Chapter 10

Graph Algorithms

Acknowledgements

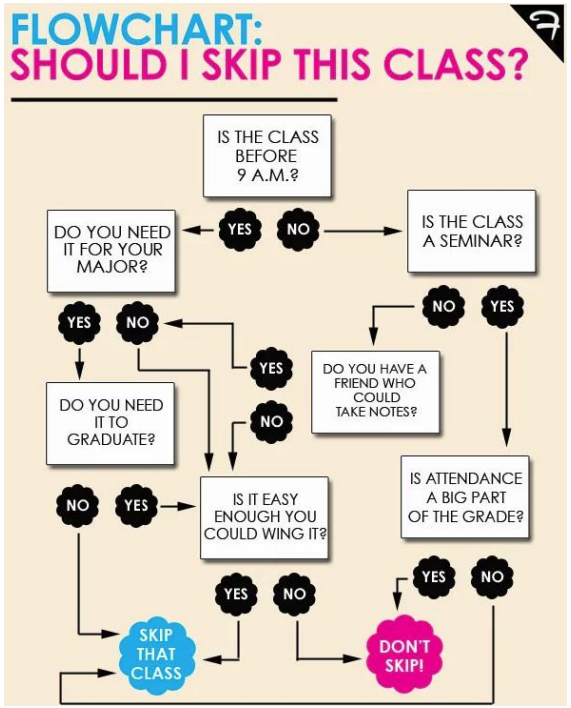
- Systems for Big-Graphs. Arijit Khan, Sameh Elnikety.
- Big Graph Search: Challenges and Techniques. Shuai Ma, Beihang University.
- Graph Search in the Big Data Era. Shuai Ma, Beihang University.
- Querying Big Social Data. Wenfei Fan. School of Informatics, University of Edinburgh.
- Algorithmic Techniques for Modeling and Mining Large Graphs (AMAzING). Alan Frieze, Aristides Gionis, Charalampos E. Tsourakakis. Carnegie Mellon University, Aalto University.
- Big Data Algorithms. Hongzhi Wang, HIT.
- Social Network Friends Suggestion System using PHP, MySQL and jQuery. http://www.codedodle.com/2016/04/social-network-friends-suggestion.html.

Chapter Outline

- What is a graph?
- Graph problems
- Computation model
- A brief revisit
- Graph search

What's a Graph?

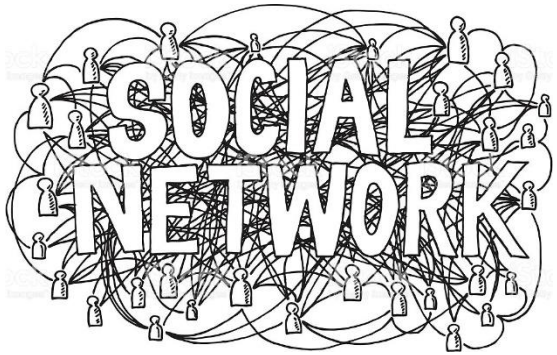
What Is A Graph?

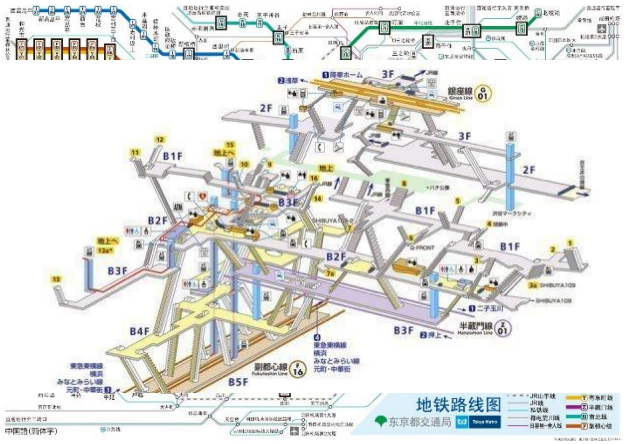


Image

Diagram 图 Map

Graph

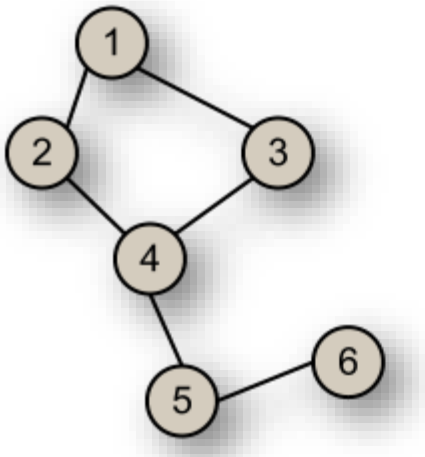




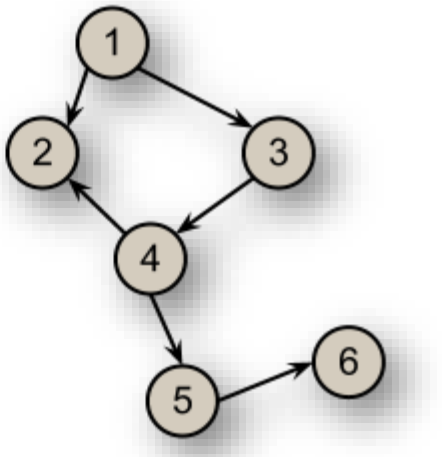
What Is the Underlying Graph?

- 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

What Is the Underlying Graph?

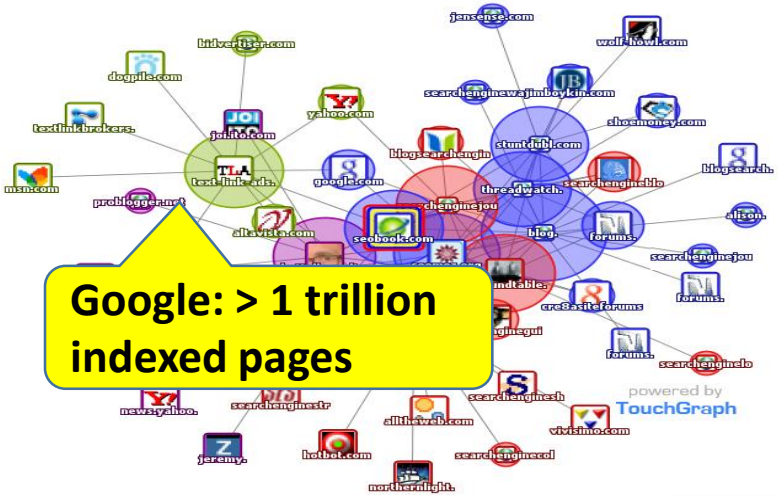


Undirected



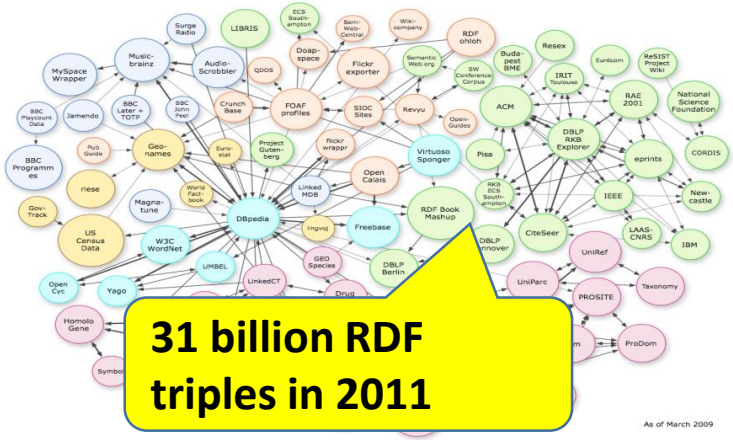
Directed

What Is Big Graphs?

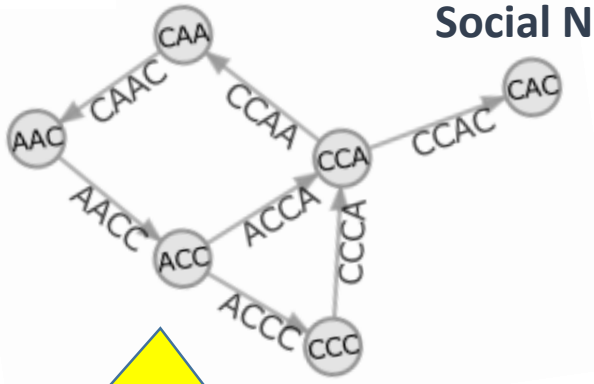


Web Graph

Social Network



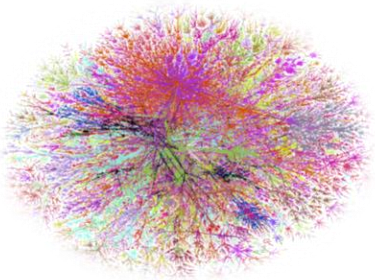
Information Network

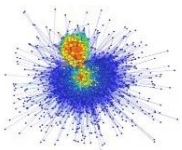


Biological Network

Graphs in Machine Learning

Big Graph Scales





100M(10^8)

Social Scale
100B (10^{11})

Web Scale
1T (10^{12})

Brain Scale, 100T (10^{14})

US Road

BTC
Semantic Web

Knowledge Graph

Internet

Web graph (Google)

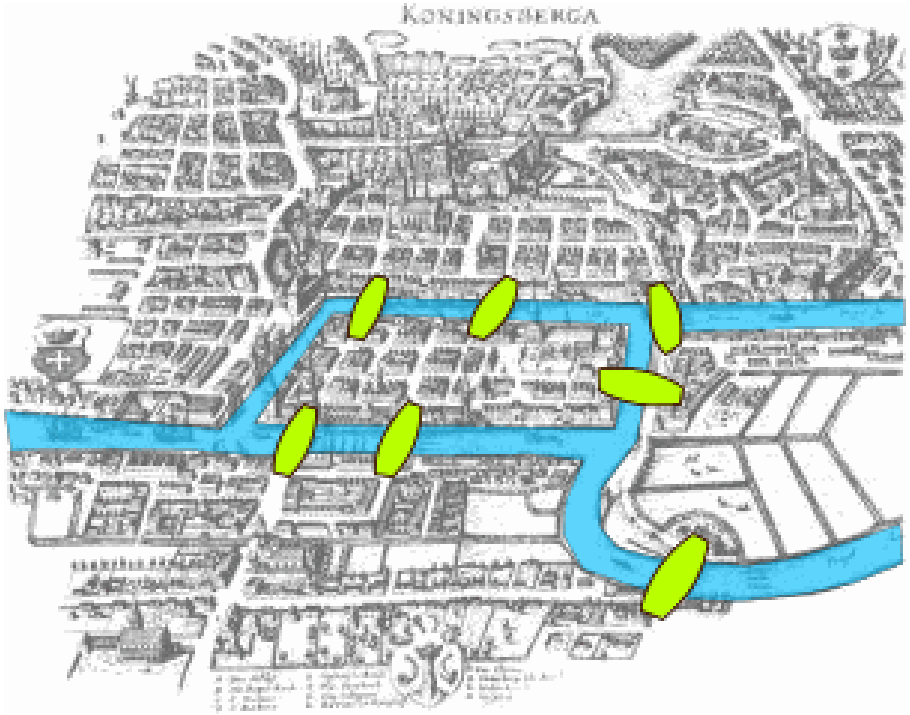
Human Connectome,
The Human Connectome Project, NIH

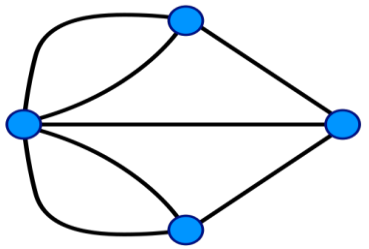
Acknowledgement: Y. Wu, WSU

Graph Applications & Problems

Applications

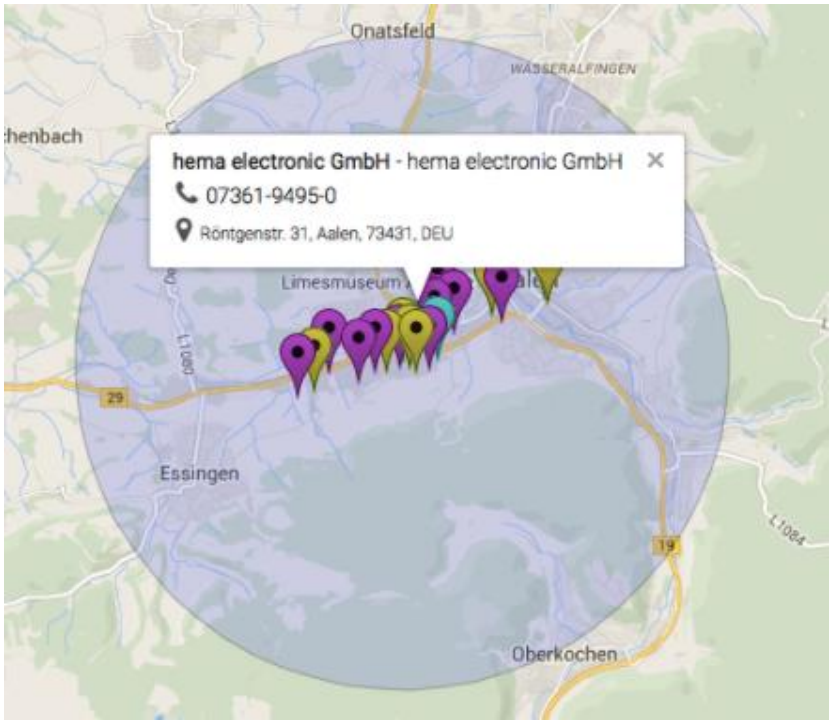
- Seven Bridges of Königsberg [Euler, 1735]
  - Return to the starting point by traveling each link of the graph once and only once.





