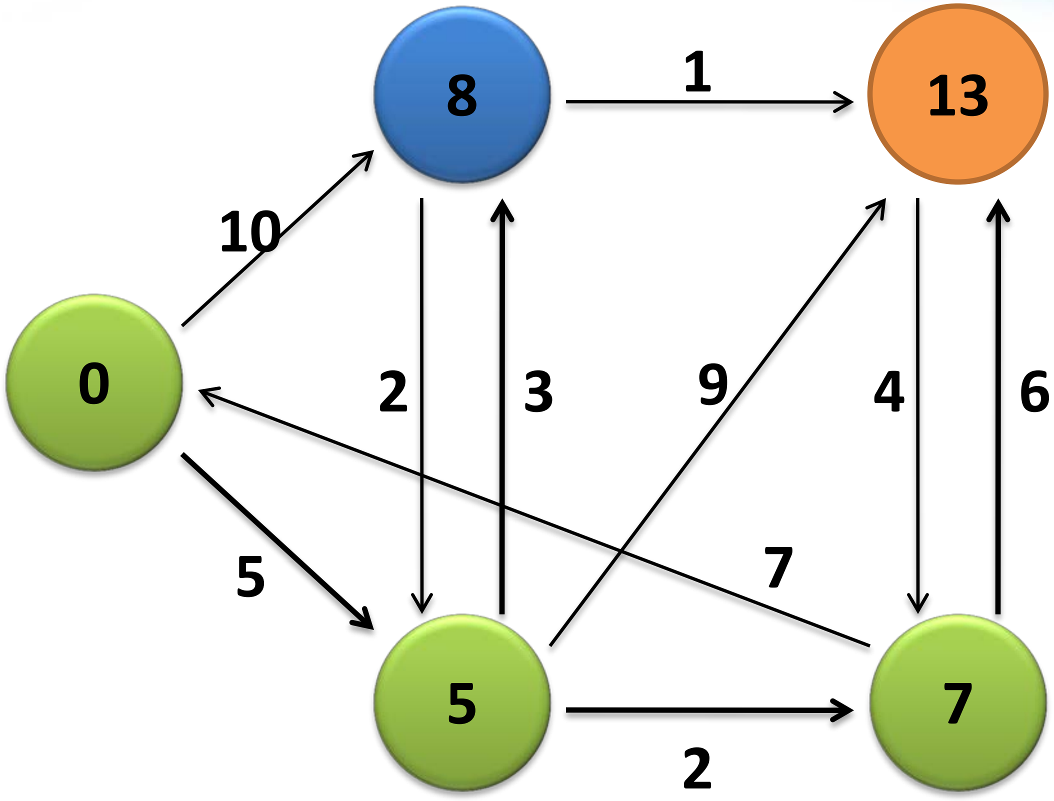
Dijkstra's Algorithm Example



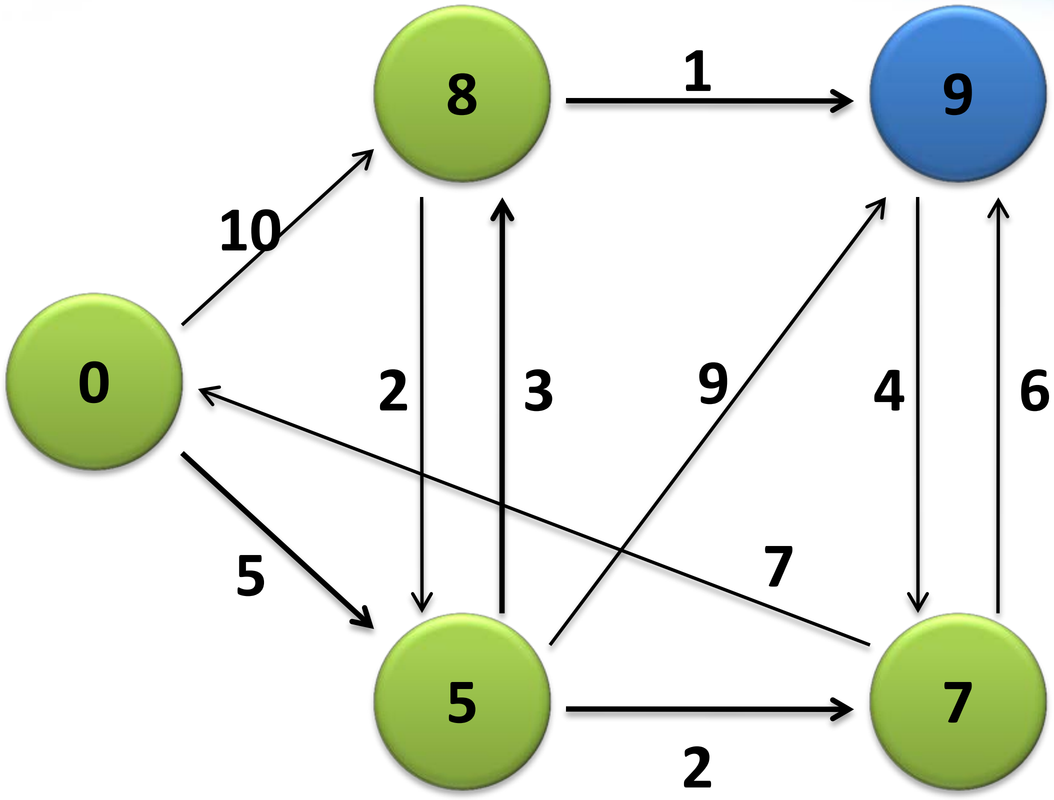
Dijkstra's Algorithm Example



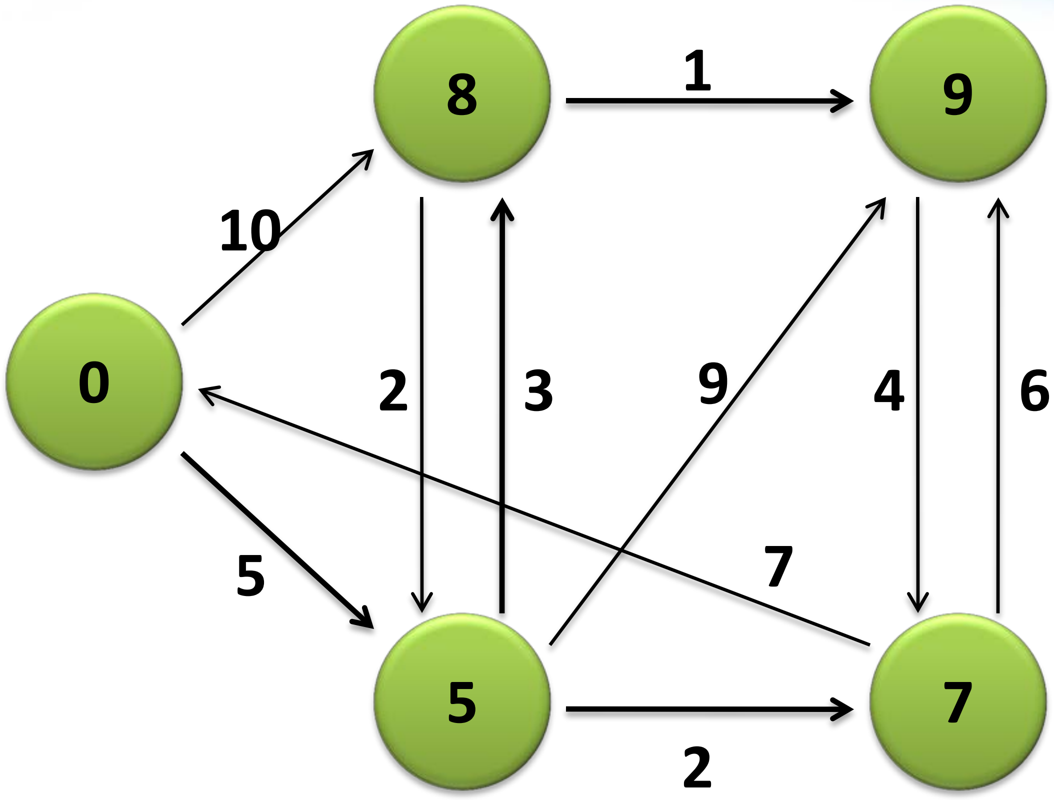
Dijkstra's Algorithm Example



