

Network Science Analytics

Option Applied Math and M.Sc. in DSBA

Lecture 1B

Graph theory and linear recap; basic network properties

Fragkiskos Malliaros

Friday, January 19, 2018

Acknowledgements

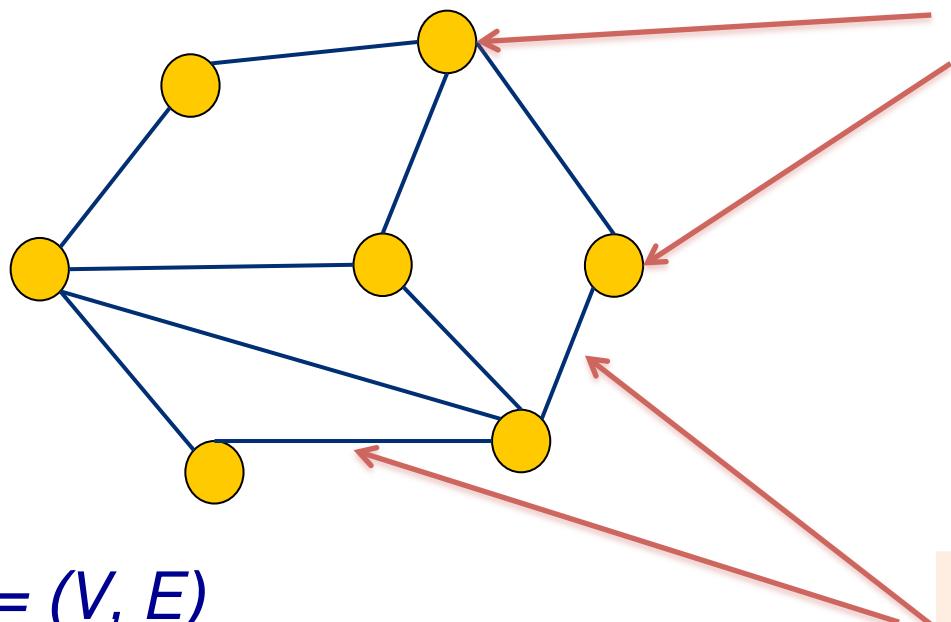
- Part of the lecture is based on material by
 - Jure Leskovec, Stanford University
 - Manos Papaggelis, York University
 - Gonzalo Mateos, University of Rochester
 - Albert-László Barabási, Northeastern University
 - Christos Faloutsos, CMU
 - Danai Koutra, University of Michigan
 - R. Zafarani, M. A. Abbasi, and H. Liu, Social Media Mining: An Introduction, Cambridge University Press, 2014. Free book and slides at <http://socialmediamining.info/>

Thank you!

Basic graph-theoretic concepts and definitions

Graphs and Networks

Graphs: modeling dependencies



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

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

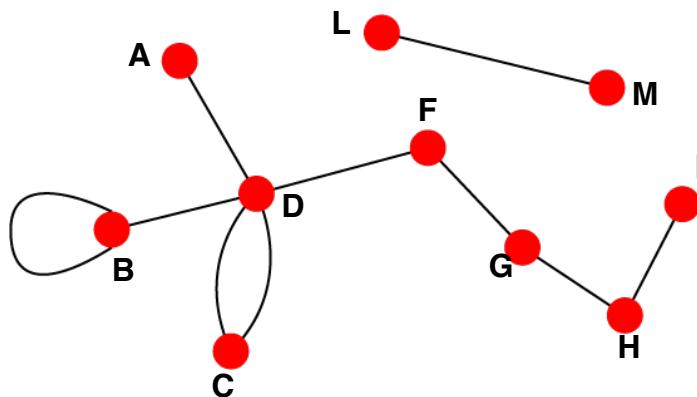
Nodes (or vertices)
(objects/entities)

Edges (or links)
(interconnections)

Undirected vs. Directed Networks

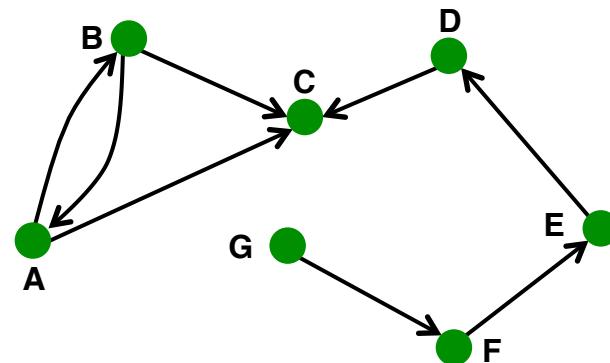
Undirected

- Links: undirected
(symmetrical, reciprocal)



Directed

- Links: directed
(arcs)



Examples

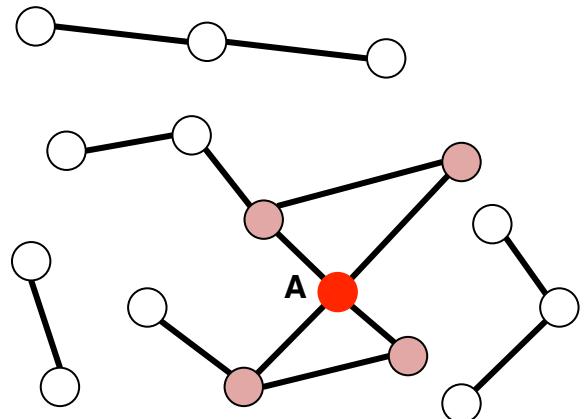
- Collaborations
- Friendship on Facebook

