CentraleSupélec

ST2: Mathematical Modeling of Propagation Phenomena

Propagation on Graphs

Lecture 1

Introduction to propagation modeling and graph analysis

Fragkiskos Malliaros

Monday, December 10, 2018

About Me

- Undergrad at the University of Patras, Greece
- Ph.D. in CS at Ecole Polytechnique, Paris
- Postdoc researcher at UC San Diego
- Assistant Professor at CentraleSupélec (since October 2017)

Research interests: Data science, ML, graph mining, text mining and NLP

Office Hours

Instructor: Fragkiskos Malliaros

Office: CVN Lab, Room SC.217

Office hours: I will be available right after the lecture

Or, send me an email and we will find a good time to meet

Email: fragkiskos.me@gmail.com

TA: Abdulkadir Çelikkanat (Ph.D. student)

Office: CentraleSupélec, Gif Sur Yvette campus, CVN Lab

Email: abdcelikkanat@gmail.com

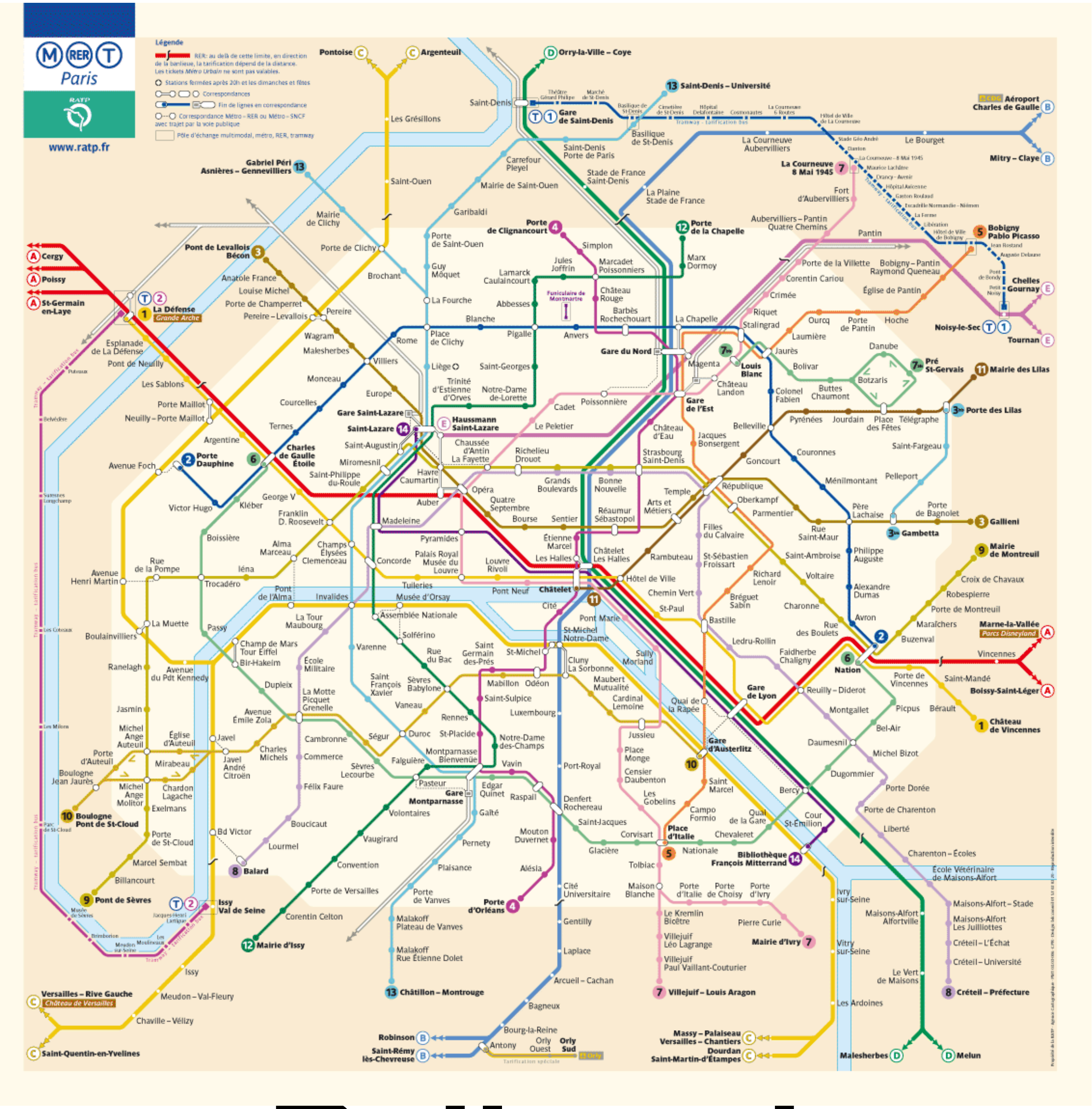
Acknowledgements

- Part of the lecture is based on material by
  - Jure Leskovec, Stanford University
  - Manos Papagelis, York University
  - B. Aditya Prakash, Virginia Tech

  Thank you!

What do the following things have in common?

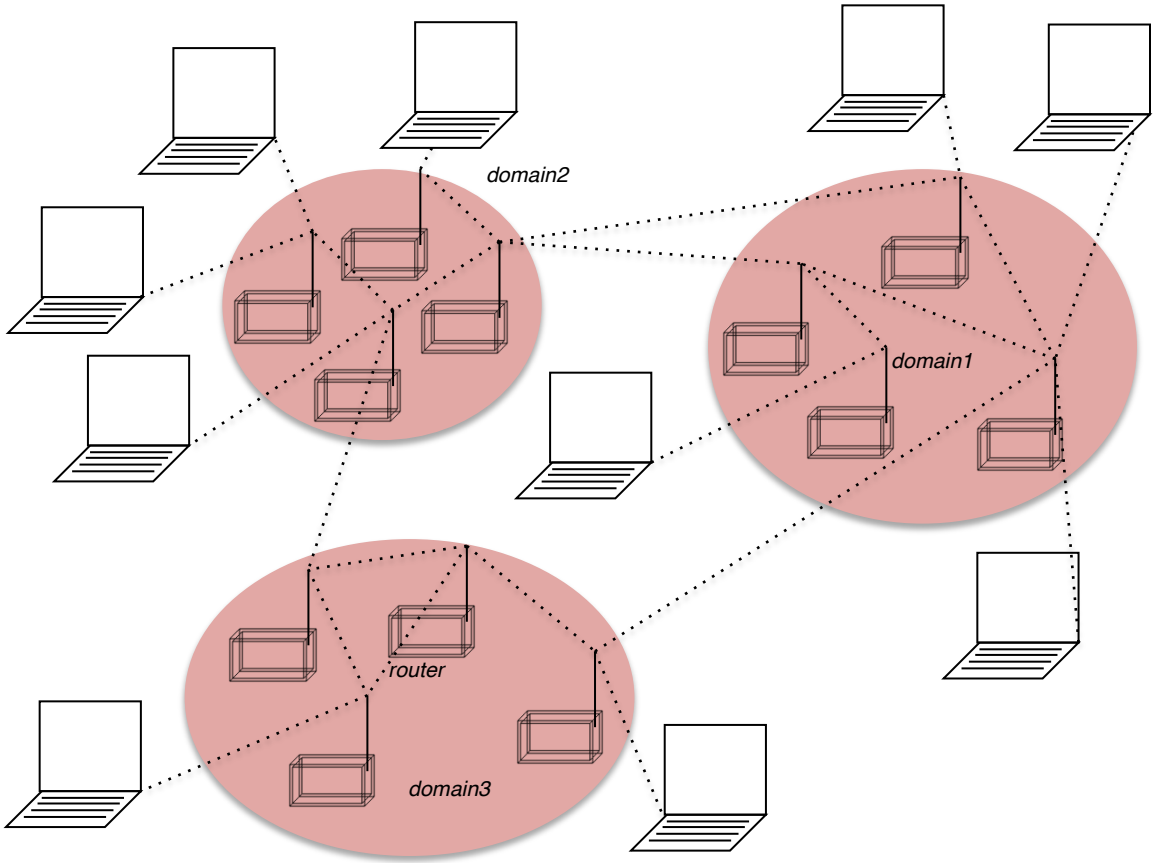
World economy



Paris
www.ratp.fr
Pontoise
Argenteuil
Orry-la-Ville – Coye
Saint-Denis – Université
Aéroport Charles de Gaulle
Mitry – Claye
Cergy
Poissy
St-Germain en-Laye
La Défense
Gare du Nord
Gare de l'Est
Gare Saint-Lazare
Châtelet
Gare de Lyon
Gare d'Austerlitz
Gare Montparnasse
Nation
Marne-la-Vallée
Boissy-Saint-Léger
Château de Vincennes
Pont de Sèvres
Mairie d'Issy
Châtillon – Montrouge
Villejuif – Louis Aragon
Créteil – Préfecture
Malesherbes
Melun
Versailles – Rive Gauche
Saint-Quentin-en-Yvelines

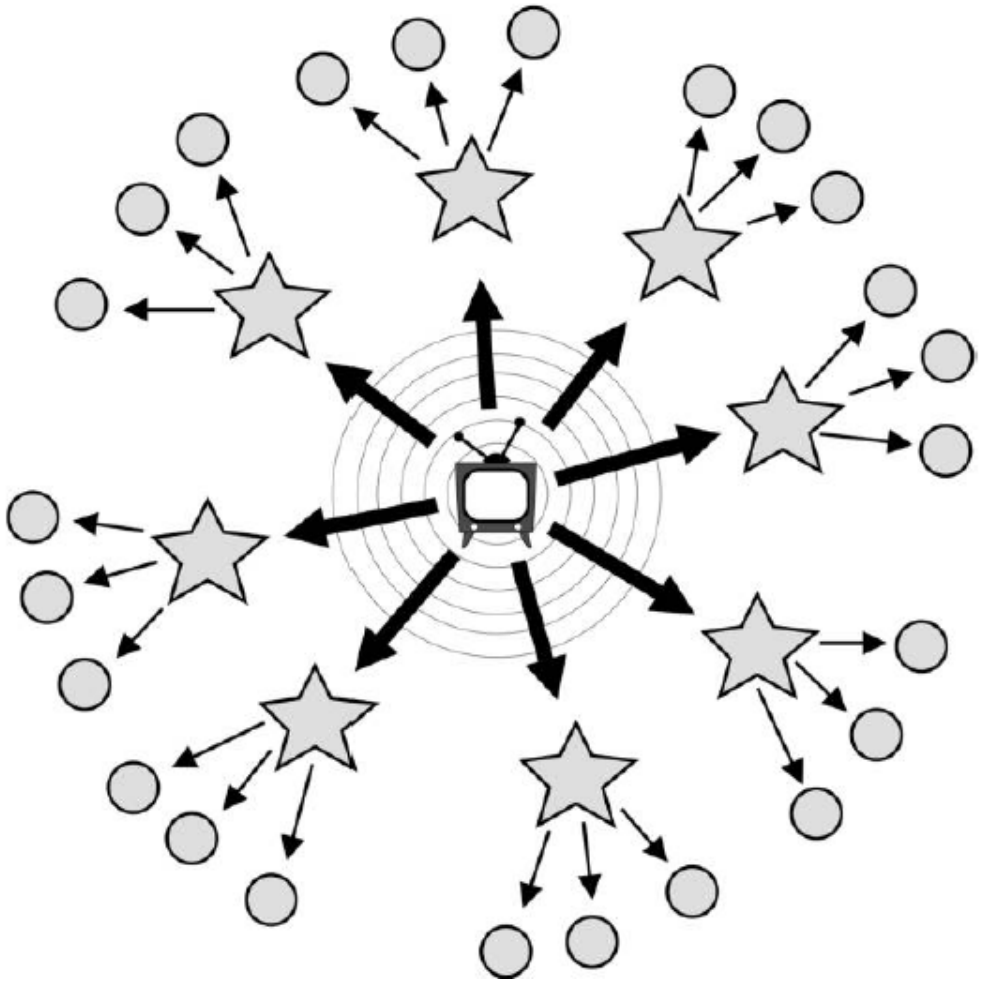
Railroads

Brain

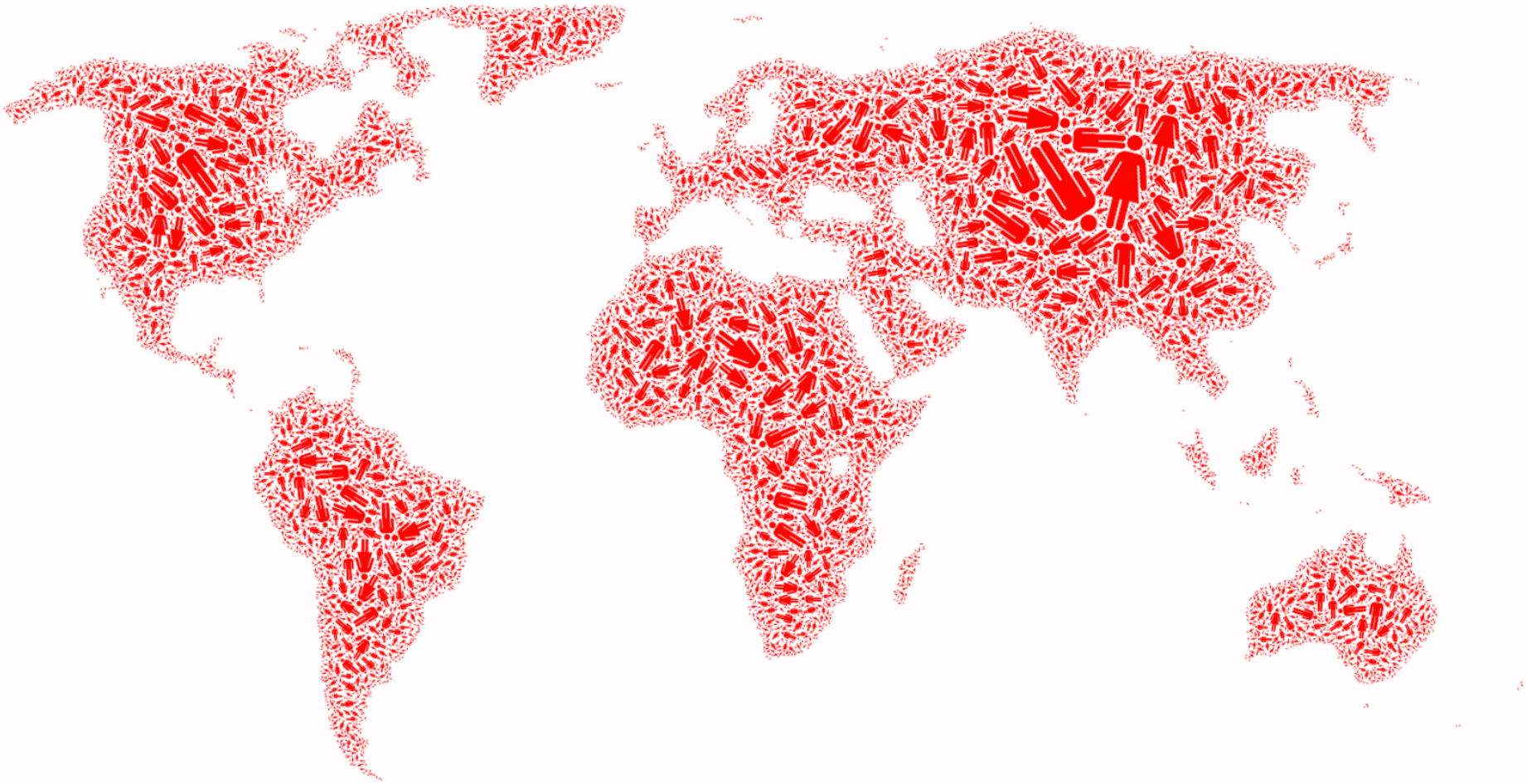


Internet

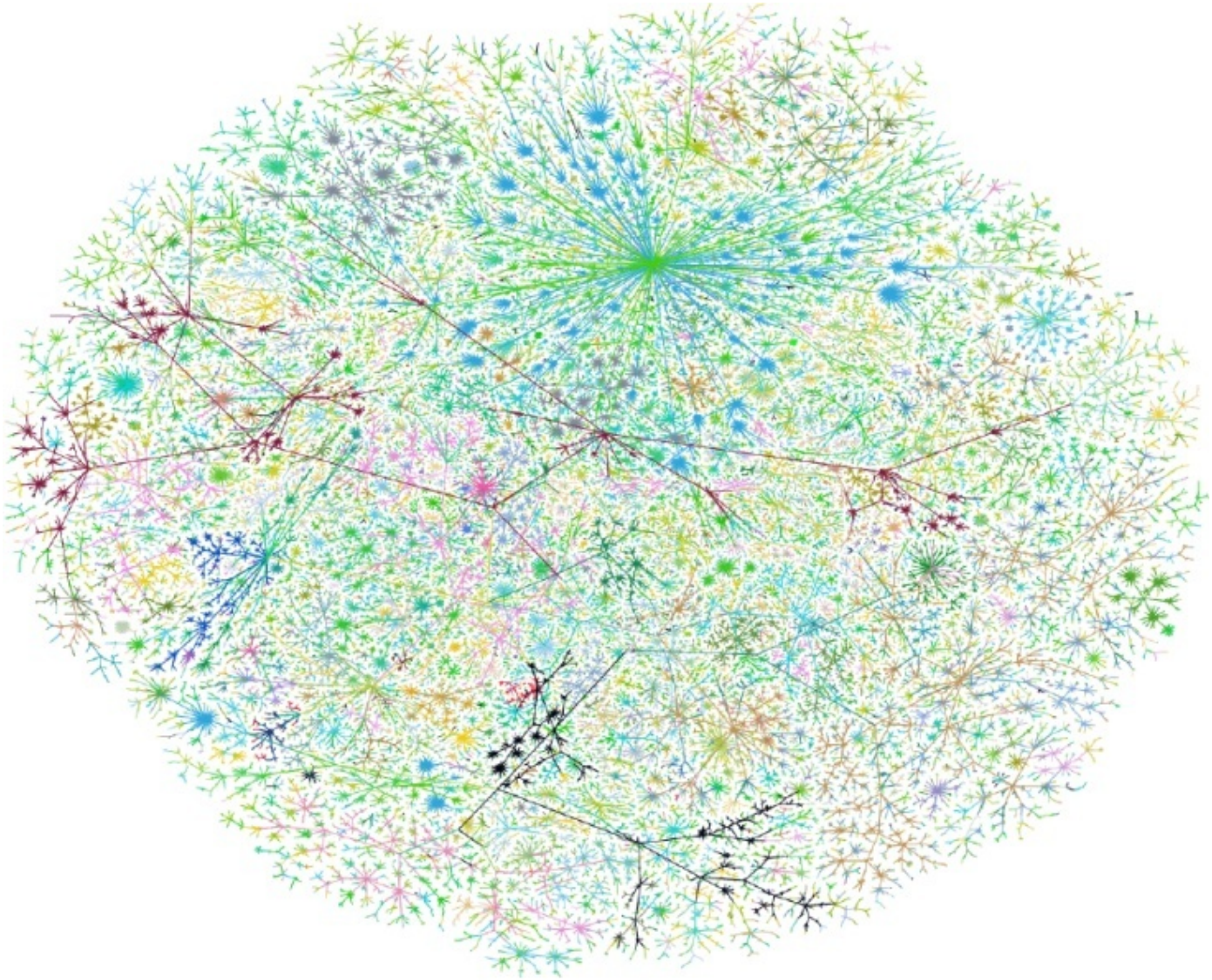
Friends & Family



Media & Information

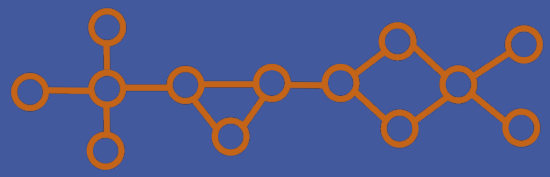


Society

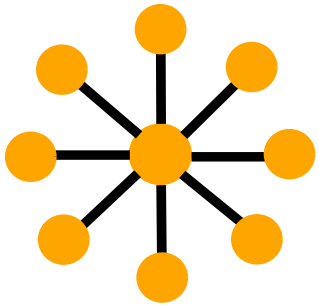


The Network!

Networks



Networks allow to model relationships between entities

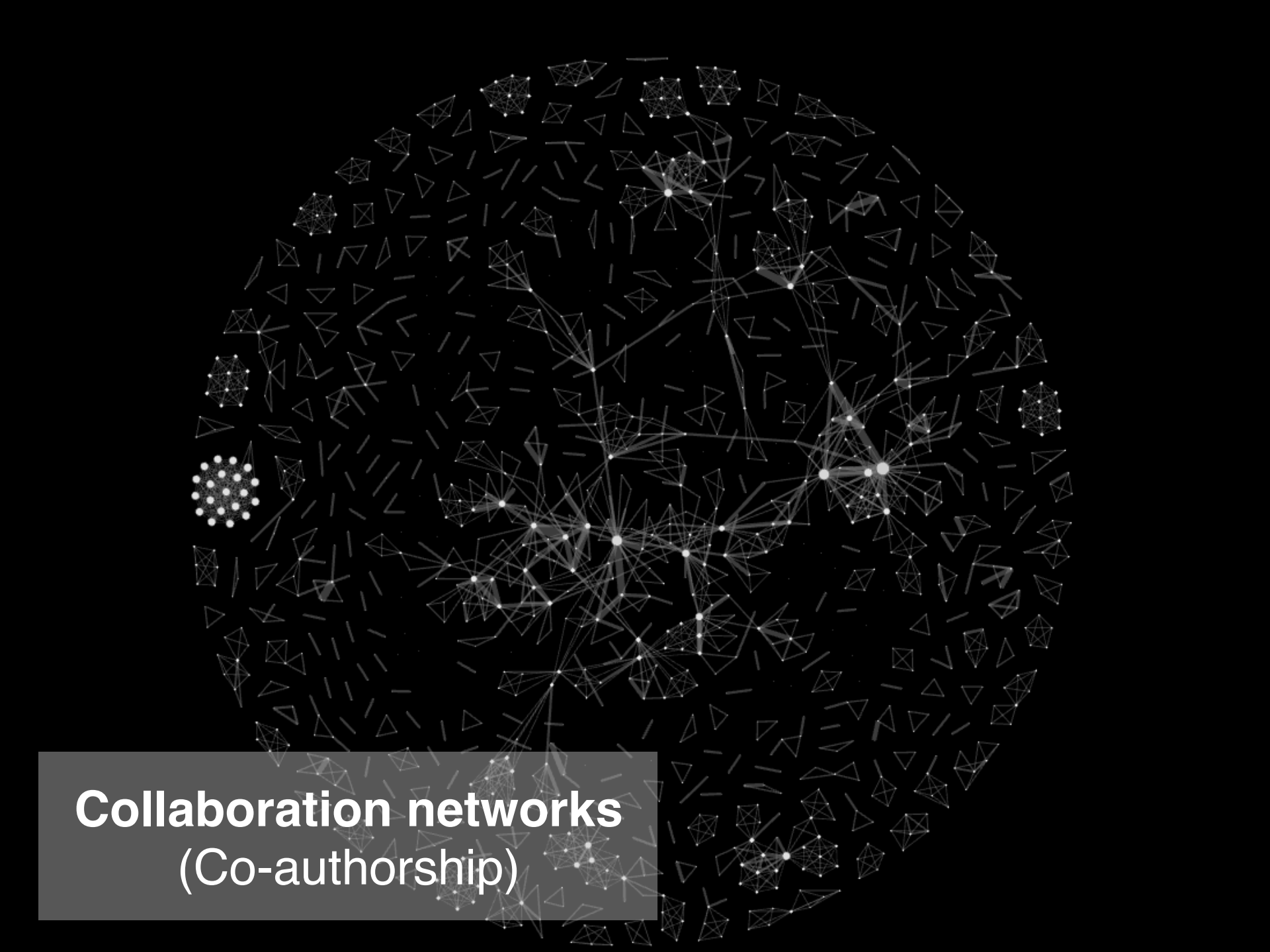


General-purpose language

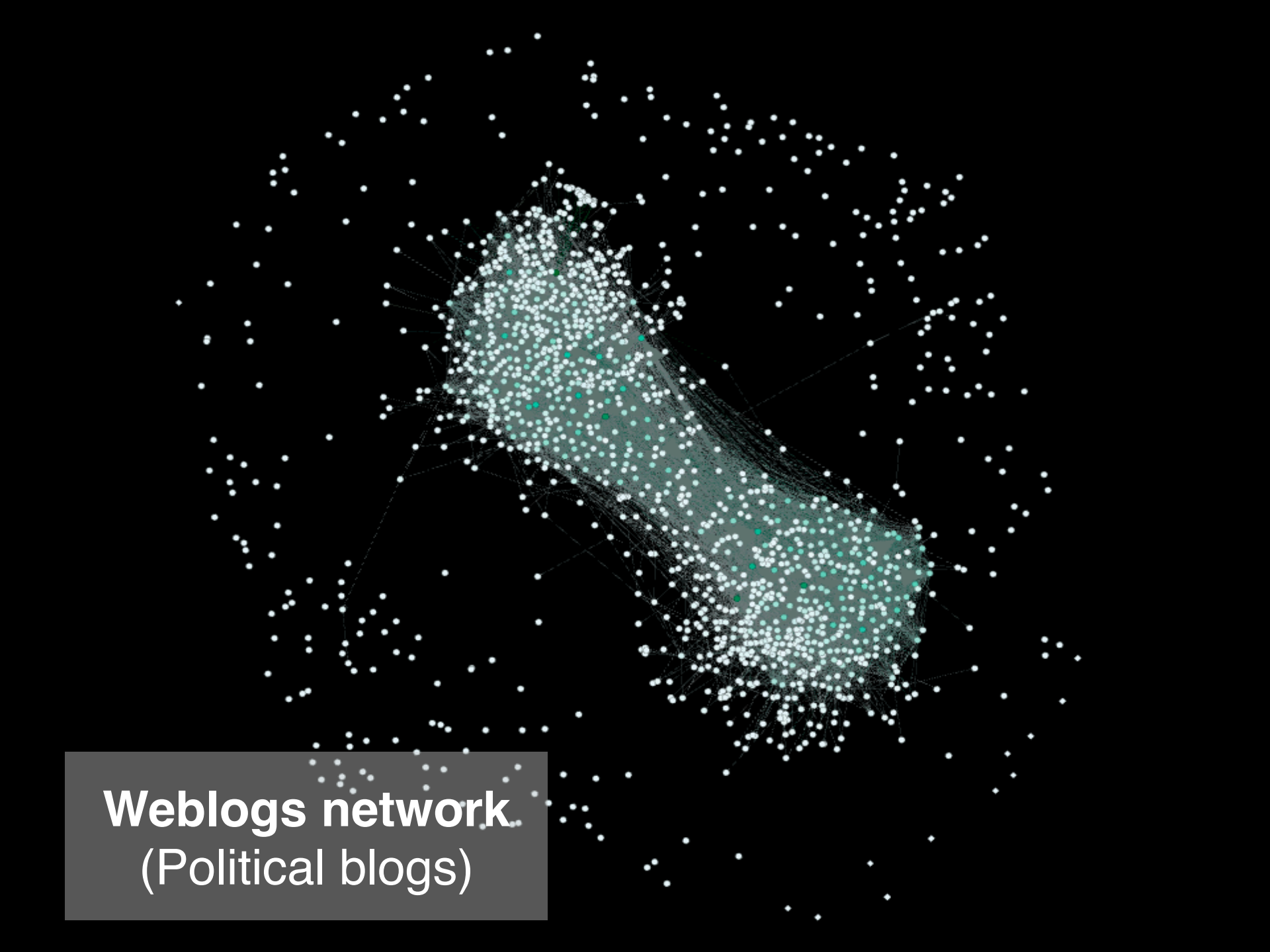
for describing real-world systems

Facebook
2.07 Billion users (Q2 of 2017)