Dijkstra's Algorithm Example



Dijkstra's Algorithm Example

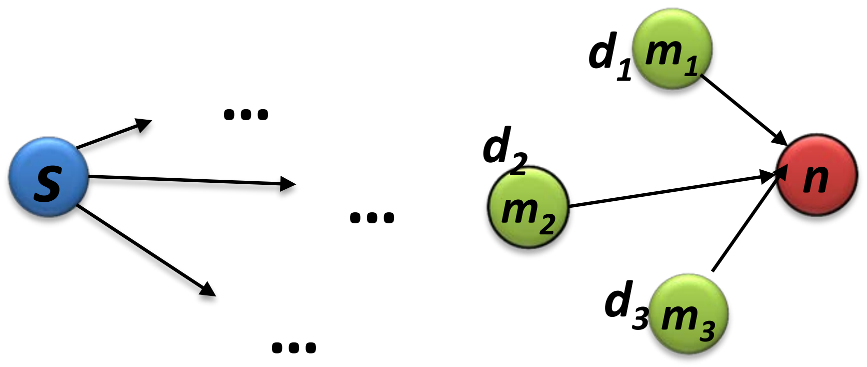


Single Source Shortest Path

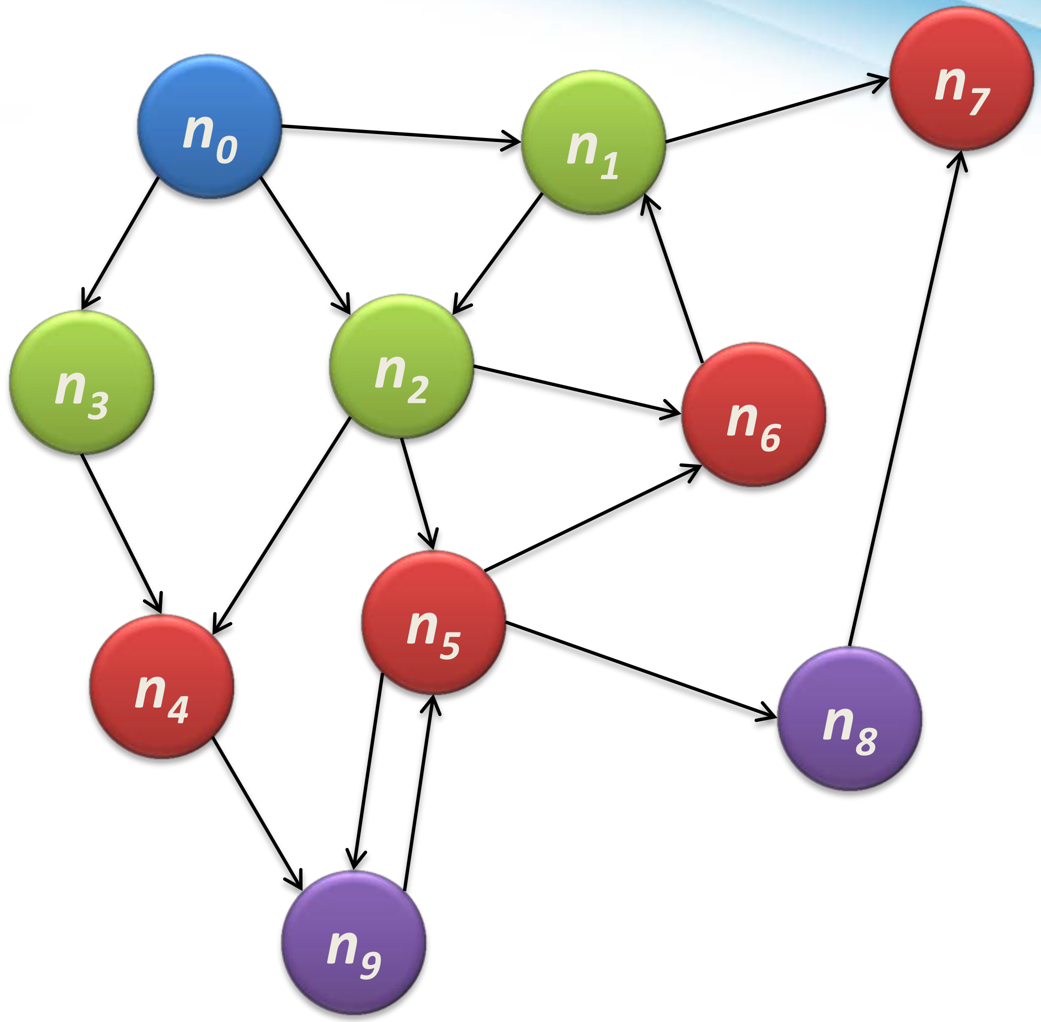
- **Problem:** find shortest path from a source node to one or more target nodes
  - Shortest might also mean lowest weight or cost
- Single processor machine: Dijkstra’s Algorithm
- MapReduce: parallel Breadth-First Search (BFS)