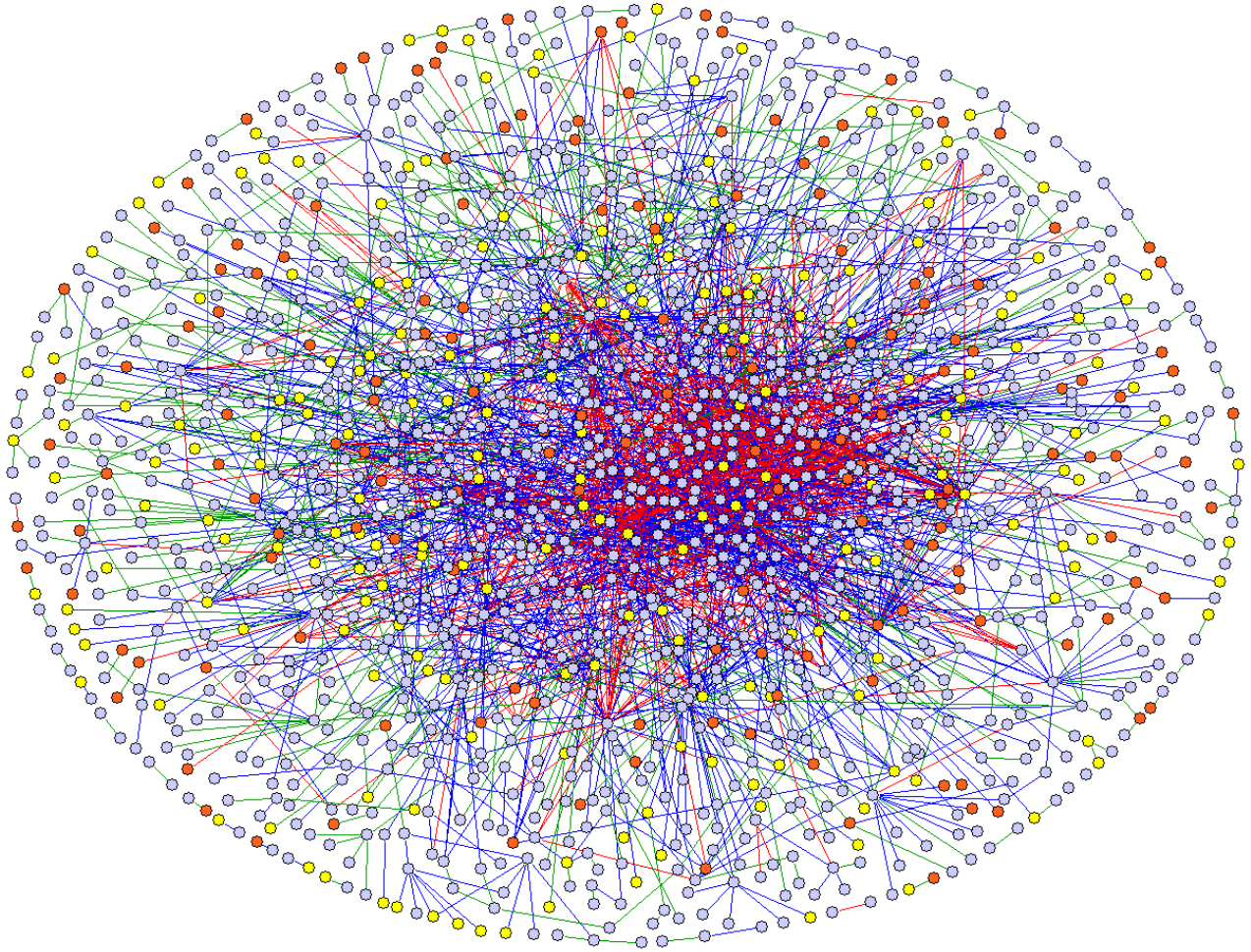
Applications

- Road network



Applications

- Protein-protein interactions (PPI networks)



Applications

- Social Network Services
  - Instant messaging
  - Social Network
  - Content Streaming
  - Sharing platform
- Applications
  - Friends recommendation
  - Community detection
  - Public opinions analysis
  - Dynamic evolution
  - Influence maximization

WordPress

六维空间

Facebook

Graph Analytics

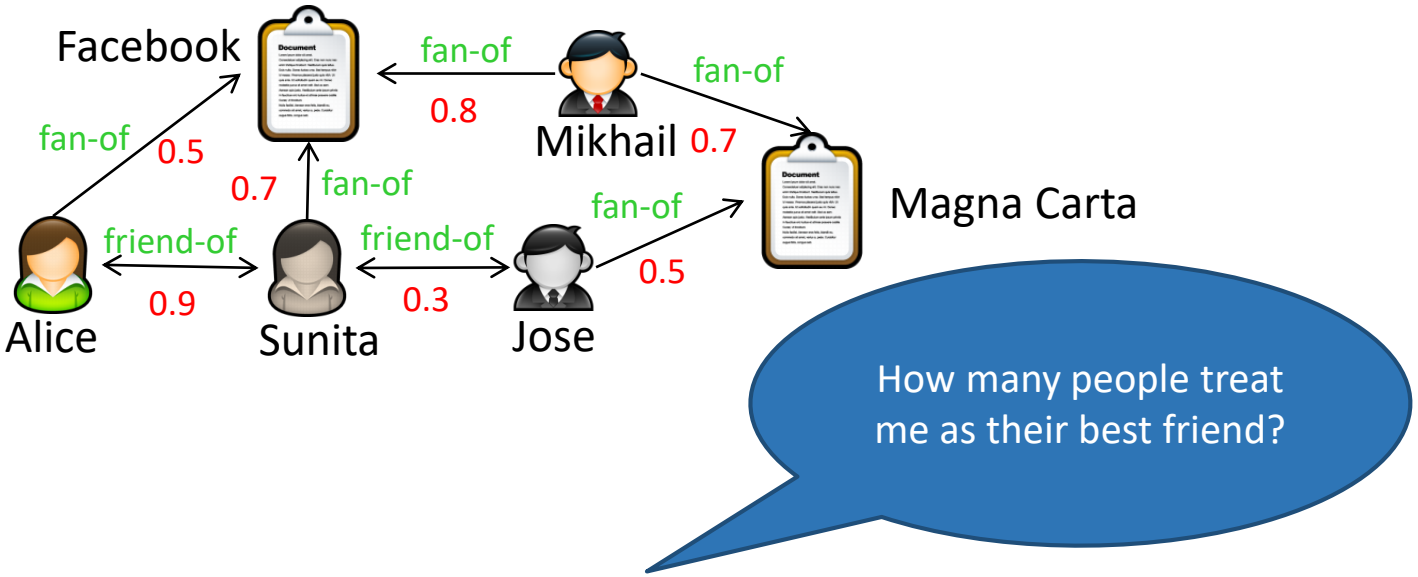
- General Graph
  - Count the number of nodes whose degree is equal to 5
  - Find the diameter of the graphs
- Web Graph
  - Rank each webpage in the web graph or each user in the twitter graph using PageRank, or other centrality measure
- Transportation Network
  - Return the shortest or cheapest flight/road from one city to another
- Social Network
  - Detect a group of users who have similar interests
- Financial Network
  - Find the path connecting two suspicious transactions;
- … …

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telco laying down fiber
- Finding Max Flow
  - Airline/Train scheduling
- Identify “special” nodes and communities
  - Breaking up terrorist cells, spread of avian flu
- Bipartite matching
  - Monster.com, Match.com

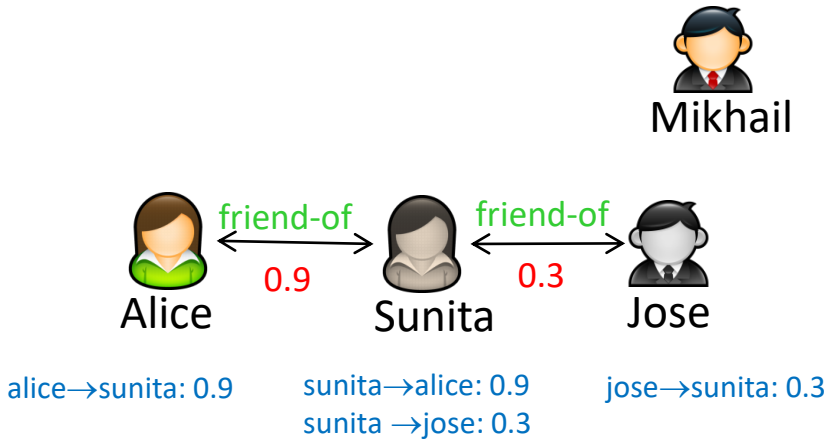
Computation Model

Example



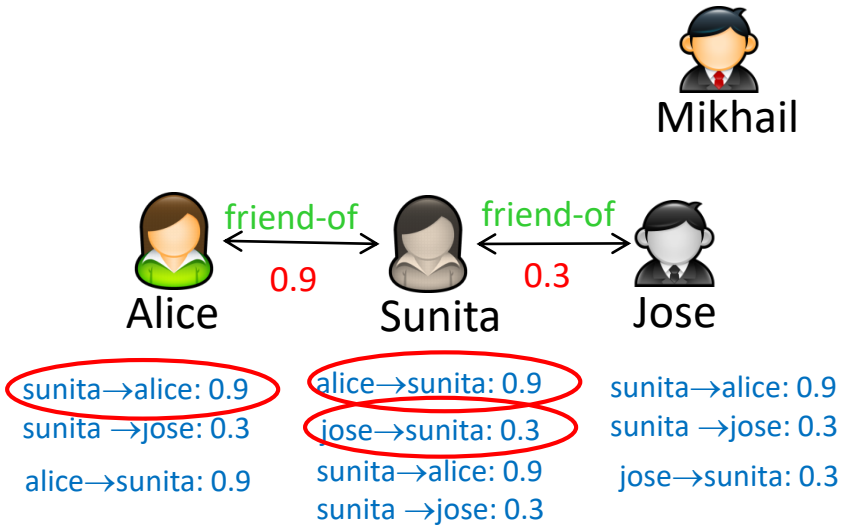
- Am I the best friend of my friend?
  - Step #1: remove irrelevant edges and nodes

Example



- Am I the best friend of my friend?
  - Step #1: remove irrelevant edges and nodes
  - Step #2: Mark each node with friends list
  - Step #3: Traverse the nodes through each edge

Example



- Am I the best friend of my friend?
  - Step #1: remove irrelevant edges and nodes
  - Step #2: Mark each node with friends list
  - Step #3: Traverse the nodes through each edge
  - Step #4: Determine the final result of each node

Realizable via MapReduce?

```
map(key: node, value: [<otherNode, relType, strength>])
{


}
reduce(key: _________, values: list of __________)
{


}
```

- Graph represented via adjacency list?

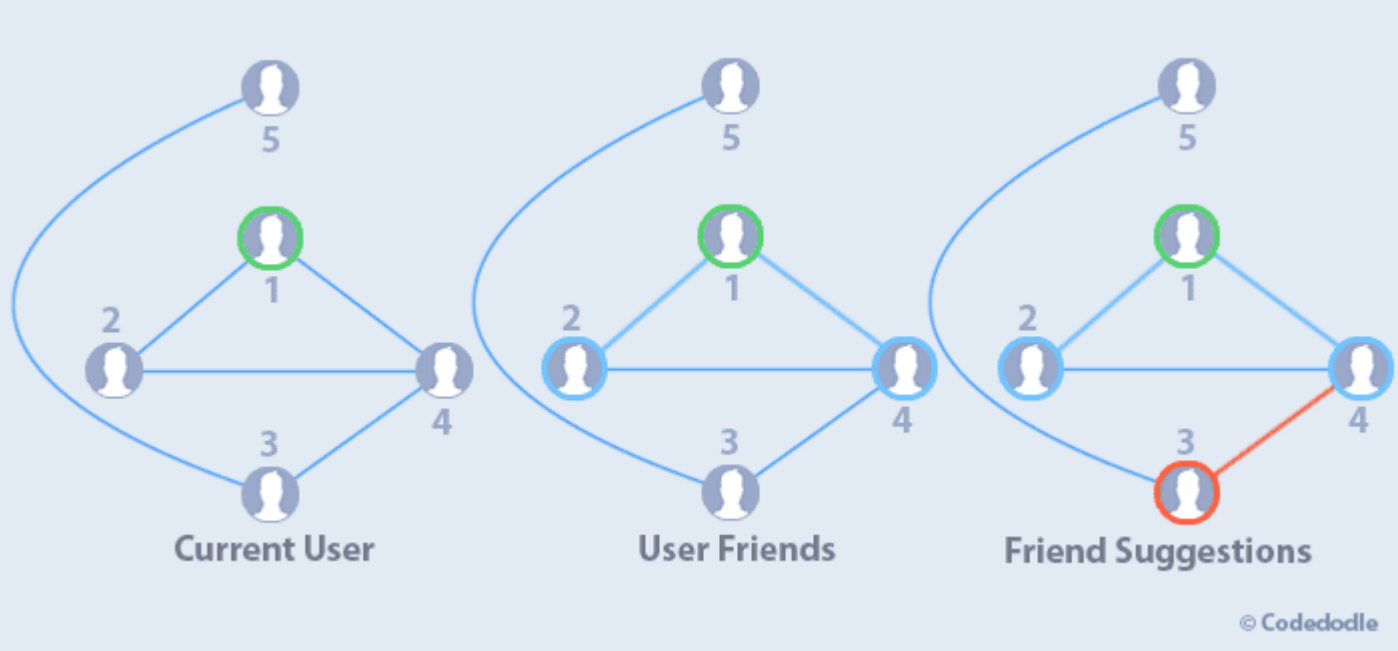
Realizable via MapReduce?

```
map(key: node, value: <otherNode, relType, strength>)
{


}
reduce(key: _________, values: list of __________)
{


}
```

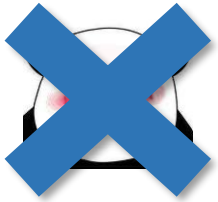
- Graph represented via adjacency matrix?

A General Case

- Who is the friend of multiple friends of mine?
- Friends recommendation!
  - He/She could be a friend of mine as well!



A Much More General Case



- Suppose we want a relationship of "more than friends"…
  - **How many people who are the friends of my friends treat me as the best friend of their best friend?**
  - How should we deal with this question?
- How about the *k*-hop (*k*>2)?
- We may need multiple rounds of MapReduce!

Iterative MapReduce

- Basic model

```
Input data from directory → dir 1
(Optional: preprocessing)

while (!Condition) {
  map from dir 1
  reduce → dir 2
  move files: from dir 2 → dir1
}

(Optional: postprocessing)
Move results from dir 2 → dir 3
```

Graphs and MapReduce

- Graph algorithms typically involve:
  - Perform local computations at each node
    - based on node features, edge features, and local link structure
  - Propagate computations
    - Traverse the graph through edges
    - *k*-hop nodes
  - Iterative computations
    - Multiple rounds of MapReduce
    - Output of round *i* → Input of round *i*+1
- Key questions:
  - How do you represent graph data in MapReduce?
  - How do you traverse a graph in MapReduce?