Source: https://www.facebook.com/zuck

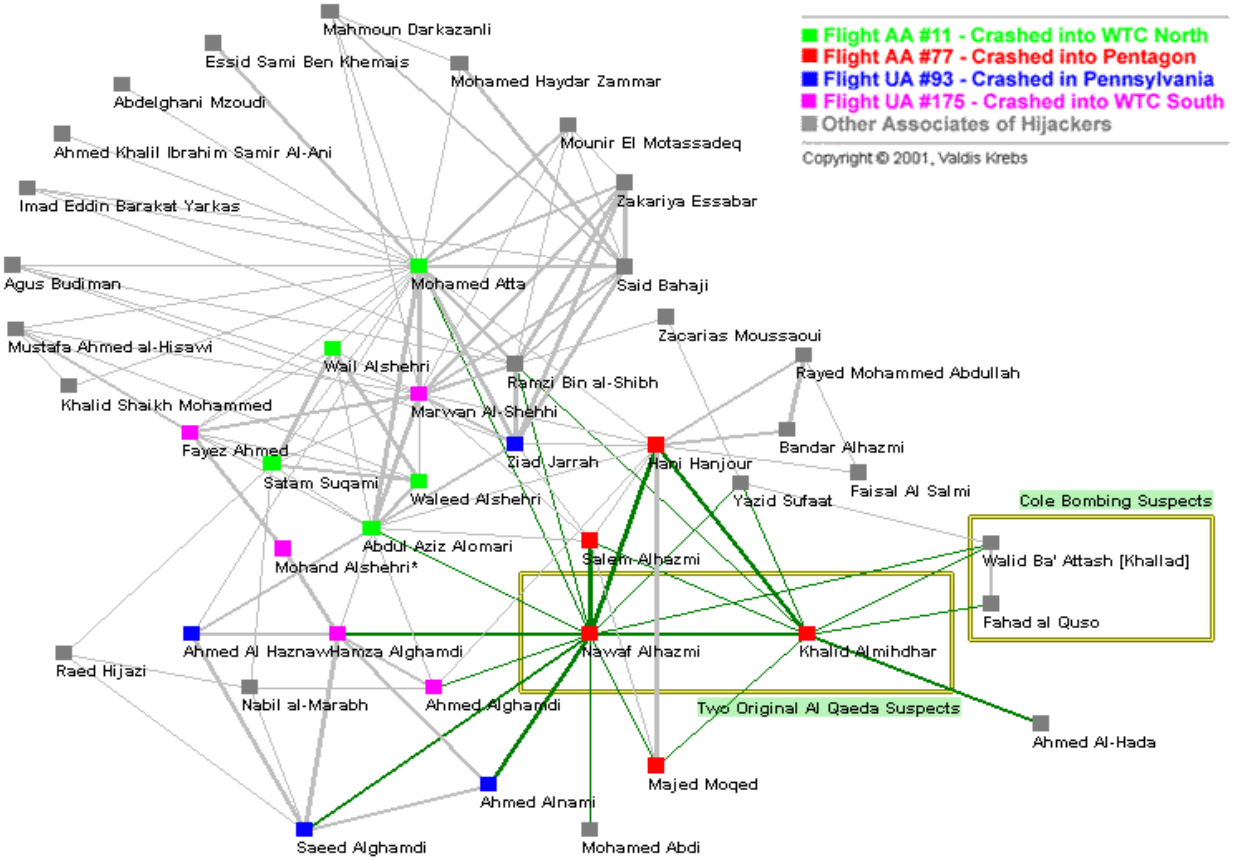


Collaboration networks
(Co-authorship)



Weblogs network
(Political blogs)

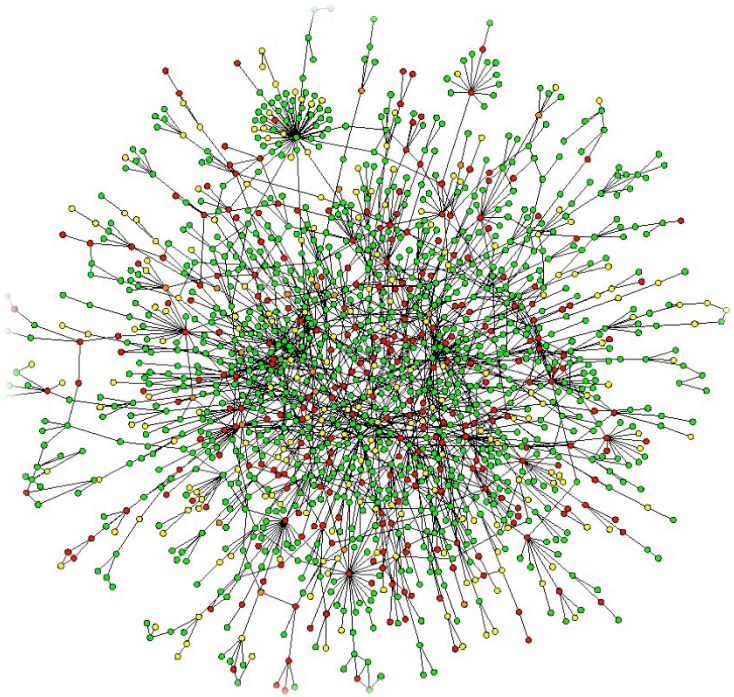
Networks of Organizations

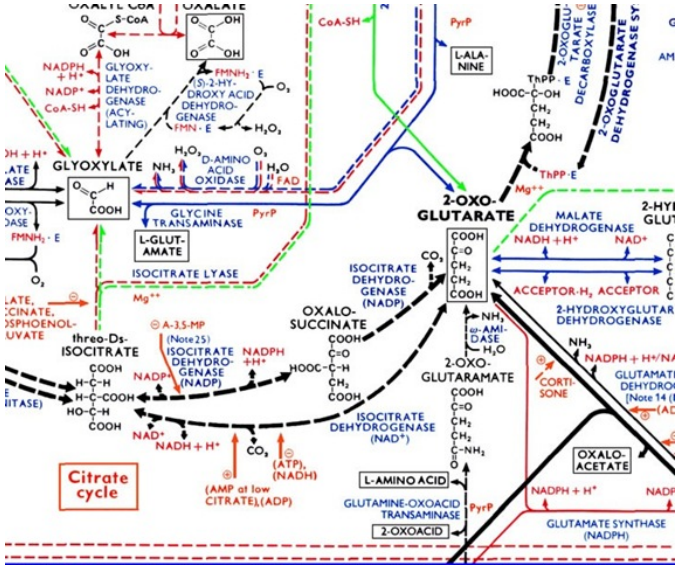


9/11 terrorist network

Source: http://www.orgnet.com/prevent.html

Biological Networks

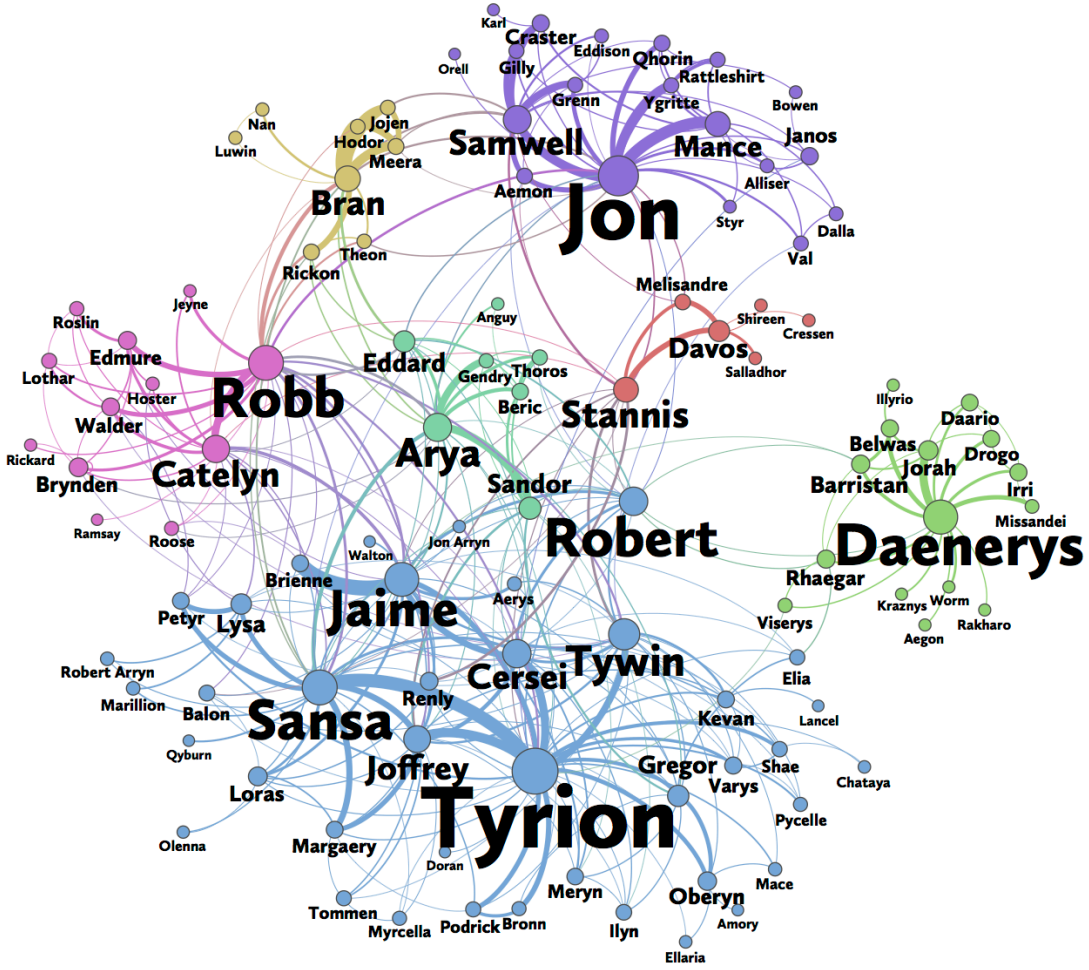




Protein-Protein Interaction Networks:
Nodes: Proteins
Edges: ‘physical’ interactions

Metabolic networks:
Nodes: Metabolites and enzymes
Edges: Chemical reactions

What Else?



In fact, anything that captures relationships between entities can be modeled as graph (any 3-way join in the DB community)

Why should I care about networks?

Why Graphs? Why Now?

- **Universal language for describing complex data**
  - Networks from science, nature, and technology are more similar than one would expect
- **Shared vocabulary (representation) between fields**
  - Computer Science, Engineering, Social Sciences, Physics, Economics, Statistics, Biology, …
  - Cross-disciplinary topic
- **Availability of big and rich data**
  - Web/mobile, bio, health, and medical
  - Computational challenges
- **Impact**
  - Social networking and social media, recommender systems, drug design, neuroscience, epidemiology, …

What Can we do With Graphs?

- **Communication networks**
  - Intrusion detection, fraud detection
  - Churn prediction (e.g., telecommunication providers)

- **Social networks**
  - Link prediction, friend recommendation
    - E.g., Facebook, LinkedIn
  - Social circle detection, community detection
  - Social recommendations
  - Identifying influential nodes, information spreading, epidemics
  - Viral marketing

In this course

- **Information networks**
  - Navigational aids

About this course

Propagation Phenomena over Networks

Dynamical processes over networks are also everywhere!

Why do we Care?

- Social collaboration
- Information Diffusion
- Viral Marketing
- Epidemiology and Public Health
- Cyber Security
- Human mobility
- Games and Virtual Worlds
- Ecology
- …

twitter

amazon

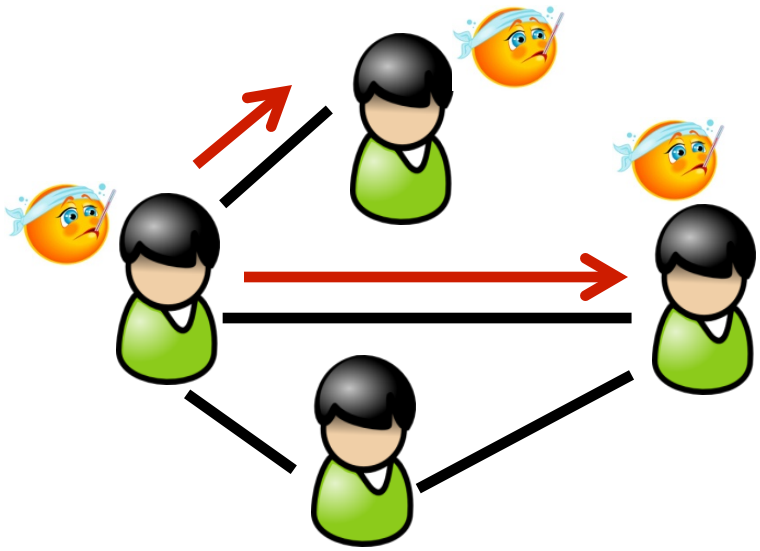
symantec.

Sprint

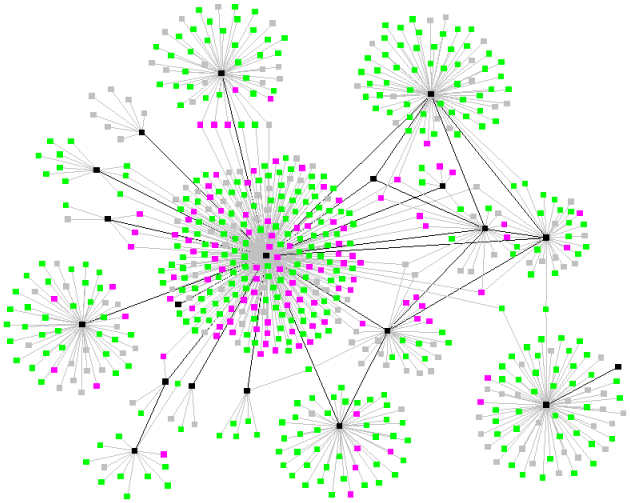
SECOND LIFE

Why do we Care? (1: Epidemiology)

- Propagation phenomena over graphs



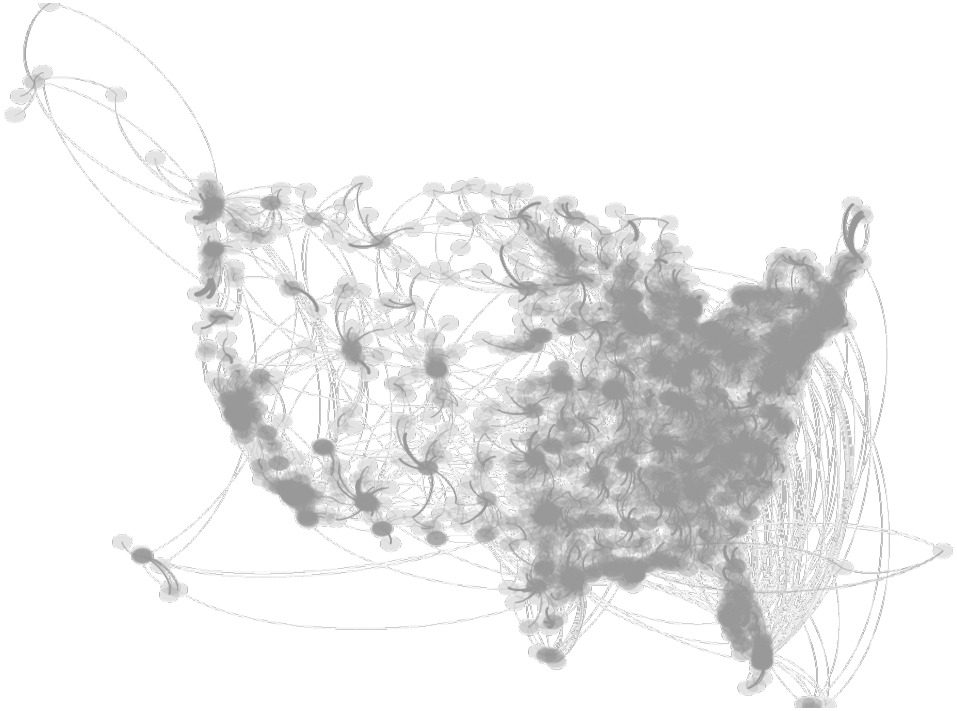
Diseases over contact networks (e.g., Ebola, H1N1)



CDC data: Visualization of the first 35 tuberculosis (TB) patients and their 1039 contacts

Why do we Care? (1: Epidemiology)

- Propagation phenomena over graphs



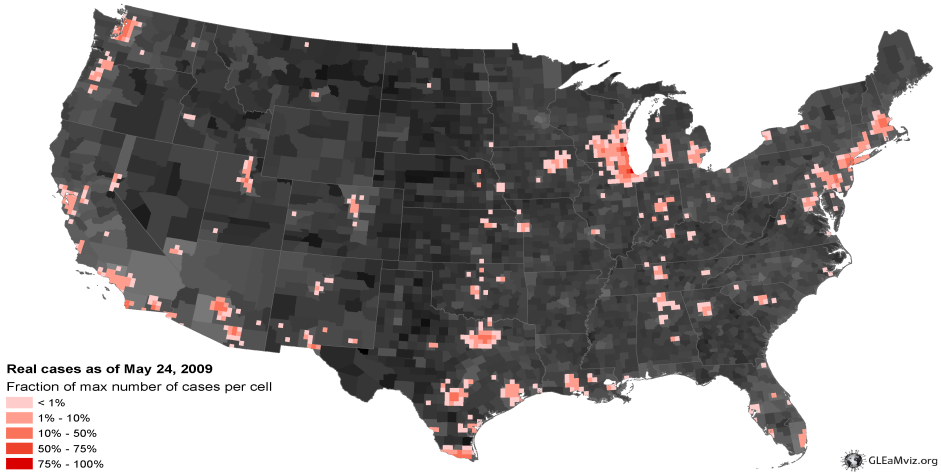
- Each circle is a hospital
- ~3000 hospitals
- More than 30,000 patients transferred

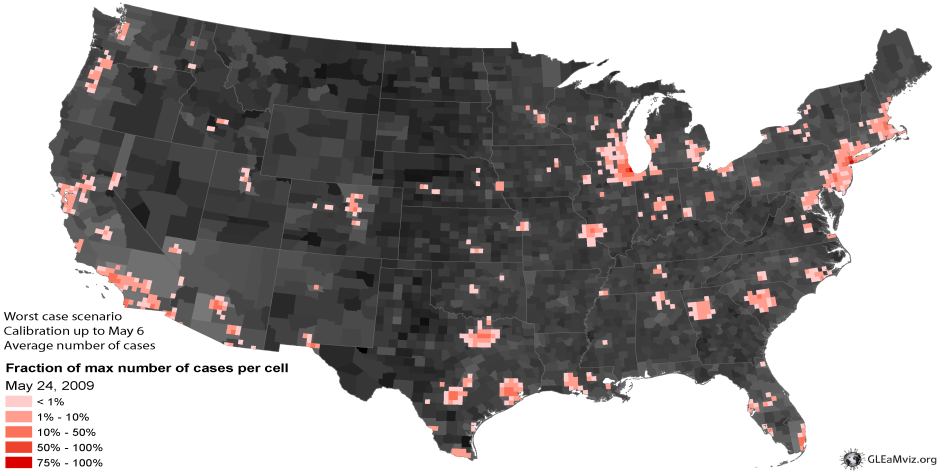
[US-MEDICARE NETWORK 2005]

Problem: Given k units of disinfectant, whom to immunize?

Why do we Care? (1: Epidemiology)

Predicting epidemics (e.g., the 2009 H1N1 pandemic)





Real

Predicted

Why do we Care? (2: Online Diffusion)

licensed under CC Attribution-NonCommercial-ShareAlike 2.0 Germany | Ludwig Gatzke | http://flickr.com/photos/stabilo-boss/

> 2B users, ~$13B revenue

facebook

~350M active users

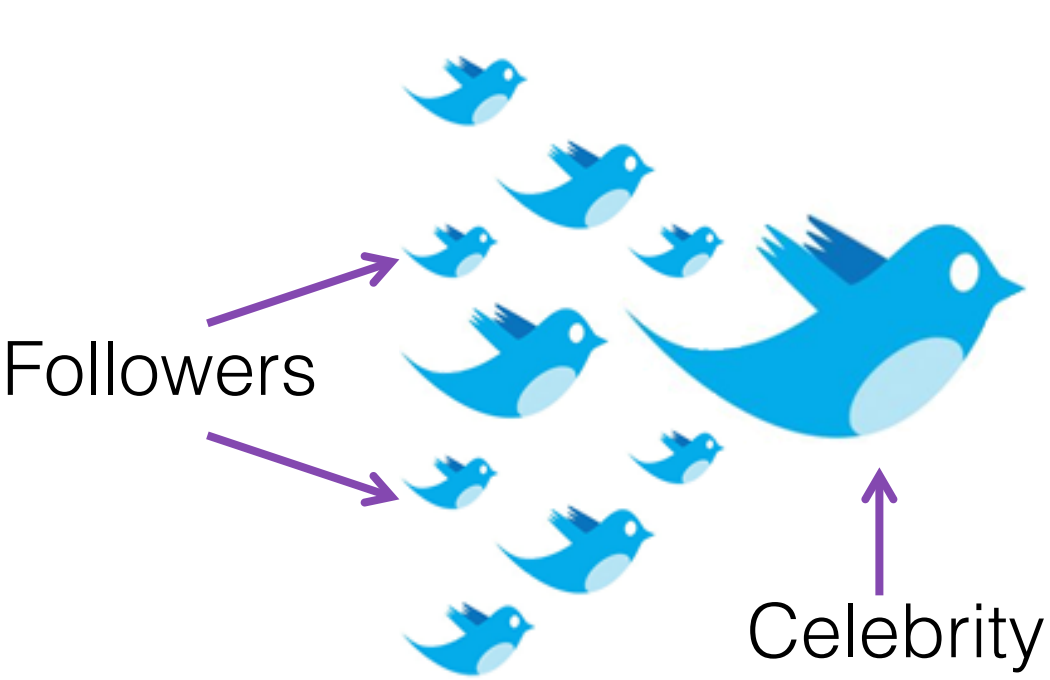
twitter

> 450M users

LinkedIn

30

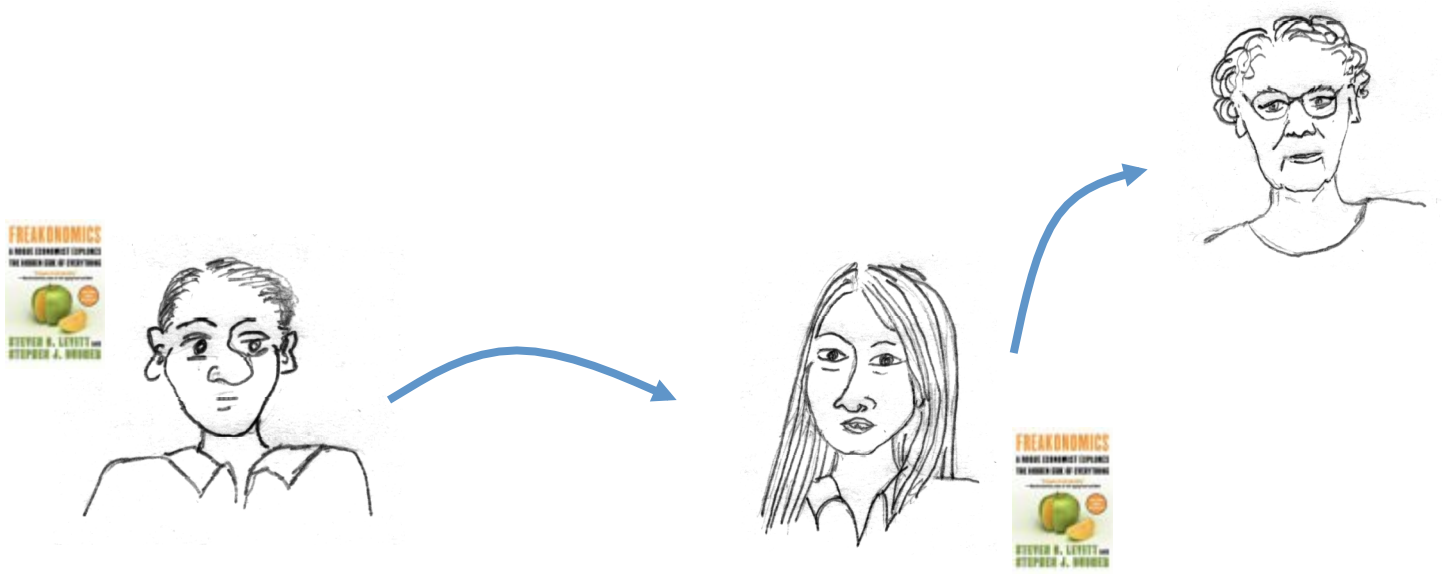
Why do we Care? (2: Online Diffusion)



Social Media Marketing

Diffusion in Viral Marketing

- Product adoption
  - Senders and followers of recommendations



Why do we Care? (3: Diffusion of Ideas)

- Propagation phenomena over graphs

Social networks and Collaborative Action

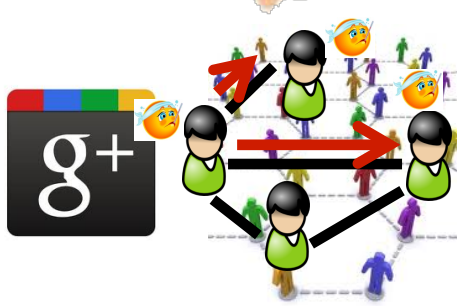
High Impact – Multiple Settings

Q: How to squash rumors faster?

epidemic out-breaks

Q: How do opinions spread?

products/viruses



Q: How to market better?

transmit s/w patches

Research Theme

ANALYSIS

Understanding

DATA

Large real-world networks & processes

POLICY/ ACTION

Managing

Research Theme – Public Health

ANALYSIS

Will an epidemic happen?

DATA

Modeling patient mobility

POLICY/ ACTION

How to control out-breaks?

Research Theme – Social Media

ANALYSIS

cascades in future?

DATA

Modeling Tweets spreading

POLICY/ ACTION

How to do better marketing ?