Examples

- Phone calls
- Following on Twitter

Node Degree

Undirected



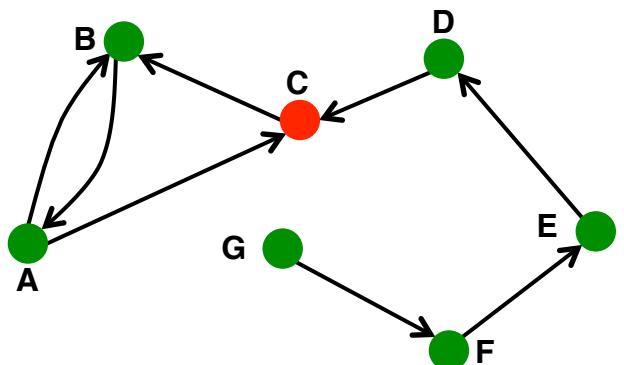
Node degree k_i : the number of edges adjacent to node i

$$k_A = 4$$

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

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

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

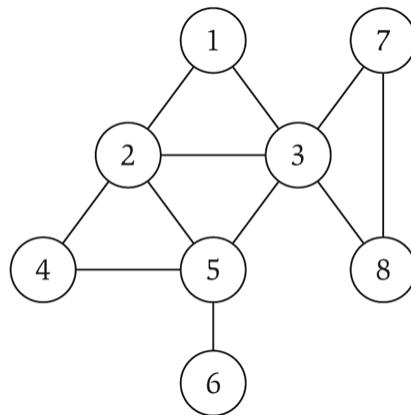
$$\sum_{i=1}^n k_i = 2 |E|$$

- **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



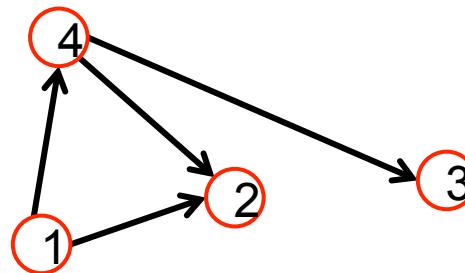
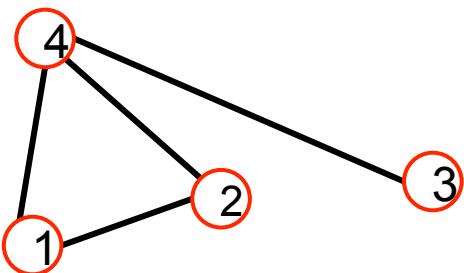
Undirected graph

More matrix representations in a while

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

Adjacency Matrix



$A_{ij} = 1$ if there is a link from node i to node j
 $A_{ij} = 0$ otherwise

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

Symmetric matrix

Undirected graph

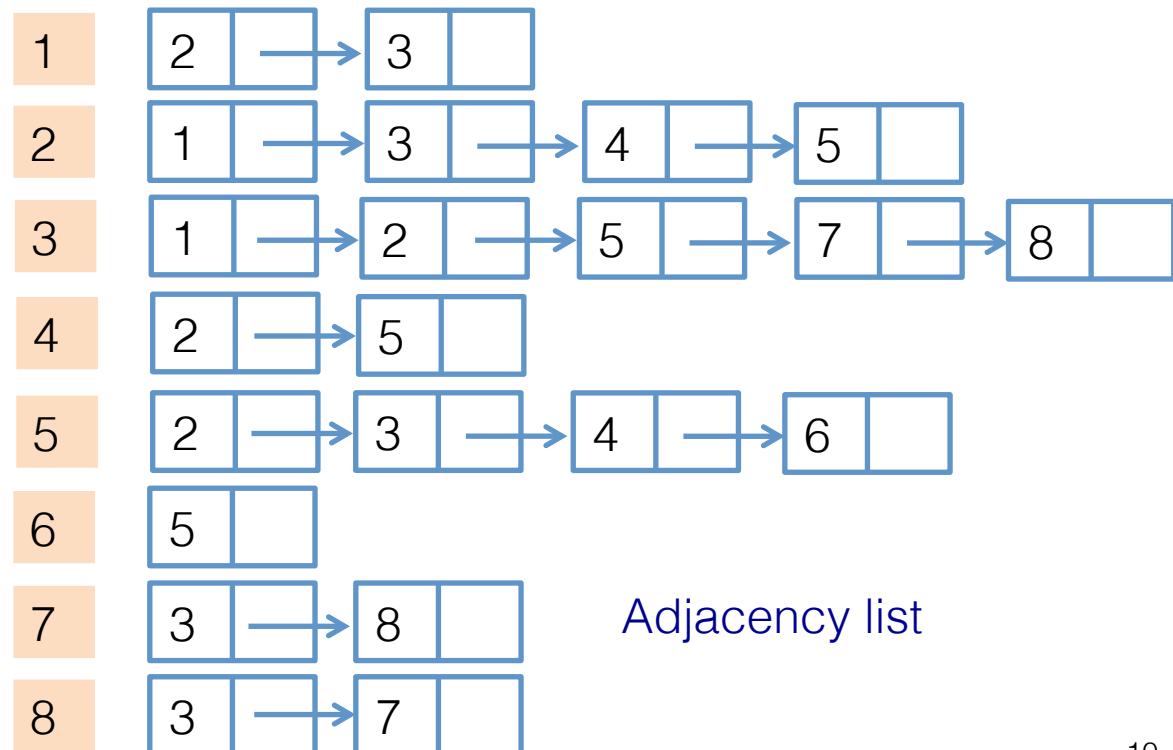
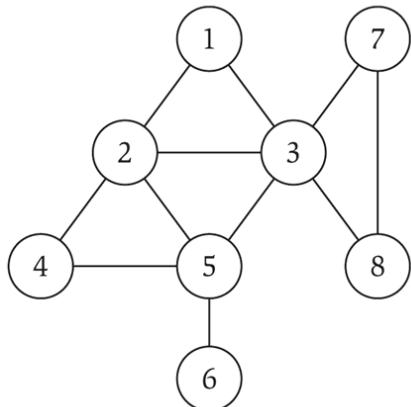
$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Nonsymmetric matrix

Directed graph

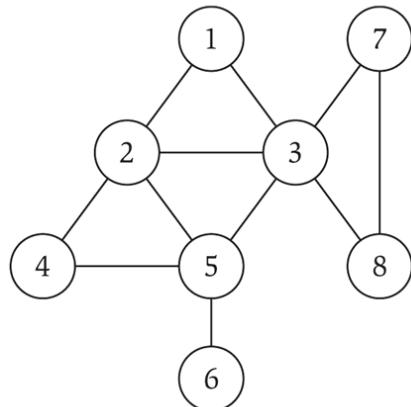
Graph Representation: Adjacency List

- 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



Graph Representation: Edge List

- Edge list
 - Very simple way to represent a graph
 - List of $|E|$ edges
 - An edge is represented as a 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)