Finding the Shortest Path

- Consider simple case of equal edge weights
- Solution to the problem can be defined inductively
- Here's the intuition:
  - Define: *b* is reachable from *a* if *b* is on adjacency list of *a*
  - DistanceTo(*s*) = 0
  - For all nodes *p* reachable from *s*, DistanceTo(*p*) = 1
  - For all nodes *n* reachable from some other set of nodes *M*, DistanceTo(*n*) = 1 + min(DistanceTo(*m*), *m* ∈ *M*)

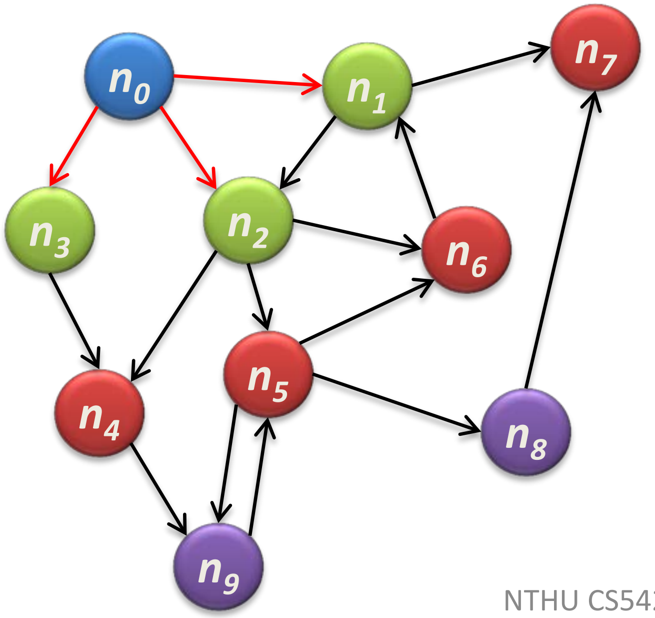


Visualizing Parallel BFS



From Intuition to Algorithm

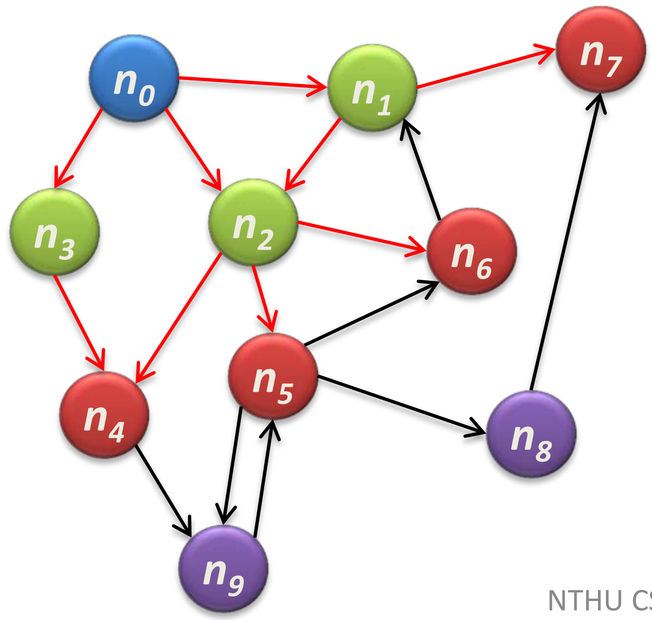
- Data representation:
  - Key: node *n*
  - Value: *d* (distance from start),
    adjacency list (list of nodes reachable from *n*)
  - Initialization: for all nodes except for start node, *d* = ∞



n0,	**0:n1:n2:n3**
n1,	**INF:n2:n7**
n2,	**INF:n4:n5:n6**
n3,	**INF:n4**
n4,	**INF:n9**
n5,	**INF:n6:n8:n9**
n6,	**INF:n1**
n7,	**INF:**
n8,	**INF:n7**
n9,	**INF:n5**