Learning Objectives

- Introduce students to the field of graph analysis for propagation modeling
  - Cover a wide range of topics, methodologies and applications related to information spreading over networks
  - Hands-on experience on dealing with graph data analysis for propagation phenomena

- By the end of the course, we expect
  - To have a thorough understanding of various data analysis tasks related to graphs and information spreading
  - Formulate and solve problems that involve propagation phenomena over graphs

Course Logistics

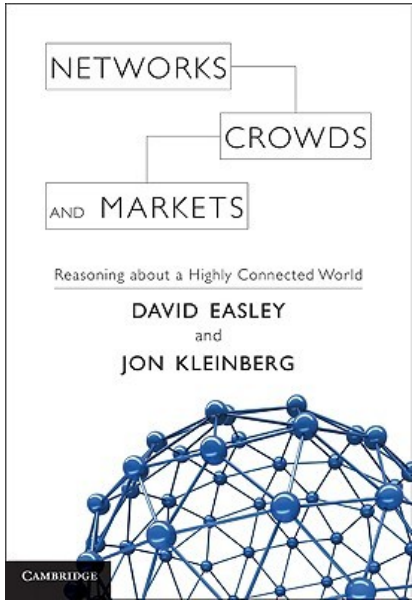
- Website
  - http://fragkiskos.me/teaching/ST2-W19/
  - Information about the course, schedule, reading material
  - Resources (helpful for the labs and the ‘integration week’)

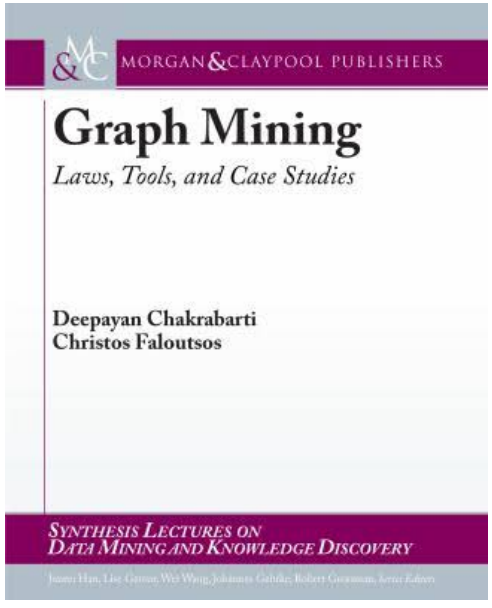
- Piazza for Q&A
  - https://piazza.com/centralesupelec/winter2019/st2/home
  - Please, participate in discussions
  - All announcements will be posted there
  - Also, lecture slides and material of the labs
  - Use key to enroll: st2

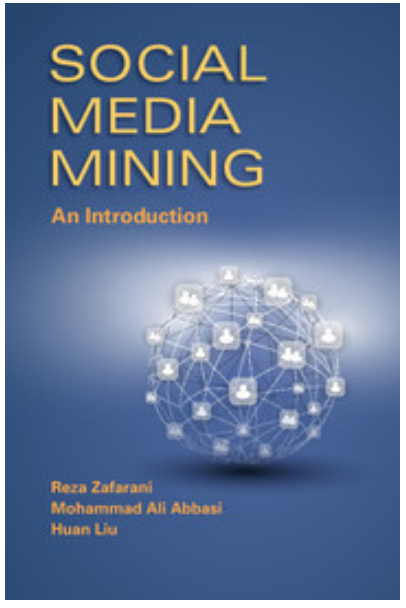
Prerequisites

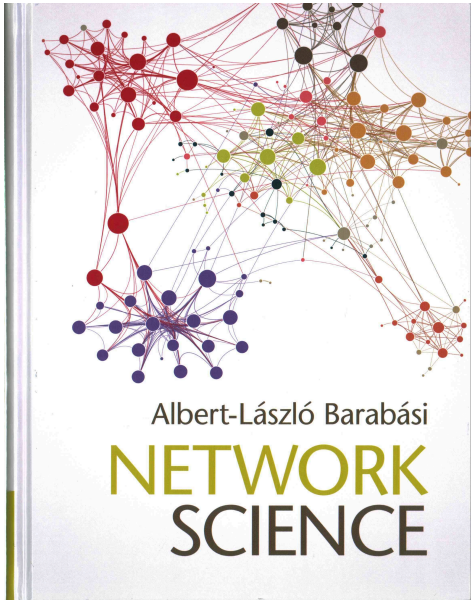
- Basic knowledge of
  - Graph theory and algorithms
  - Linear algebra
  - Probabilities and statistics
- We will review most of the main concepts in today's lecture
- Programming is necessary
  - Python (or any other language of your preference)

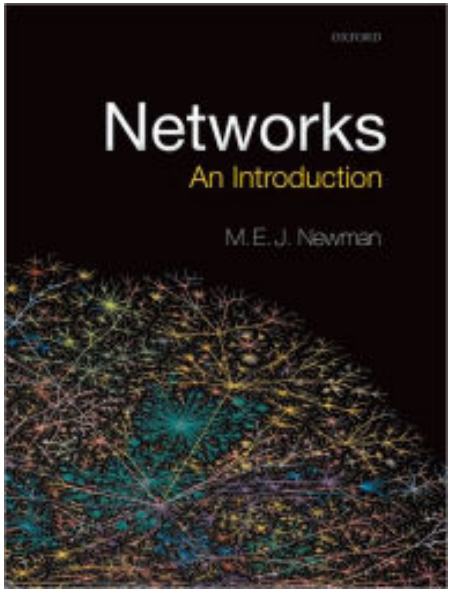
Reading Material











- The books are publicly available in electronic form (except the "Networks: An Introduction)
  - Pointers to chapters for every lecture (see the website)
- Research articles for some topics

Software Tools

- We strongly advise to use Python
  - NetworkX library
  - igraph library (also for C++ and R)
- Snap library
  - C++ and Python
- Gephi, Jung and graph-tool for network visualization
- See the Resources section

Some Personal Notes ☺

- Come to the class ☺
  - Please ask questions, participate in discussions on piazza
- Check out the additional suggested material on the website
  - Some topics are research-oriented
  - Search the web, google is your friend!
  - For every topic covered in the class, you can find the original research articles – take a look on them
  - Typically, the suggested reading material is overlapping – read selectively
- Play with software tools
  - This is the actual goal of the assignments
- Give us your feedback!

Schedule of the Course

http://fragkiskos.me/teaching/ST2-W19/

Schedule and Lectures

The slides for each lecture will be posted in piazza just before the start of the class.

	Date	Topic	Material
1	December 10	○ Introduction to propagation modeling and graph analysis	Lecture 1
2	December 12	○ Centrality criteria and link analysis algorithms	Lecture 2
3	December 17	○ Models of information propagation on graphs	Lecture 3
4	December 19	○ Identification of influential spreaders on graphs	Lecture 4
5	January 7	○ Influence maximization in social networks	Lecture 5

Networks or Graphs?

- Network often refers to real systems
  - Web, Social network, Internet Metabolic network

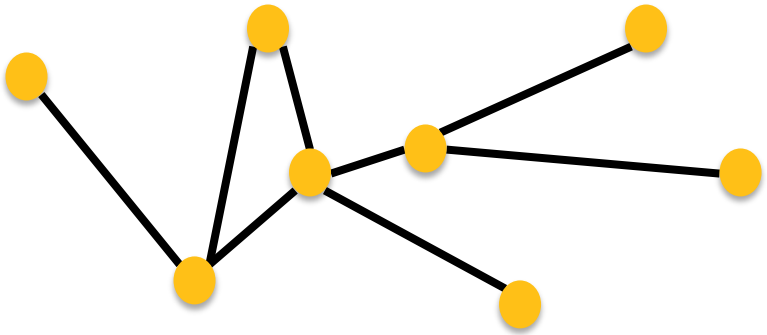
  Language: network, node, link

- Graph is mathematical representation of a network (a model)
  - Web graph, Social graph (a Facebook term)

  Language: graph, vertex, edge

We will try to make this distinction whenever it is appropriate, but in most cases we will use the two terms interchangeably

Basics in Graph Theory and Linear Algebra



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Adjacency matrix

- Objects: nodes, vertices *V*
- Interactions: links, edges *E*
- System: network, graph *G = (V, E)*

Graph-theoretic algorithms

- E.g., graph concepts, types of graphs, subgraphs, traversal, shortest paths, connectivity, complexity issues

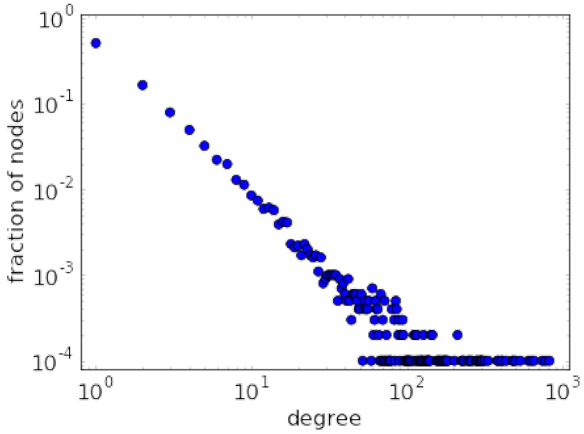
Linear algebra

- E.g., matrix-based graph representations, matrix decomposition, properties of the adjacency matrix, Laplacian matrix, spectral graph theory, graph spectrum

Network Properties and Patterns

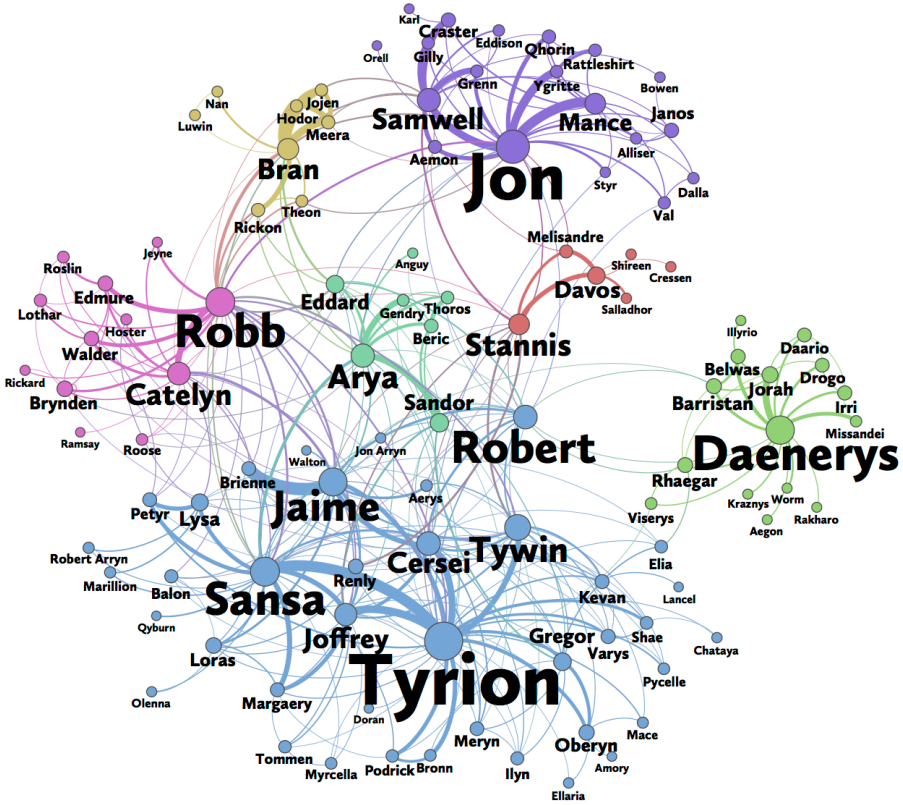
Q1: How does a real-network look like?

- Interesting measures?



Q2: Properties and patterns?

Centrality and Ranking in Networks

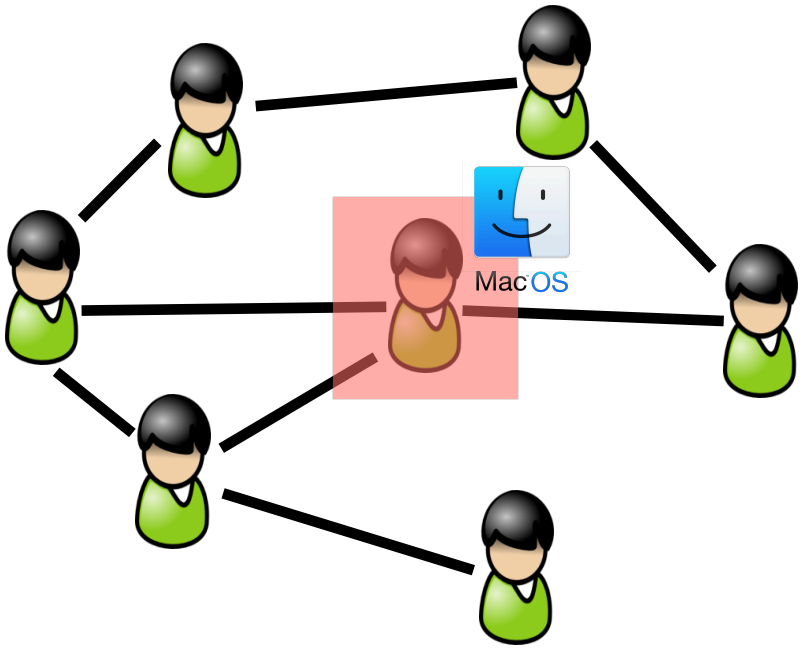


Q: How to determine the importance of a node in the graph?

- Centrality criteria (e.g., degree, closeness, betweenness)
- HITS and PageRank algorithms
- Scalability issues

Source: https://www.macalester.edu/~abeverid/thrones.html

Influential Nodes and Influence Maximization



[Viral marketing] How to organize an effective product promotion campaign?

[Opinion dynamics] How do opinions/rumors spread?

[Epidemiology] How do viruses/ diseases propagate?

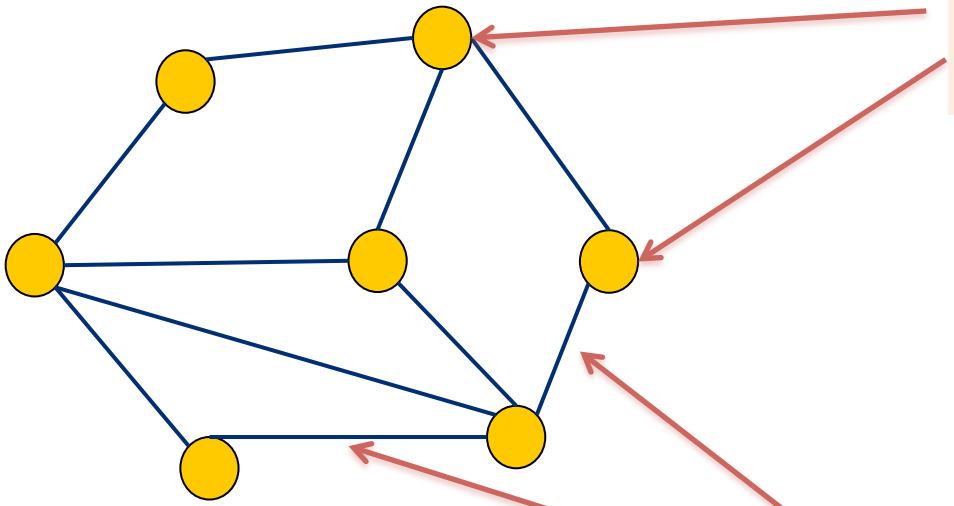
- Epidemic models on networks
- Detection of influential nodes (spreaders) in networks
- Influence maximization algorithms

[Prakash, Ramakrishnan, KDD '16]

Basic graph-theoretic concepts and definitions

Graphs and Networks

Graphs: modeling dependencies



Nodes (or vertices)
(objects/entities)

Edges (or links)
(interconnections)

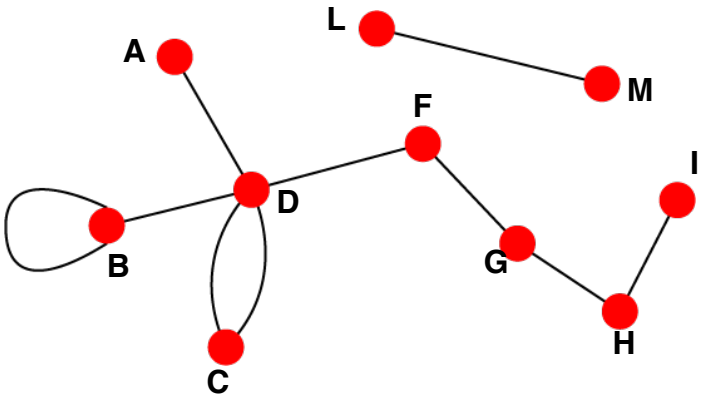
$G = (V, E)$
(network or graph)

$n = |V|$ is the number of nodes
$m = |E|$ is the number of edges

Undirected vs. Directed Networks

Undirected

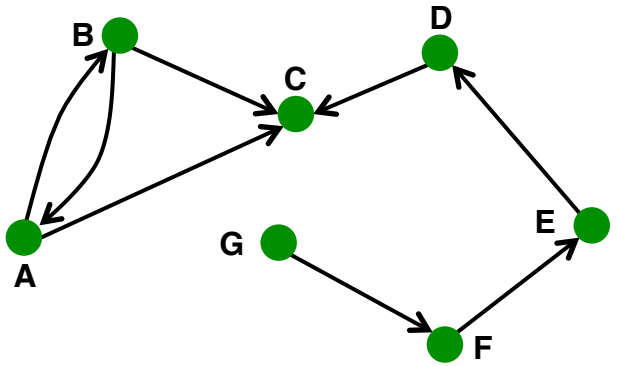
- **Links:** undirected (symmetrical, reciprocal)



- Examples
  - Collaborations
  - Friendship on Facebook

Directed

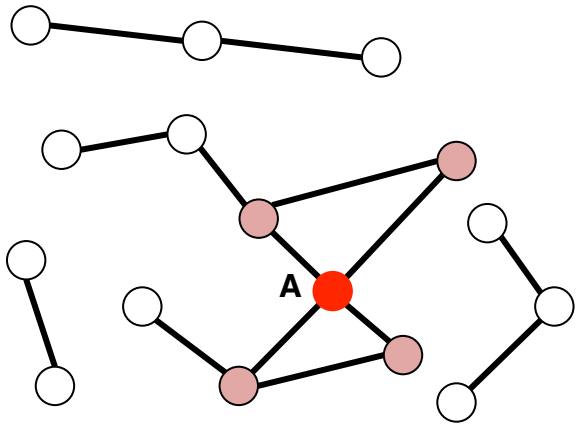
- **Links:** directed (arcs)



- Examples
  - Phone calls
  - Following on Twitter

Node Degree

Undirected



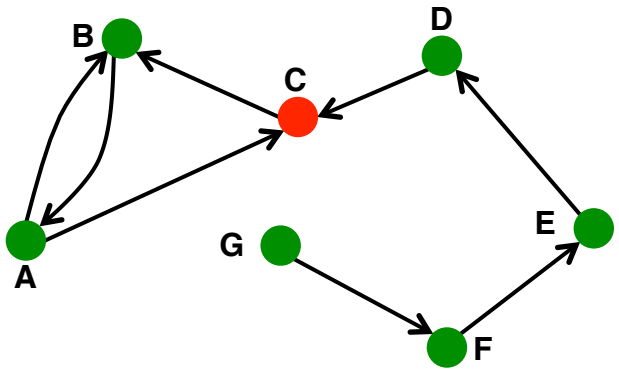
Node degree k_i: the number of edges adjacent to node i

$$k_A = 4$$

Average degree:

$$\bar{k} = \langle k \rangle = \frac{1}{n} \sum_{i=1}^{n} k_i = \frac{2|E|}{n}$$

Directed



In directed networks we define an in-degree and out-degree

The (total) degree of a node is the sum of in- and out-degrees

$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

Source: Node with $k^{in} = 0$

Sink: Node with $k^{out} = 0$

Average: $\bar{k^{in}} = \bar{k^{out}}$

Properties of the Degree

- Property 1: The sum of the degrees in an undirected graph is twice the number of edges

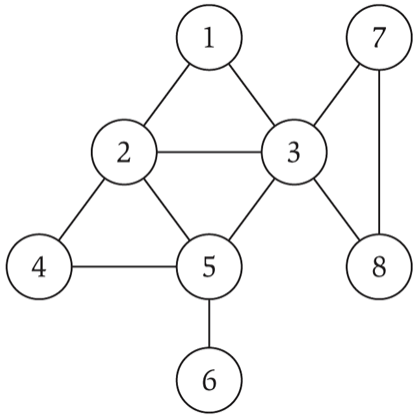
$$\sum_{i=1}^{n} k_i = 2\ |E|$$

- Property 2: In any directed graph, the sum of in-degrees is equal to the sum of out-degrees

$$\sum_{i=1}^{n} k_i^{out} = \sum_{j=1}^{n} k_j^{in}$$

Graph Representation: Adjacency Matrix

- A graph can be represented by the adjacency matrix A
  - Matrix of size $n \times n$, where $n=|V|$ is the number of nodes
  - $A_{ij} > 0$, if i and j are connected
  - $A_{ij} = 0$, if i and j are not connected
  - In case of **unweighted** graphs, $A_{ij} = 1$, if (i, j) is an edge of the graph
  - Space proportional to n^2



Undirected graph

Node indexing

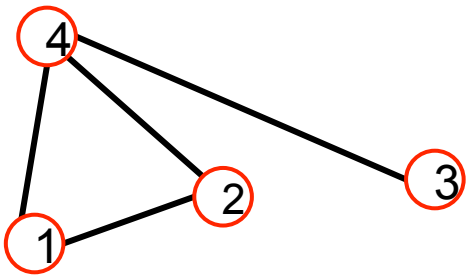
Node indexing

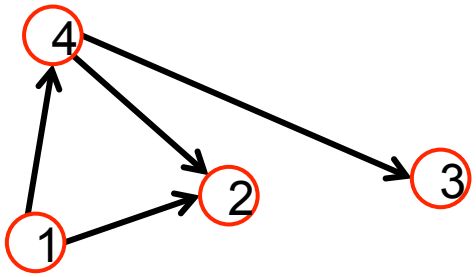
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency matrix

More matrix representations in a while

Adjacency Matrix





$A_{ij} = 1$ if there is a link from node *i* to node *j*

$A_{ij} = 0$ otherwise

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Undirected graph

Symmetric matrix

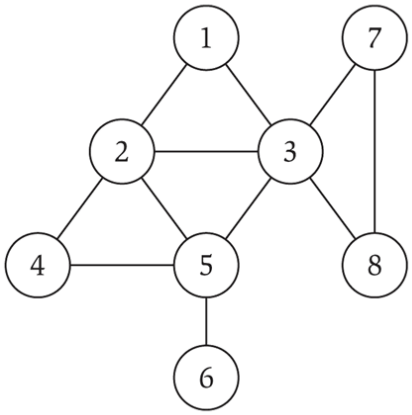
$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

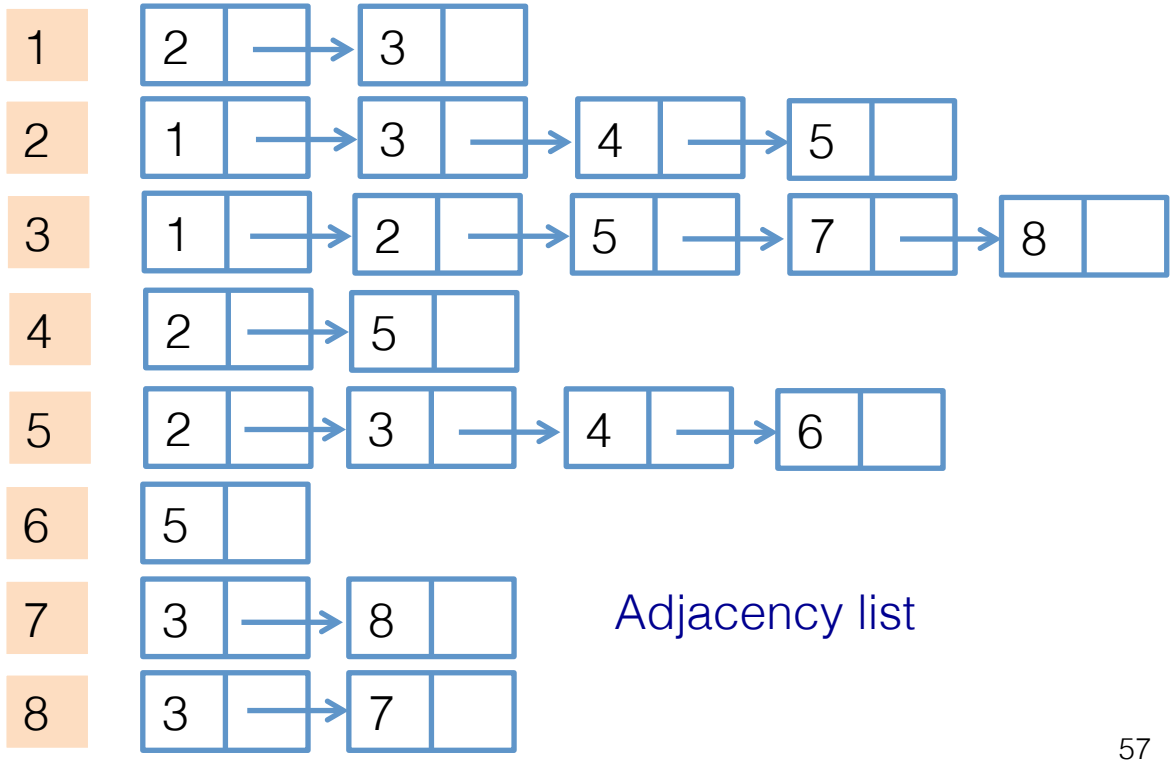
Directed graph

Nonsymmetric matrix

Graph Representation: Adjacency List

- Adjacency lists
  - Representation of a graph with n nodes using an array of n lists of nodes
  - List i contains node j if there is an edge (i, j)
  - A weighted graph can be represented with a list of node/weight pairs
  - Space proportional to $\Theta(m+n)$
  - Checking if (i, j) is an edge takes $O(k_i)$ time

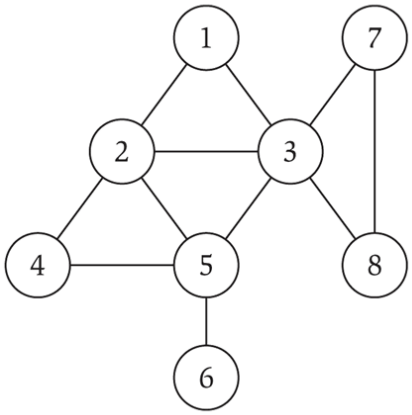




Adjacency list

Graph Representation: Edge List

- Edge list
  - Very simple way to represent a graph
  - List of $|E|$ edges
  - An edge is represented as as pair of vertices (e.g., source, destination)
  - Space proportional to $\Theta(|E|)$
  - Checking if (i, j) is an edge takes $O(|E|)$ time (if the edges appear in no particular order)

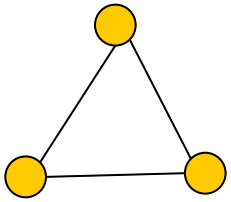


1	2
1	3
2	3
2	4
2	5
3	5
3	7
3	8
4	5
5	6
7	8

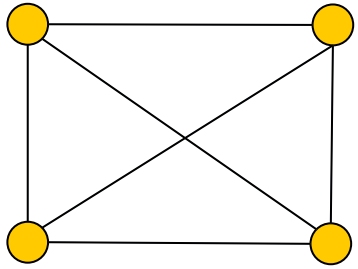
Edge list

Complete Graph

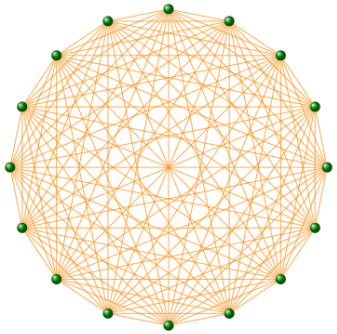
- **Definition:** A graph $G = (V, E)$ is called complete K_n if every pair of nodes is connected by an edge



Complete graph with 3 nodes: triangle (K_3)



K_4



K_{16}

- Q: What is the number of edges in K_n (complete graph with n nodes)?

$$\binom{n}{2} = \frac{n!}{2!(n-1)!} = \frac{n(n-1)}{2}$$

Real Networks are Sparse Graphs

Most real-world networks are sparse: $|E| << |E_{max}|$ (or avg. $k << |V|-1$)

Network	# of nodes	Avg. degree
WWW (Stanford-Berkeley)	319,717	9.65
Social networks (LinkedIn)	6,946,668	8.87
Communication (MSN IM)	242,720,596	11.1
Coauthorships (DBLP)	317,080	6.62
Internet (AS-Skitter):	1,719,037	14.91
Roads (California)	1,957,027	2.82
Proteins (S. Cerevisiae)	1,870	2.39

Consequence:
The adjacency matrix is filled with zeros!