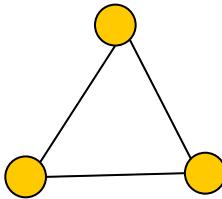


1	2
1	3
2	3
2	4
2	5
3	5
3	7
3	8
4	5
5	6
7	8

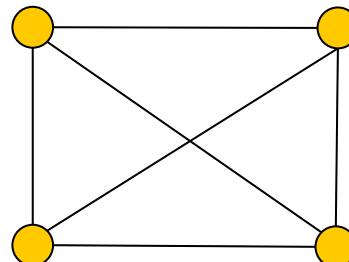
Edge list

Complete Graph

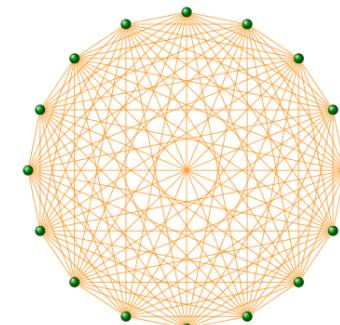
- **Definition:** A graph $G = (V, E)$ is called complete K_n if every pair of nodes is connected by an edge



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**: $|E| \ll |E_{max}|$ (or avg. $k \ll |V|-1$)

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:

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

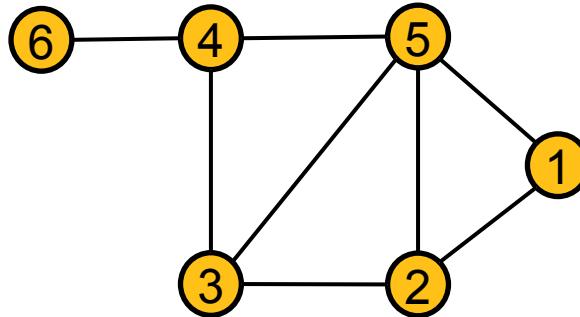
Density of the graph:

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

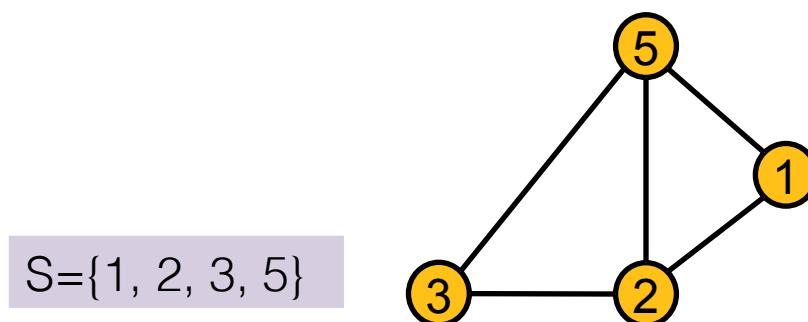
Source: Leskovec et al., Internet Mathematics, 2009

Subgraphs

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

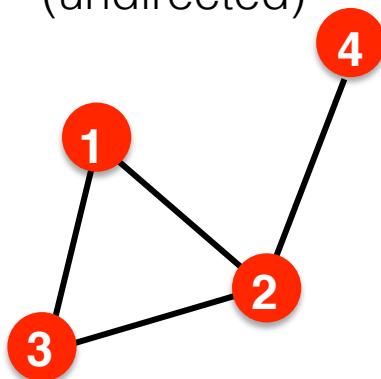


- Definition:** The induced subgraph $G[S] = (S, E')$ is the graph whose vertex set is S and its edge set consists of all of the edges in E that have both endpoints in S



More Types of Graphs (1/2)

Unweighted
(undirected)



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

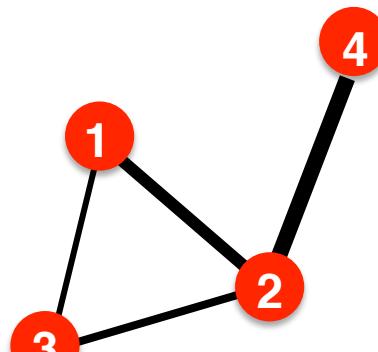
$$A_{ii} = 0$$

$$A_{ij} = A_{ji}$$

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

Examples: Friendship, Hyperlink

Weighted
(undirected)



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

$$A_{ii} = 0$$

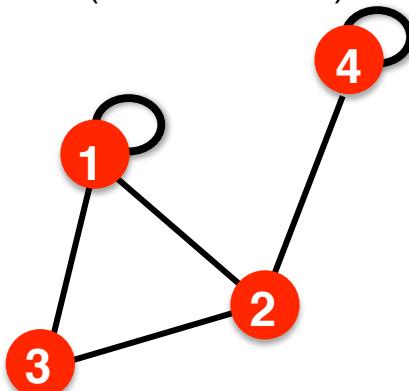
$$A_{ij} = A_{ji}$$

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

Examples: Collaboration, Internet, Roads

More Types of Graphs (2/2)

Self-edges (self-loops)
(undirected)



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

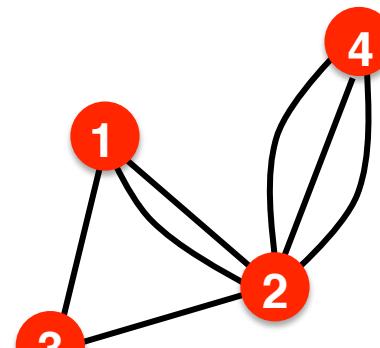
$$A_{ii} \neq 0$$

$$A_{ij} = A_{ji}$$

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

Examples: Proteins, Hyperlinks

Multigraph
(undirected)