From Intuition to Algorithm

- Mapper:
  - $\forall m \in$ adjacency list: emit $(m, d + 1)$
- Sort/Shuffle
  - Groups distances by **reachable nodes**
- Reducer:
  - **Selects minimum distance** path for each reachable node



n0,0:n1:n2:n3
n1,1:n2:n7
n2,1:n4:n5:n6
n3,1:n4
n4,INF:n9
n5,INF:n6:n8:n9
n6,INF:n1
n7,INF:
n8,INF:n7
n9,INF:n5

n1,1
n2,1
n3,1

n2,2
n7,2

n4,2
n5,2
n6,2

n4,2

n1,1

n2,1
n2,2

n3,1

n4,2
n4,2

n5,2

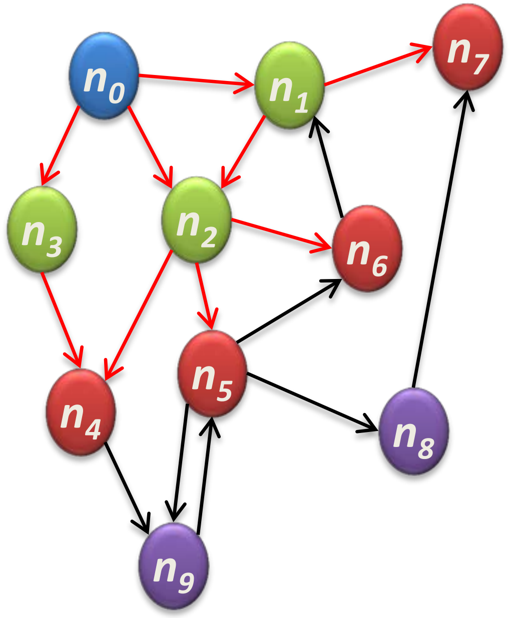
n6,2

n7,2

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

Start 2nd iter:



n0,0:n1:n2:n3
n1,1:n2:n7
n2,1:n4:n5:n6
n3,1:n4
n4,INF:n9
n5,INF:n6:n8:n9
n6,INF:n1
n7,INF:
n8,INF:n7
n9,INF:n5

MAP

n0,0:n1:n2:n3

n1,1:n2:n7
n1,1

n2,1:n4:n5:n6
n2,1
n2,2

n3,1:n4
n3,1

n4,INF:n9
n4,2
n4,2

n5,INF:n6:n8:n9
n5,2

n6,INF:n1
n6,2

n7,INF:
n7,2

n8,INF:n7

n9,INF:n5

REDUCE

n0,0:n1:n2:n3
n1,1:n2:n7
n2,1:n4:n5:n6
n3,1:n4
n4,2:n9
n5,2:n6:n8:n9
n6,2:n1
n7,2:
n8,INF:n7
n9,INF:n5

Next Iteration

BFS Pseudo-Code

n: the node id (an integer)
N: the node's data structure
(adjacency list and current distance)

```
1: class Mapper
2:     method Map(nid $n$, node $N$)
3:         $d \leftarrow N$.Distance
4:         Emit(nid $n, N$)                    Ex: <n0,{0,n1:n2:n3}>        ▷ Pass along graph structure
5:         for all nodeid $m \in N$.AdjacencyList do
6:             Emit(nid $m, d+1$)                                       ▷ Emit distances to reachable nodes
                                   Ex:<n1,1>, <n2,1>, <n3,1>
1: class Reducer
2:     method Reduce(nid $m, [d_1, d_2, \ldots]$)  Ex: n1, {3,n2:n4}, {1}, {4}}
3:         $d_{min} \leftarrow \infty$
4:         $M \leftarrow \emptyset$
5:         for all $d \in$ counts $[d_1, d_2, \ldots]$ do
6:             if IsNode($d$) then  Ex: {3,n2:n4}
7:                 $M \leftarrow d$                                       ▷ Recover graph structure
8:             else if $d < d_{min}$ then Ex: {1}                        ▷ Look for shorter distance
9:                 $d_{min} \leftarrow d$
10:        $M$.Distance $\leftarrow d_{min}$                                ▷ Update shortest distance
11:        Emit(nid $m$, node $M$)  Ex: <n1, {1,n2:n4} >
```

Weighted Edges

- 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
- Additional bookkeeping needed to keep track of actual path
  - <nid, dist, previous nid, adjacent list>

Stopping Criterion

- Execution of an iterative MapReduce algorithm
  - Typically, requires a **non-MapReduce driver program**, which submits a MapReduce job to iterate the algorithm until a **termination condition** has been met
- How many iterations are needed in parallel BFS
  - Convince yourself: when a node is first "discovered", we've found the shortest path
  - Graph diameter (unit edge length)
  - Number of nodes (weighted edge)
  - Six degrees of separation: everyone on the planet is connected to everyone else by at most six steps

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"
- How can we do better using MapReduce?