Graph density:

$$D = \frac{2|E|}{|V|(|V|-1)}$$

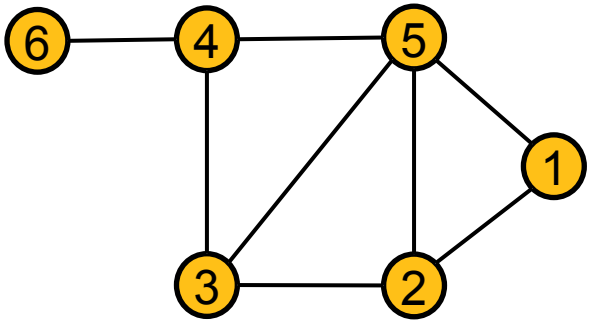
Density of the graph:

- WWW=1.51×10^{-5}
- MSN IM = 2.27×10^{-8}

Source: Leskovec et al., Internet Mathematics, 2009

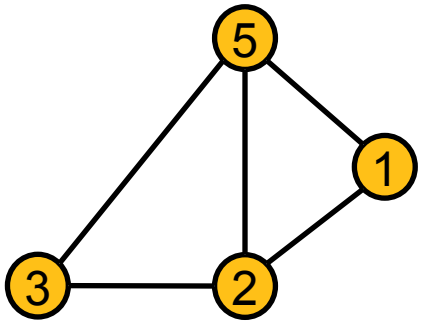
Subgraphs

- Let *G = (V, E)* be a graph and let $S \subseteq V$ be any subset of its vertices



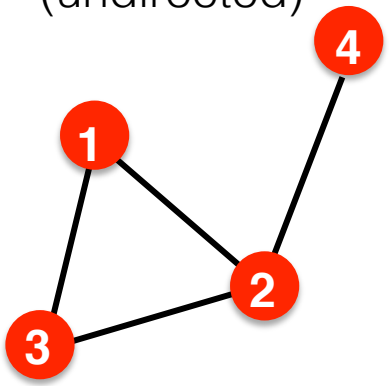
- Definition: The induced subgraph *G[S] = (S, E')* is the graph whose vertex set is *S* and its edge set consists of all of the edges in *E* that have both endpoints in *S*

S={1, 2, 3, 5}



More Types of Graphs (1/2)

Unweighted
(undirected)



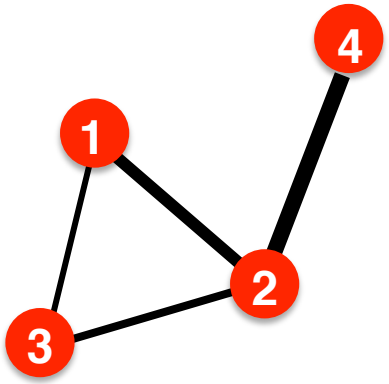
$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$|E| = \frac{1}{2}\sum_{i,j=1}^{n} A_{ij} \qquad \bar{k} = \frac{2|E|}{n}$$

Examples: Friendship, Hyperlink

Weighted
(undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

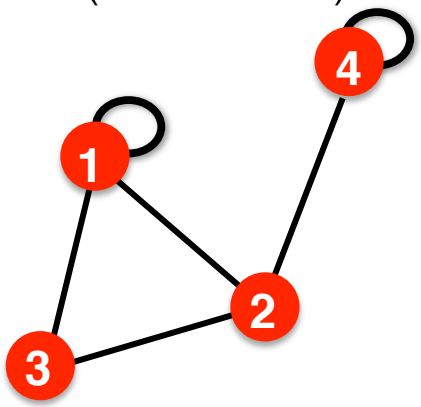
$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$|E| = \frac{1}{2}\sum_{i,j=1}^{n} nonzero(A_{ij}) \qquad \bar{k} = \frac{2|E|}{n}$$

Examples: Collaboration, Internet, Roads

More Types of Graphs (2/2)

Self-edges (self-loops
(undirected)



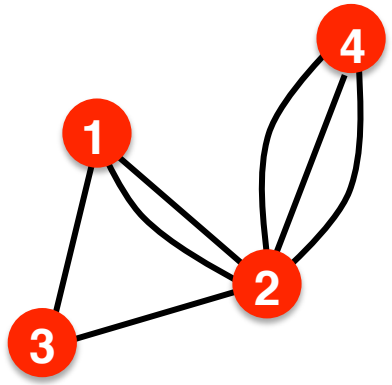
$$A_{ij} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$A_{ii} \neq 0$ $\qquad$ $A_{ij} = A_{ji}$

$$|E| = \frac{1}{2}\sum_{i,j=1,i\neq j}^{n} A_{ij} + \sum_{i=1}^{n} A_{ii}$$

Examples: Proteins, Hyperlinks

Multigraph
(undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

$A_{ii} = 0$ $\qquad$ $A_{ij} = A_{ji}$

$$|E| = \frac{1}{2}\sum_{i,j=1}^{n} nonzero(A_{ij}) \qquad \bar{k} = \frac{2|E|}{n}$$

Examples: Communication, Collaboration

Network Representations

direction, edge weight, self-edges, acyclic, multigraph

WWW > directed multigraph with self-edges

Facebook friendships > undirected, unweighted

Citation networks > directed, unweighted, acyclic

Collaboration networks > undirected multigraph or weighted graph

Mobile phone calls > directed, (weighted?) multigraph

Twitter graph > directed, weighted (?)

Protein Interactions > undirected, unweighted with self-interactions

Representation Matters!

Choice of the proper network representation of a given system determines our ability to use networks successfully

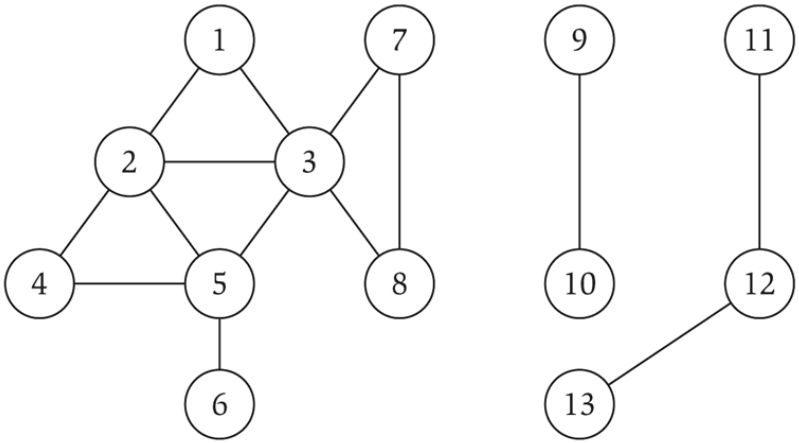
Paths, graph connectivity and distance

Paths and Connectivity in Graphs

- Definition: A path in an undirected graph $G=(V,E)$ is a sequence of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair v_{i-1}, v_i is joined by an edge in E

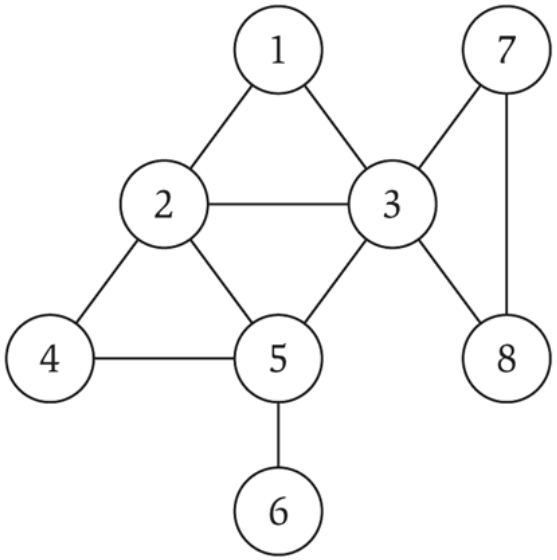
- Definition: An undirected graph is connected if for every pair of nodes u and v, there is a path between u and v

Is this graph connected?



Cycles in Graphs

- Definition: A cycle is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$, $k > 2$ and the first $k-1$ nodes are all distinct

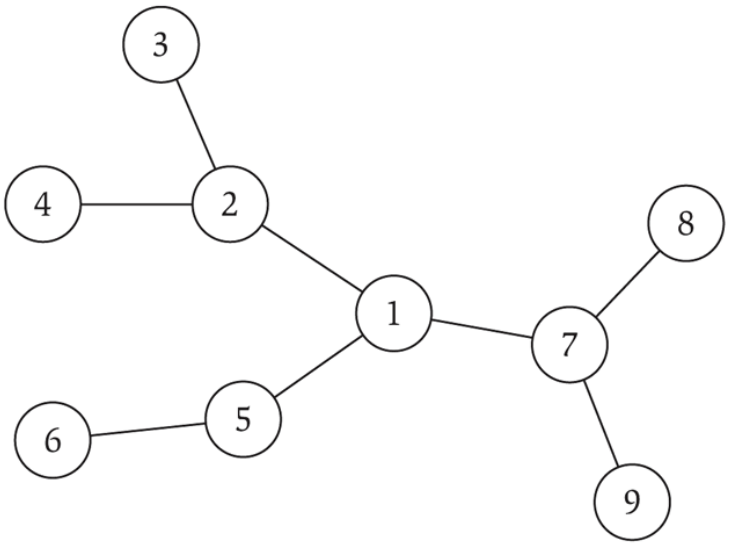


Cycle C = 1 – 2 – 4 – 5 – 3 – 1

Trees

- Definition: An undirected graph is a tree if it is connected and does not contain a cycle

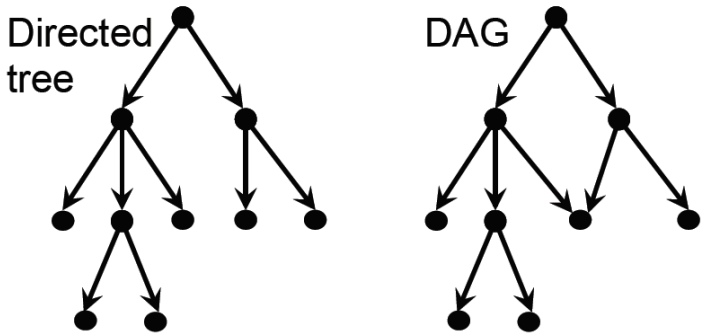
- Theorem: Let *G* be an undirected graph with *n* nodes. Then, any two of the following statements imply the third:
  - *G* is connected
  - *G* does not contain a cycle
  - *G* has *n-1* edges



Directed Acyclic Graph

- Definition: A directed tree is a directed graph whose underlying undirected graph is a tree
  - Root is the only vertex with paths to all the other nodes
  - Vertex terminology: parent, children, ancestor, descendant, leaf

- A directed acyclic graph (DAG) is a directed graph with no directed cycles

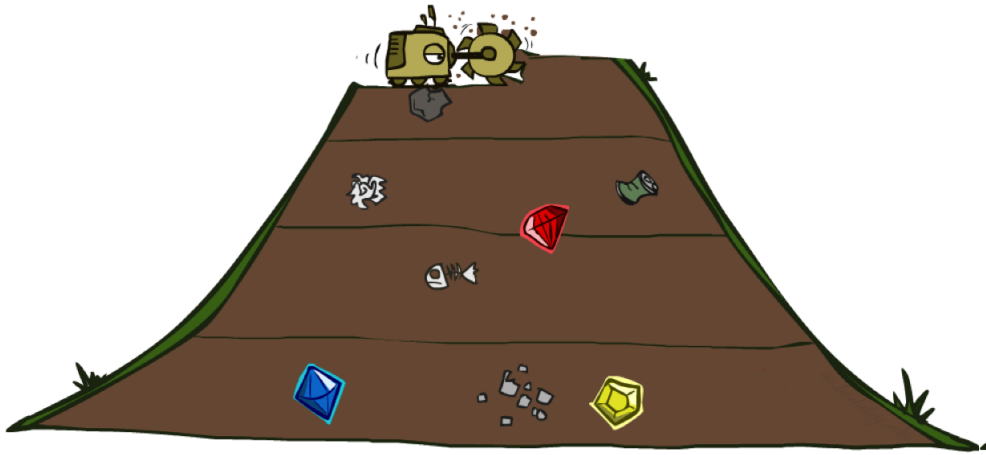


Graph Traversal

- Graph traversal is the problem of visiting all the nodes in the graph
  - Also known as graph search

- Graph traversal algorithms
  - Breadth-first search (BFS)
  - Depth-first search (DFS)

BFS vs. DFS



BFS

DFS

Source: A. Shehu, F. Li, GMU

Breadth-First Search (BFS)

- Strategy for searching in graphs when search is limited to two operations:
  - Visit and inspect a node of the graph
  - Visit the neighborhood nodes of currently visited node

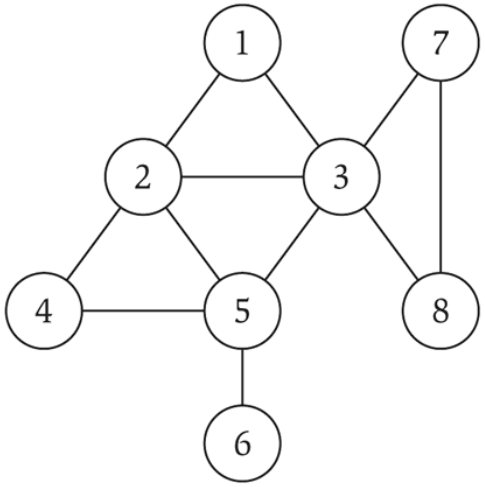
```
1  procedure BFS(G, v):
2      create a queue Q
3      add v onto Q
4      mark v
5      while Q is not empty:
6          t ← Q.dequeue()
7          if t is what we are looking for:
8              return t
9          for all edges e in G.adjacentEdges(t) do
12             o ← G.adjacentVertex(t,e)
13             if o is not marked:
14                 mark o
15                 add o in Q
```

Input: Graph *G* and a node *v*

Output: An assignment (labeling) of the nodes of the graph into layers (or the node closest to *v* in *G* satisfying some conditions)

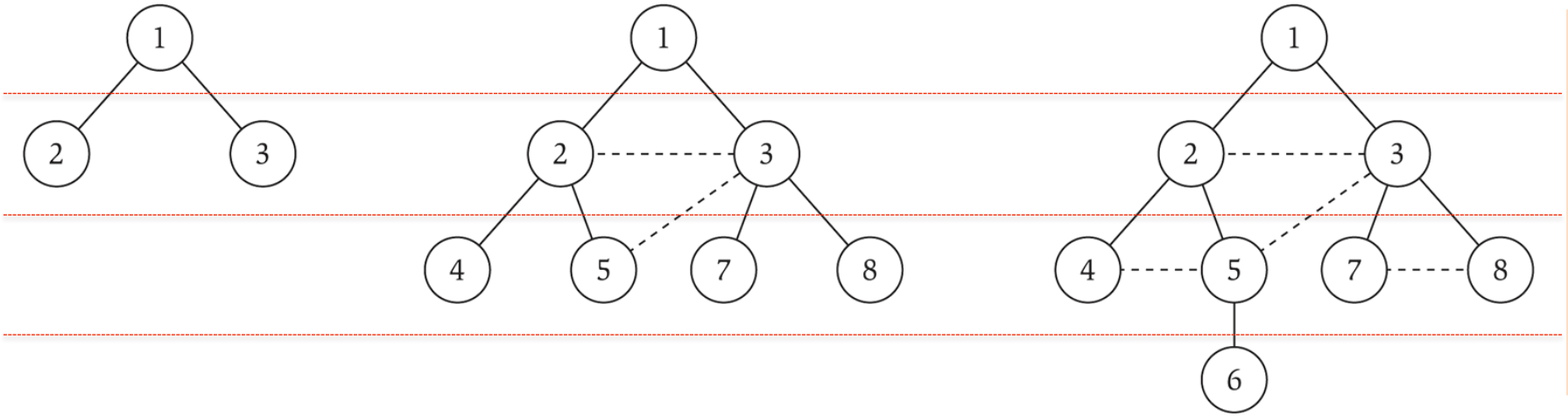
Complexity: $\mathcal{O}(|V| + |E|)$

Breadth-First Search (BFS)
Example



Start BFS from node 1

Property: Let T be a BFS tree of *G* and let *(u, v)* be an edge of *G*. Then, the levels of *u* and *v* differ by at most 1



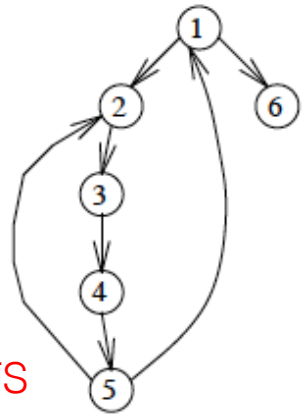
Depth-First Search (DFS)

- Strategy for traversing graphs:
  - Visits the children nodes before visiting the sibling nodes
  - Traverses the depth of any particular path before exploring the breadth

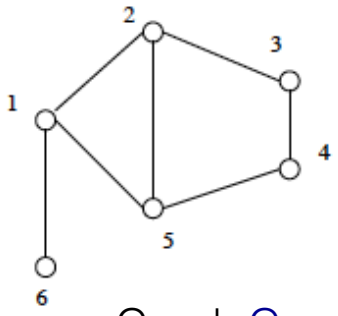
```
1  procedure DFS(G,v):
2      label v as explored
3      for all edges e in G.adjacentEdges(v) do
4          if edge e is unexplored then
5              w ← G.adjacentVertex(v,e)
6              if vertex w is unexplored then
7                  label e as a discovery edge
8                  recursively call DFS(G,w)
9              else
10                 label e as a back edge
```

Input: Graph *G* and a node *v*

Output: A labeling of the edges of the graph as discovery and back edges



DFS

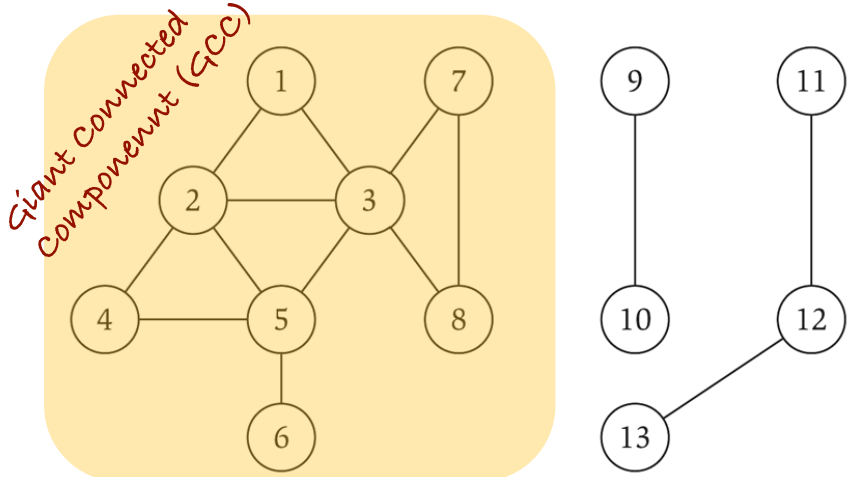


Graph G

Complexity: $\mathcal{O}(|V| + |E|)$

Connected Components

- A connected component is a maximal connected subgraph of a graph G (there is a path between any pair of nodes)
  - Maximal means adding another vertex will ruin connectivity



Graph with 3 connected components

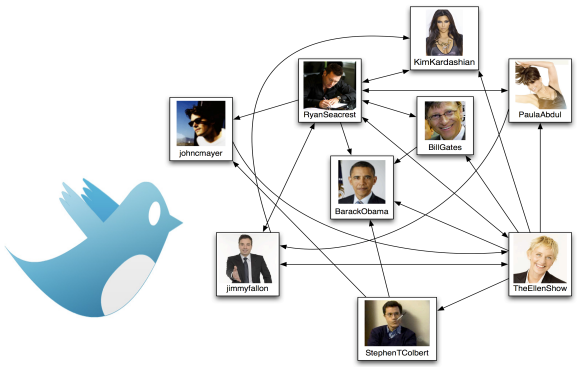
Connected component containing node 1: {1, 2, 3, 4, 5, 6, 7, 8}

Question: How can we compute the connected components of a graph?

A: Apply BFS

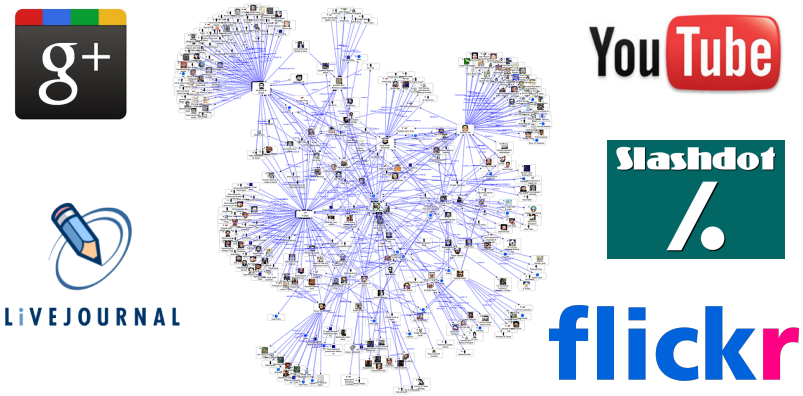
Connectivity in Directed Graphs (1/2)

- A plethora of network data from several applications is from their nature directed

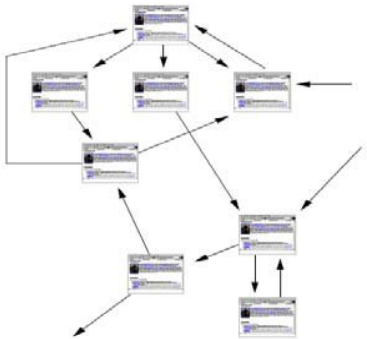


Twitter

[Image: http://sites.davidson.edu/mathmovement/]

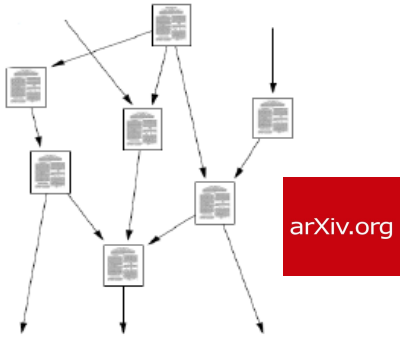


Online Social Networks



Web Graph

Wikipedia

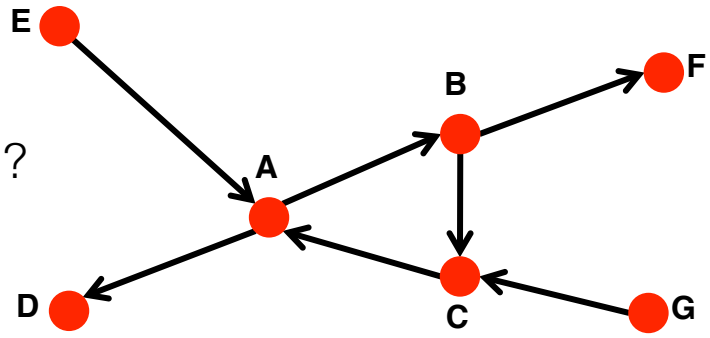


Citation Graph

Connectivity in Directed Graphs (2/2)

- Directed reachability: Given a node *s* in a directed graph *G*, find all nodes reachable from *s*
  - Web crawler: start from a webpage *s*, find all webpages linked from s, either directly or indirectly
- The notion of connectivity in directed graphs is replaced by the
  - Strong connectivity: a graph is called strongly connected if it is possible to reach any node taking into account the directionality of the edges
  - Weak connectivity: a graph is called weakly connected if it is possible to reach any node if we do not take into account the directionality of the edges

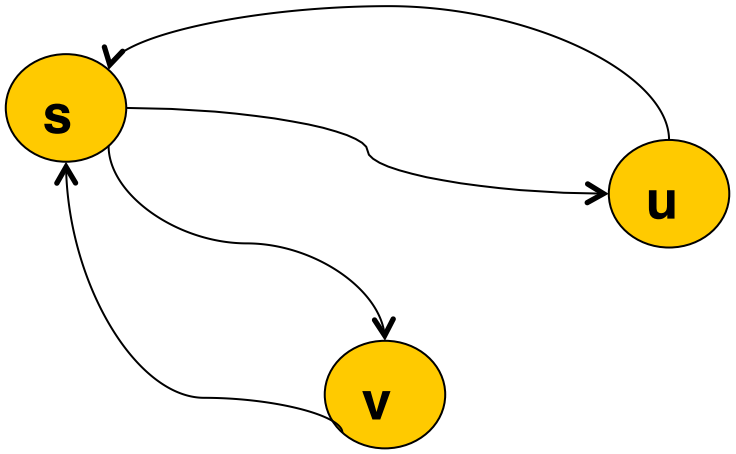
Q: Is this graph strongly connected?



Strong Connectivity

- Lemma: Let *s* be any node. Graph *G* is strongly connected iff every node is reachable from *s*, and *s* is reachable from every node
- Proof:
  - --> Follows from definition
  - <-- Path from *u* to *v*: concatenate *u* --> *s* path with *s* --> *v* path

    Path from *v* to *u*: concatenate *v* --> *s* path with *s* --> *u* path



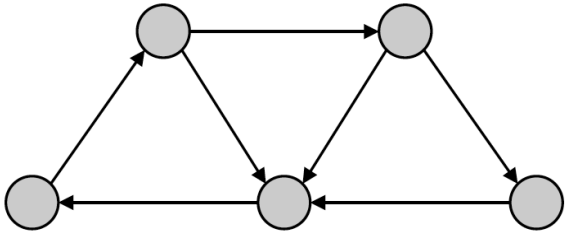
Strong Connectivity: Algorithm

- Theorem: We can determine if G is strongly connected in $O(m+n)$ (i.e., linear) time

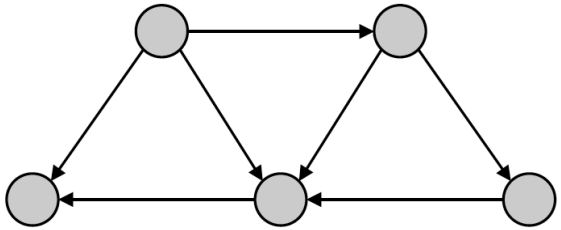
- Proof:

  Idea: use BFS

  – Pick any node s
  – Run BFS from s in G
  – Run BFS from s in $G_{reverse}$(reverse the orientation of the edges)
  – Return true iff all nodes reached in both BFS executions



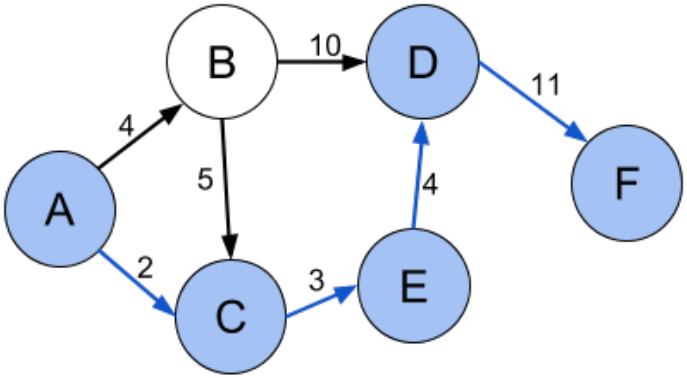
Strongly connected



Not strongly connected

Shortest Paths

- Definition: find a path between two nodes in a graph, in such a way that the sum of the weights of its constituent edges is minimized
  - Many applications (e.g., road networks, community detection, communications)
- Variants
  - Single-source shortest path problem
  - Single-destination shortest path problem
  - All-pairs shortest path problem



Shortest path (A, C, E, D, F) between vertices A and F in the weighted directed graph

Various algorithms:
- Dijkstra
- Bellman-Ford
  (works with negative edge weights)

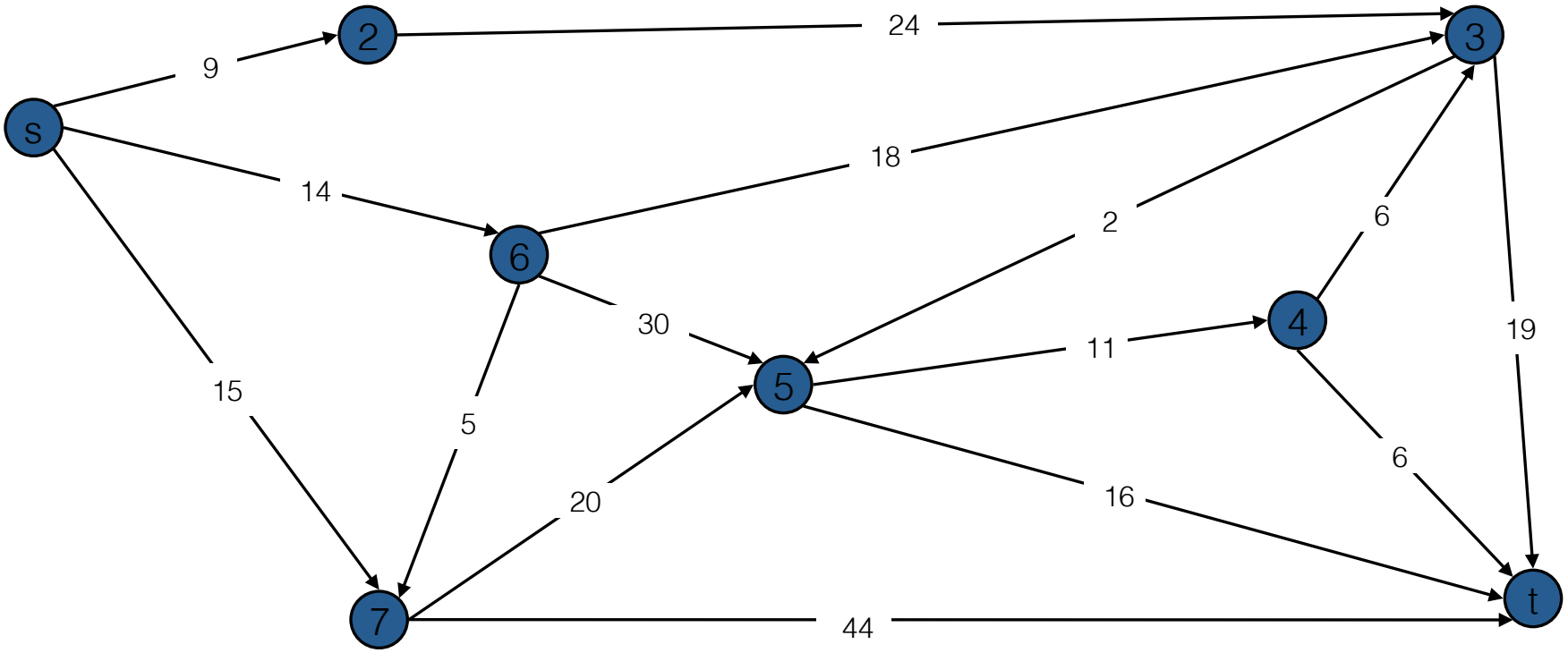
See: https://en.wikipedia.org/wiki/Shortest_path_problem