Revisit

Revisit

Representation

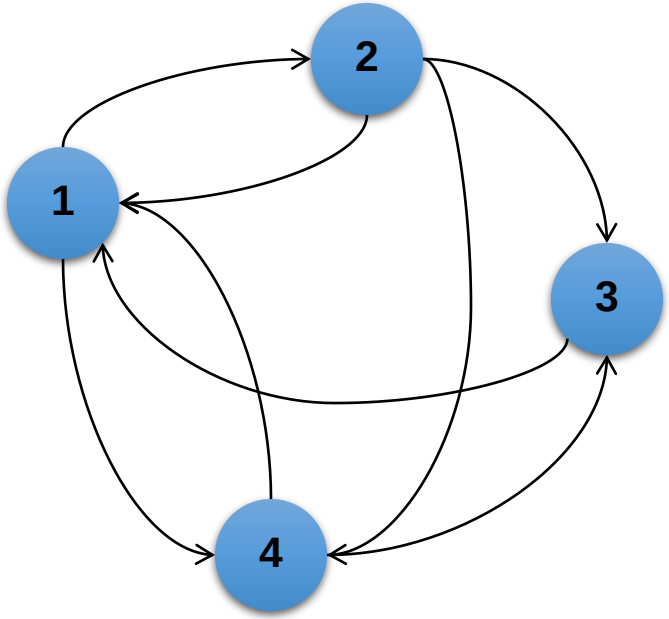
Shortest Path

PageRank

Representing Graphs

- 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*

	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 outlinks and inlinks
- Disadvantages:
  - Lots of zeros for sparse matrices
  - Lots of wasted space

Representing Graphs

- 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

Revisit

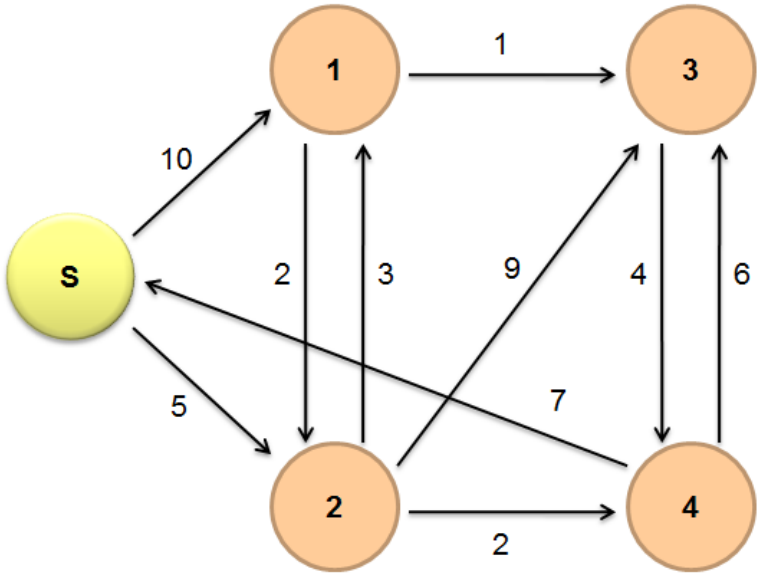
Representation

Shortest Path

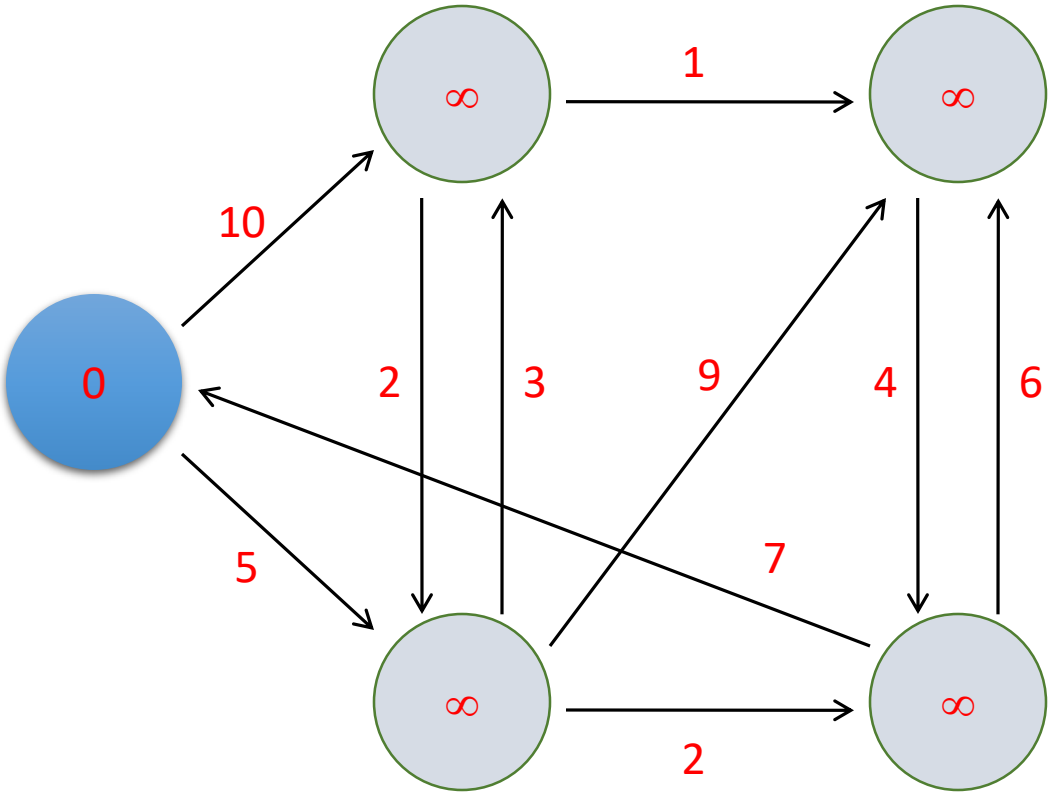
PageRank

Single-Source Shortest Path

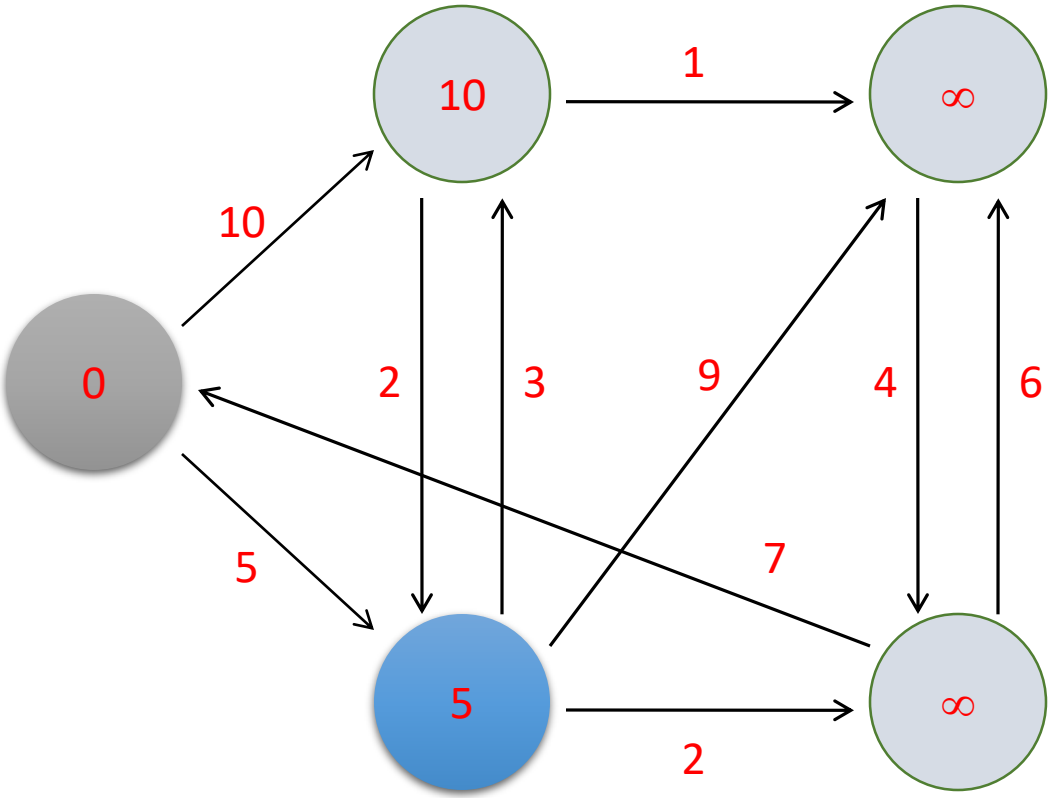
- **Problem:** find shortest path from a source node to one or more target nodes
  - Shortest might also mean lowest weight or cost
- Dijkstra's Algorithm:
  - For a given source node in the graph, the algorithm finds the shortest path between that node and every other