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

$$A_{ii} = 0$$

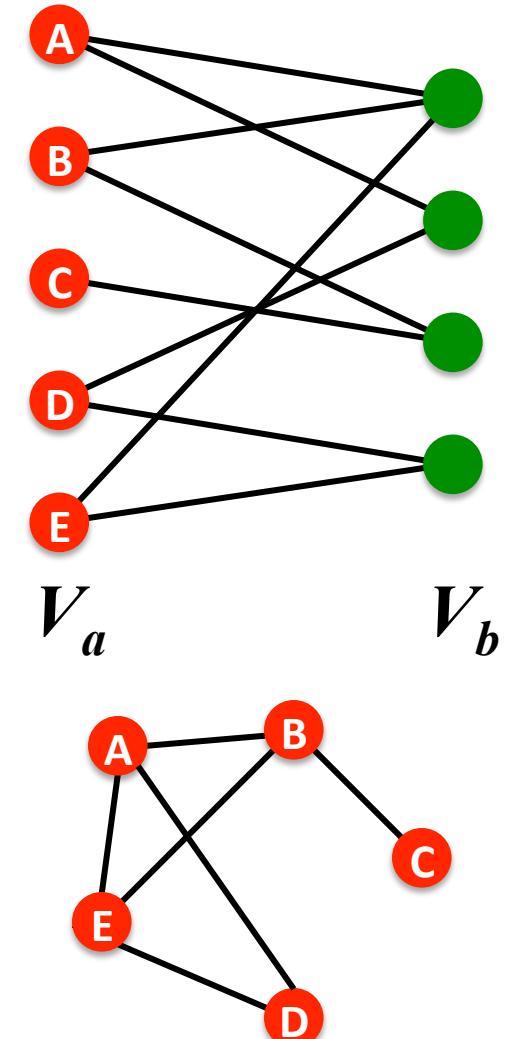
$$A_{ij} = A_{ji}$$

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

Examples: Communication, Collaboration

Bipartite Graph

- **Definition:** A graph $G=(V,E)$ is called **bipartite** if the node set V can be partitioned into two disjoint sets V_a, V_b and every edge (u,v) connects a node of V_a to a node of V_b
- **Examples**
 - Authors-to-papers (they authored)
 - Actors-to-Movies (they appeared in)
 - Users-to-Movies (they rated)
 - Customers-to-Products (they purchased)
- **Folded** (or one-mode) networks
 - Author collaboration networks
 - Movie co-rating networks



Folded version of the graph above

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!

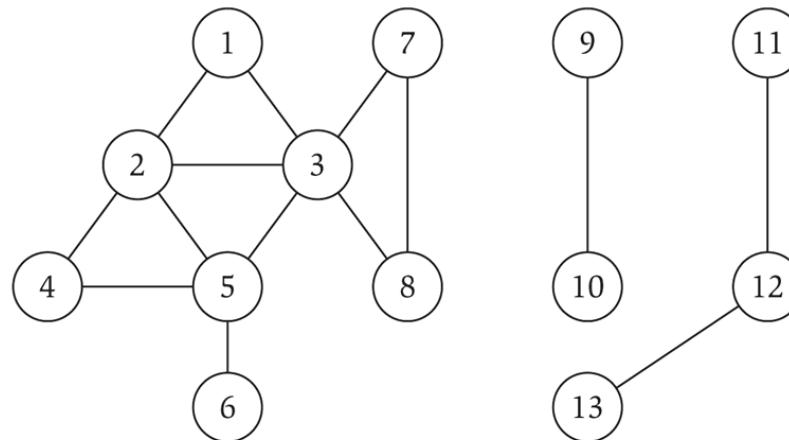
Choice of the proper network representation of a given system determines our ability to use networks successfully

Paths, graph connectivity and distance

Paths and Connectivity in Graphs

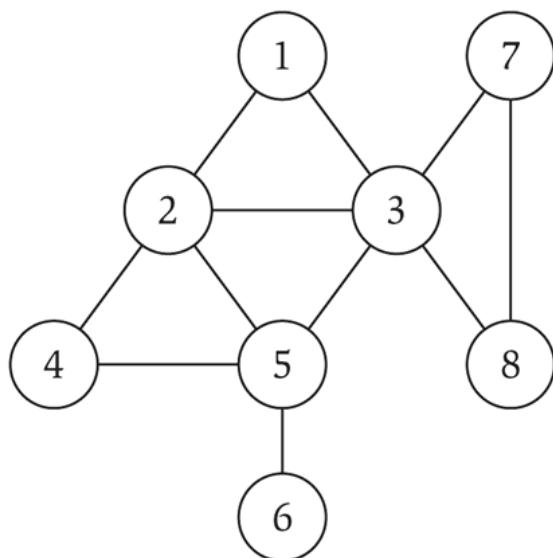
- **Definition:** A **path** in an undirected graph $G=(V,E)$ is a sequence of nodes v_1, v_2, \dots, v_k with the property that each consecutive pair v_{i-1}, v_i is joined by an edge in E
- **Definition:** An undirected graph is **connected** if for every pair of nodes u and v , there is a path between u and v

Is this graph
connected?



Cycles in Graphs

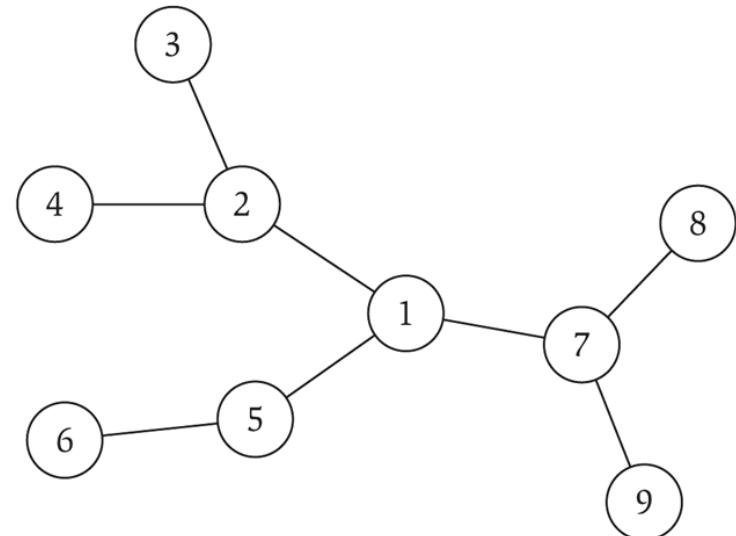
- **Definition:** A cycle is a path v_1, v_2, \dots, v_k in which $v_1 = v_k$, $k > 2$ and the first $k-1$ nodes are all distinct