Dijkstra's Algorithm

```
1   function Dijkstra(Graph, source):
2
3       create vertex set Q
4
5       for each vertex v in Graph:                 // Initialization
6           dist[v] := infinity                     // initial distance from source to vertex v is set to infinite
7           previous[v] := undefined                // Previous node in optimal path from source
8           add v to Q                              // All nodes initially in Q (unvisited nodes)
9
10      dist[source] := 0                           // Distance from source to source
11
12
13      while Q is not empty:                       // Main loop
14          u := node in Q with smallest dist[ ]
15          remove u from Q
16
17          for each neighbor v of u:               // Where v has not been removed yet from Q
18              alt := dist[u] + length(u, v)
19              if alt < dist[v]                    // A shorter path to v has been found
20                  dist[v] := alt
21                  previous[v] := u
22
23  return  dist[], previous[ ]
```

Dijkstra's Shortest Path Algorithm

- Find shortest path from ***s*** to ***t***

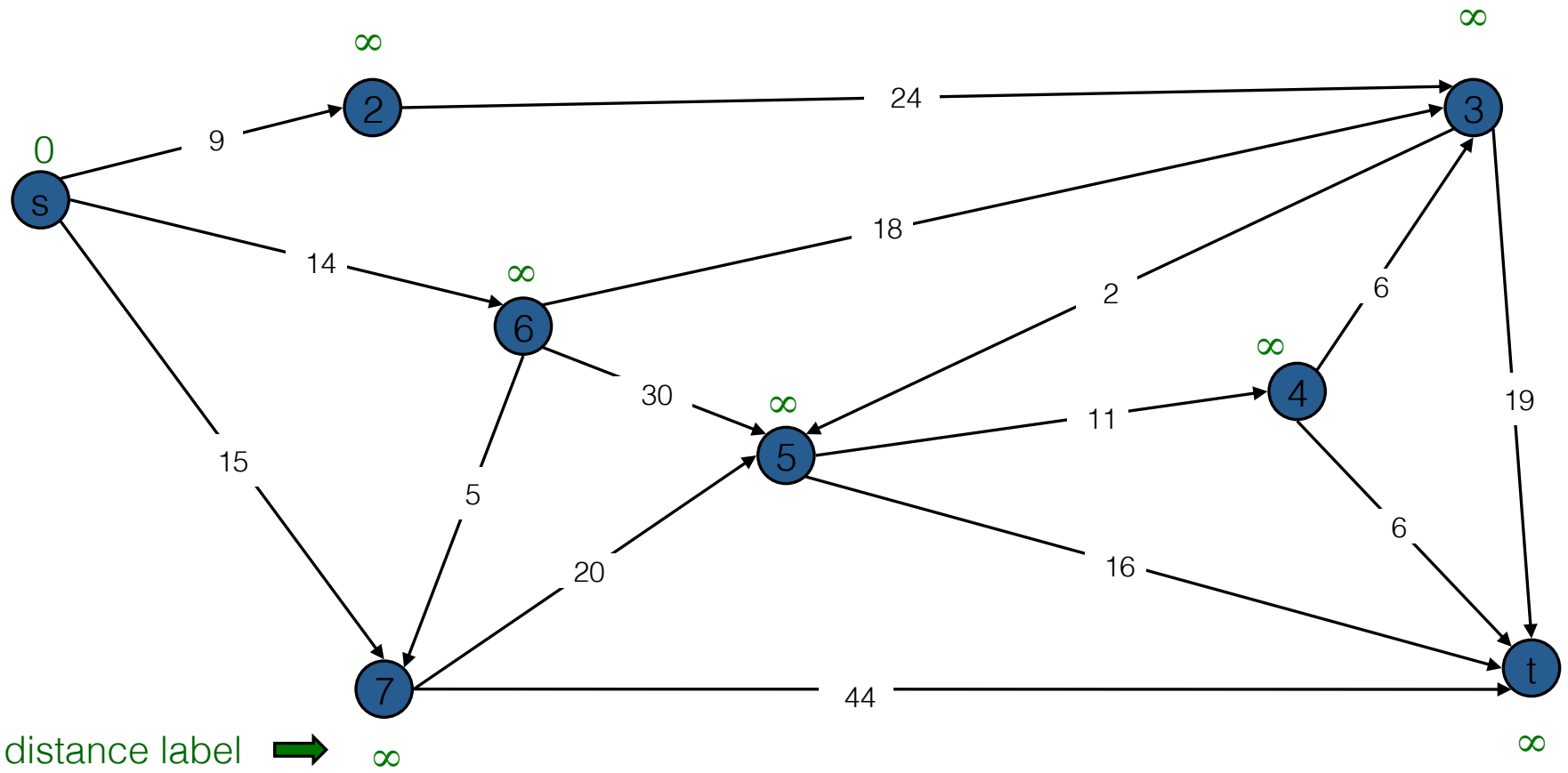


Source: L. Wang, City Univ. of Hong Kong

Dijkstra's Shortest Path Algorithm

Visited = { }

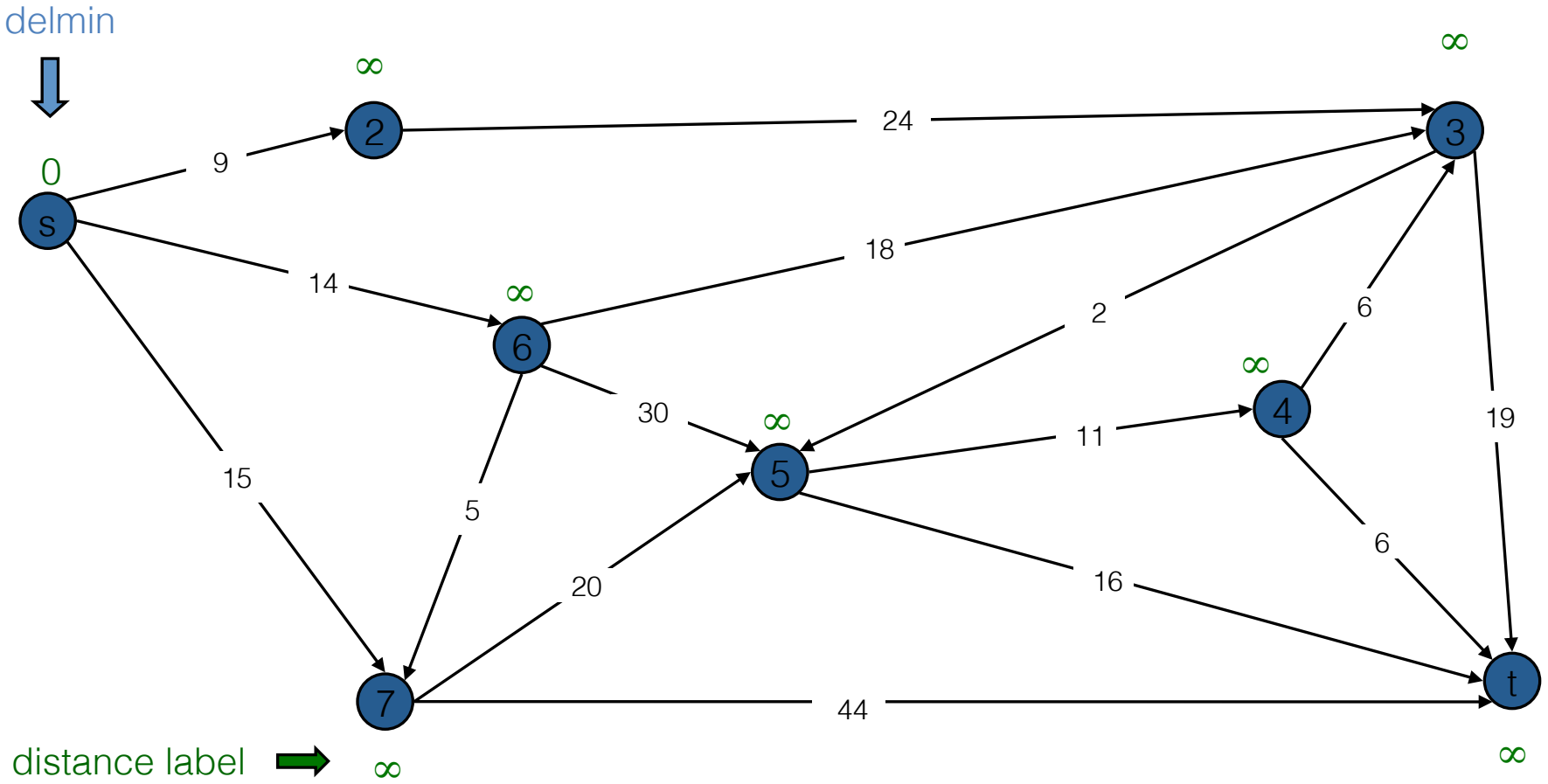
Unvisited = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { }

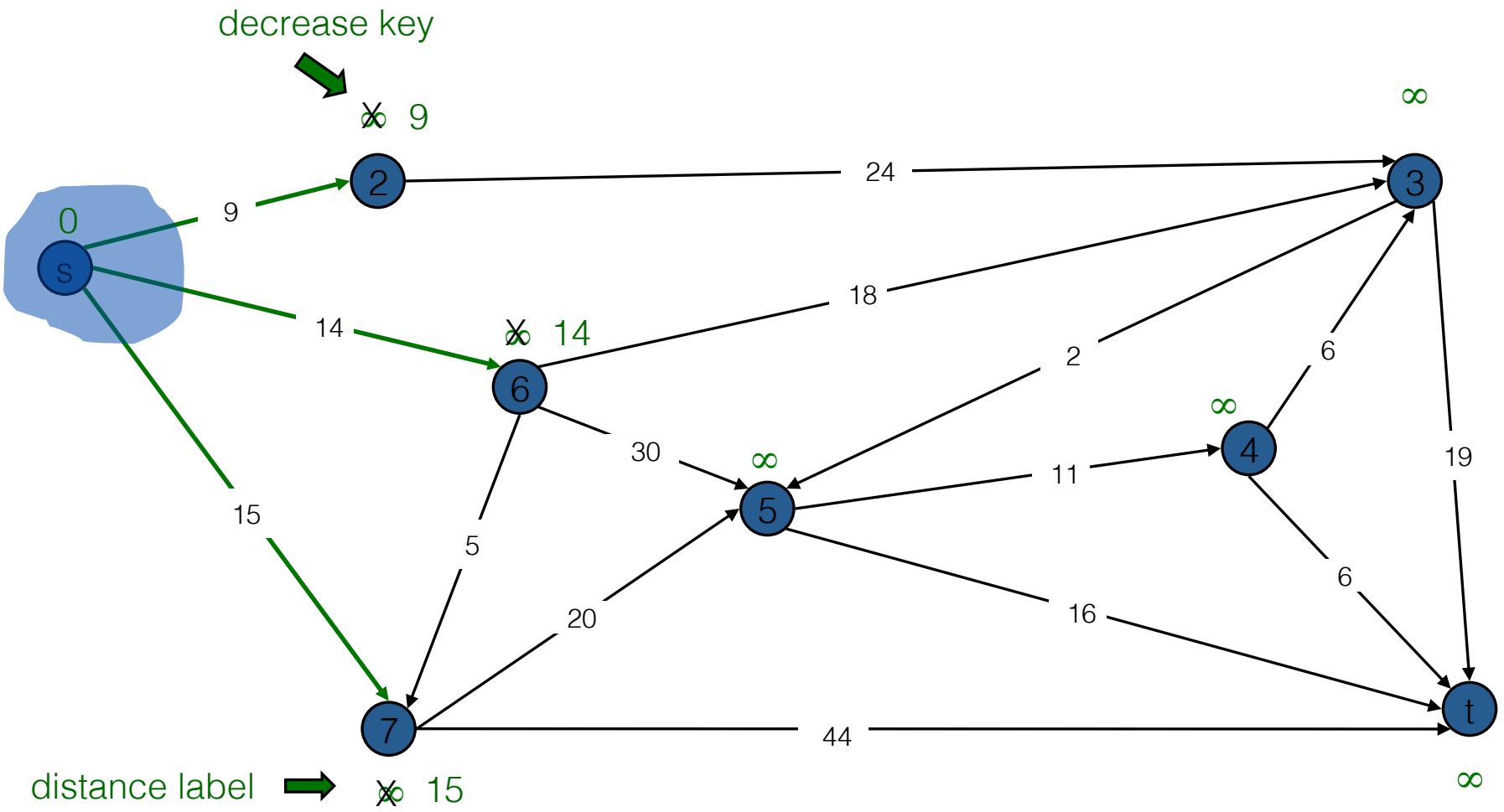
Unvisited = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s }

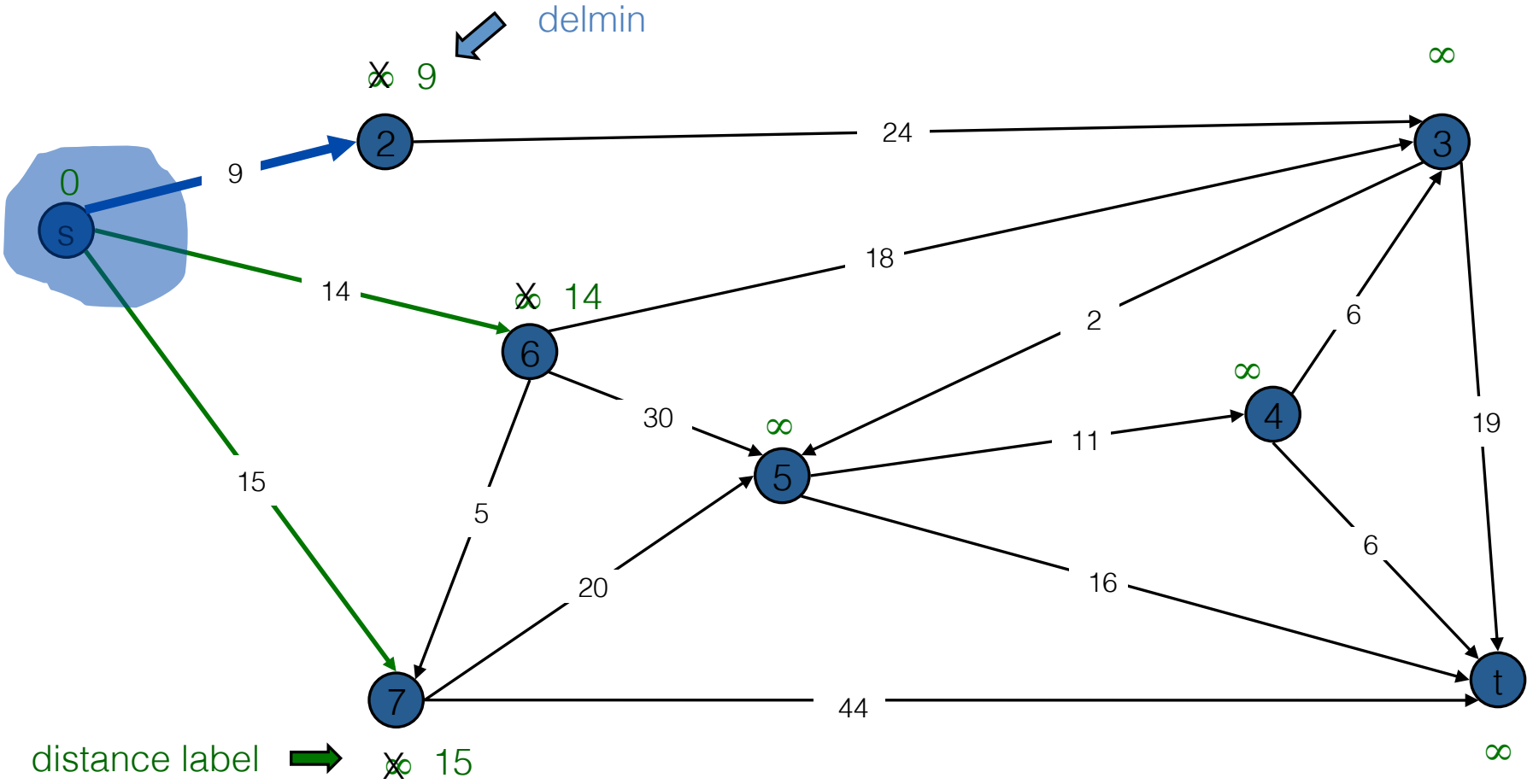
Unvisited = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s }

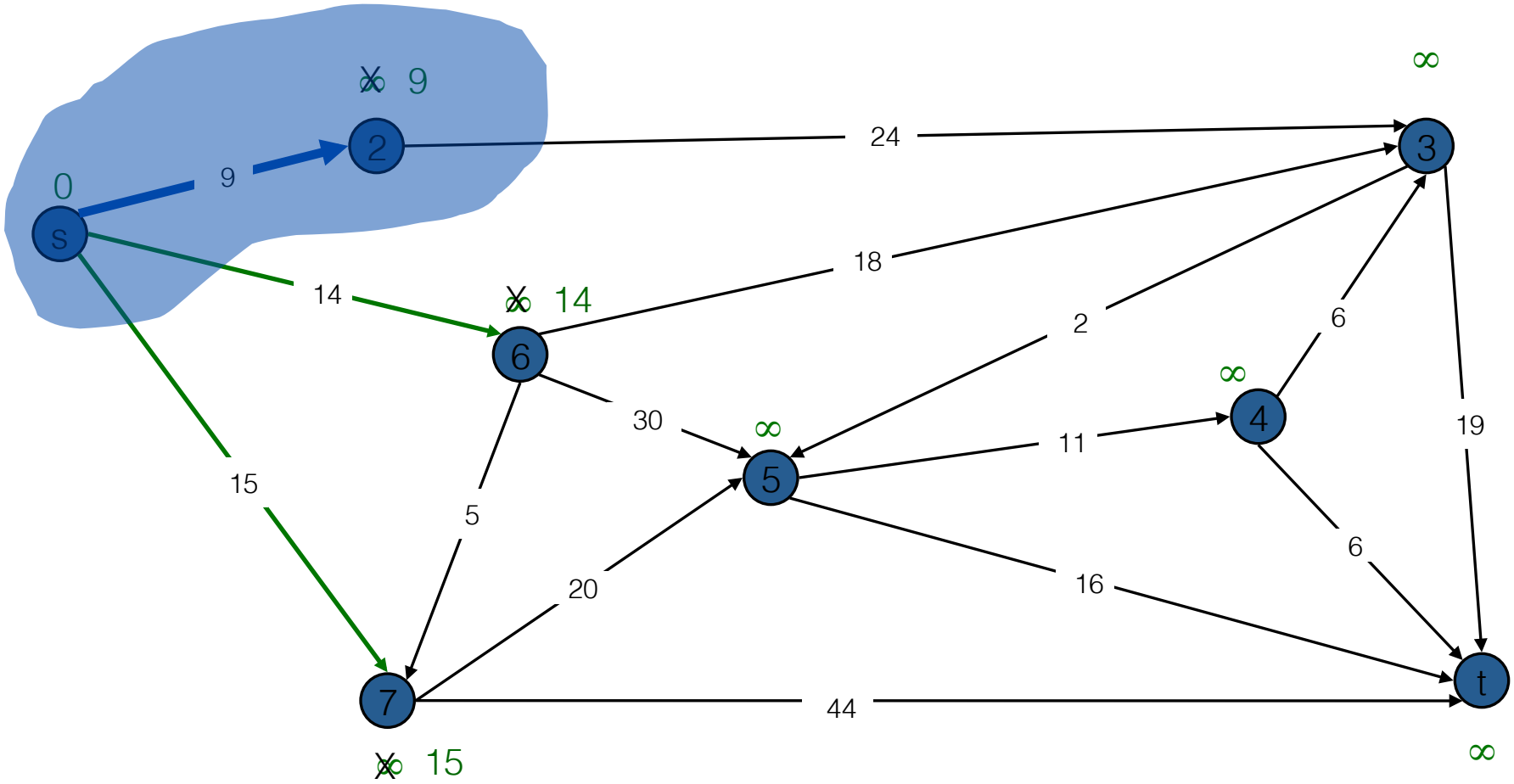
Unvisited = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

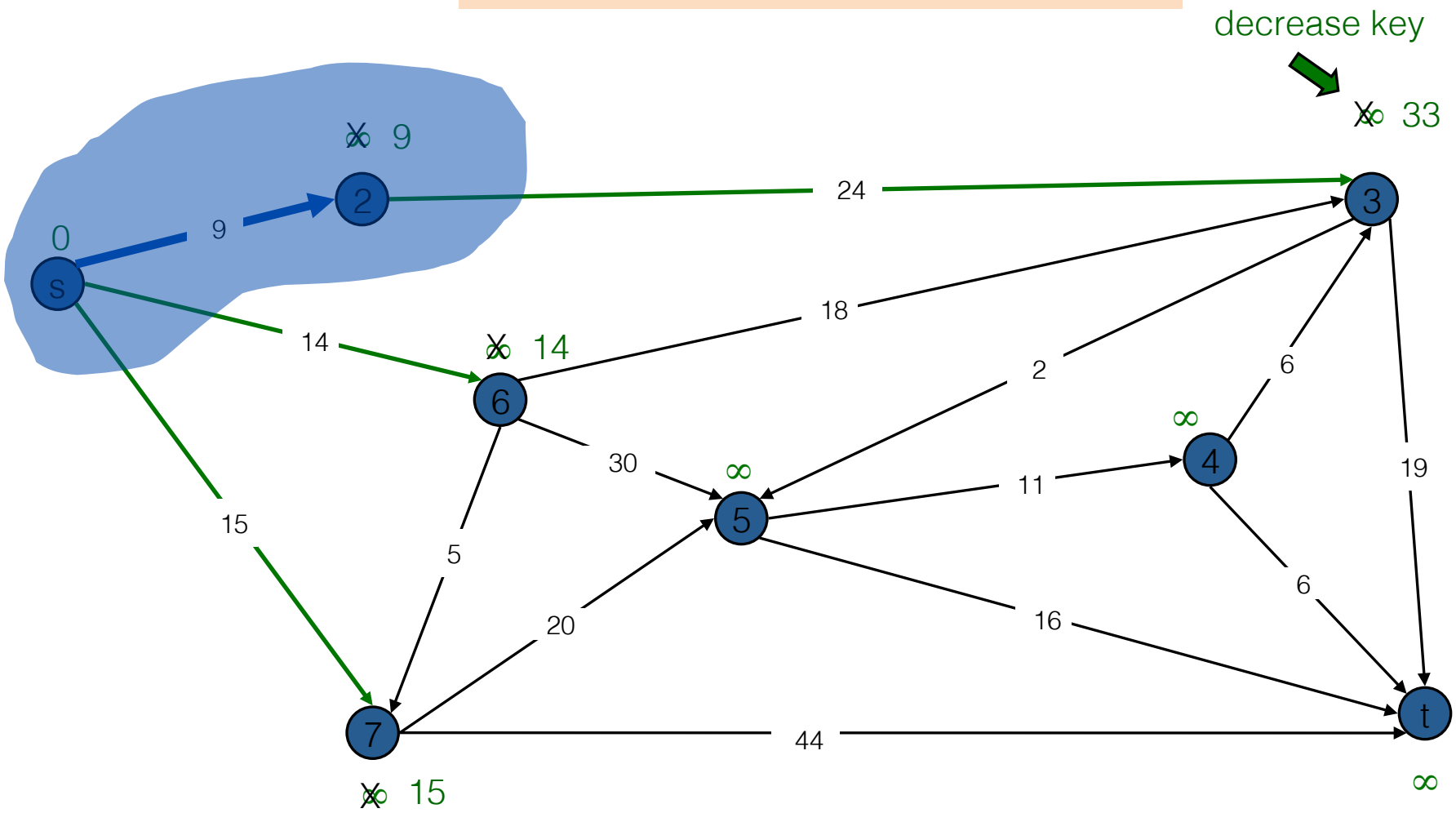
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

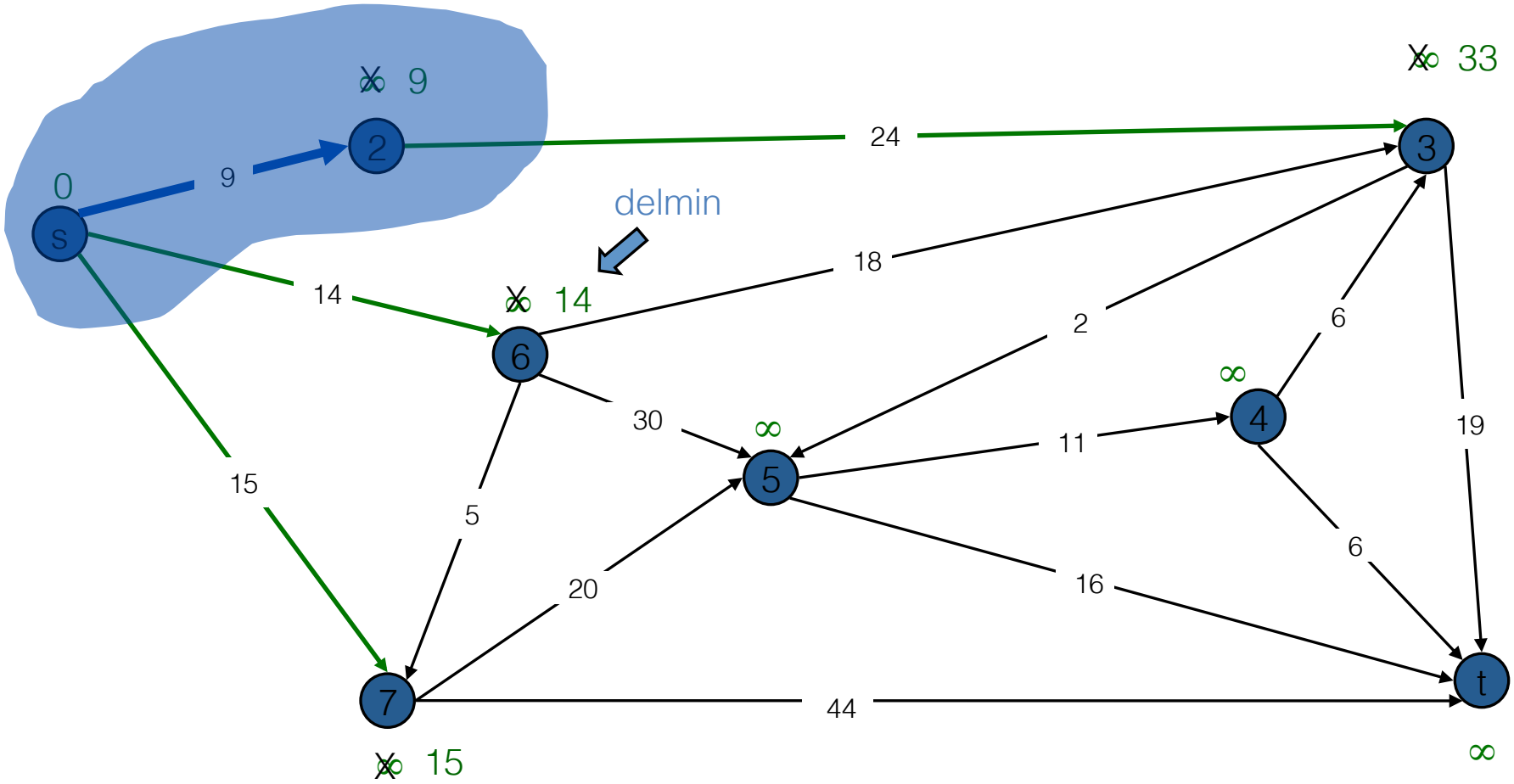
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