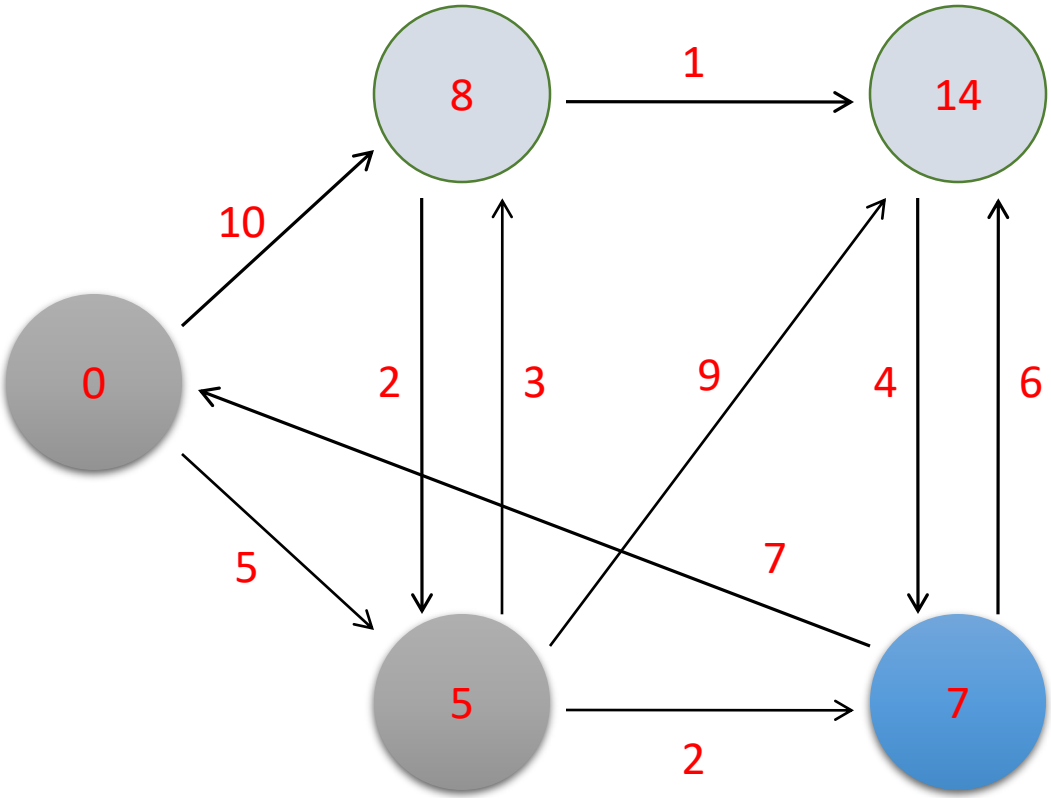
Dijkstra's Algorithm Example



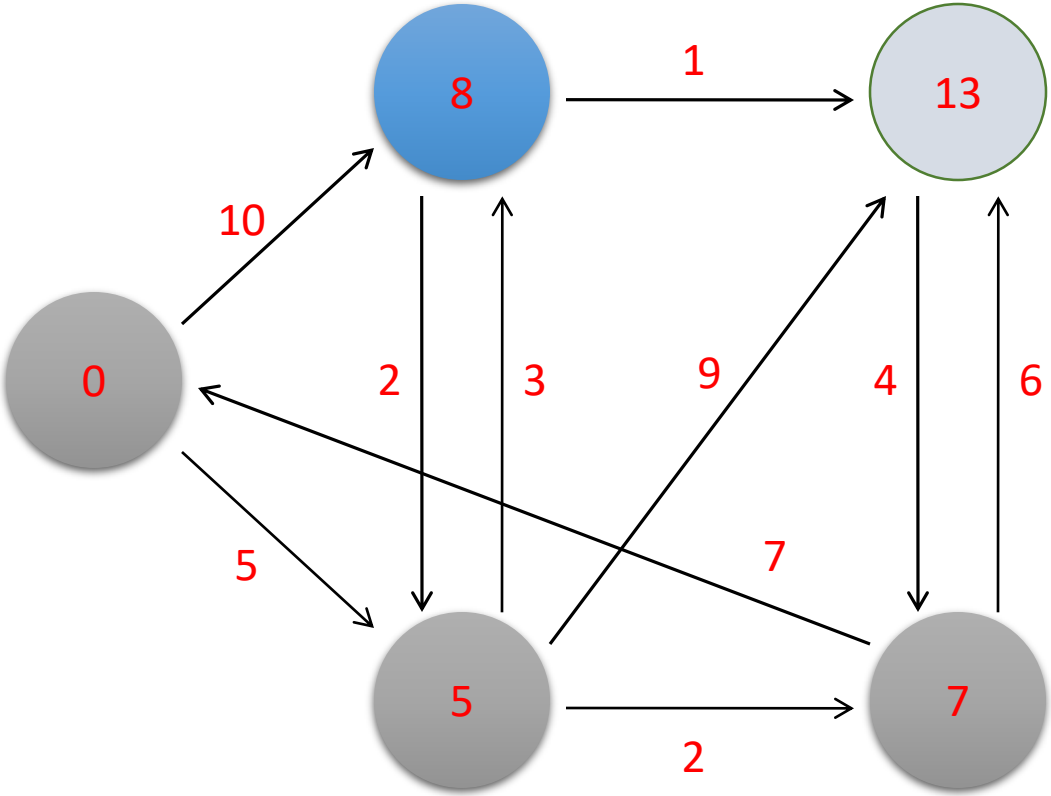
Dijkstra's Algorithm Example



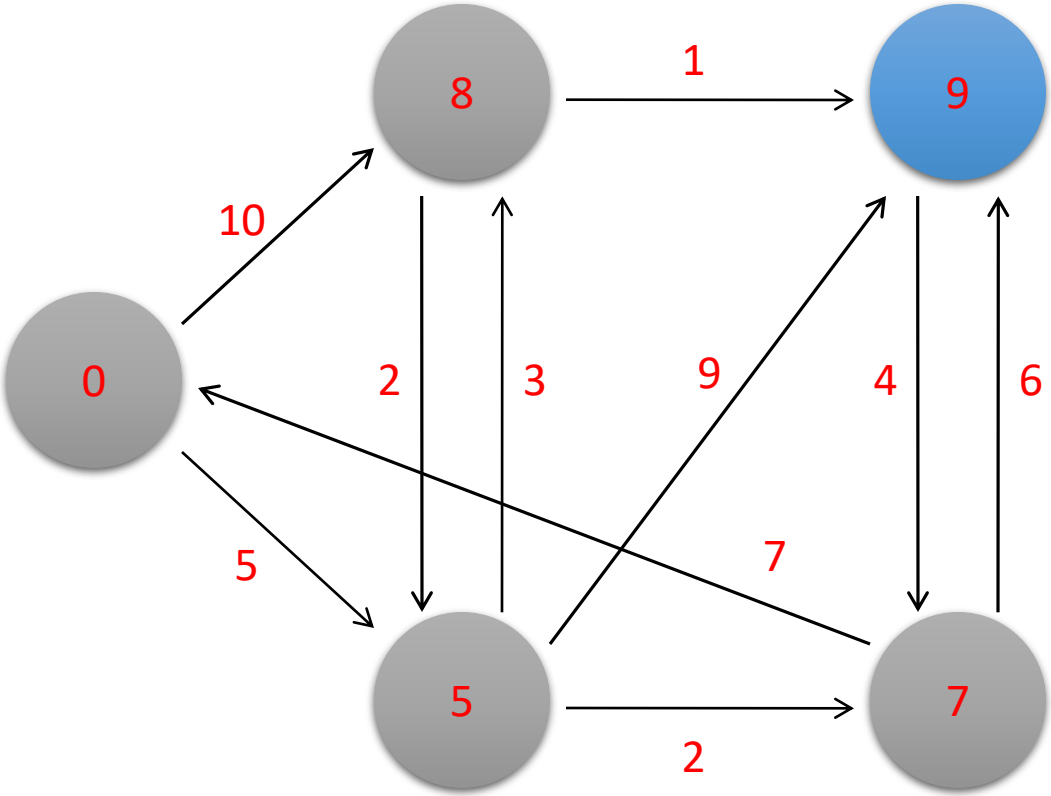
Dijkstra's Algorithm Example



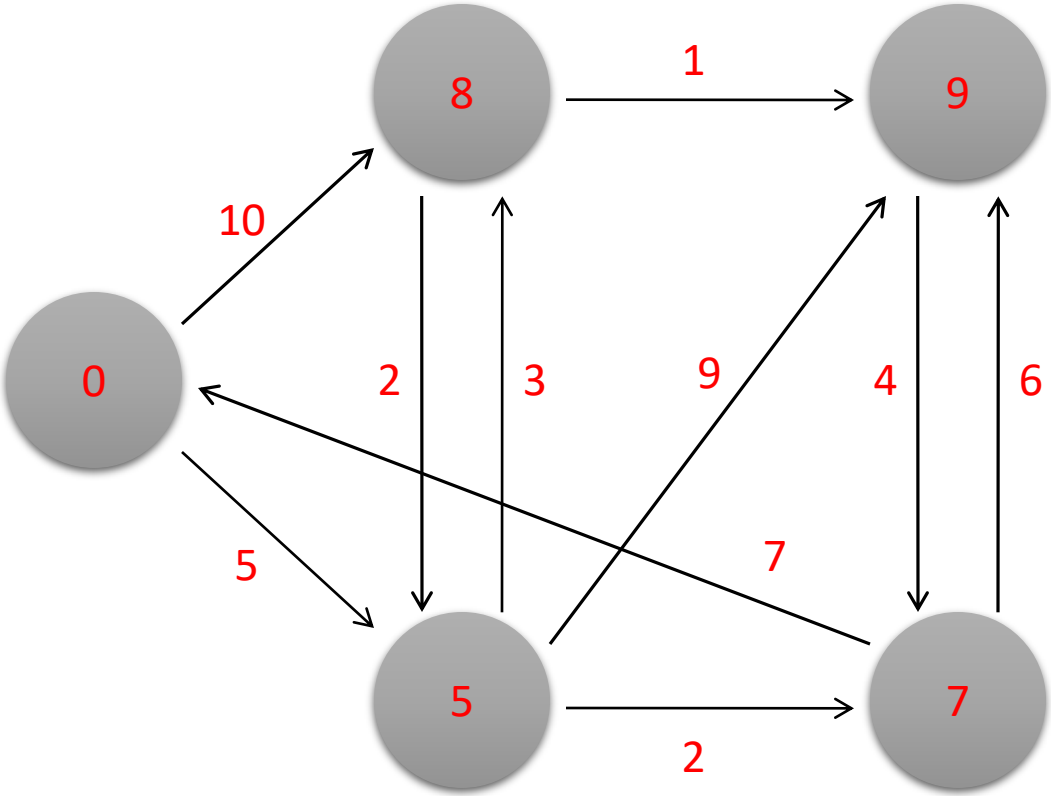
Dijkstra's Algorithm Example



Dijkstra's Algorithm Example



Dijkstra's Algorithm Example

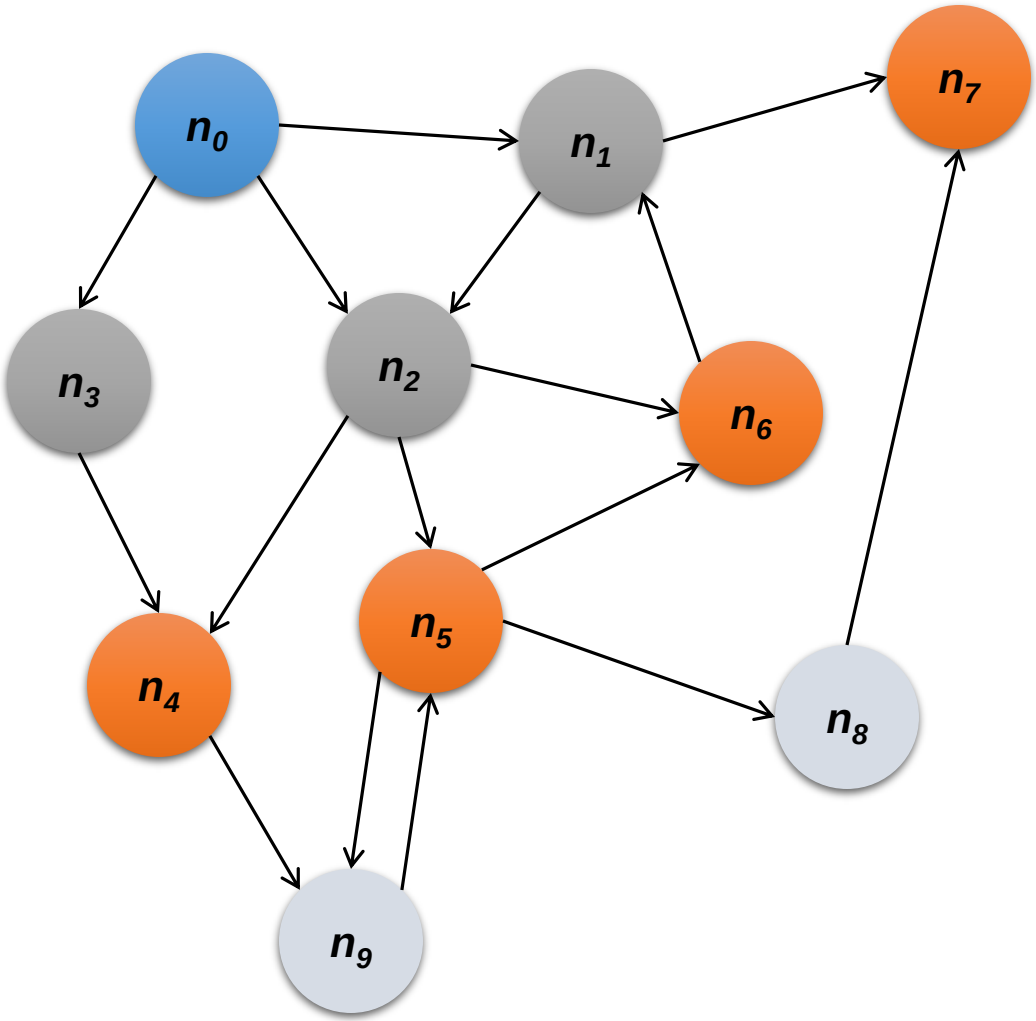


Finish!

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)

Visualizing Parallel BFS



From Intuition to Algorithm

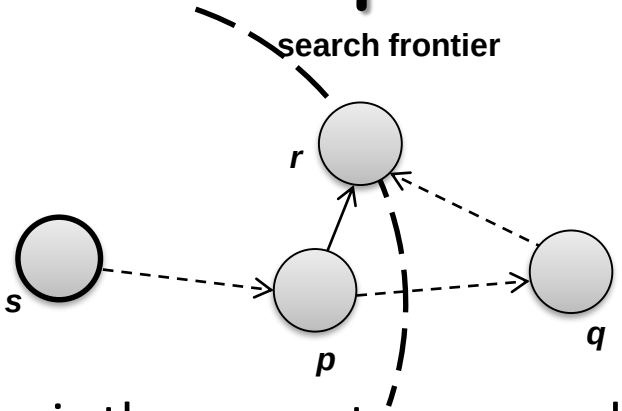
- Data representation:
  - Key: node *n*
  - Value: *d* (distance from start), adjacency list (list of nodes reachable from *n*)
  - Initialization: for all nodes except for start node, $d = \infty$
- Mapper:
  - $\forall m \in$ adjacency list: emit (m, $d + 1$)
- Sort/Shuffle
  - Groups distances by reachable nodes
- Reducer:
  - Selects minimum distance path for each reachable node
  - Additional bookkeeping needed to keep track of 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
  - The input of Mapper is the output of Reducer in the previous iteration
  - 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

Additional Complexities



- Assume that *p* is the current processed node
  - In the current iteration, we just "discovered" node r for the very first time.
  - We've already discovered the shortest distance to node *p*, and that the shortest distance to *r* so far goes through *p*
  - Is *s*->*p*->*r* the shortest path from *s* to *r*?
- The shortest path from source *s* to node *r* may go outside the current search frontier
  - It is possible that *p*->*q*->*r* is shorter than *p*->*r*!
  - We will not find the shortest distance to *r* until the search frontier expands to cover *q*.

Example (only distances)

- Input file:

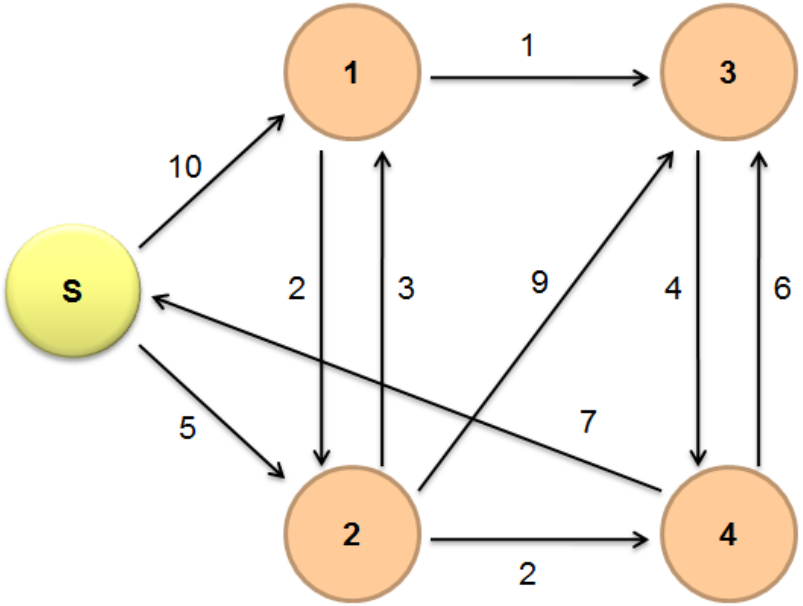
s --> 0 | n1: 10, n2: 5

n1 --> ∞ | n2: 2, n3:1

n2 --> ∞ | n1: 3, n3:9,

n3 --> ∞ | n4:4

n4 --> ∞ | s:7, n3:6



Iteration 1

- Map:

Read s --> 0 | n1: 10, n2: 5

Emit: (n1, 10), (n2, 5), and the adjacency list (s, n1: 10, n2: 5)

The other lists will also be read and emit, but they do not contribute, and thus ignored

- Reduce:

Receives: (n1, 10), (n2, 5), (s, <0, (n1: 10, n2: 5)>)

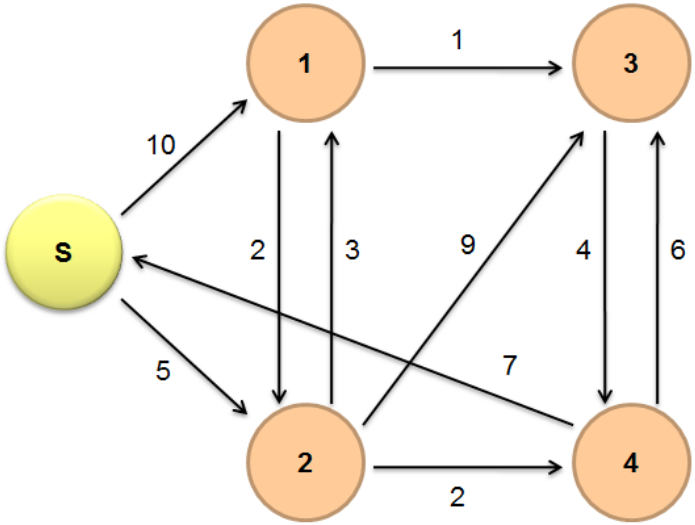
The adjacency list of each node will also be received, ignored in example

Emit:

s --> 0 | n1: 10, n2: 5

n1 --> 10 | n2: 2, n3:1

n2 --> 5 | n1: 3, n3:9， n4:2



Iteration 2

- Map:

Read: n1 --> 10 | n2: 2, n3:1

Emit: (n2, 12), (n3, 11), (n1, <10, (n2: 2, n3:1)>)

Read: n2 --> 5 | n1: 3, n3:9， n4:2

Emit: (n1, 8), (n3, 14), (n4, 7)，(n2, <5, (n1: 3, n3:9， n4:2)>)

Ignore the processing of the other lists

- Reduce:

Receives: (n1, (8, <10, (n2: 2, n3:1)>)), (n2, (12, <5, n1: 3, n3:9，n4:2>)), (n3, (11, 14)), (n4, 7)

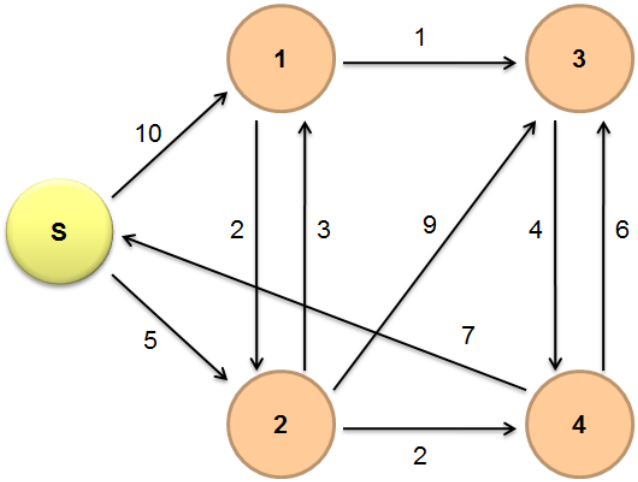
Emit:

n1 --> 8 | n2: 2, n3:1

n2 --> 5 | n1: 3, n3:9， n4:2

n3 --> 11 | n4:4

n4 --> 7 | s:7, n3:6



Iteration 3

- Map:

Read: n1 --> 8 | n2: 2, n3:1

Emit: (n2, 10), (n3, 9), (n1, <8, (n2: 2, n3:1)>)

Read: n2 --> 5 | n1: 3, n3:9， n4:2 (**Again!**)

Emit: (n1, 8), (n3, 14), (n4, 7)， (n2, <5, (n1: 3, n3:9， n4:2)>)

Read: n3 --> 11 | n4:4

Emit: (n4, 15)， (n3, <11, (n4:4)>)

Read: n4 --> 7 | s:7, n3:6

Emit: (s, 14), (n3, 13), (n4, <7, (s:7, n3:6)>)

- Reduce:

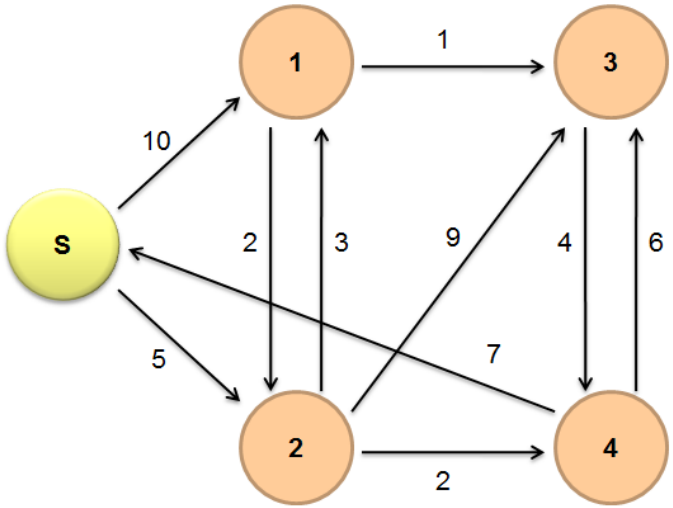
Emit:

n1 --> 8 | n2: 2, n3:1

n2 --> 5 | n1: 3, n3:9， n4:2

n3 --> 9 | n4:4

n4 --> 7 | s:7, n3:6



Iteration 4

- Map:

Read: n1 --> 8 | n2: 2, n3:1 (**Again!**)

Emit: (n2, 10), (n3, 9), (n1, <8, (n2: 2, n3:1)>)

Read: n2 --> 5 | n1: 3, n3:9， n4:2 (**Again!**)

Emit: (n1, 8), (n3, 14), (n4, 7)， (n2, <5, (n1: 3, n3:9， n4:2)>)

Read: n3 --> 9 | n4:4

Emit: (n4, 13)， (n3, <9, (n4:4)>)

Read: n4 --> 7 | s:7, n3:6 (**Again!**)

Emit: (s, 14), (n3, 13), (n4, <7, (s:7, n3:6)>)

- Reduce:

Emit:

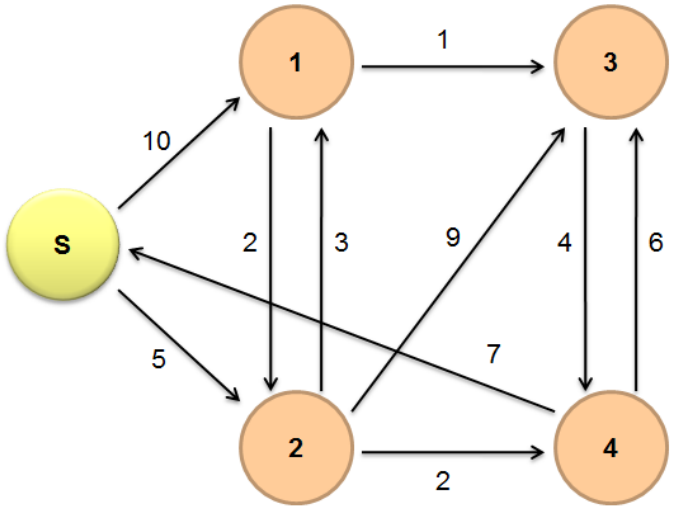
n1 --> 8 | n2: 2, n3:1

n2 --> 5 | n1: 3, n3:9， n4:2

n3 --> 9 | n4:4

n4 --> 7 | s:7, n3:6

In order to avoid duplicated computations, you can use a status value to indicate whether the distance of the node has been modified in the previous iteration.



No updates. Terminate.

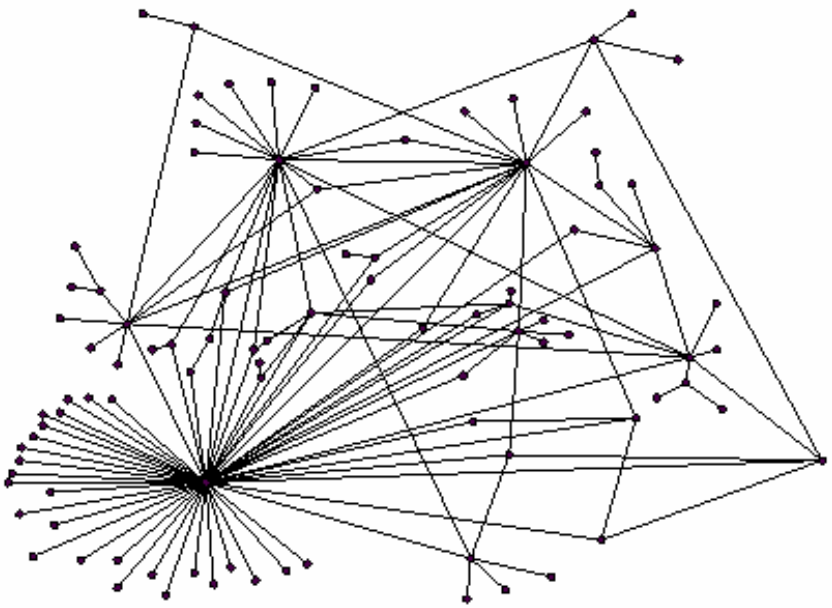
Revisit

Representation
Shortest Path
PageRank

Ranking Nodes on the Graph

- All web pages are not equally “important”
  - http://xxx.github.io/ vs. http://www.unsw.edu.au/

- There is large diversity in the web-graph node connectivity. Let’s rank the pages by the link structure!



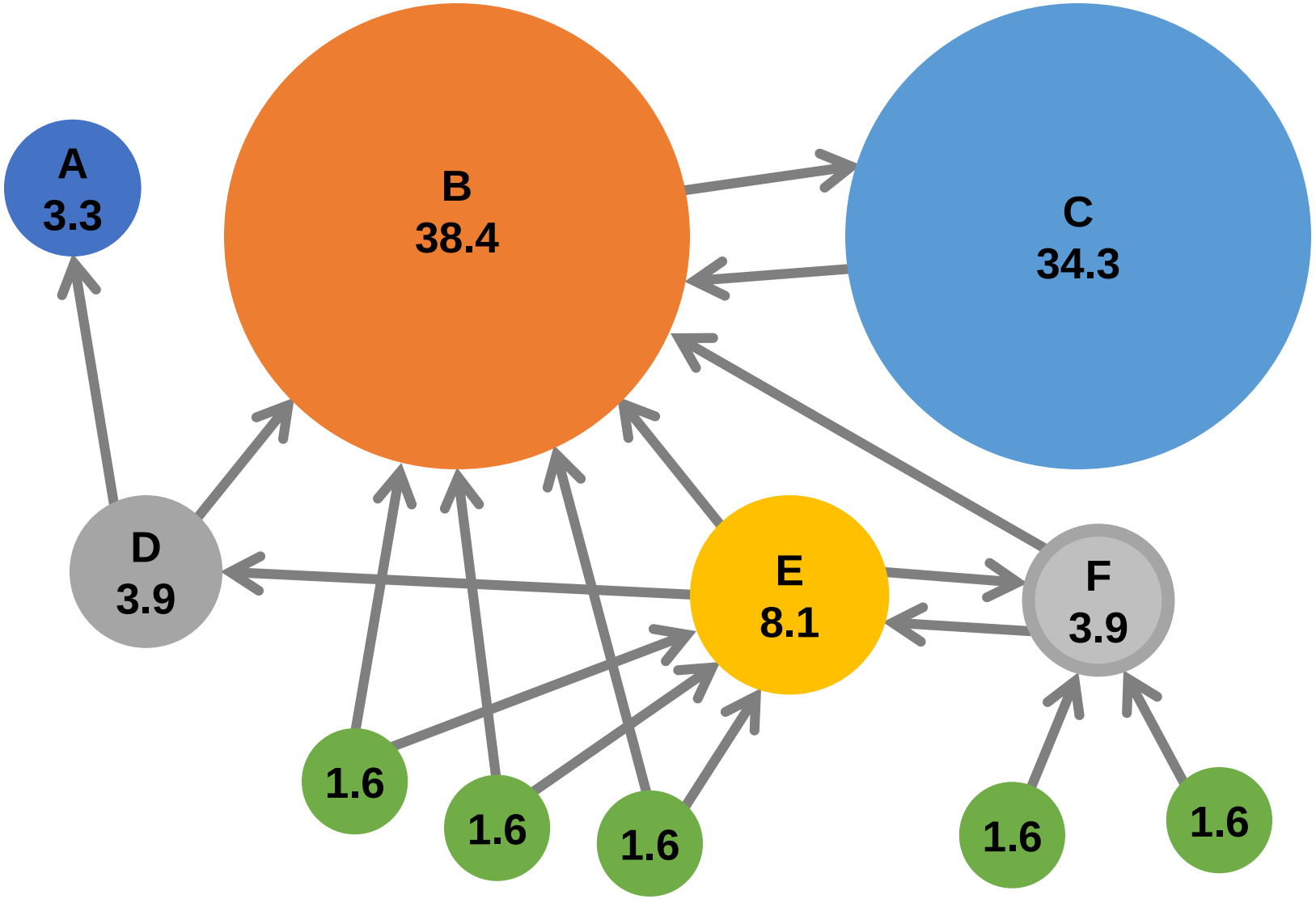
Links as Votes

- Idea: Links as votes
  - Page is more important if it has more links
  - In-coming links? Out-going links?

- Think of in-links as votes:
  - http://www.unsw.edu.au/ has 23,400 in-links
  - http://xxx.github.io/ has 1 in-link

- Are all in-links equal?
  - Links from important pages count more
  - Recursive question!

Example: PageRank Scores



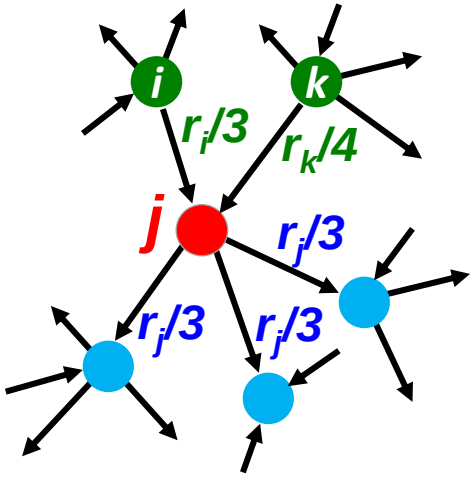
Simple Recursive Formulation

- 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

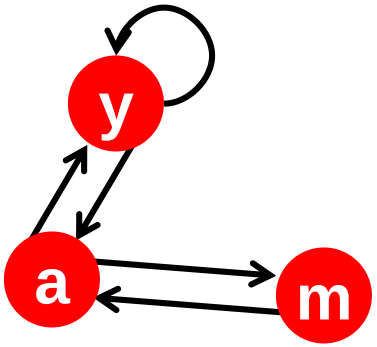
- Page ***j***'s own importance is the sum of the votes on its in-links

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



PageRank: How to solve?

- Power Iteration:
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

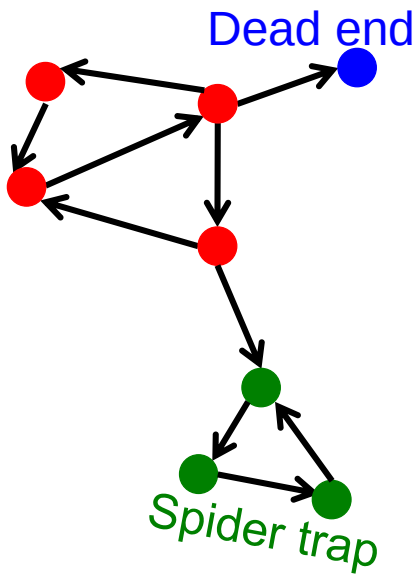
- Example:

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

PageRank: Problems

2 problems:

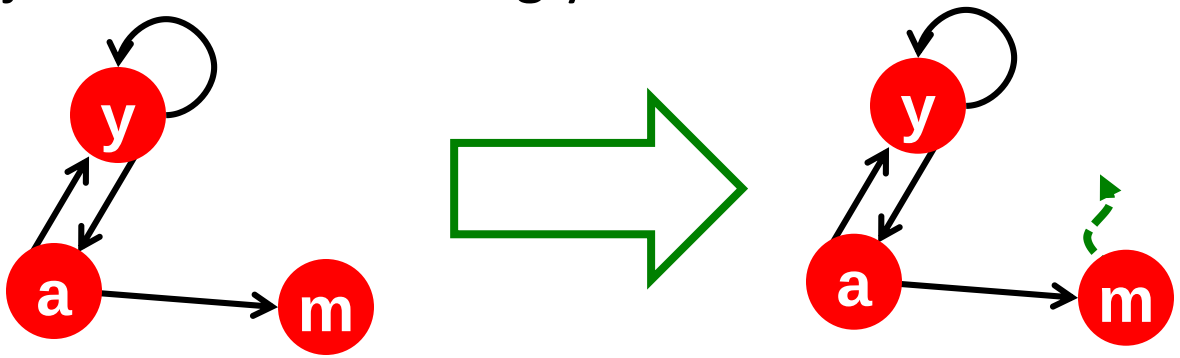
- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"



- **(2) Spider traps:** (all out-links are within the group)
  - Random walked gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Solution: Teleport!

- Teleports: Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly

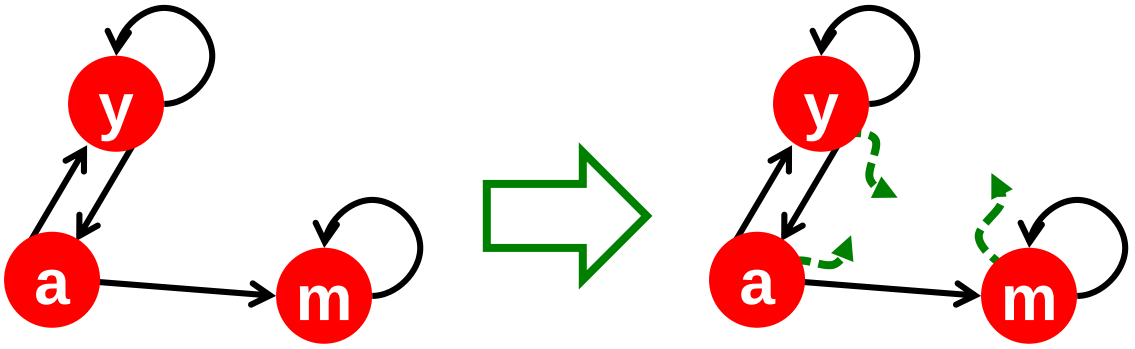


	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Solution: Always Teleports!