Cycle C = 1 – 2 – 4 – 5 – 3 – 1

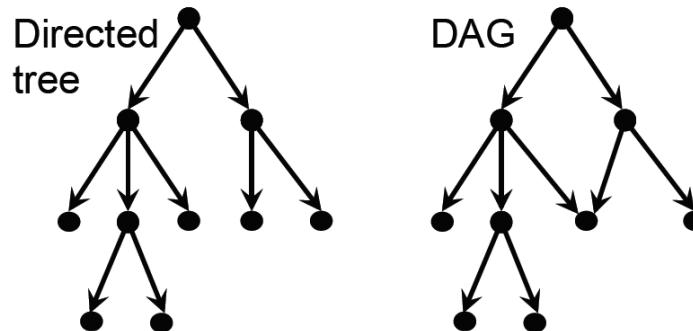
Trees

- **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

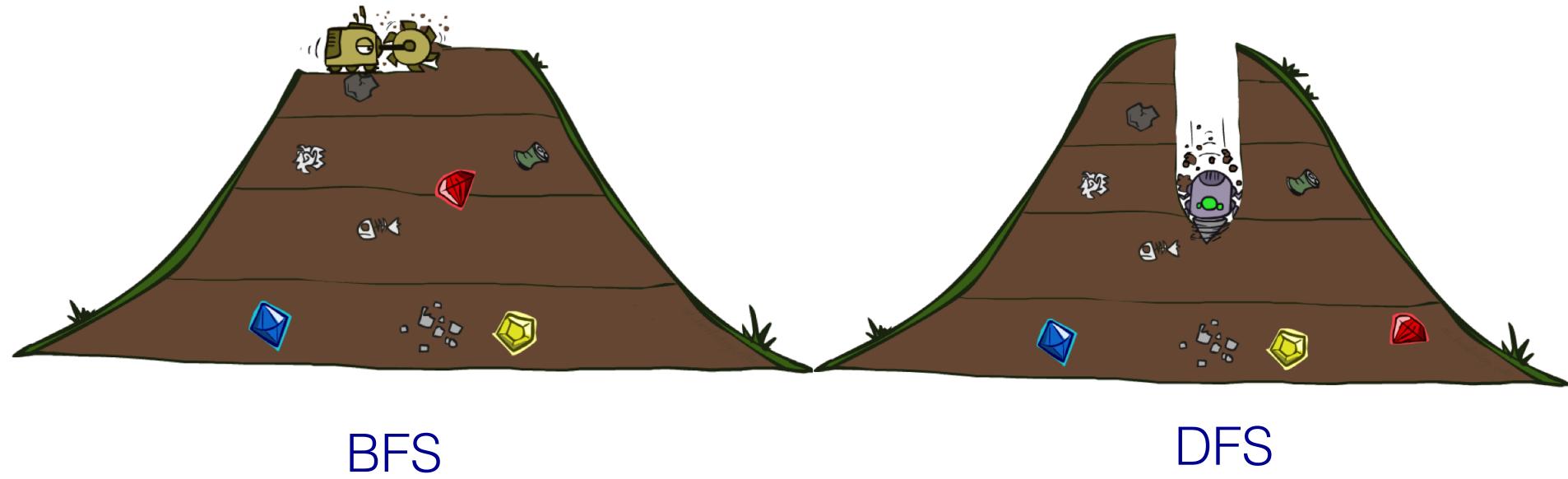
- **Definition:** A **directed tree** is a directed graph whose underlying undirected graph is a tree
 - **Root** is the only vertex with paths to all the other nodes
 - **Vertex terminology:** parent, children, ancestor, descendant, leaf
- A **directed acyclic graph (DAG)** is a directed graph with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



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  $\leftarrow$  Q.dequeue()
7         if t is what we are looking for:
8             return t
9         for all edges e in G.adjacentEdges(t) do
10            o  $\leftarrow$  G.adjacentVertex(t,e)
11            if o is not marked:
12                mark o
13                add o in Q
```

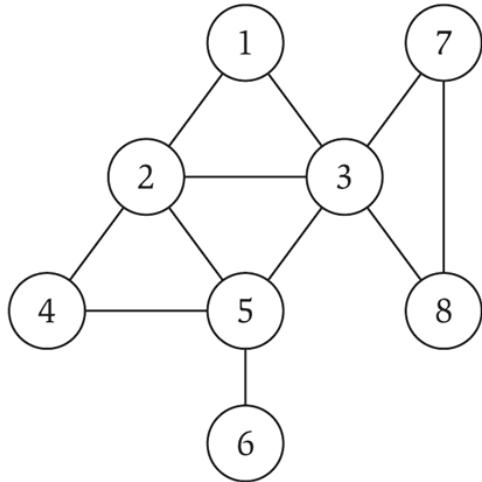
Complexity: $\mathcal{O}(|V| + |E|)$

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)

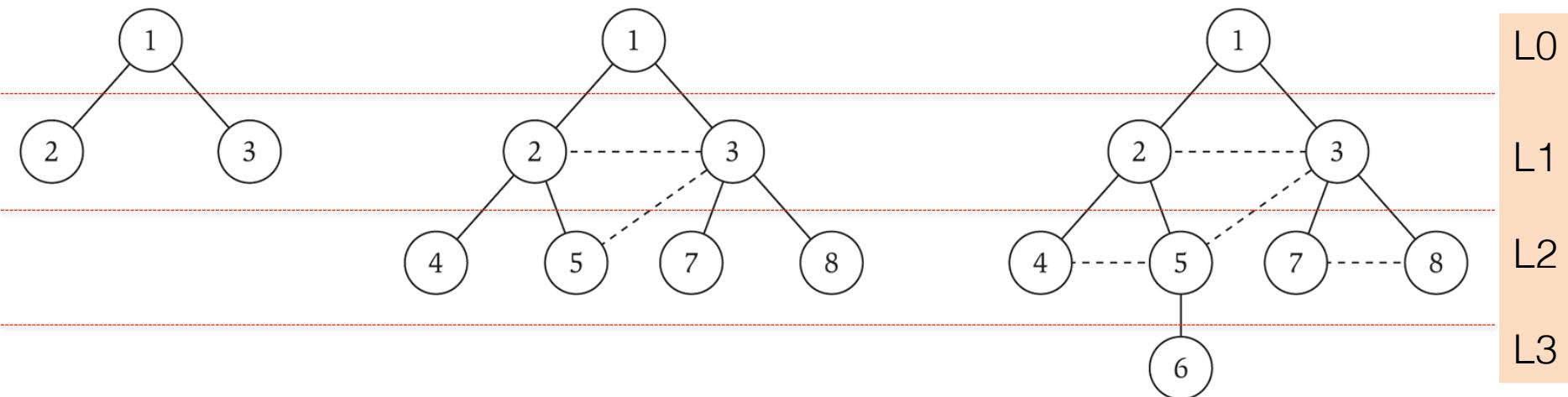
Breadth-First Search (BFS)

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)

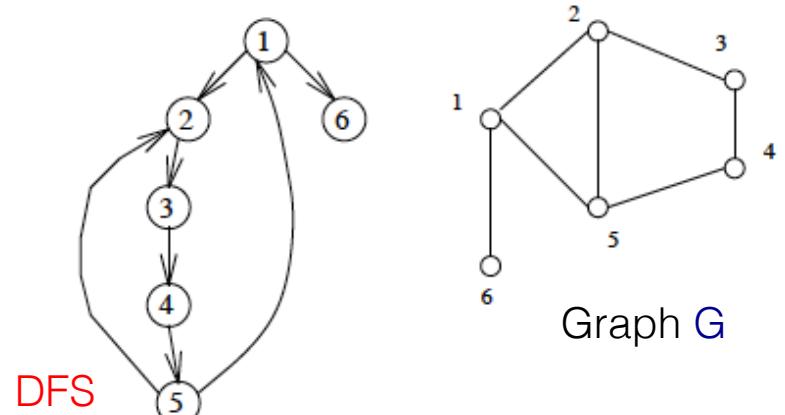
- Strategy for traversing graphs:
 - Visits the children nodes before visiting the sibling nodes
 - Traverses the depth of any particular path before exploring the breadth