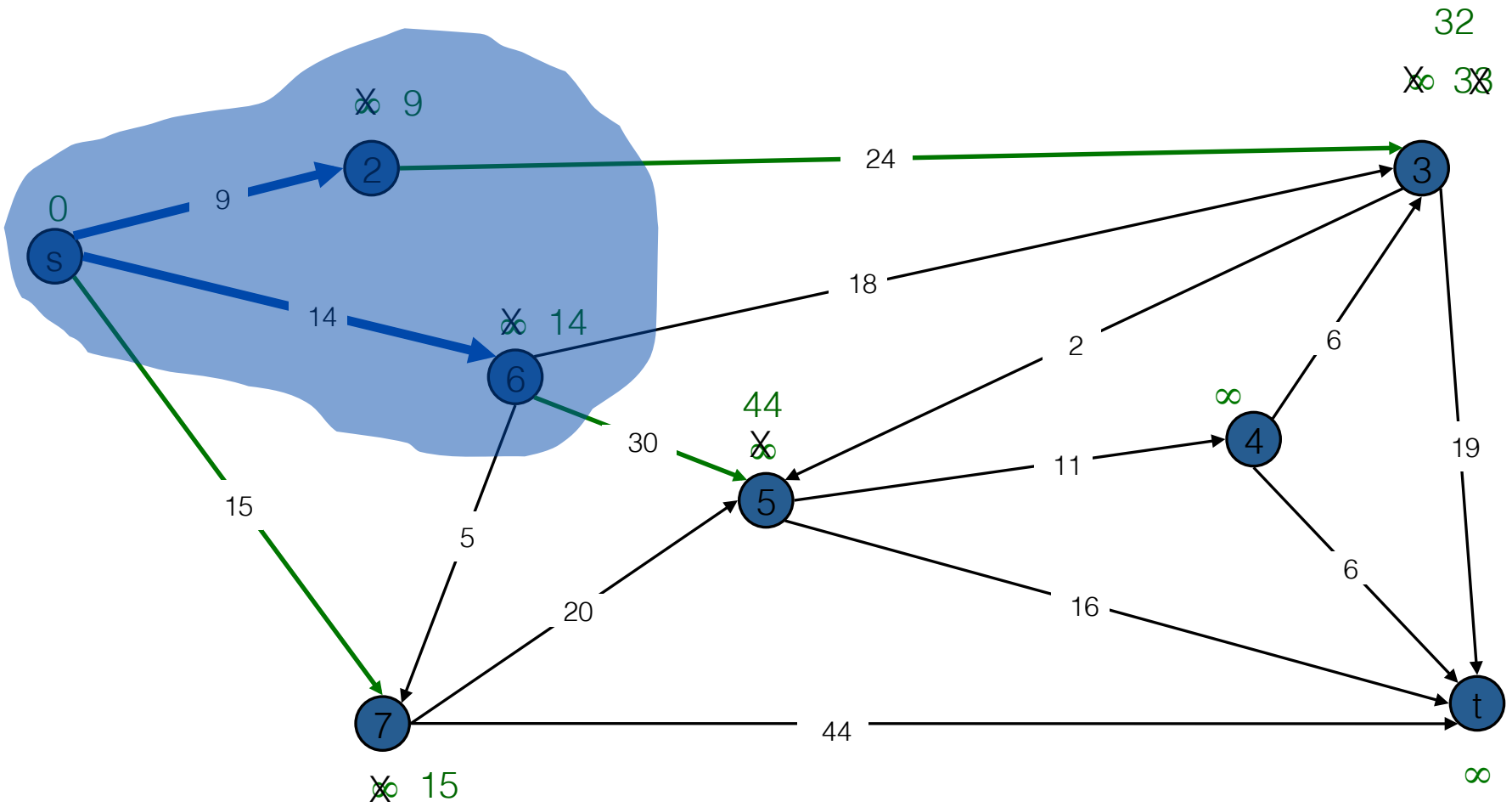
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6 }

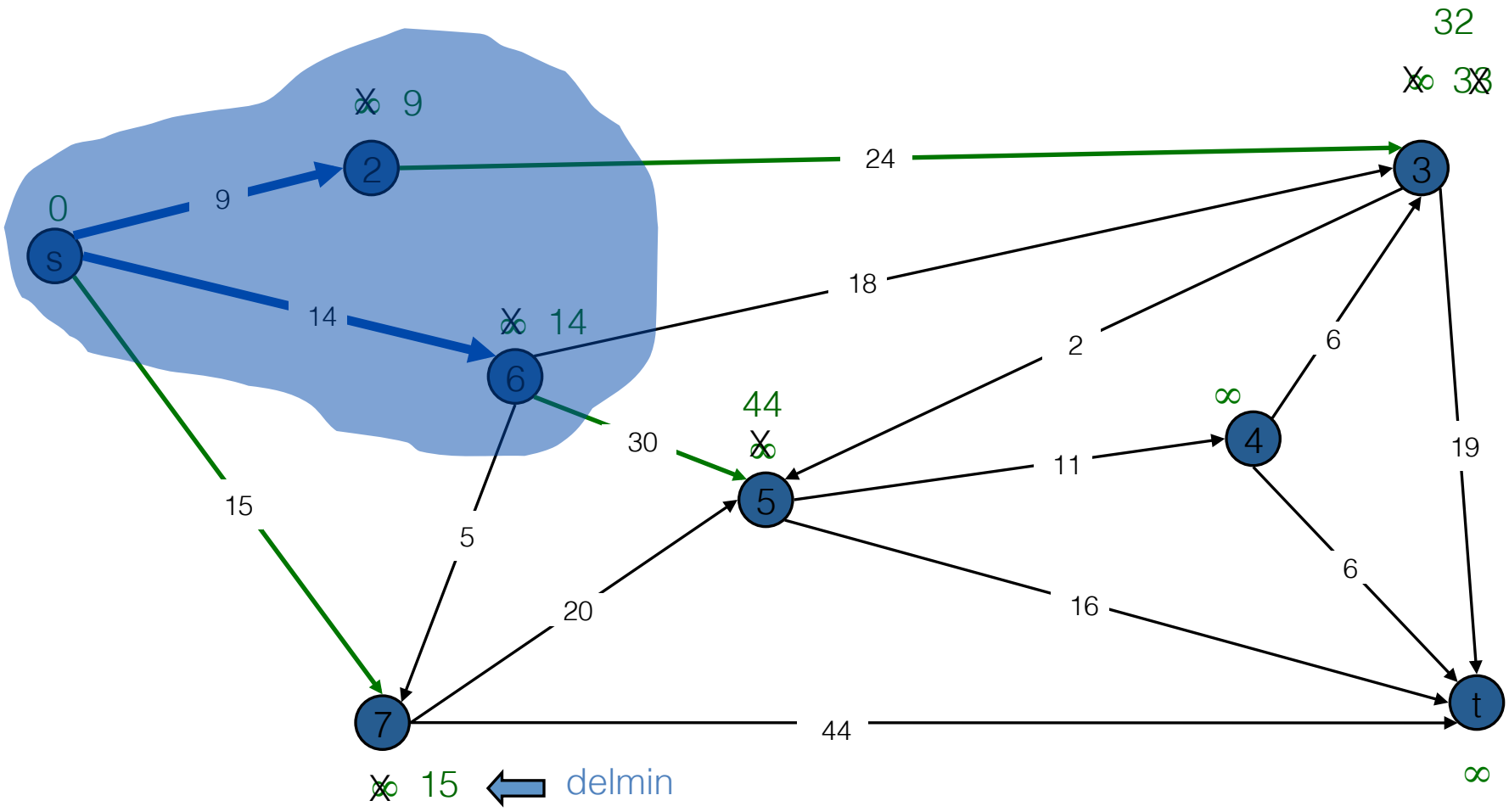
Unvisited = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6 }

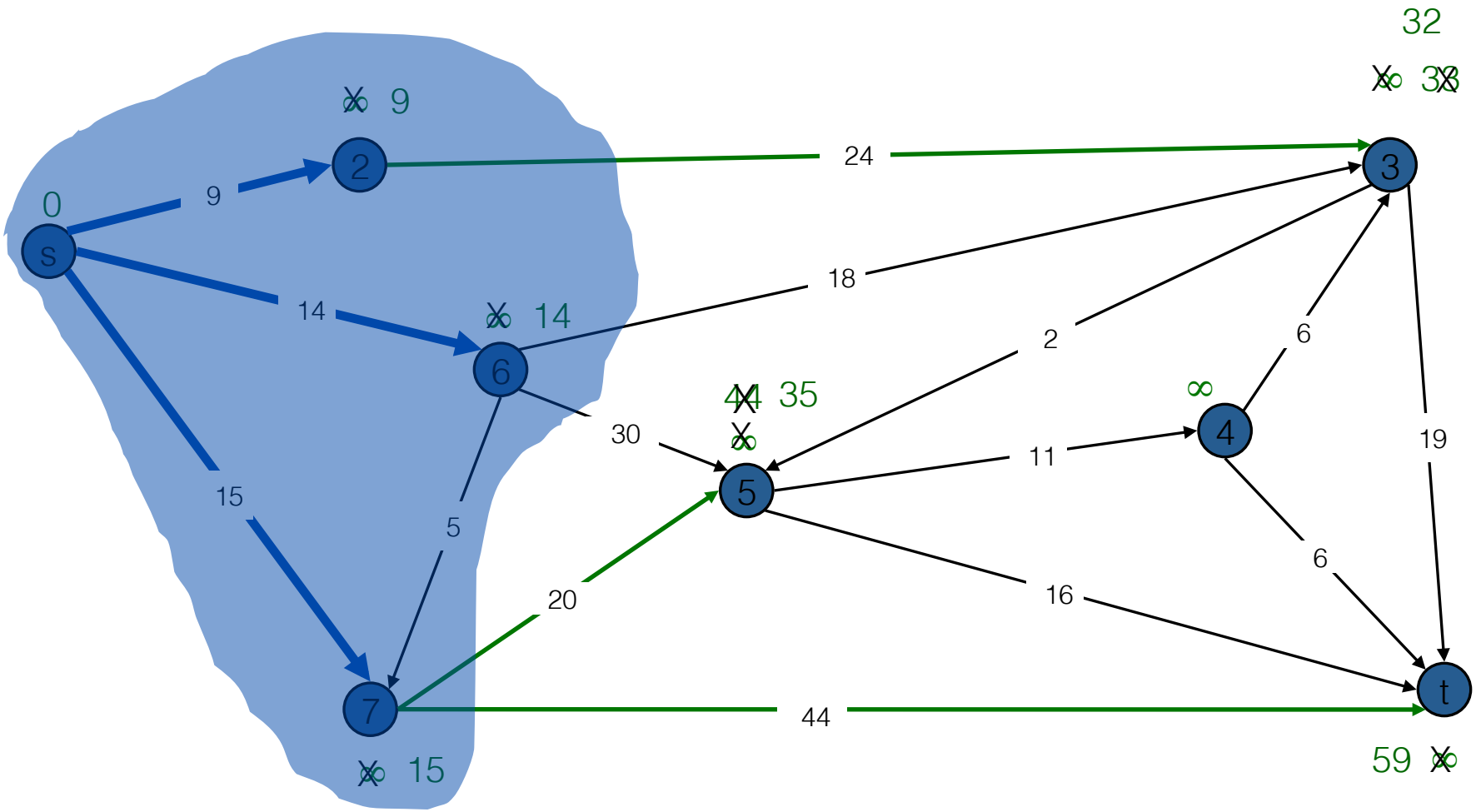
Unvisited = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6, 7 }

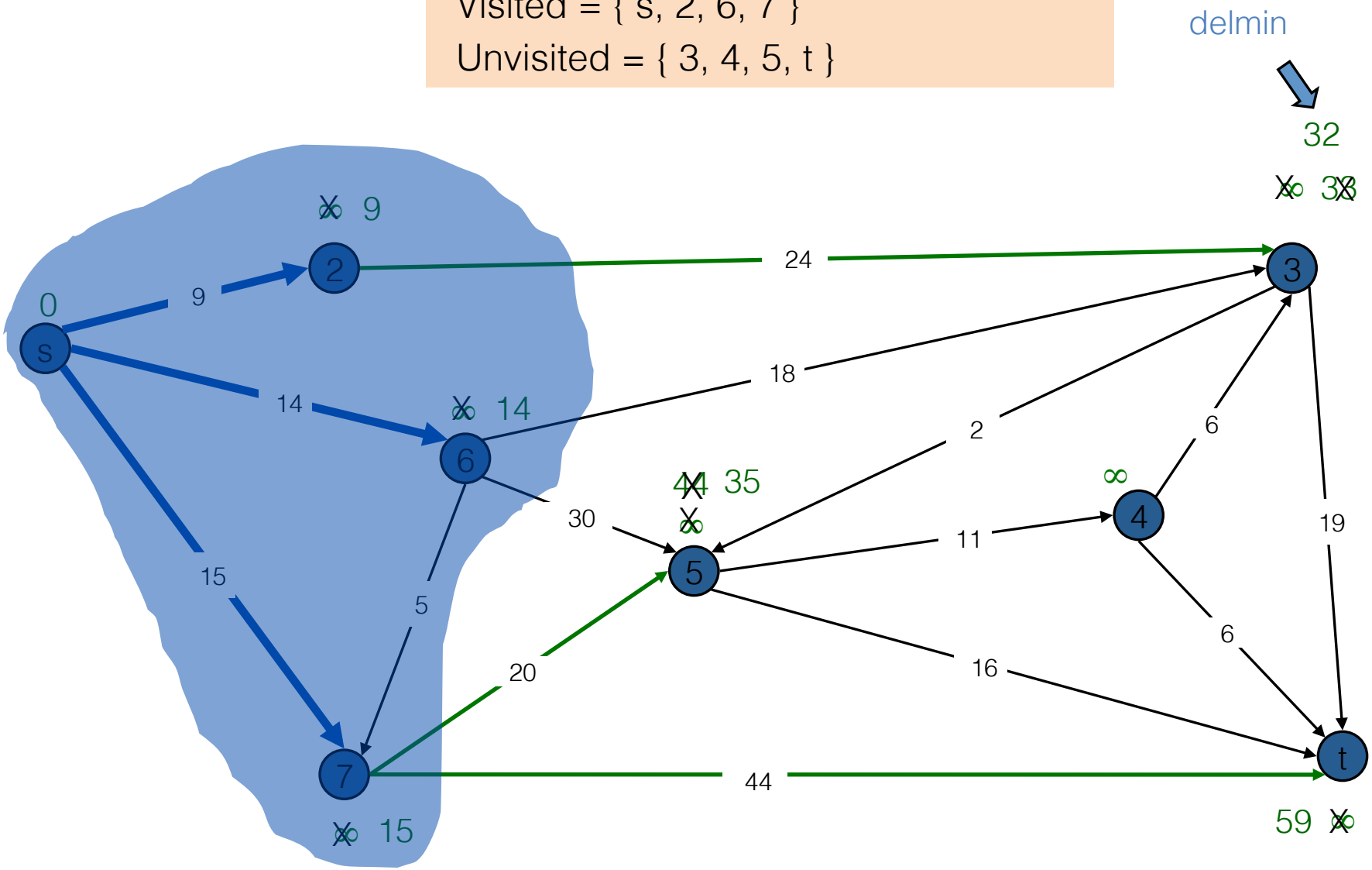
Unvisited = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6, 7 }

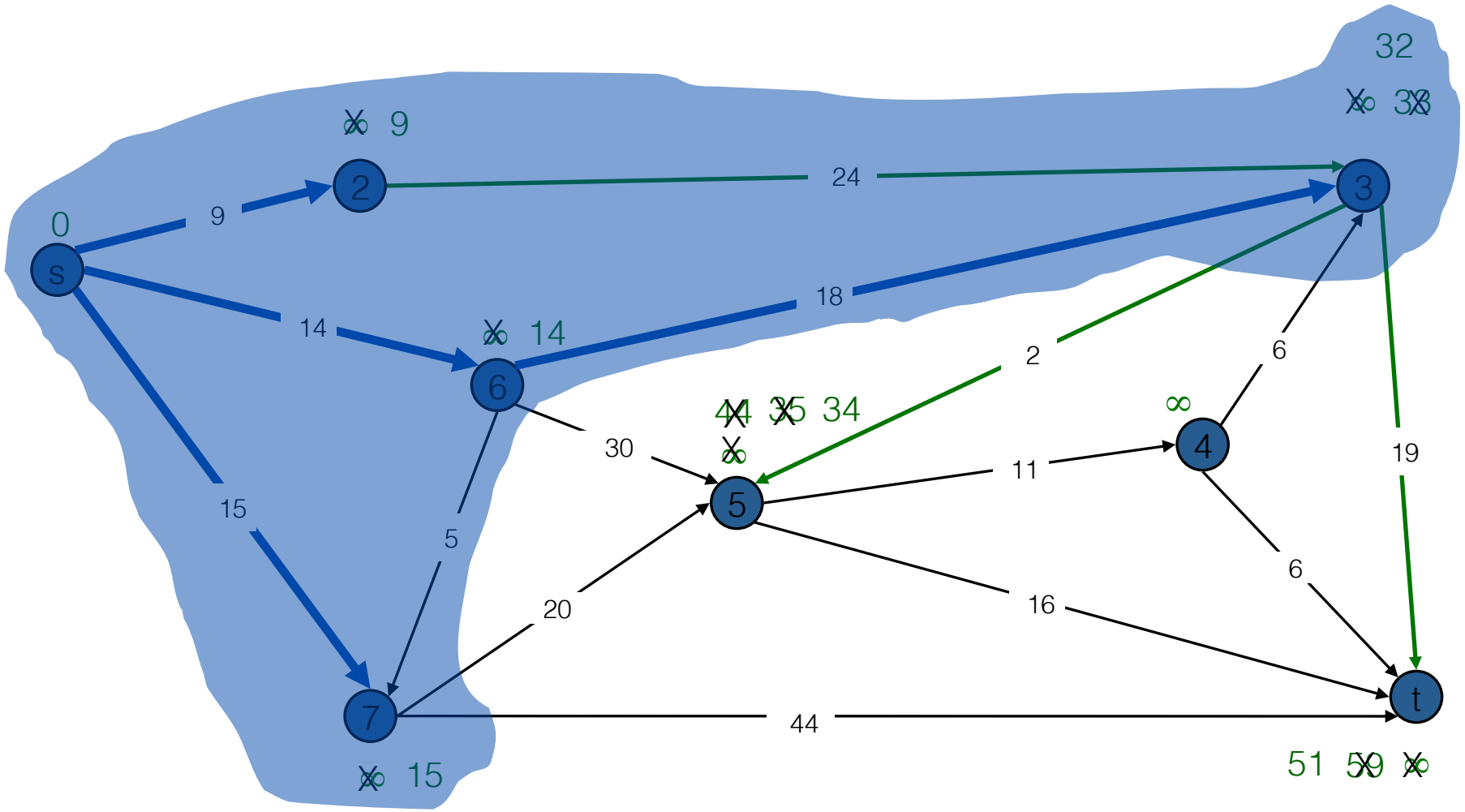
Unvisited = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 3, 6, 7 }

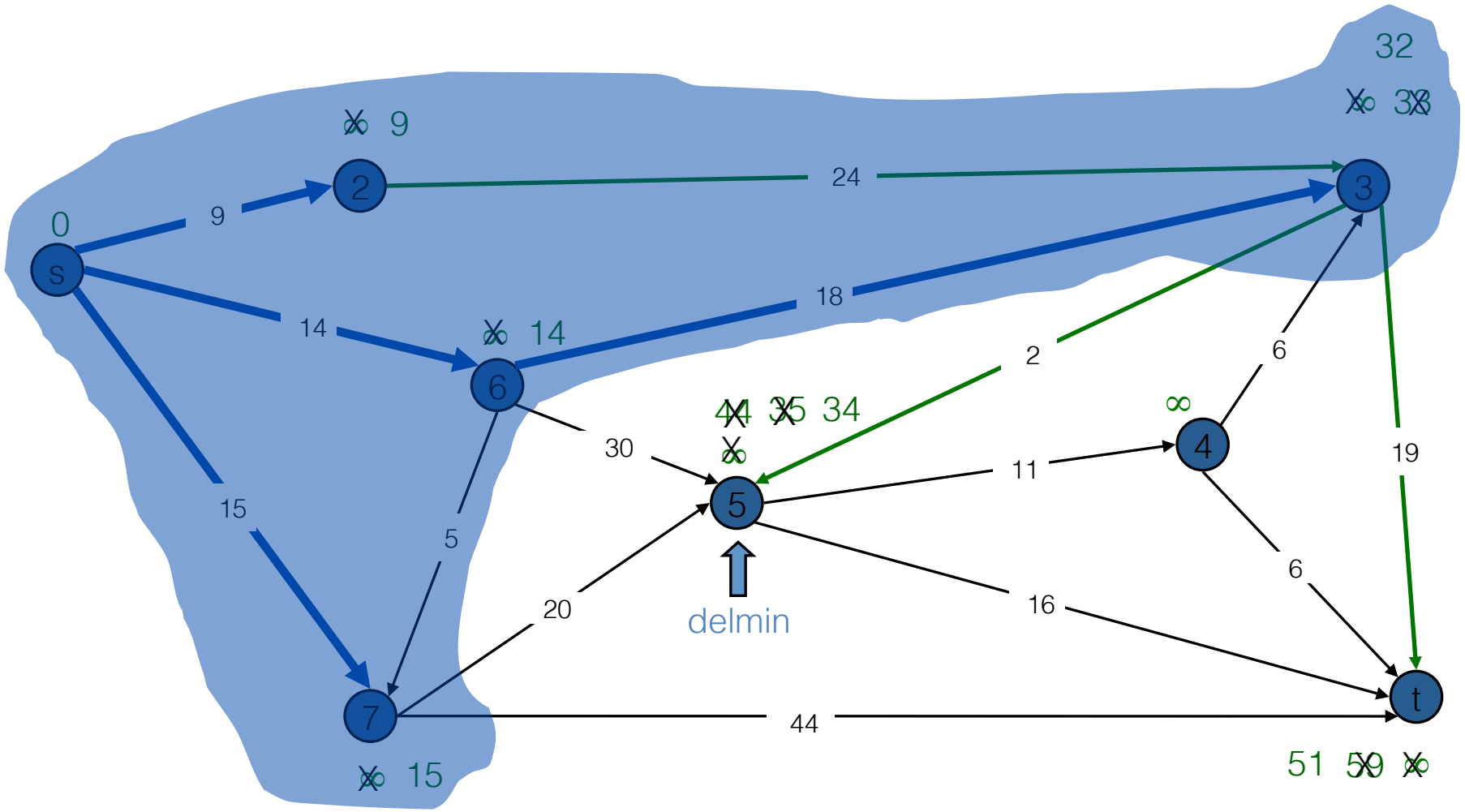
Unvisited = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 3, 6, 7 }

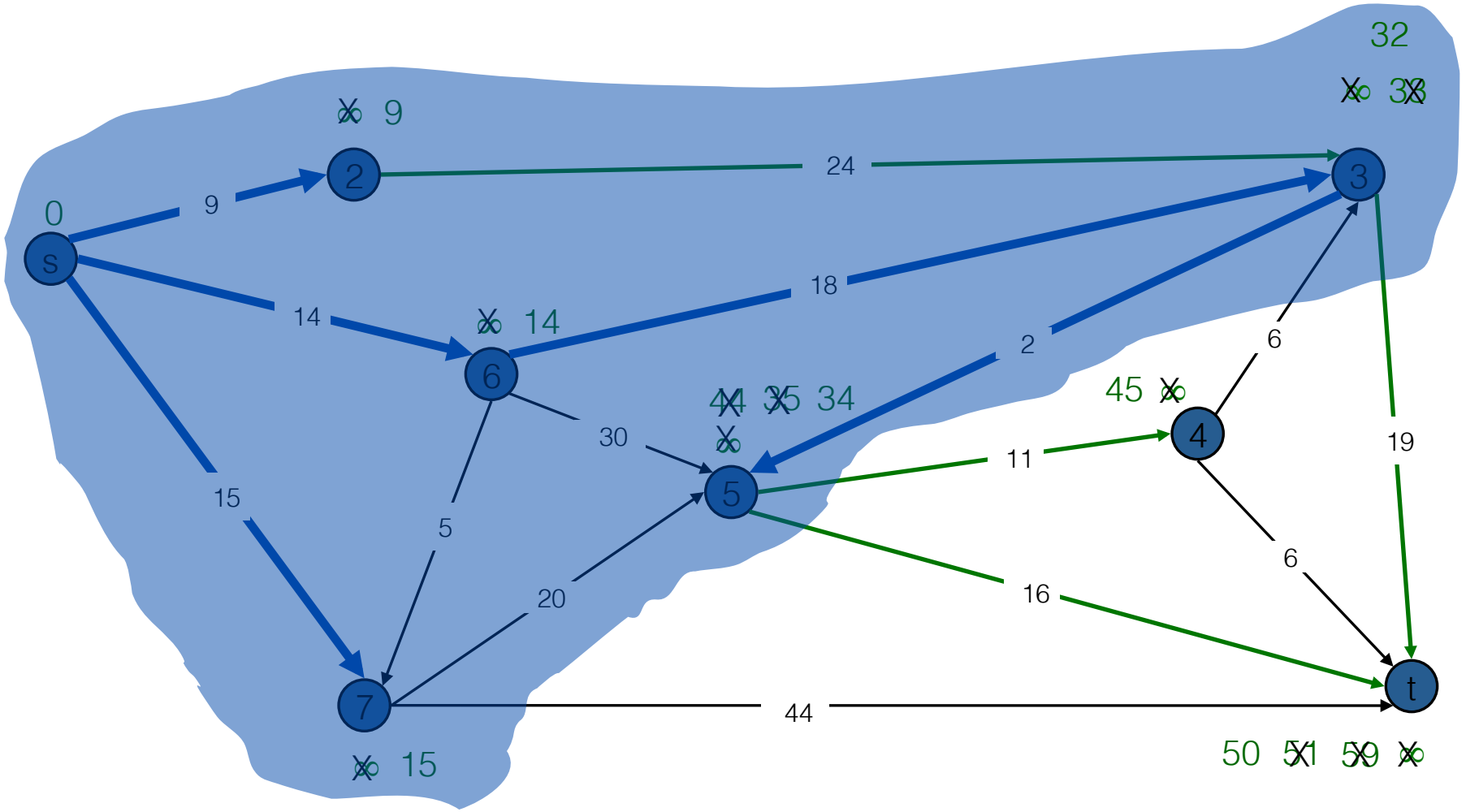
Unvisited = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 3, 5, 6, 7 }