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



Why Teleports Solve the Problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

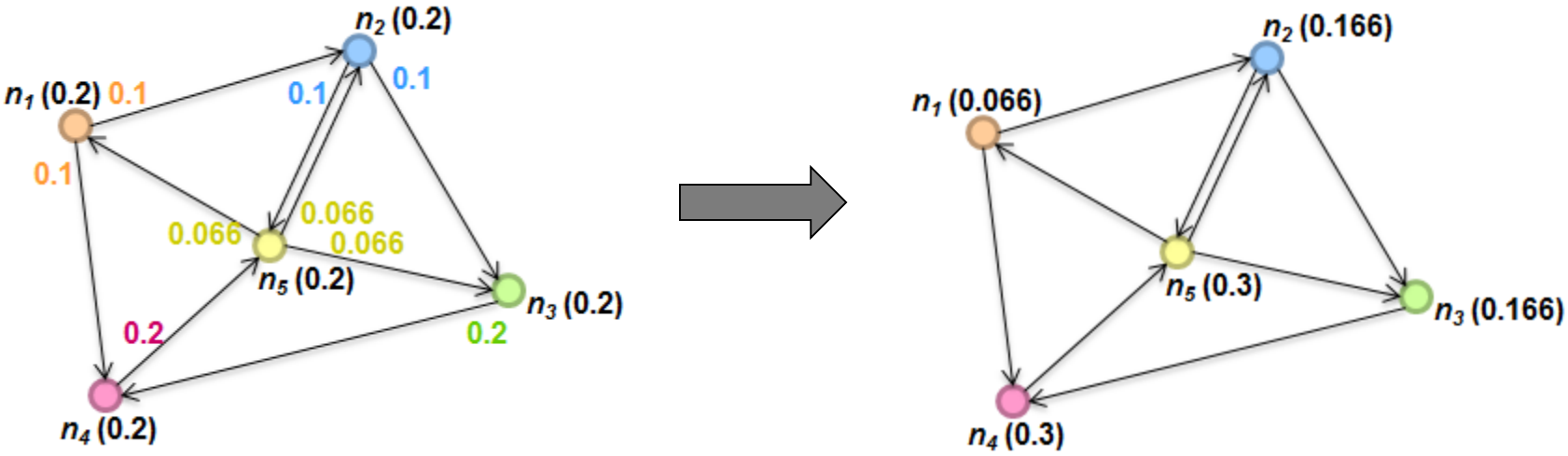
PageRank Computation Review

- Properties of PageRank
  - Can be computed iteratively
  - Effects at each iteration are local
- Sketch of algorithm:
  - Start with seed r_i values
  - Each page distributes r_i "credit" to all pages it links to
  - Each target page t_j adds up "credit" from multiple in-bound links to compute r_j
  - Iterate until values converge

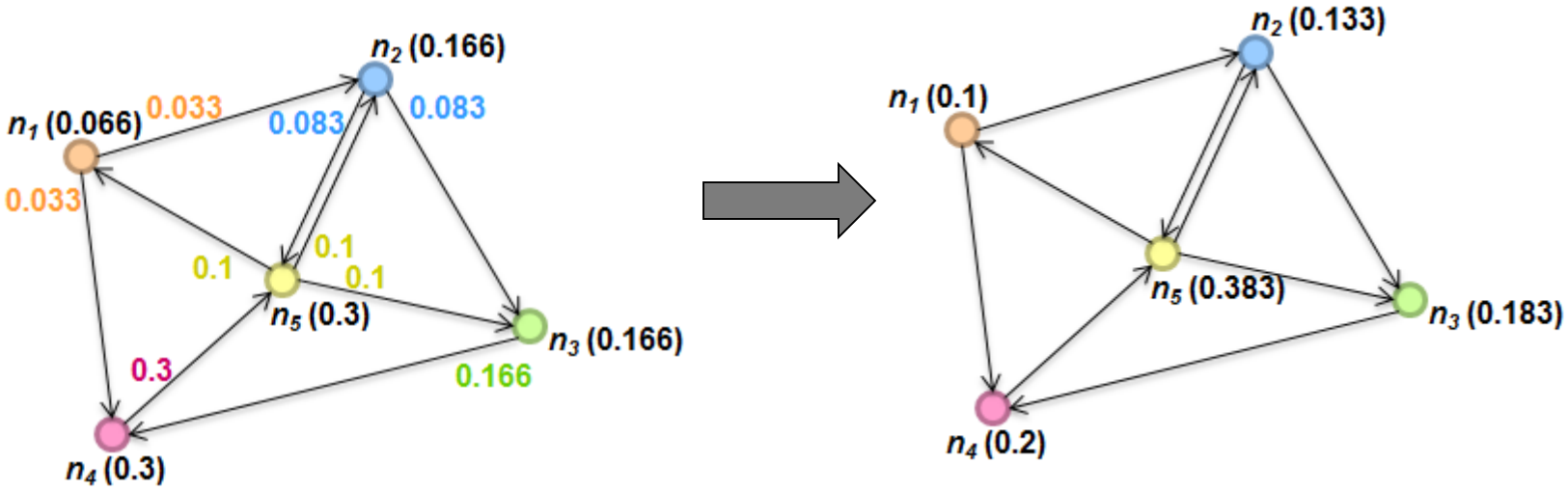
Simplified PageRank

- First, tackle the simple case:
  - No teleport
  - No dead ends
- Then, factor in these complexities…
  - How to deal with the teleport probability?
  - How to deal with dead ends?

Sample PageRank Iteration (1)

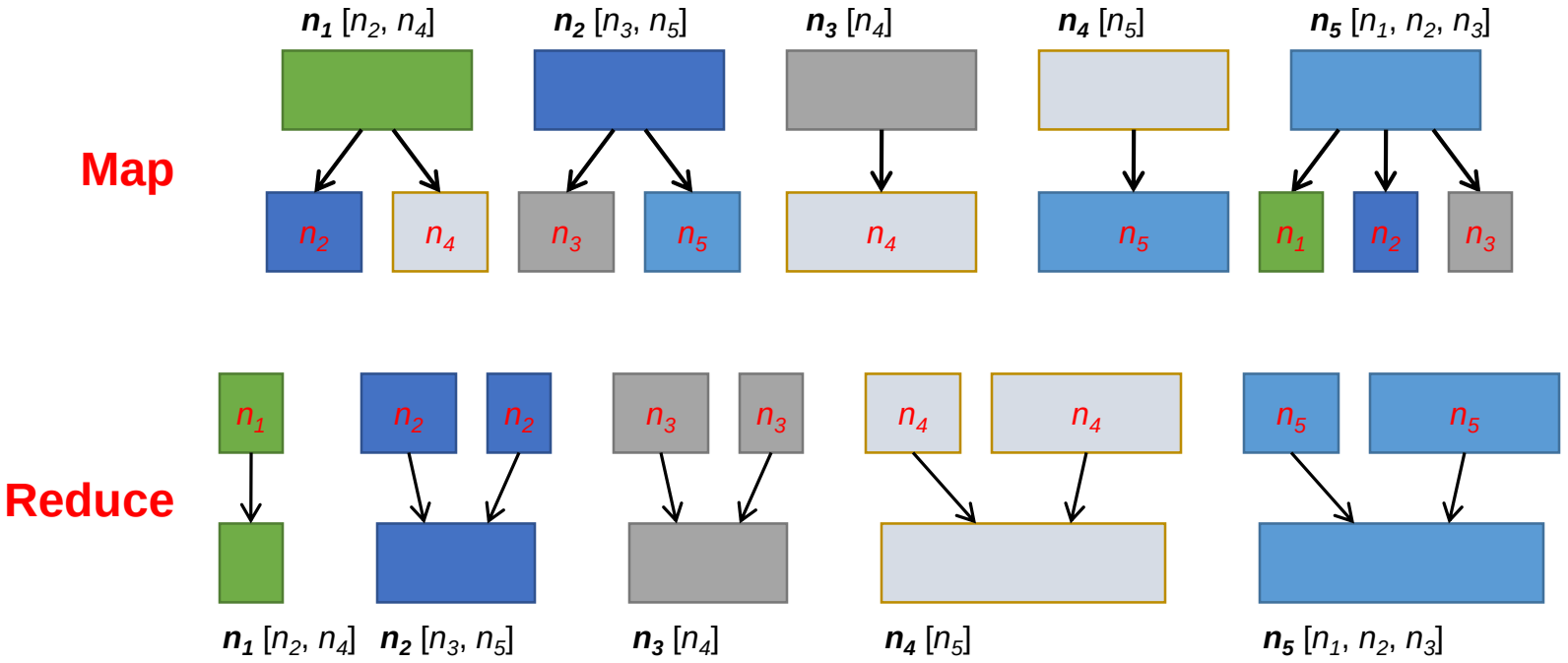


Sample PageRank Iteration (2)



PageRank in MapReduce

- One Iteration



Complete PageRank

- Two additional complexities
  - What is the proper treatment of dangling nodes?
  - How do we factor in the random jump factor?
- Solution:
  - If a node's adjacency list is empty, distribute its value to all nodes evenly.
    - In mapper, for such a node *i*, emit (nid m, r_i/N) for each node m in the graph
  - Add the teleport value
    - In reducer, M.PageRank = $\beta * s + (1-\beta) / N$

More Tools on Big Graph

- Graph databases: Storage and Basic Operators
  - http://en.wikipedia.org/wiki/Graph_database
  - Neo4j (an open source graph database)
  - InfiniteGraph
  - VertexDB

- Distributed Graph Processing (mostly in-memory-only)
  - Google's Pregel (vertex centered computation)
  - Giraph (Apache)
  - GraphX (Spark)
  - GraphLab
  - … …

Graph Search

Graph Search

What Is Graph Search?
Reachability Queries
Subgraph Isomorphism
Large Graph Processing

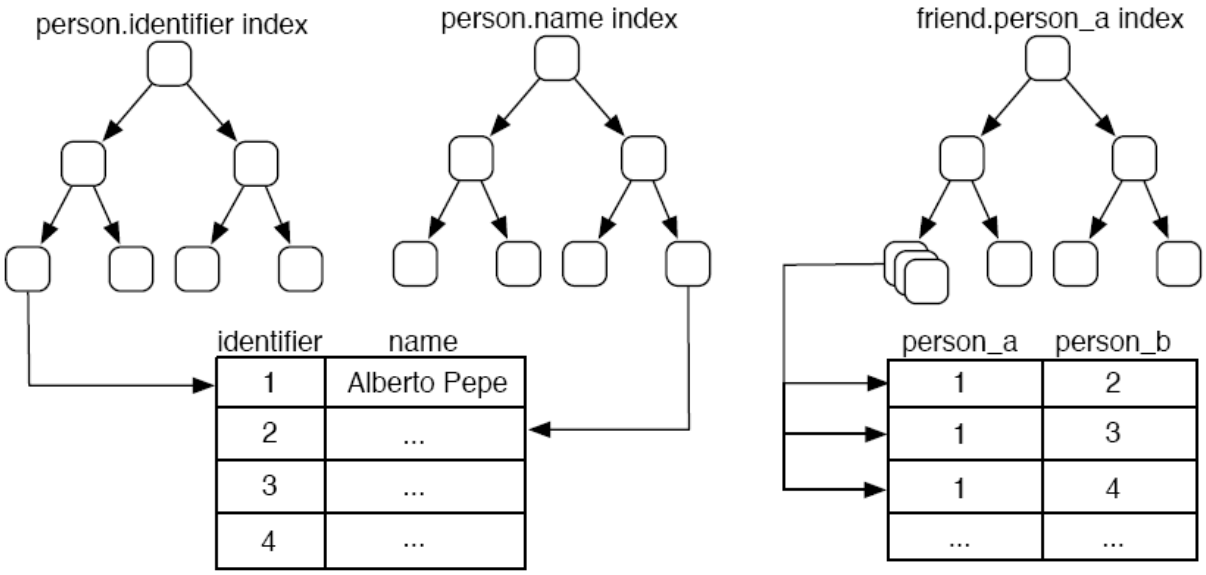
What is Graph Search?

- A unified definition (in the name of graph matching)
  - Given a pattern graph G_p and a data graph G:
    - check whether G_p ‘‘matches’’ G; and
    - identify all ‘‘matched’’ subgraphs.
- Remarks
  - Two classes of queries:
    - Boolean queries (Yes or No)
    - Functional queries, which may use Boolean queries as a subroutine
  - Graphs contain a set of nodes and a set of edges, typically with labels
  - Pattern graphs are typically small (e.g., 10), but data graphs are usually huge (e.g., 10^8)
- Matching Semantics
  - Traditional: Subgraph Isomorphism
  - Emerging applications: Graph Simulation and its extensions, etc.

What is Graph Search?

- Different semantics of "match" implies different "types" of graph search, including, but not limited to, the following:
  - Shortest paths/distances
  - Subgraph isomorphism
  - Graph homomorphism and its extensions
  - Graph simulation and its extensions
  - Graph keyword search
  - Neighborhood queries
  - …

Graph Search vs. RDBMS



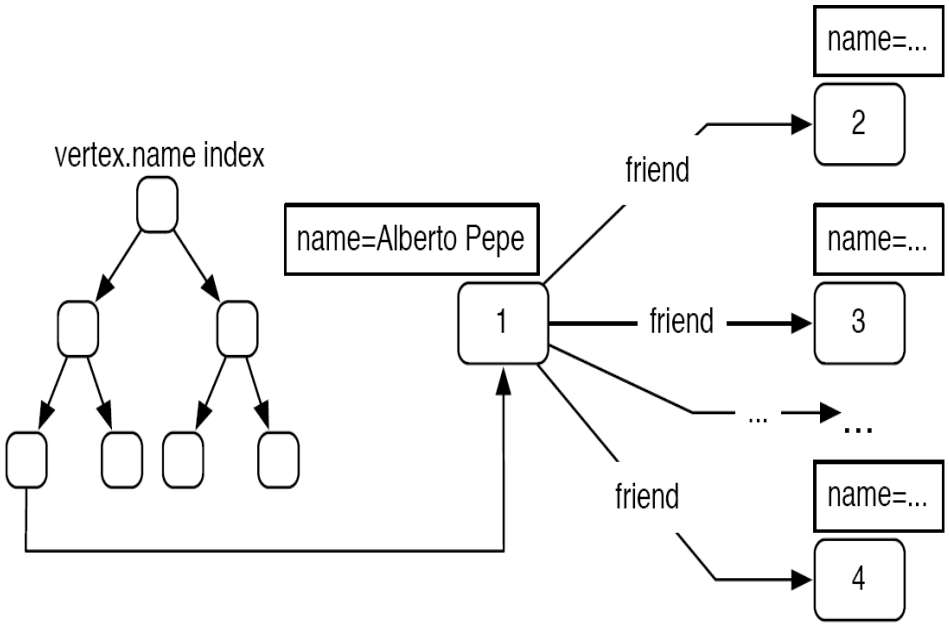
Query:
Find the name of all of Alberto Pepe's friends.