```
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
```

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

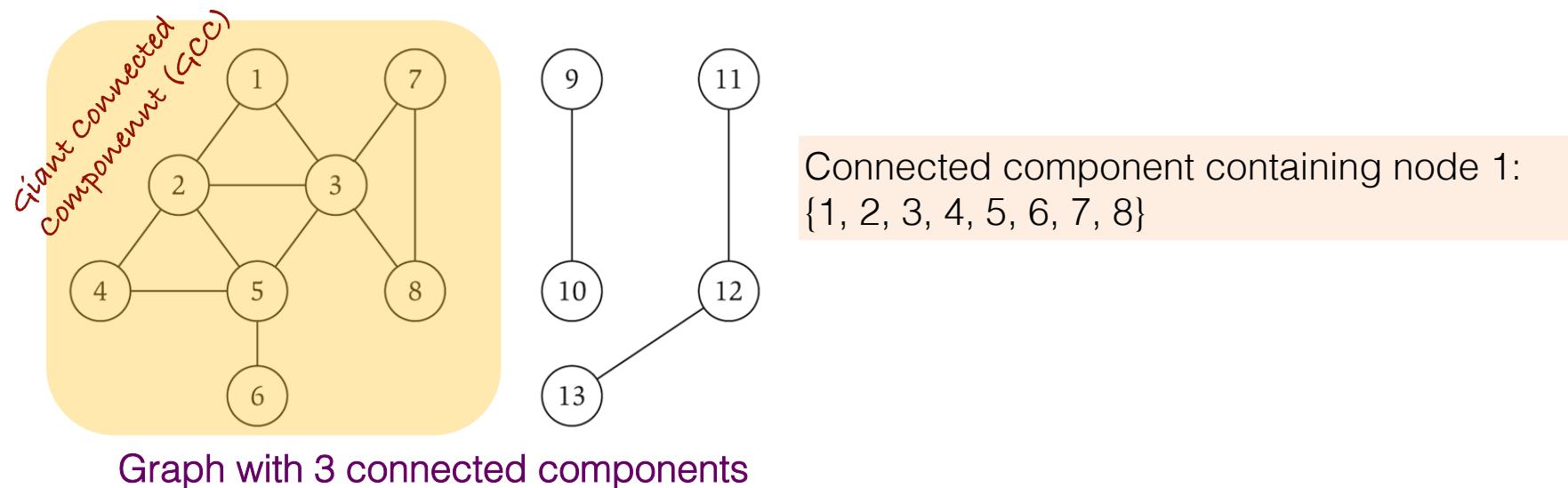
Input: Graph G and a node v

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



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

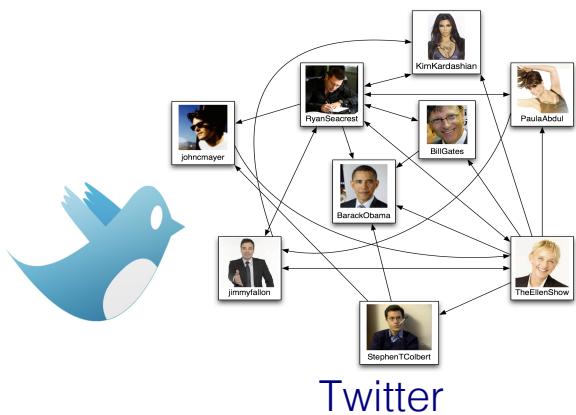


Question: How can we compute the connected components of a graph?

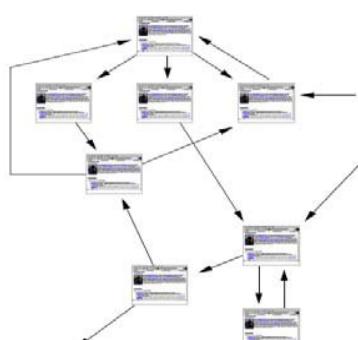
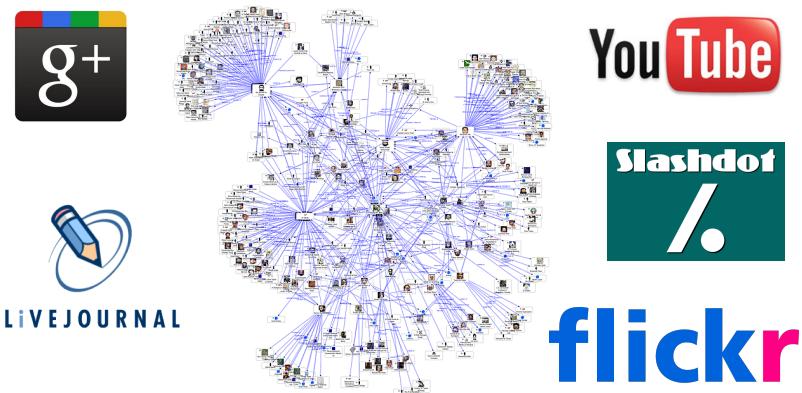
A: Apply BFS

Connectivity in Directed Graphs (1/2)

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



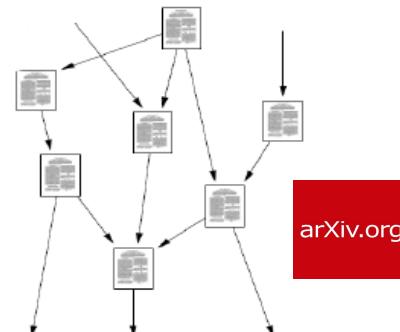
[Image: <http://sites.davidson.edu/mathmovement/>]



Web Graph



Wikipedia

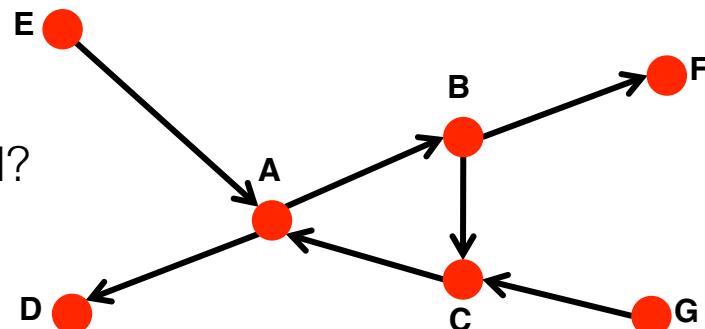


Citation Graph

Connectivity in Directed Graphs (2/2)

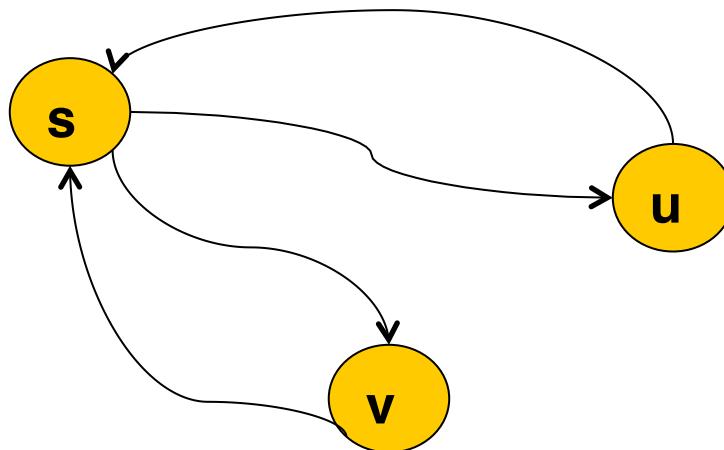
- **Directed reachability:** Given a node s in a directed graph G , find all nodes reachable from s
 - **Web crawler:** start from a webpage s , find all webpages linked from s , either directly or indirectly
- The notion of connectivity in directed graphs is replaced by the