Outline

- Information retrieval
  - Boolean Retrieval
  - Ranked Retrieval
  - Inverted indexing in MapReduce
- Graphic Problem
  - Parallel breadth-first search
  - PageRank

Random Walks Over the Web

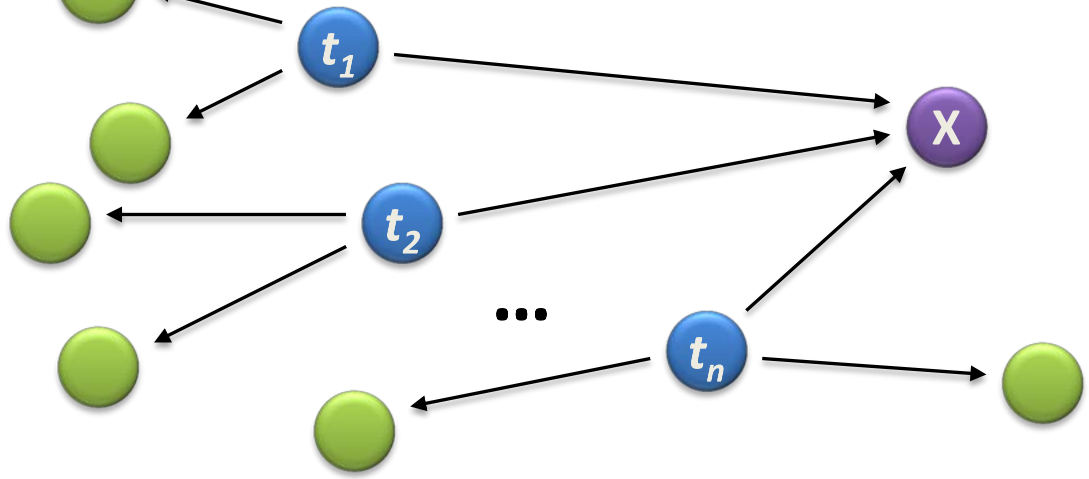
- Random surfer model:
  - User starts at a random Web page
  - User randomly clicks on links, surfing from page to page
- PageRank
  - Characterizes the amount of time spent on any given page
  - Mathematically, a probability distribution over pages
- PageRank captures notions of page importance
  - Correspondence to human intuition?
  - One of thousands of features used in web search
  - Note: query-independent

PageRank: Defined

Given page *x* with inlinks $t_1 \ldots t_n$, where

- *C(t)* is the out-degree of *t*
- α is probability of **random jump**
- *N* is the total number of nodes in the graph

$$PR(x) = \alpha\left(\frac{1}{N}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$