Step 1: The person.name index -> the identifier of Alberto Pepe. [$O(\log_2 n)$]

Step 2: The friend.person index -> k friend identifiers. [$O(\log_2 x) : x<<m$]

Step 3: The k friend identifiers -> k friend names. [$O(k \log_2 n)$]

Graph Search vs. RDBMS

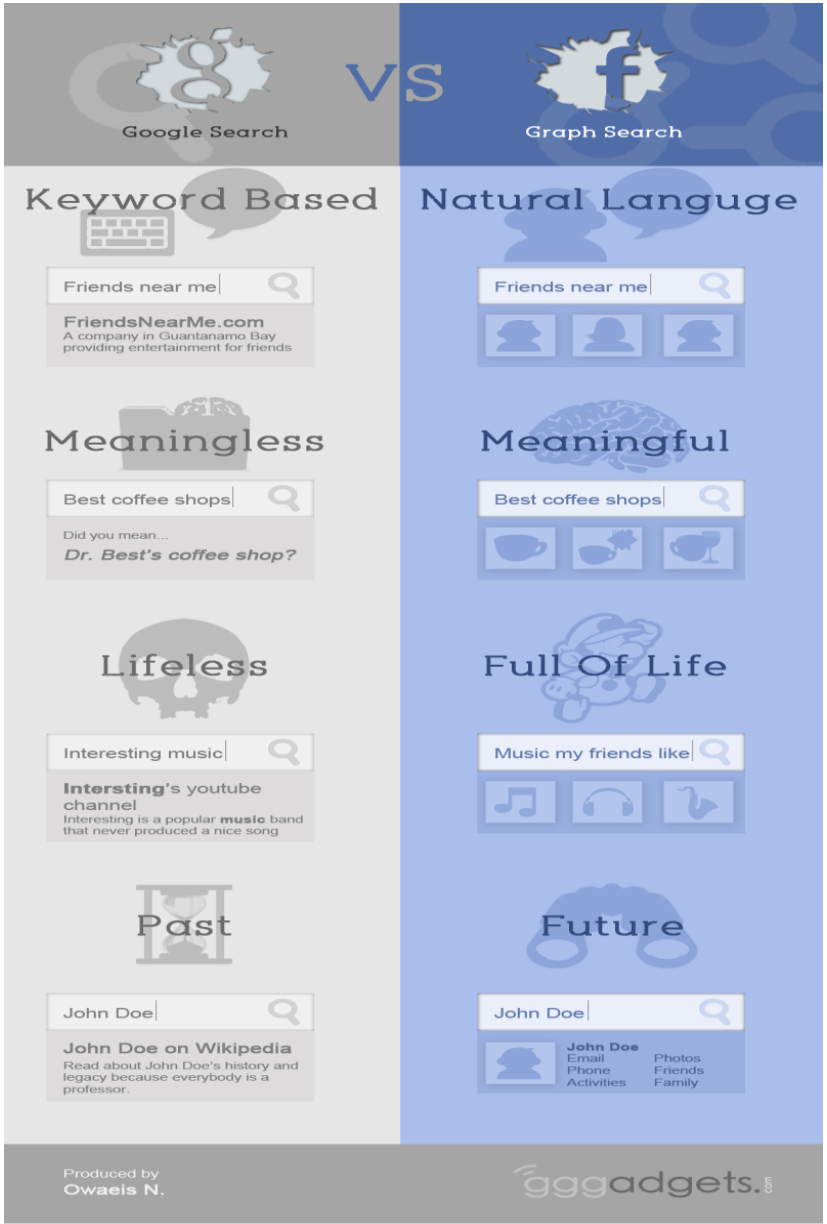


Query:
Find the name of all of Alberto Pepe's friends.

Step 1: The vertex.name index -> the vertex with the name Alberto Pepe. [$O(\log_2 n)$]

Step 2: The vertex returned -> the k friend names. [$O(k + x)$]

Social Search vs. Web Search



- key words only vs. Phrases、 short sentences
- (Simple Web) pages vs. Entities
- Lifeless vs. Full of life
- History vs. Future

it's interesting, and over the last 10 years, people have been trained on how to use search engines more effectively.
[Keywords & Search In 2013: Interview With A. Goodman & M. Wagner]

International Conference on Application of Natural Language to Information Systems (NLDB) **started from 1995**

Graph Search

What Is Graph Search?

Reachability Queries

Subgraph Isomorphism

Large Graph Processing

Distributed graph pattern matching

The cost of a batch matching algorithm: **f**(|G|, |Q|)

reduce the parameter?

manageable sizes

Divide and conquer

- ✓ partition G into fragments (G1, …, Gn), distributed to various sites
- ✓ upon receiving a query Q,

evaluate Q on smaller Gi

  - evaluate Q(Gi) *in parallel*
  - collect partial matches at a coordinator site, and assemble them to find the answer Q(G) in the entire G

Social graphs are already geometrically distributed

How to compute Q(Gi) with performance guarantees?

Partial evaluation

compute **f**(x) ⇒ **f**(**s**, **d**)

the part of known input

yet unavailable input

only on **s**

- ✓ generate a partial answer

at each site, Gi as the known input

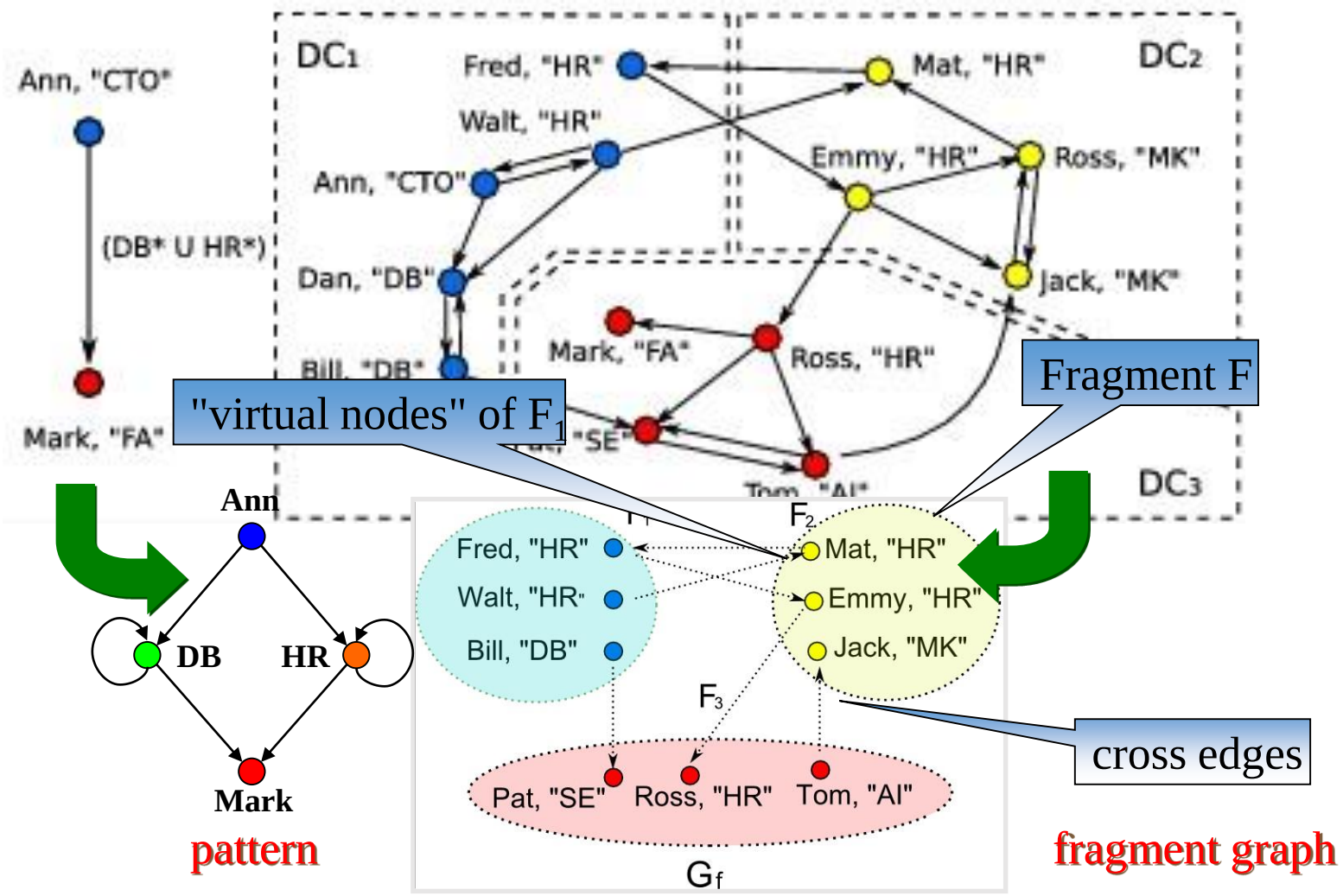
- ✓ Partial evaluation in distributed query processing
  - evaluate Q(Gi) *in parallel*
  - collect partial matches at a coordinator site, and assemble them to find the answer Q(G) in the entire G

Gj as the yet unavailable input

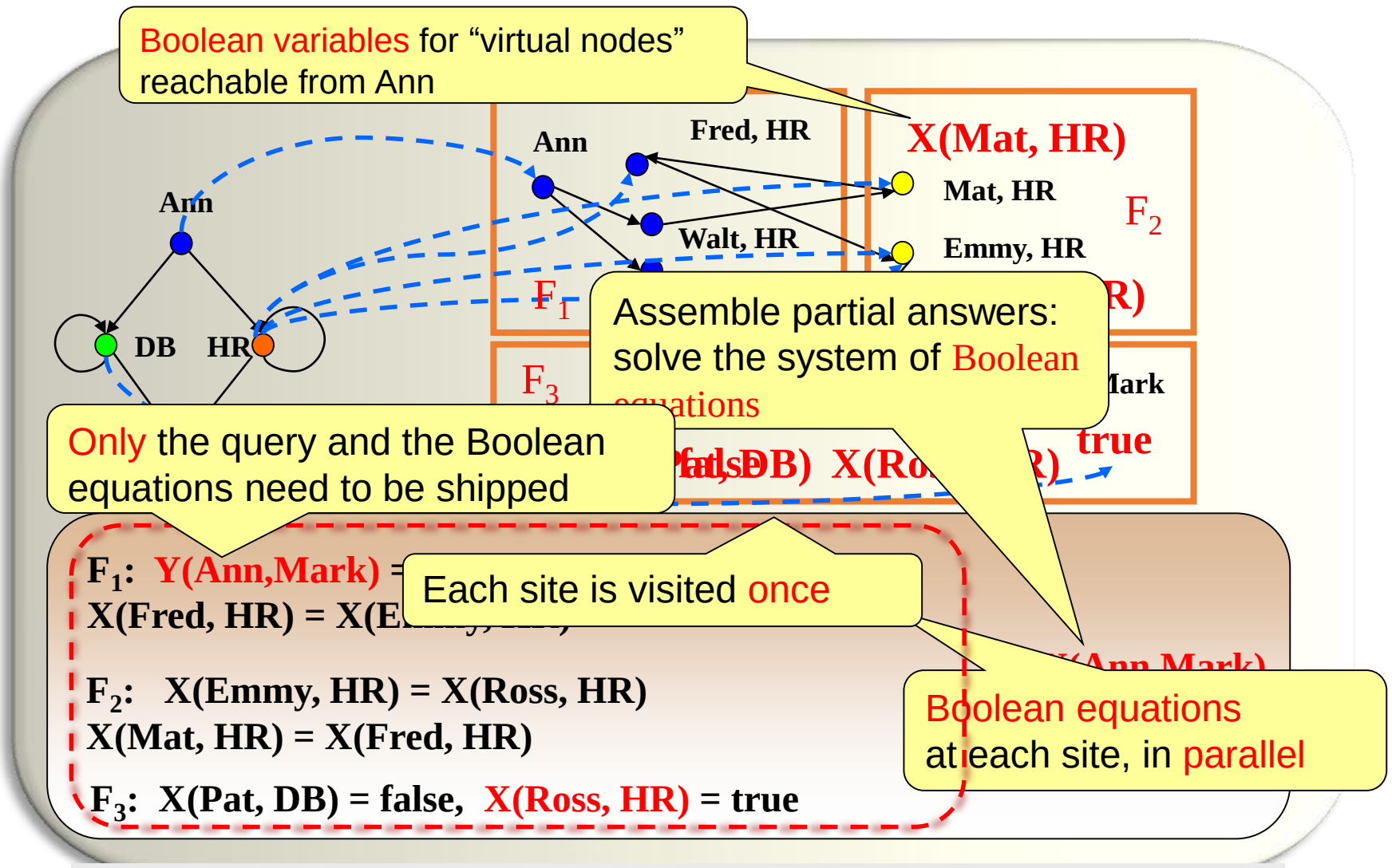
functions

Partial evaluation: a promising approach

Regular reachability queries



Regular reachability queries



The same query is partially evaluated at each site in parallel

Partial results

- Input: G = (G1, …, Gn), Q = (s, t, U)
- Output: true if there exists a path from s to t in G satisfying U

General regular expression

✓ Performance guaran

network traffic

nodes with edges across different fragments

The largest fragment

- the total amount of data shipped is in O($|Vf|^2 |U|^2$)
- takes O($|Gm| |Vf|^2 + |Vf|^2 |U|^2$) time