 - **Strong connectivity:** 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?



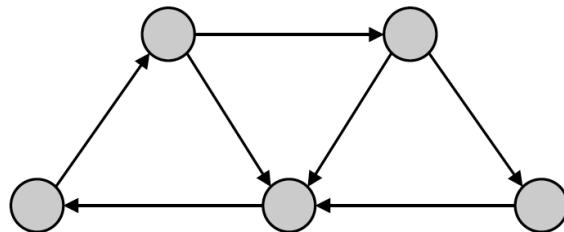
Strong Connectivity

- **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:**
 - \rightarrow Follows from definition
 - \leftarrow Path from u to v : concatenate $u \rightarrow s$ path with $s \rightarrow v$ path
Path from v to u : concatenate $v \rightarrow s$ path with $s \rightarrow u$ path

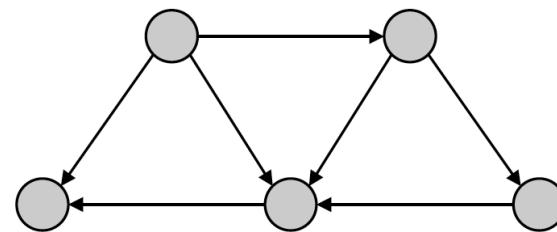


Strong Connectivity: Algorithm

- **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



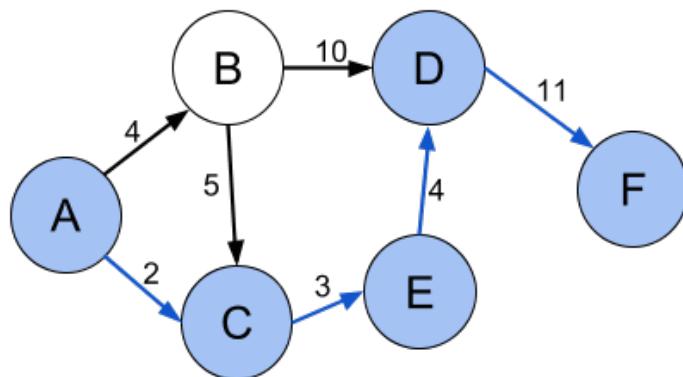
Strongly connected



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:
6          dist[v] := infinity
7          previous[v] := undefined
8          add v to Q
9
10     dist[source] := 0
11
12
13     while Q is not empty:
14         u := node in Q with smallest dist[ ]
15         remove u from Q
16
17         for each neighbor v of u:
18             alt := dist[u] + length(u, v)
19             if alt < dist[v]
20                 dist[v] := alt
21                 previous[v] := u
22
23     return dist[], previous[ ]
```