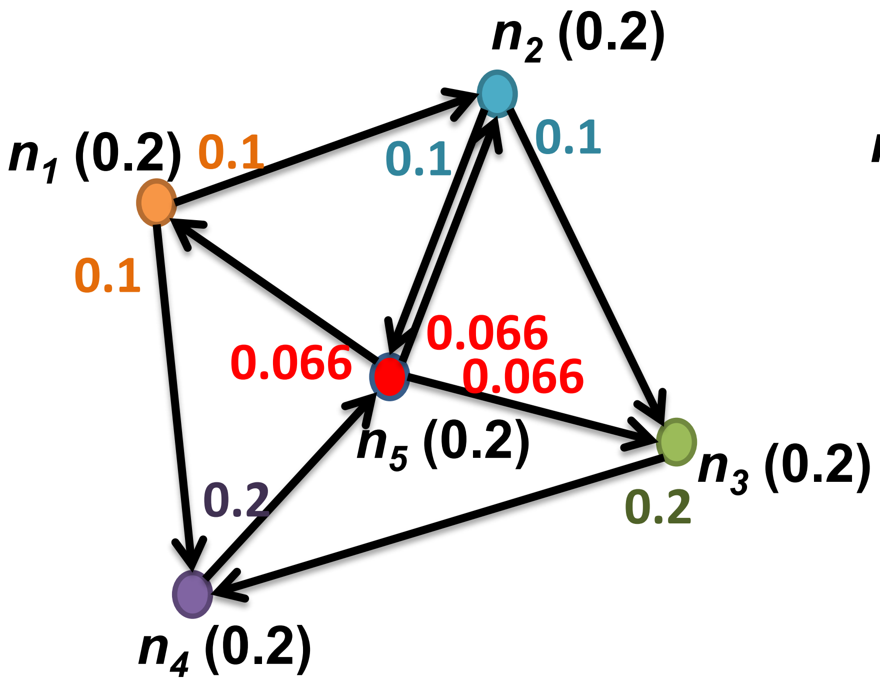
Computing PageRank

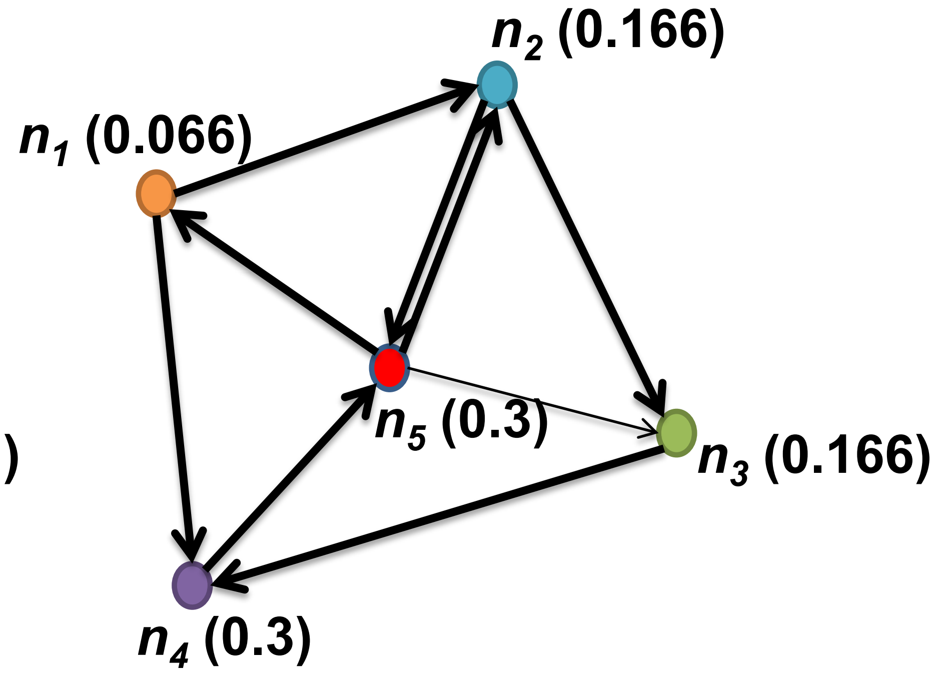
- Properties of PageRank
  - Can be computed iteratively
  - Effects at each iteration are local
- Sketch of algorithm:
  - Start with seed PR_i values
  - Each page distributes PR_i "credit" to all pages it links to
  - Each target page adds up "credit" from multiple in-bound links to compute PR_{i+1}
  - Iterate until values **converge**