Network traffic and response time: Independent of |G|

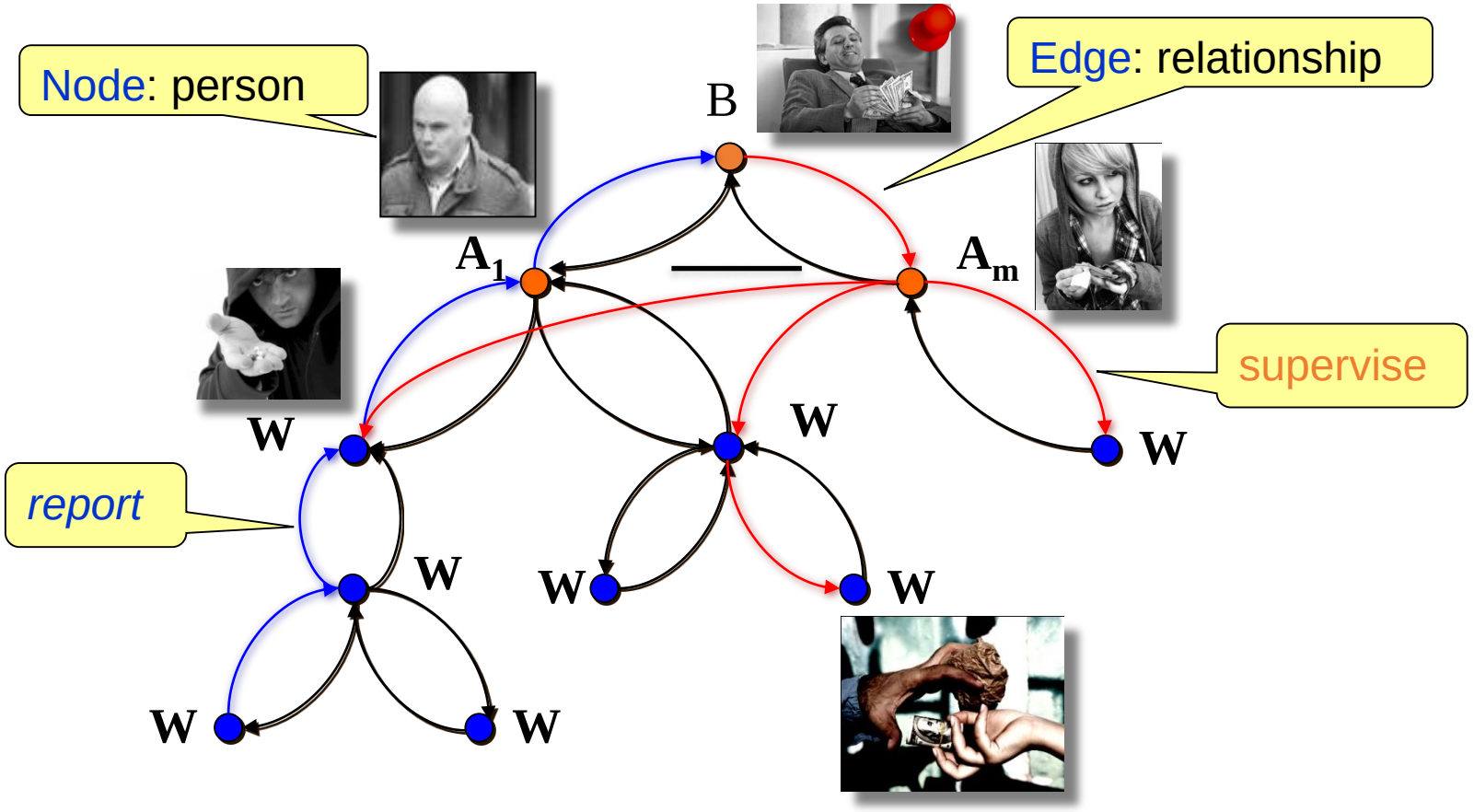
- ✓ the same results hold on any partitioning strategy
- ✓ similar results for XPath on distributed XML documents
- ✓ MapReduce

Distributed query processing with performance guarantees

Graph Search

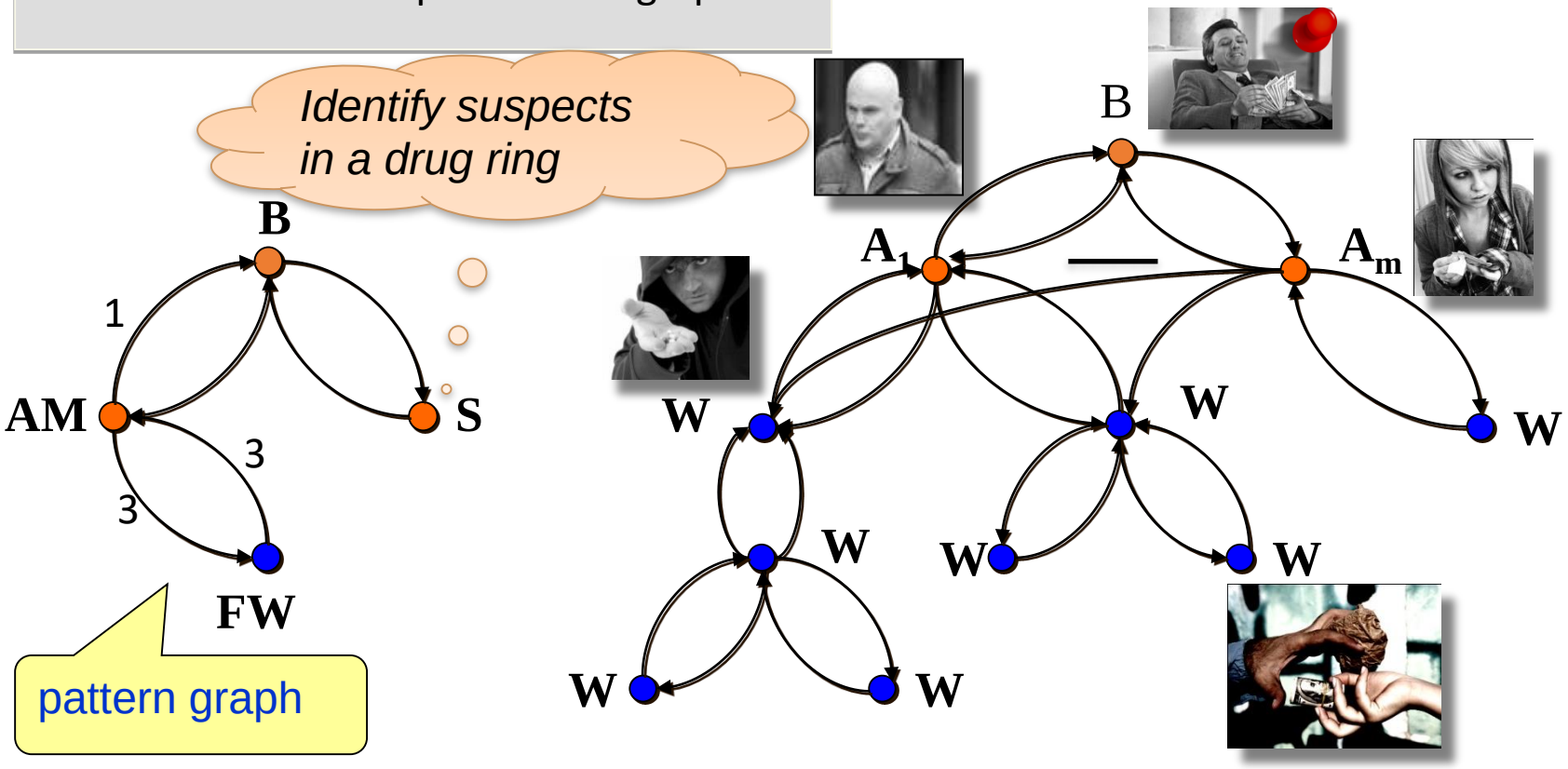
What Is Graph Search?
Reachability Queries
Subgraph Isomorphism
Large Graph Processing

Social networks modeled as graphs



Pattern matching in social graphs

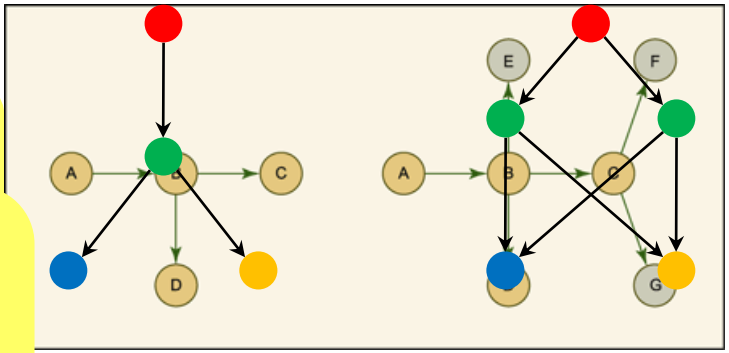
Find all *matches* of a pattern in a graph



"Understanding the structure of drug trafficking organizations"

Traditional Graph Pattern Matching

- Given a pattern graph Q and a data graph G, find all the matches of Q in G.
  - subgraph isomorphism



a bijective function *f* on nodes:

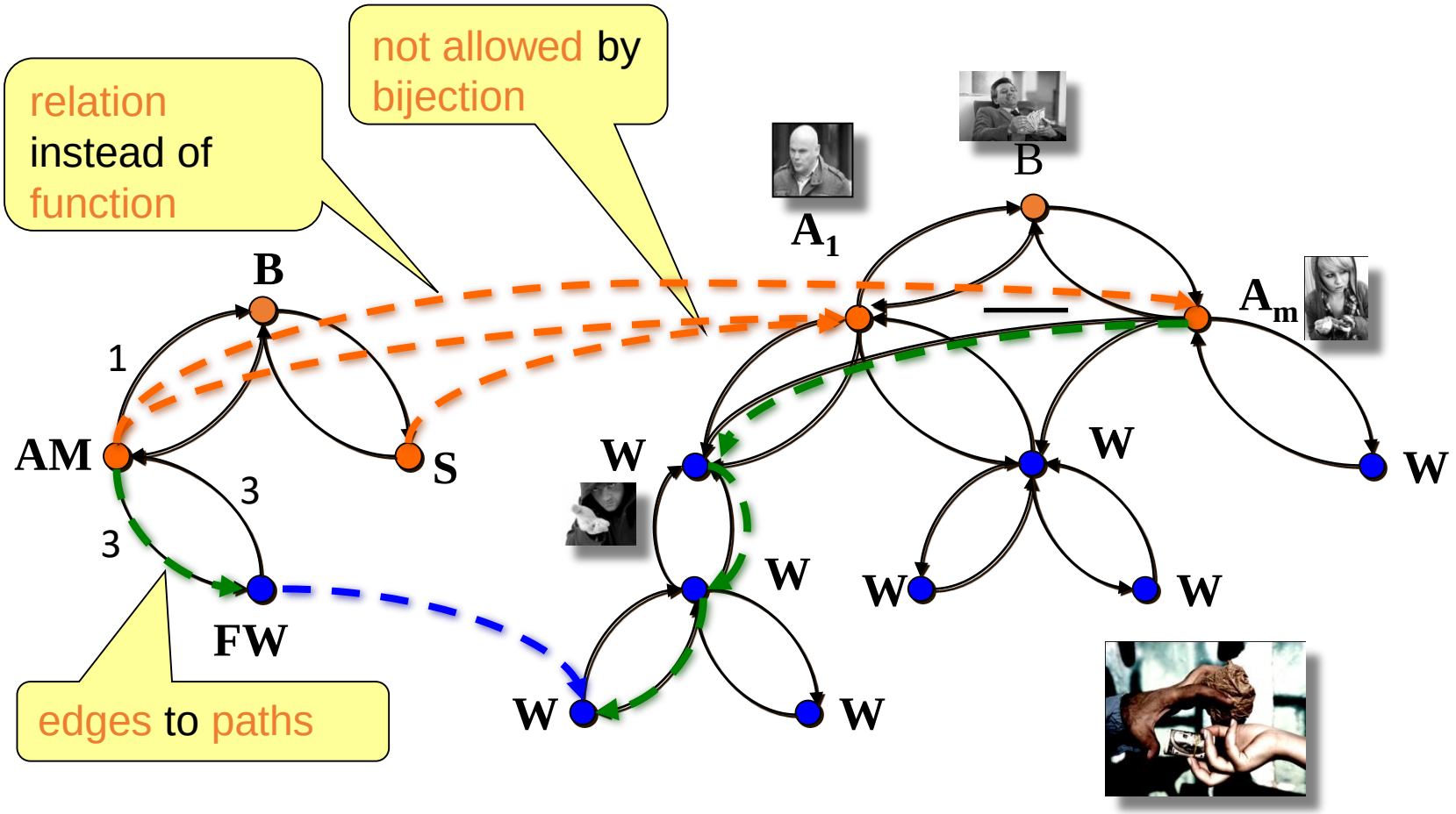
a binary relation *S* on nodes

for each $(u,v) \in S$, each edge (u,u') in Q is mapped to an edge (v, v') in G, *such that* $(u',v') \in S$

  - Web site classification,
  - social position detection …

Suffice for social network analysis?

Pattern matching in social graphs



The quest for a new form of graph pattern matching

Graph Search

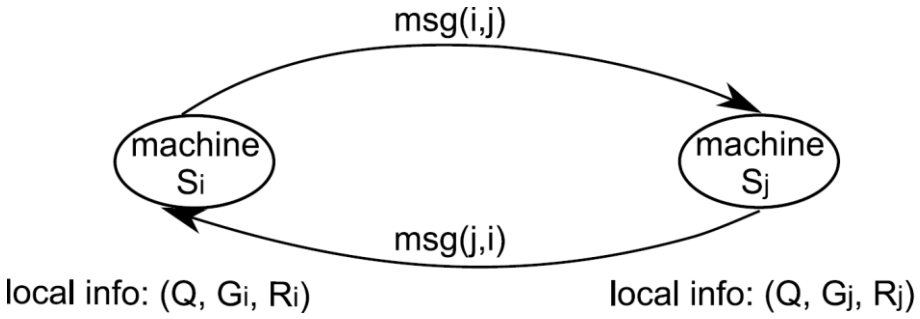
What Is Graph Search?
Reachability Queries
Subgraph Isomorphism
Large Graph Processing

Distributed Processing

- Real-life graphs are typically way too large
- Real-life graphs are naturally distributed
  - Google, Yahoo! and Facebook have large-scale data centers
- Distributed graph processing is inevitable

Distributed Processing

- Model of Computation
  - A cluster of identical machines (with one acted as coordinator);
  - Each machine can directly send arbitrary number of messages to another one;
  - All machines co-work with each other by local computations and message-passing.



- Complexity measures
  - **Visit times**: the maximum visiting times of a machine (**interactions**)
  - **Makespan**: the evaluation completion time (**efficiency**)
  - **Data shipment**: the size of the total messages shipped among distinct machines (**network band consumption**)

Incremental Techniques

- Google Percolator:
  - Converting the indexing system to an incremental system
  - Reduce the average document processing latency by a factor of 100
  - Process the same number of documents per day, while reducing the average age of documents in Google search results by 50%.
- **It is a great waste to compute everything from scratch!**

Data Preprocessing

- Data Sampling
  - Instead of dealing with the entire data graphs, it reduces the size of data graphs by sampling and allows a certain loss of precision.
  - In the sampling process, ensure that the sampling data obtained can reflect the characteristics and information of the original data graphs as much as possible.
- Data Compression
  - It generates small graphs from original data graphs that preserve the information only relevant to queries.
  - A specific compression method is applied to a specific query application, such that data graph compression is not universal for all query applications.
  - Reachability query, Neighbor query

End of Chapter 10