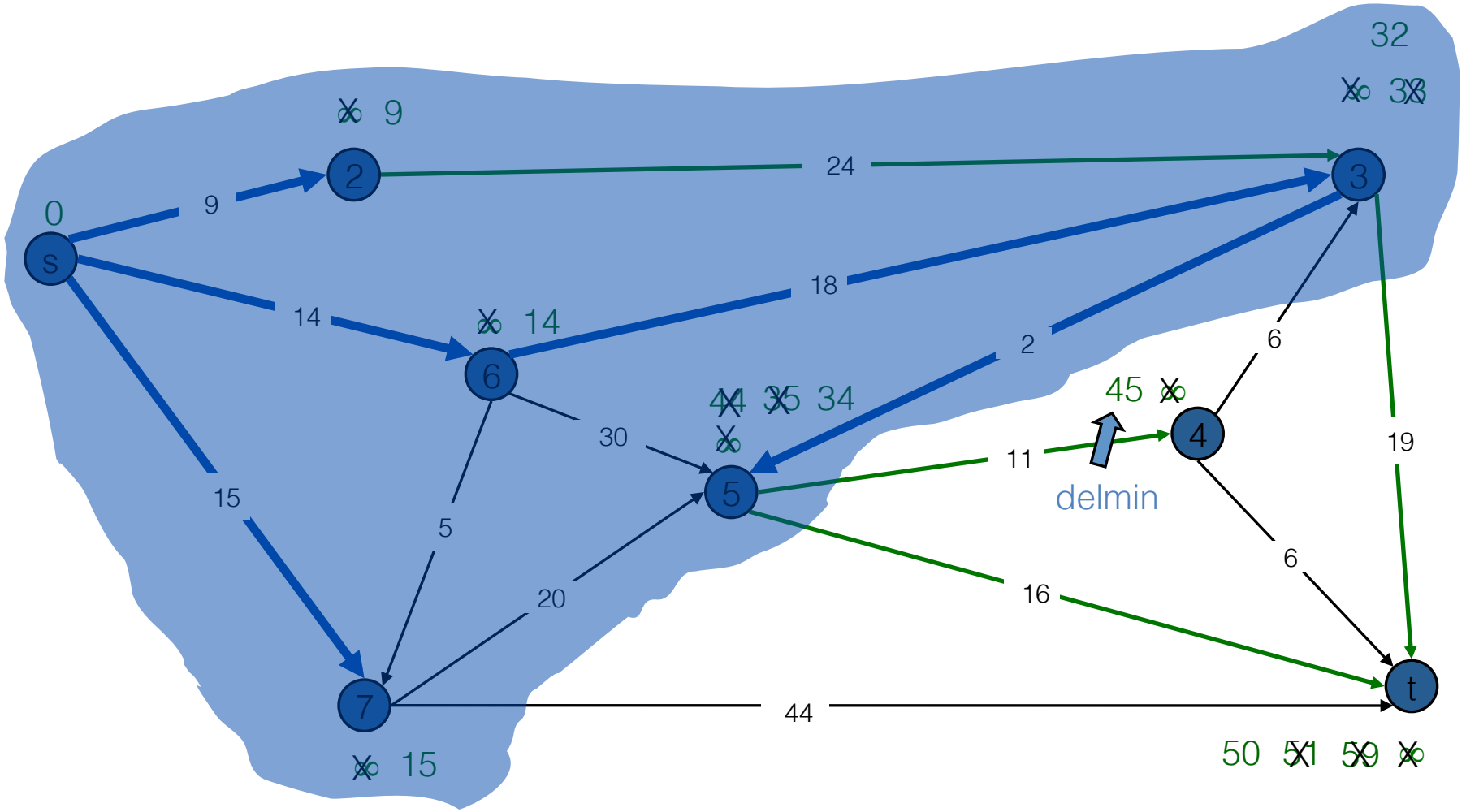
Unvisited = { 4, t }



Dijkstra's Shortest Path Algorithm

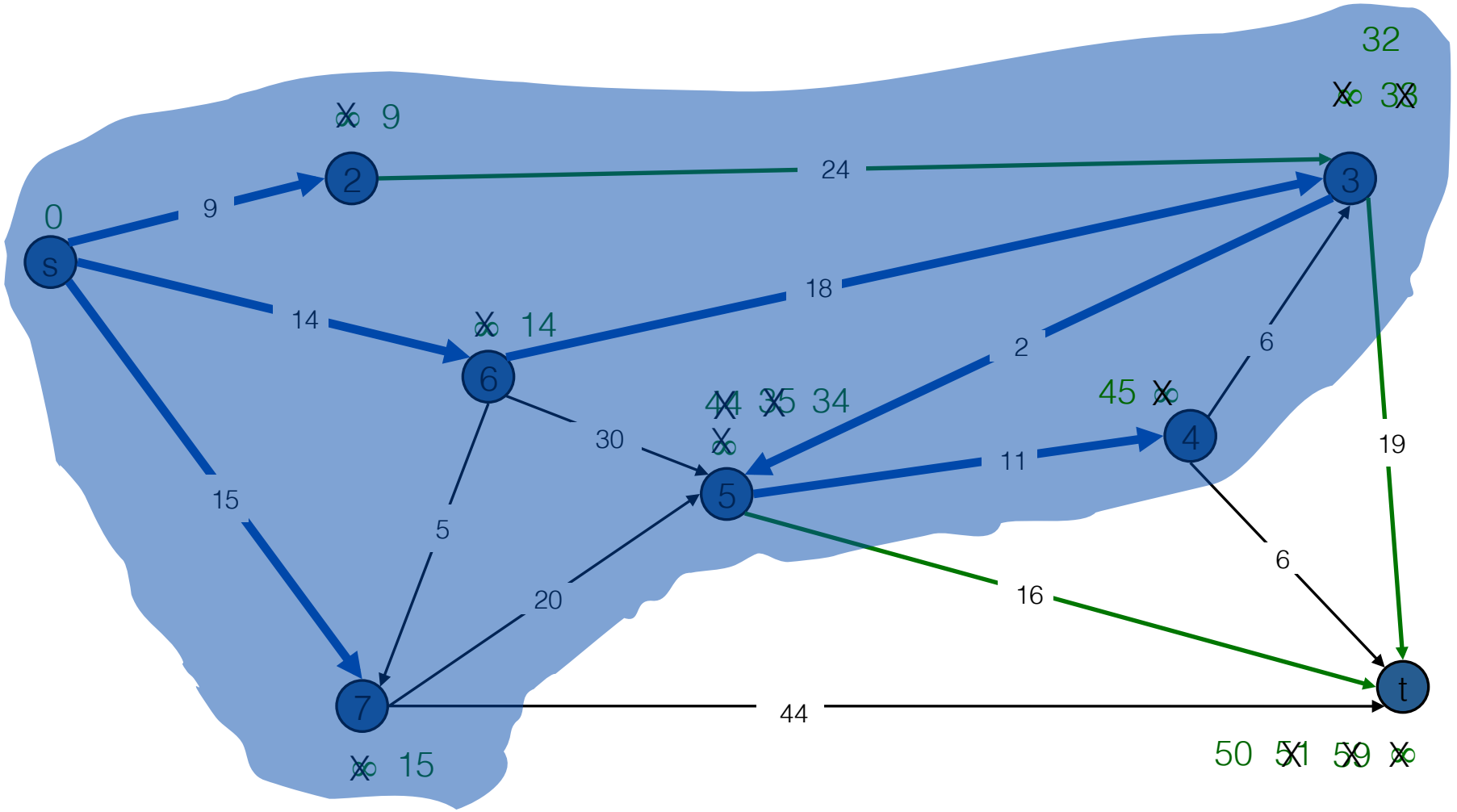
Visited = { s, 2, 3, 5, 6, 7 }

Unvisited = { 4, t }



Dijkstra's Shortest Path Algorithm

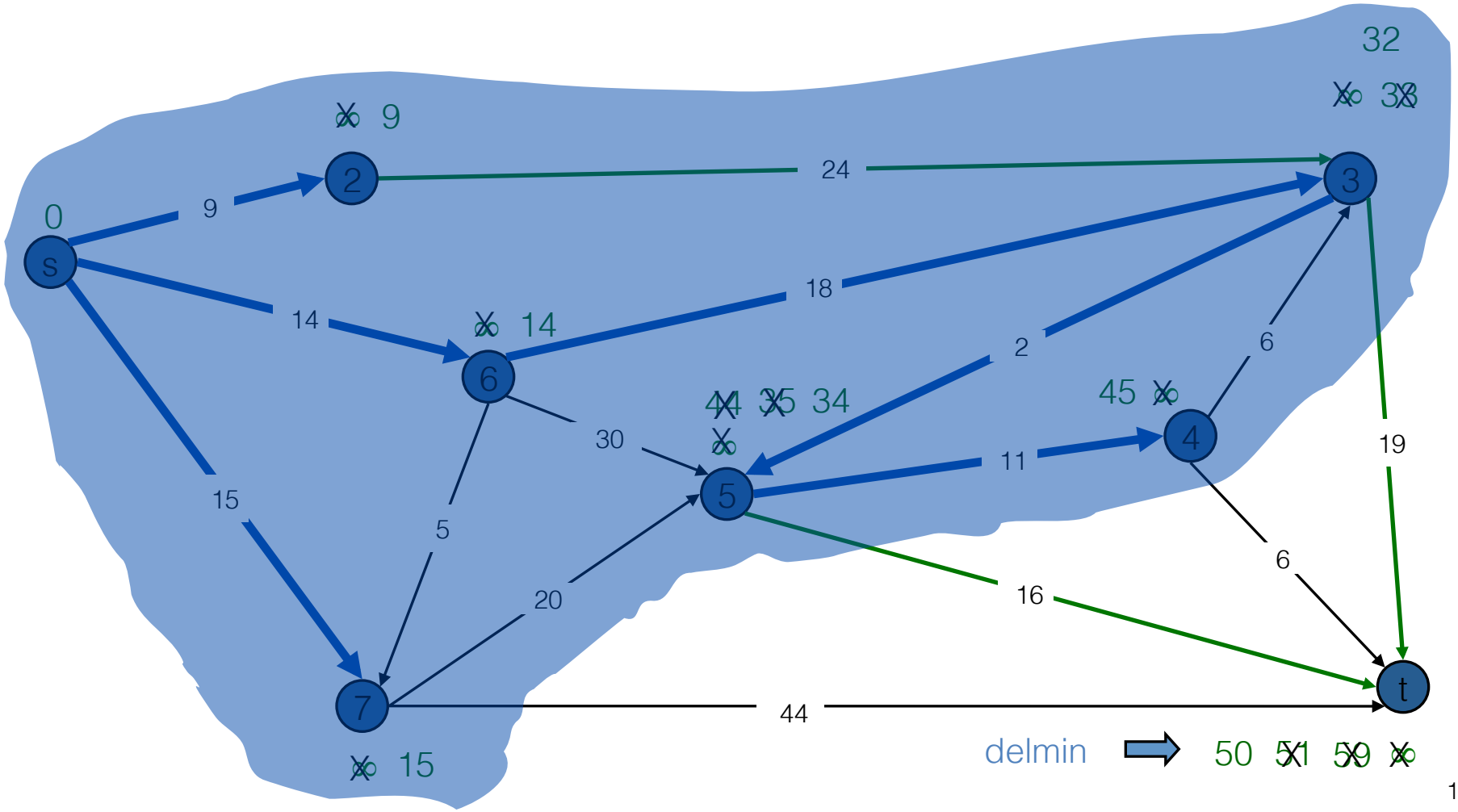
Visited = { s, 2, 3, 4, 5, 6, 7 }
Unvisited = { t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 3, 4, 5, 6, 7 }

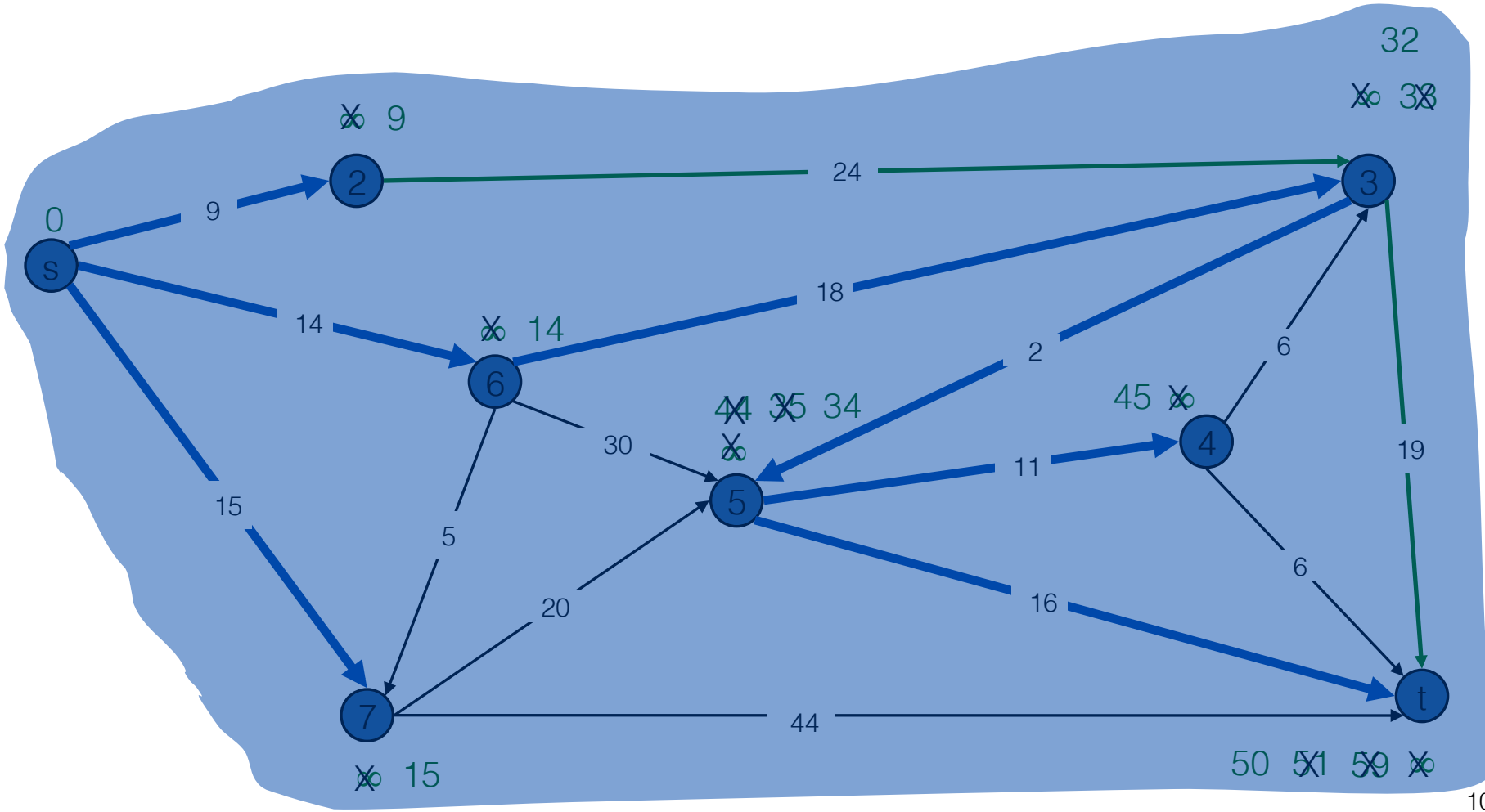
Unvisited = { t }



Dijkstra's Shortest Path Algorithm

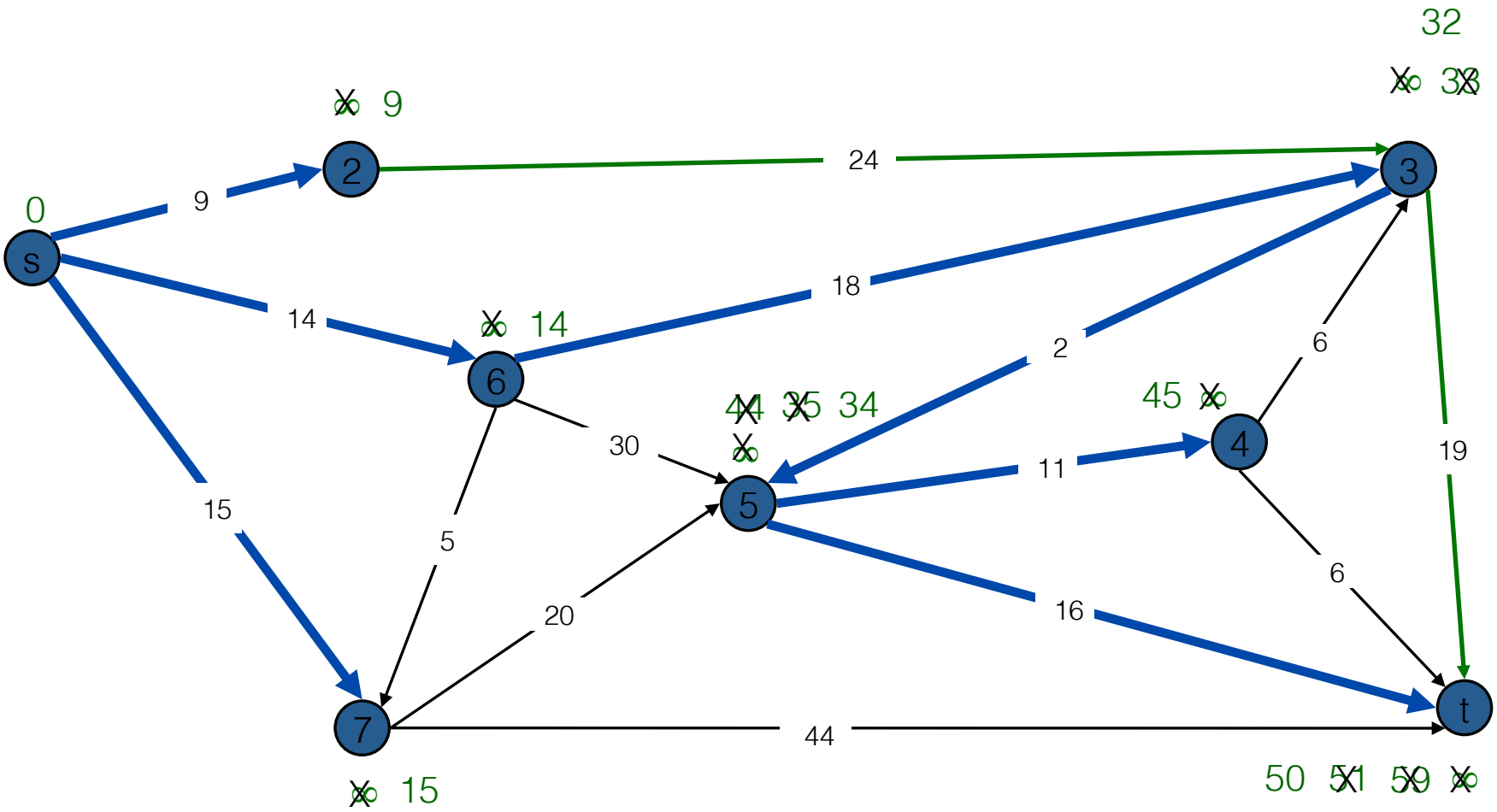
Visited = { s, 2, 3, 4, 5, 6, 7, t }

Unvisited = { }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 3, 4, 5, 6, 7, t }
Unvisited = { }



Basics in Linear Algebra

Eigenvalues and Eigenvectors

- Definition: Let A be a symmetric $n \times n$ matrix. A vector u is defined as eigenvector of A if and only if $A\,u = \lambda\,u$, where λ is a scalar called eigenvalue corresponding to u
  - In other words, λ is an eigenvalue of A iff the equation

    $$(A - \lambda I)\,u = 0$$

    has a non trivial solution

- Then, A can be written as

$$\mathbf{A} = \mathbf{U}\ \mathbf{\Lambda}\ \mathbf{U}^T$$

  - where the orthogonal matrix U contains as columns the eigenvectors $u_1, \ldots, u_n$ that correspond to $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$
  - $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$

Singular Value Decomposition (SVD)

- Definition: The SVD of a real matrix $A \in R^{m \times n}$ is defined as

$$\mathbf{A} = \mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V}^T$$

  - *U* is the *m x m* orthogonal matrix that contains the left-singular vectors of *A*
  - *V* is the *n x n* orthogonal matrix that contains the right-singular vector of *A*
  - *Σ* is the *m x n* diagonal matrix with nonnegative entries comprised of the singular values σ_i sorted from high to low
  - Note: for symmetric matrices, the singular values correspond to the absolute values of the eigenvalues

- Rank *r* of a matrix *A*
  - Number of linearly independent rows or columns
  - $r \leq min\{m,n\}$
  - *r* = number of nonzero singular values σ_i $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$

Orthogonal matrix:
$Q^TQ = QQ^T = I$

Low Rank Approximation with SVD

- How to approximate matrix $A_{m \times n}$ with a rank-k matrix ($k << n$)?
  - Meaning: All the rows are linear combinations of a set of merely k rows (same for columns)
  - Super important: dimensionality reduction (e.g., PCA)
  - Choosing a rank-k matrix boils down to choosing sets of k vectors
  - Do we have any way to do that? Yes, SVD does exactly that!

- Compute SVD of $A = U \Sigma V^T$
  - Keep only the top k left singular vectors of U (first k columns): U_k ($m \times k$ matrix)
  - Keep only the top k right singular vectors of V^T (first k rows): V_k^T ($k \times n$ matrix)
  - Keep only the top k singular values: set Σ_k to be the first k rows/columns of Σ ($k \times k$ diagonal matrix) – k largest singular values

$$\mathbf{A}_k = \mathbf{U}_k \ \mathbf{\Sigma}_k \ \mathbf{V}_k^T$$

The Frobenious norm $|| A - A_k ||_F$ is minimized

$$\| \mathbf{A} \|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2}$$

How do we measure and characterize a network?

More properties

Key Network Properties

Degree distribution: $P(k)$

Path length: h

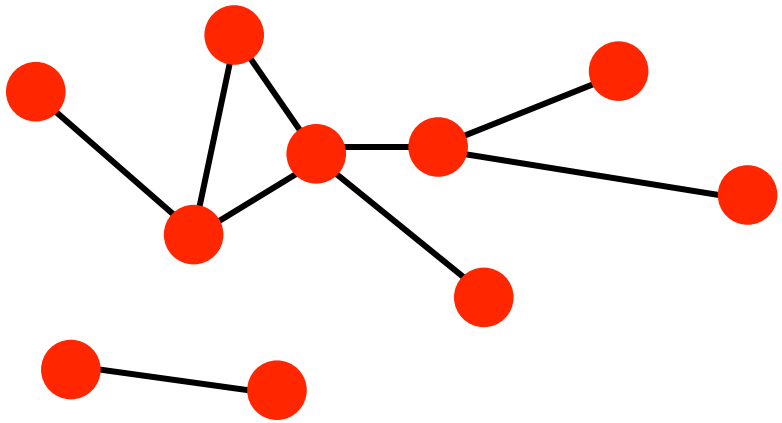
Clustering coefficient: C

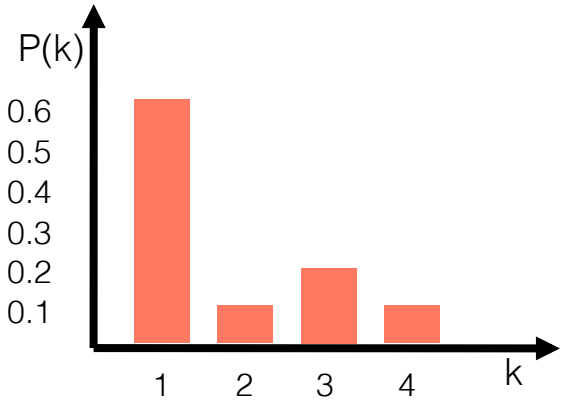
Degree Distribution

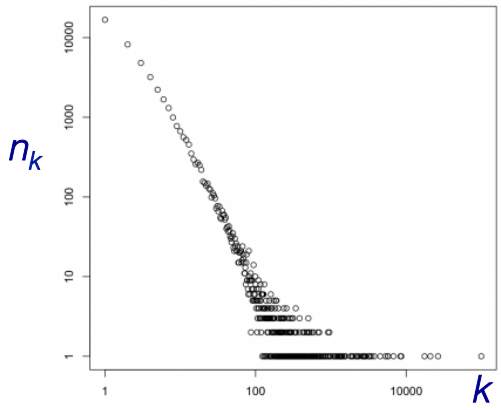
- Degree distribution *P(k)*: Probability that a randomly chosen node has degree *k*

  n_k = # nodes with degree *k*

- Normalized histogram:
  $P(k) = n_k / n$ ➔ plot







Number of Paths

Property: Let A^h denote the h-th power of A, with entries A^h_{uv}

- Then, element A^h_{uv} captures the number of $u - v$ paths of length h in G

  - # of paths of length $h=1$: If there is a link between u and v, $A_{uv}=1$ else $A_{uv}=0$

  - # of paths of length $h=2$: If there is a path of length two between u and v through k, the product $A_{uk} A_{kv}=1$ else $A_{uk} A_{kv}=0$

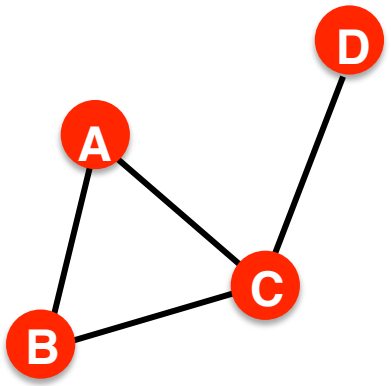
$$H^{(2)}_{uv} = \sum_{k=1}^{n} \mathbf{A}_{uk}\mathbf{A}_{kv} = \mathbf{A}^2_{uv}$$

  - # of paths of length h: If there is a path of length h between u and v then A_{uk} .... $A_{kv}=1$ else A_{uk} .... $A_{kv}=0$

$$H^{(h)}_{uv} = \mathbf{A}^h_{uv}$$

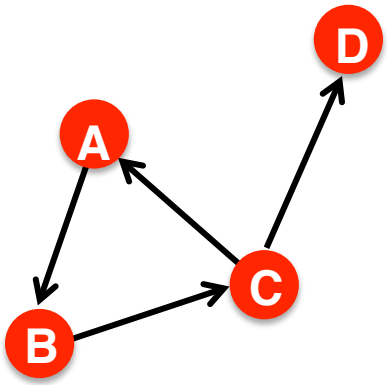
(holds for both directed and undirected graphs)

Distance in a Graph



$h_{B,D} = 2$

- Distance (shortest path, geodesic) between a pair of nodes is defined as the number of edges along the shortest path connecting the nodes
  - If the two nodes are disconnected, the distance is usually defined as infinite



$h_{B,C} = 1, h_{C,B} = 2$

- In directed graphs paths need to follow the direction of the arrows
  - Consequence: Distance is **not symmetric**: $h_{A,C} \neq h_{C,A}$

Network Diameter

- Diameter: the maximum (shortest path) distance between any pair of nodes in a graph

- Average path length for a connected graph (component) or a strongly connected (component of a) directed graph
  - Many times we compute the average only over the connected pairs of nodes (that is, we ignore "infinite" length paths)

$$\bar{h} = \frac{1}{n(n-1)} \sum_{i,j \neq i} h_{ij}$$

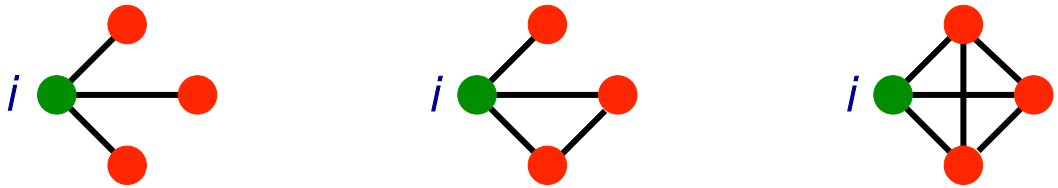
where h_{ij} is the distance from node i to node j

Clustering Coefficient

- Clustering coefficient
  - What portion of node i's neighbors are connected?
  - Node *i* with degree k_i
  - $C_i \in [0,1]$

$$C_i = \frac{2e_i}{k_i(k_i-1)}$$

where e_i is the number of edges between the neighbors of node *i*



C_i=0 C_i=1/3 C_i=1

Average clustering coefficient: $C = \frac{1}{|V|}\sum_{i=1}^{|V|} C_i$

Key Network Properties

Degree distribution:	*P(k)*
Path length:	*h*
Clustering coefficient:	*C*

They can inform us about the structure of a network

What is happening in real networks?