PageRank Example

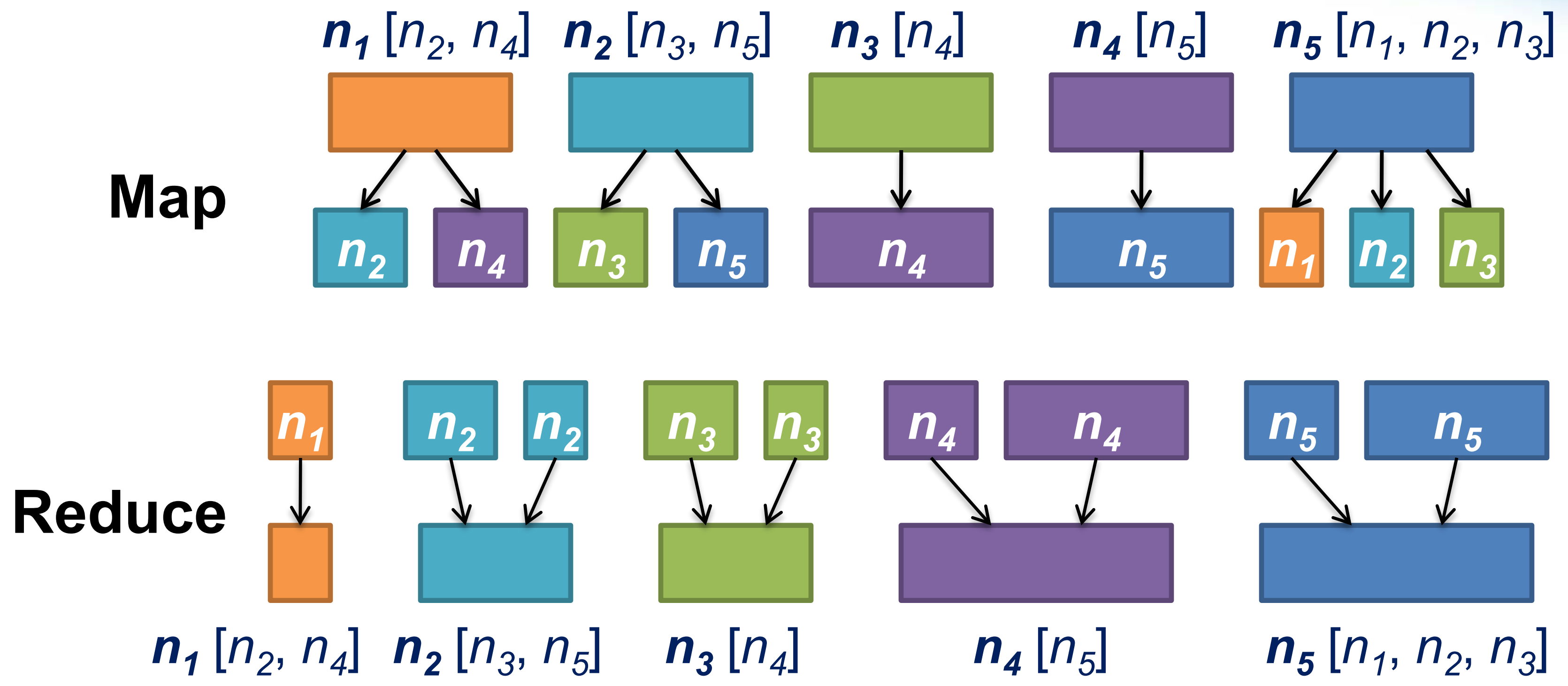
- Simplcase without random walk: $\alpha = 0$

$$PR(x) = \alpha\left(\frac{1}{N}\right) + (1-\alpha)\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$





PageRank in MapReduce



PageRank Pseudo-Code

```
class Mapper
    method Map(nid $n$, node $N$)
        $p \leftarrow N.\textsc{PageRank}/|N.\textsc{AdjacencyList}|$
        Emit(nid $n, N$)                                  ▷ Pass along graph structure
        for all nodeid $m \in N.\textsc{AdjacencyList}$ do
            Emit(nid $m, p$)                              ▷ Pass PageRank mass to neighbors

class Reducer
    method Reduce(nid $m, [p_1, p_2, \ldots]$)
        $M \leftarrow \emptyset$
        for all $p \in$ counts $[p_1, p_2, \ldots]$ do
            if IsNode($p$) then
                $M \leftarrow p$                          ▷ Recover graph structure
            else
                $s \leftarrow s + p$                      ▷ Sums incoming PageRank contributions
        $M.\textsc{PageRank} \leftarrow s$
        Emit(nid $m$, node $M$)
```

PageRank Convergence

- Alternative convergence criteria
  - Iterate until PageRank values don’t change
  - Iterate until PageRank rankings don’t change
  - Fixed number of iterations

Reference

- http://hadoop.apache.org/common/docs/r1.0.3/mapred_tutorial.html
- http://hadoop.apache.org/common/docs/r1.0.3/api/org/apache/hadoop/mapred/JobConf.html
- http://developer.yahoo.com/hadoop/tutorial/module5.html
- **The PageRank Citation Ranking: Bringing Order to the Web.** Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) Technical Report. Stanford InfoLab.