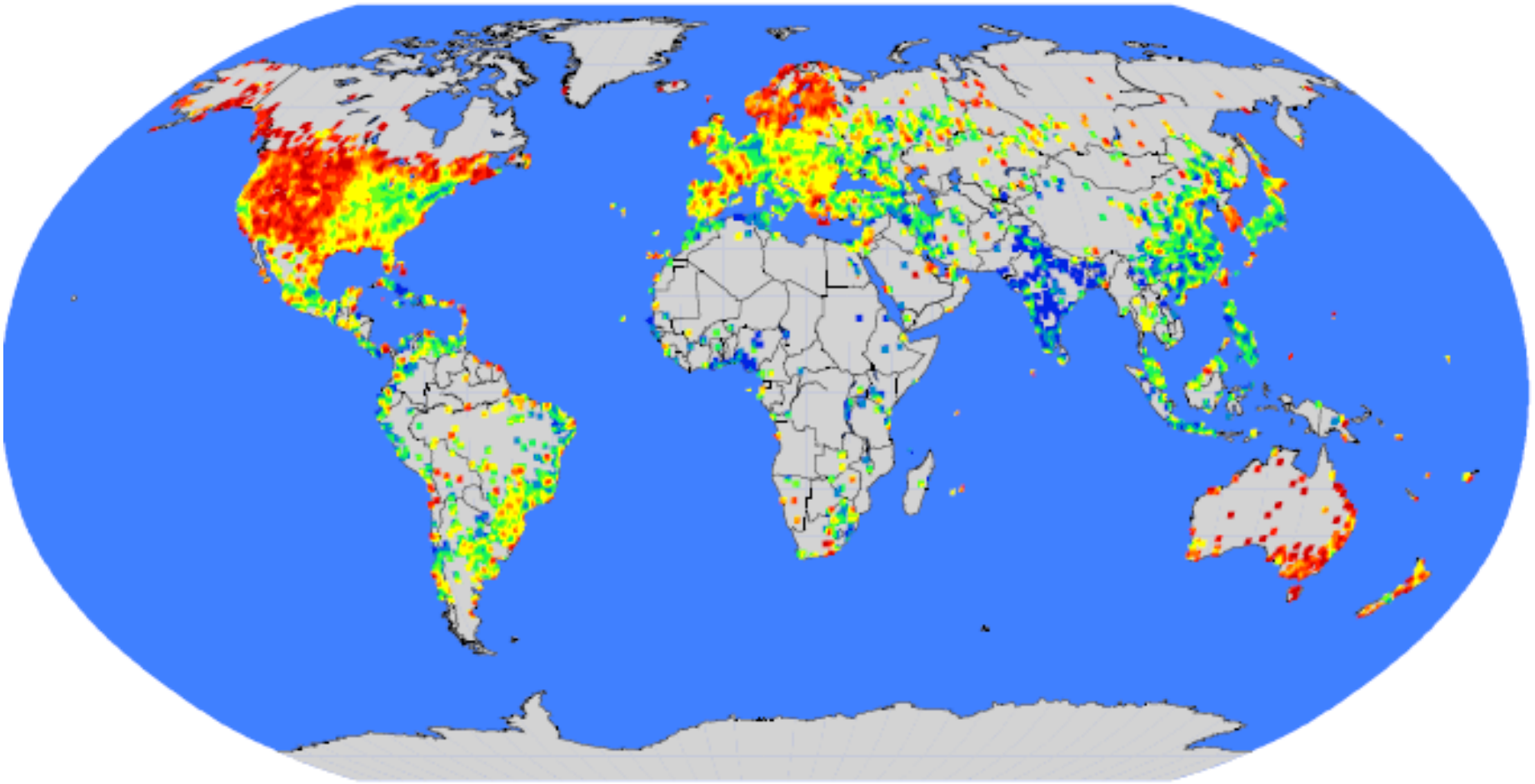
Let’s see the properties of a real network:

The MSN messenger case

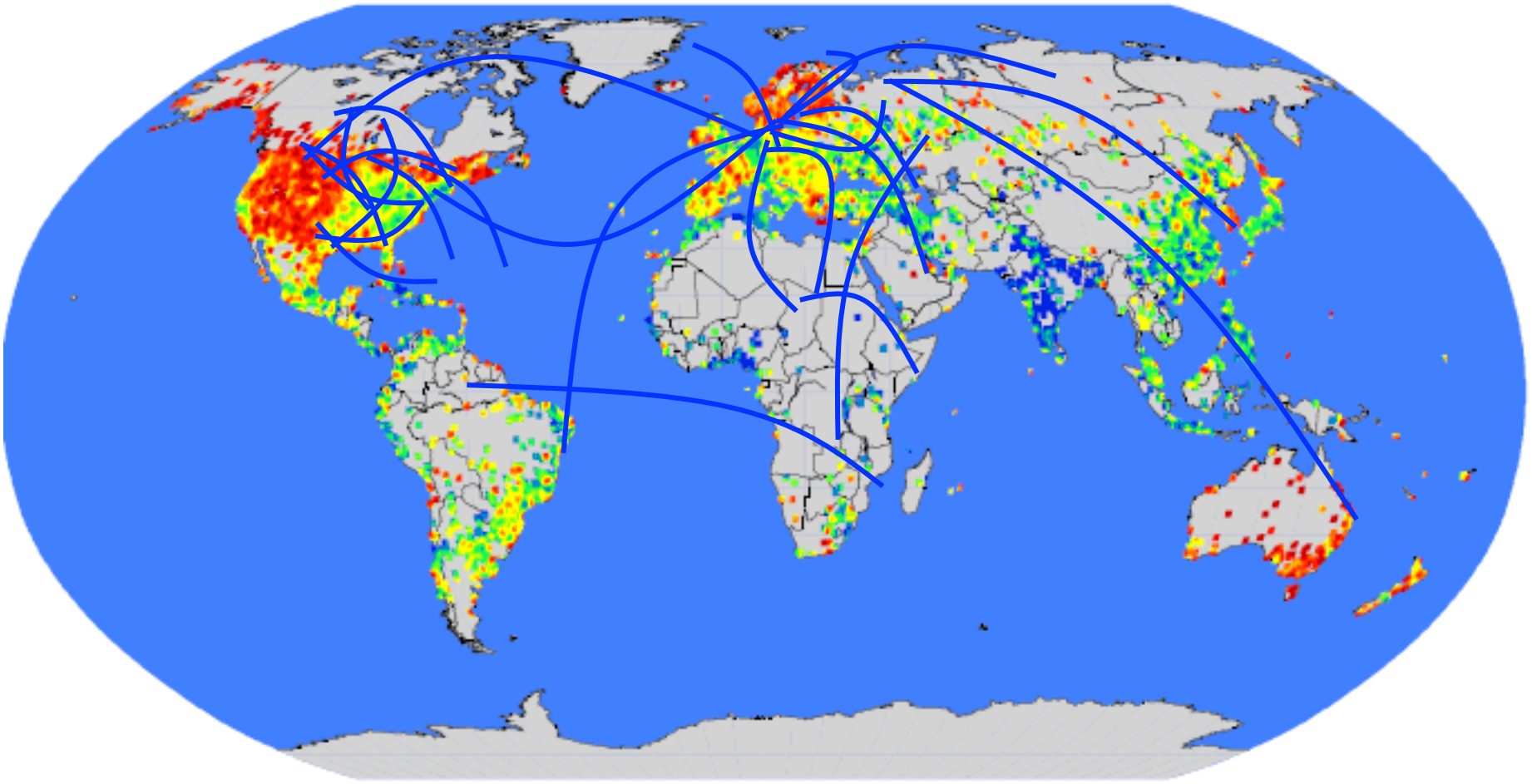
The MSN Messenger

- MSN Messenger activity in June 2006:
  - 245 million users logged in
  - 180 million users engaged in conversations
  - More than 30 billion conversations
  - More than 255 billion exchanged messages
  - Now called *Windows Live Messenger*

Communication: Geography

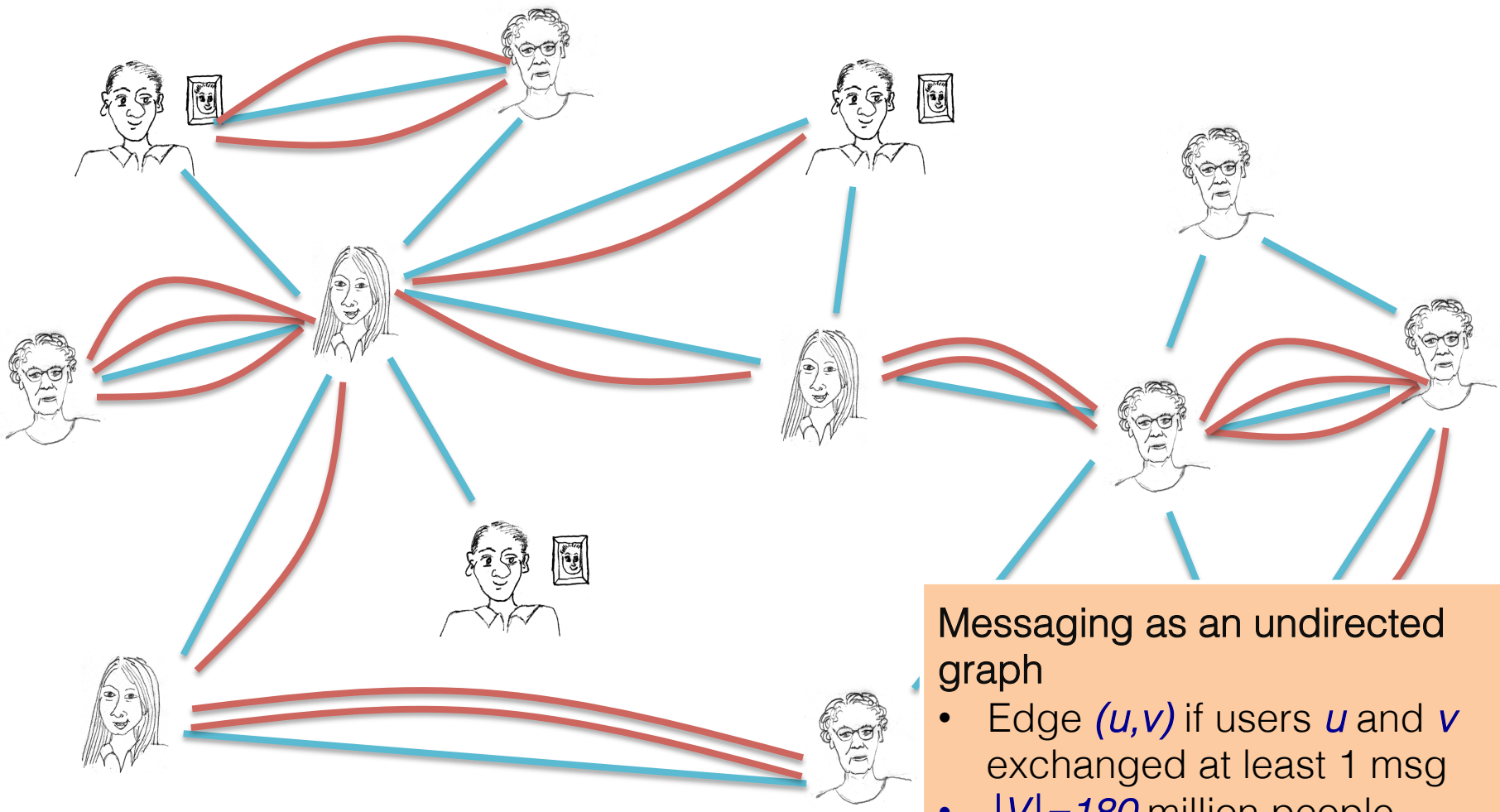


Communication network



Network: 180M people, 1.3B edges

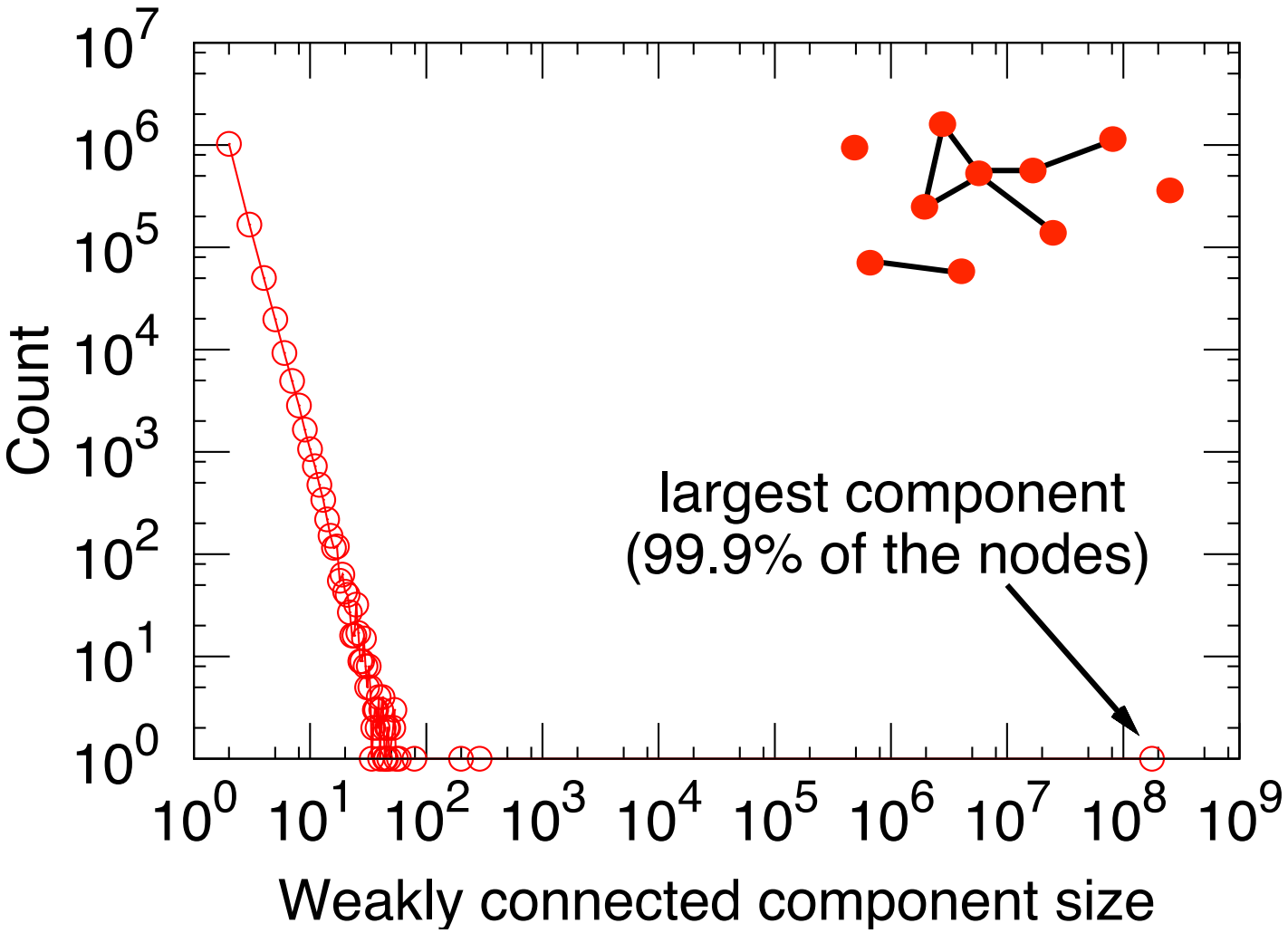
Messaging as a Multigraph



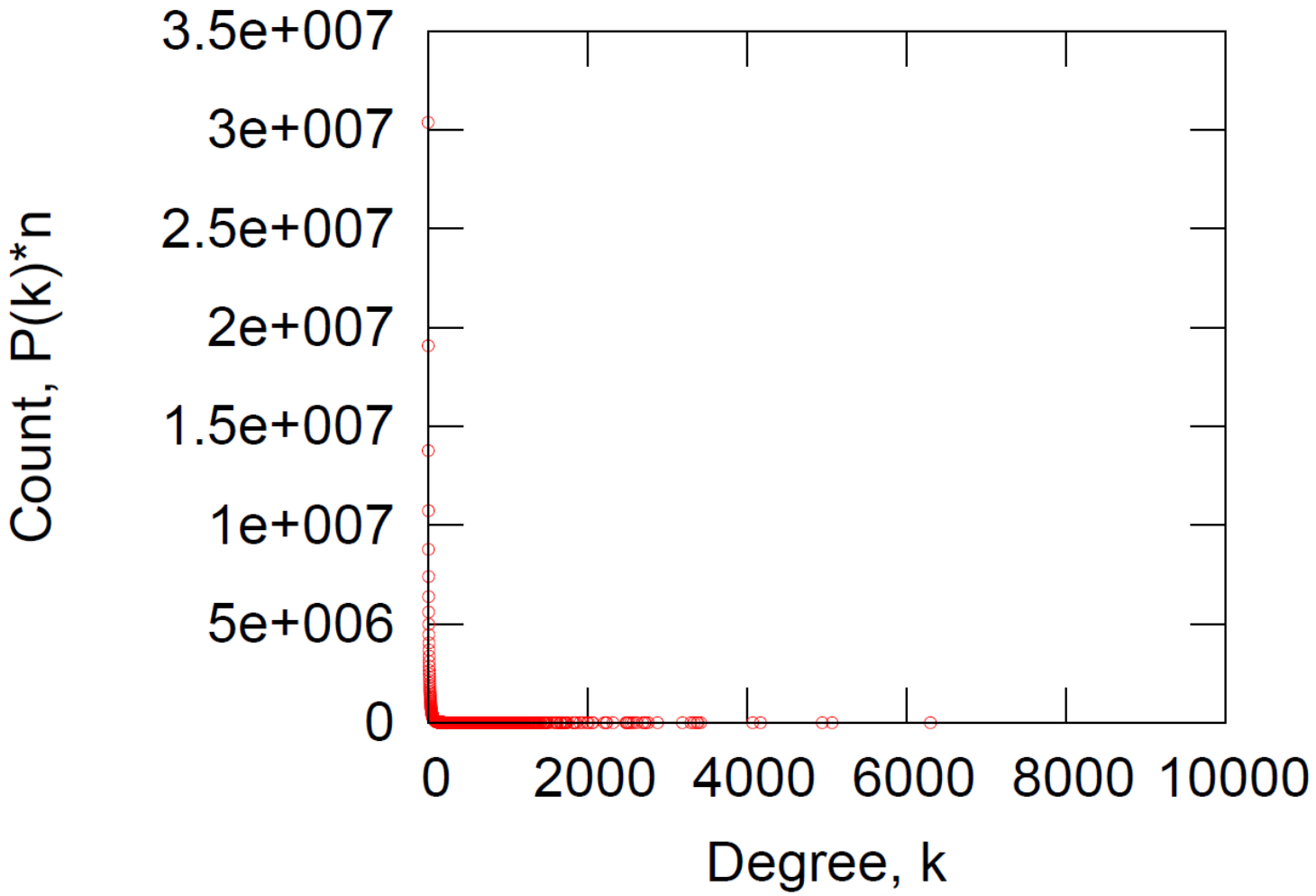
Messaging as an undirected graph

- Edge *(u,v)* if users *u* and *v* exchanged at least 1 msg
- $|V|=180$ million people
- $|E|=1.3$ billion edges

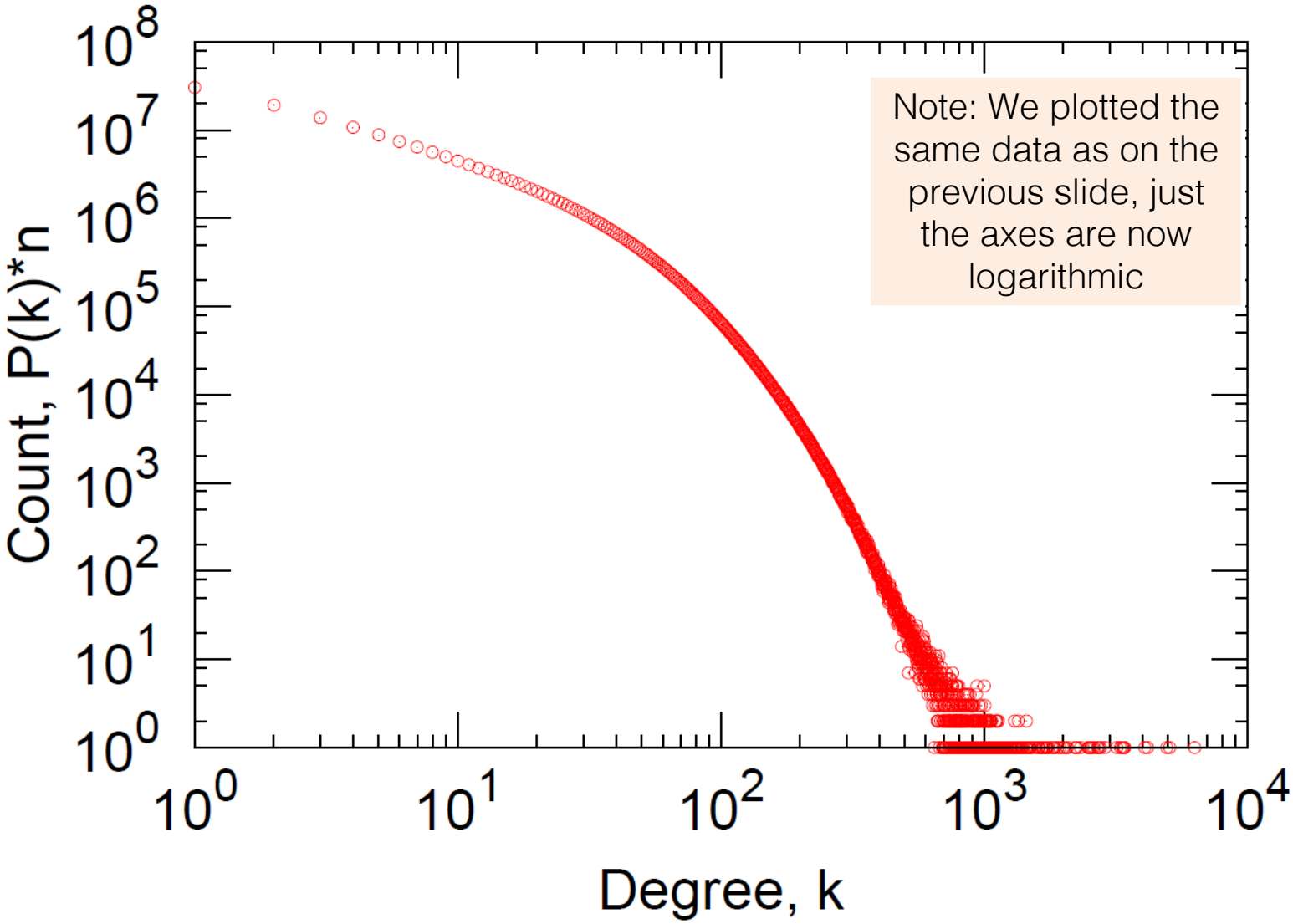
MSN: Connectivity



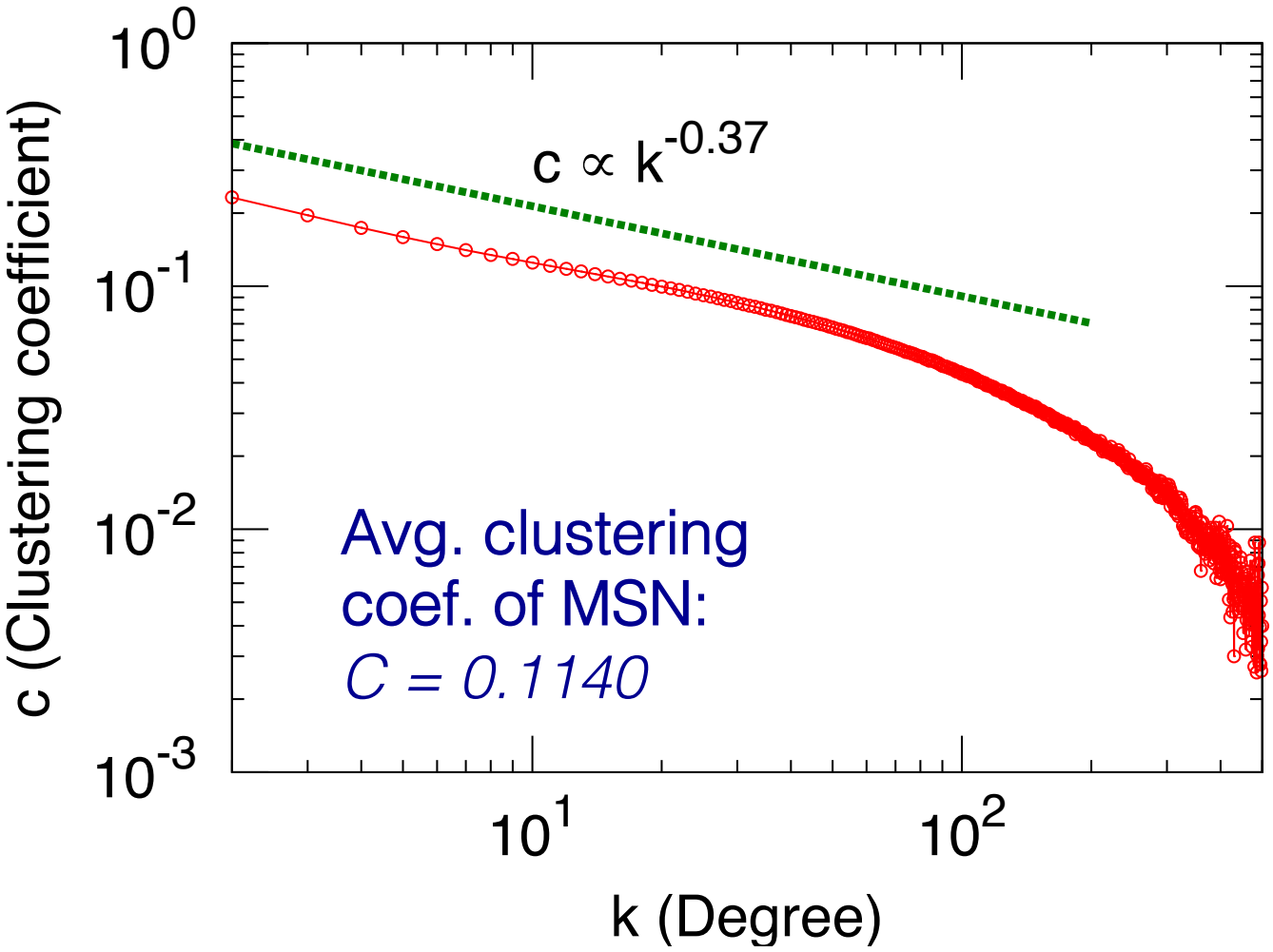
MSN: Degree Distribution



MSN: Log-Log Degree Distribution

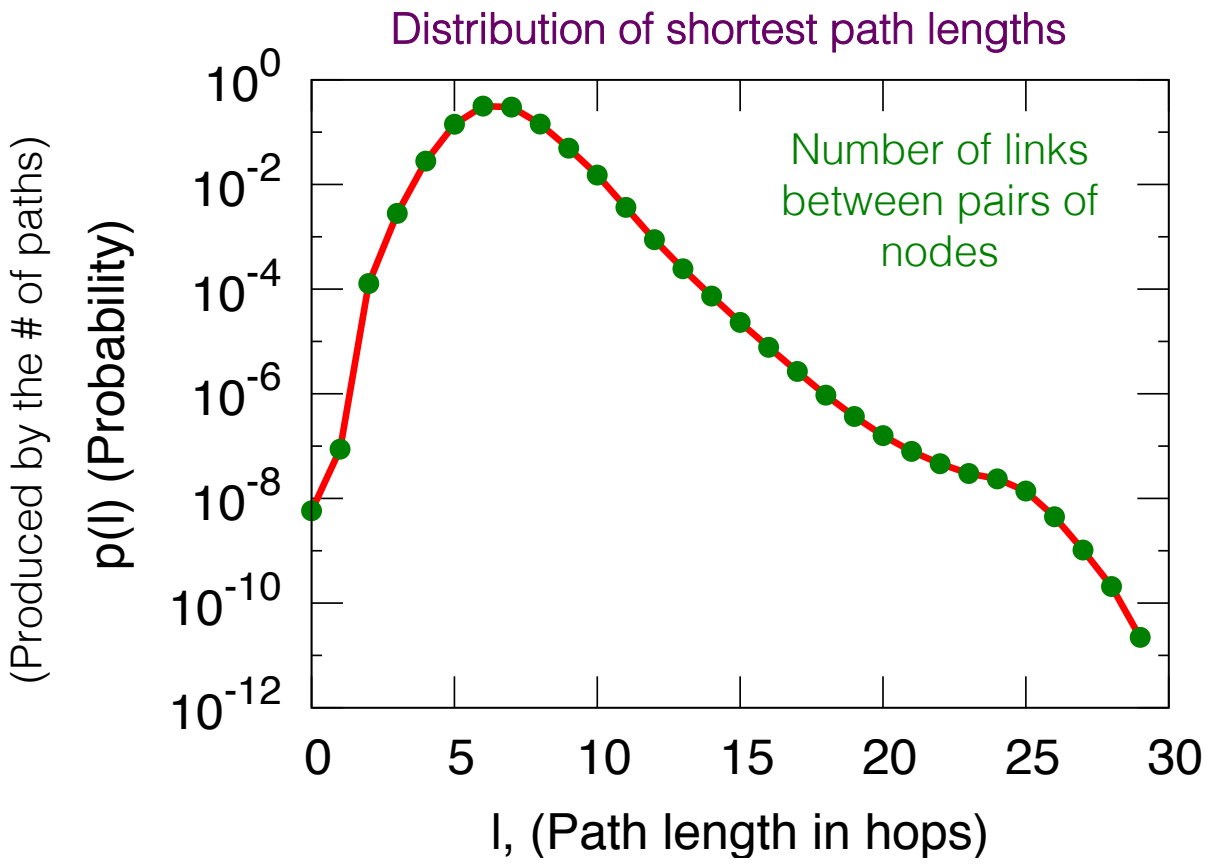


MSN: Clustering



C_k: average C_i of nodes i of degree k: $$C_k = \frac{1}{n_k} \sum_{i:k_i=k}^{n} C_i$$

MSN: Diameter



Avg. path length **6.6**
90% of the nodes can be reached in < 8 hops

nodes as we do BFS out of a random node

Steps	#Nodes
0	1
1	10
2	78
3	3,96
4	8,648
5	3,299,252
6	28,395,849
7	79,059,497
8	52,995,778
9	10,321,008
10	1,955,007
11	518,410
12	149,945
13	44,616
14	13,740
15	4,476
16	1,542
17	536
18	167
19	71
20	29
21	16
22	10
23	3
24	2
25	3

MSN: Key Network Properties

Degree distribution:	*Heavily skewed* *avg. degree= 14.4*
Path length:	*6.6*
Clustering coefficient:	*0.11*

What is the impact of those measurements in information propagation?

Next Lecture

- Centrality criteria on graphs
  - How to determine the importance of a node
  - Important nodes ≈ influential nodes

Thank You!