// Initialization
// initial distance from source to vertex v is set to infinite
// Previous node in optimal path from source
// All nodes initially in Q (unvisited nodes)

// Distance from source to source

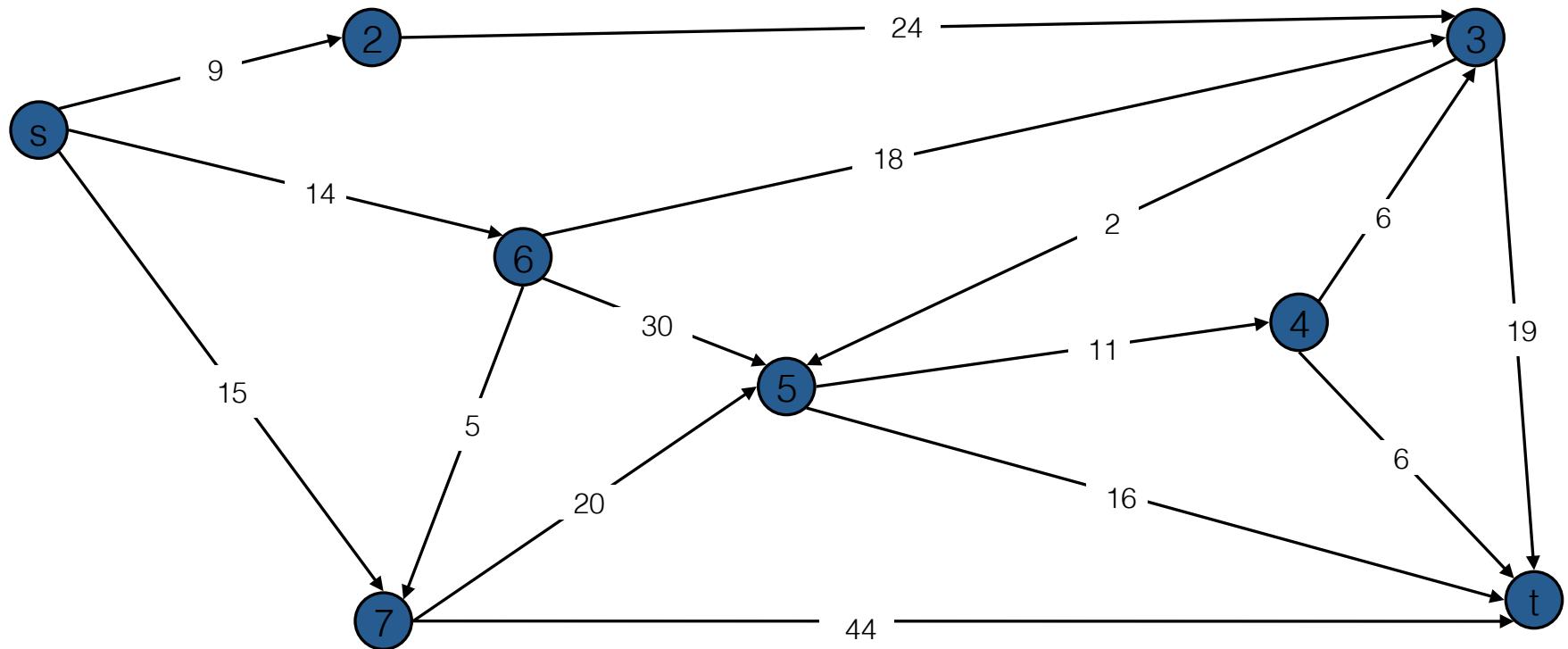
// Main loop

// Where v has not been removed yet from Q

// A shorter path to v has been found

Dijkstra's Shortest Path Algorithm

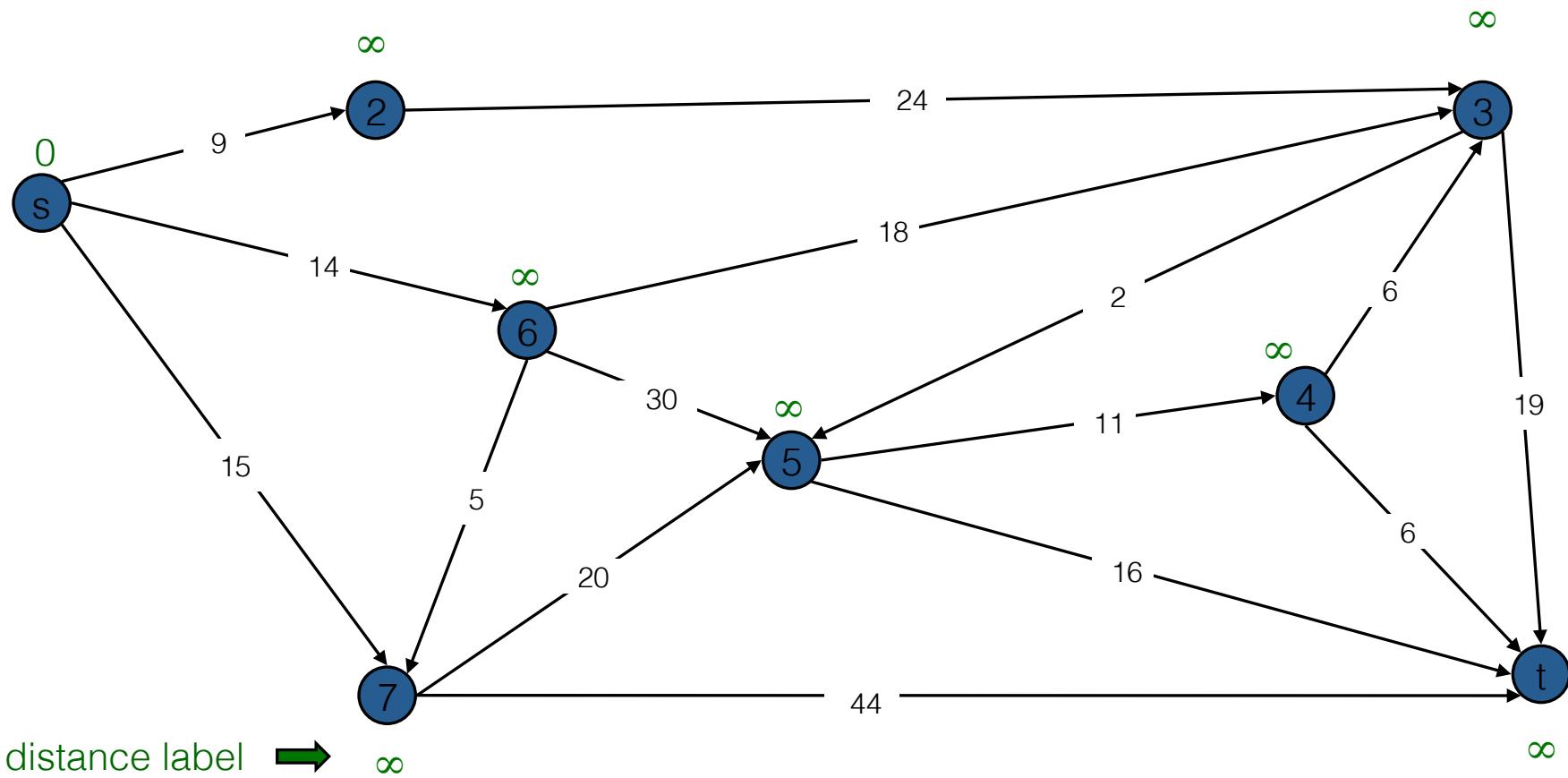
- Find shortest path from s to t



Dijkstra's Shortest Path Algorithm

Visited = { }

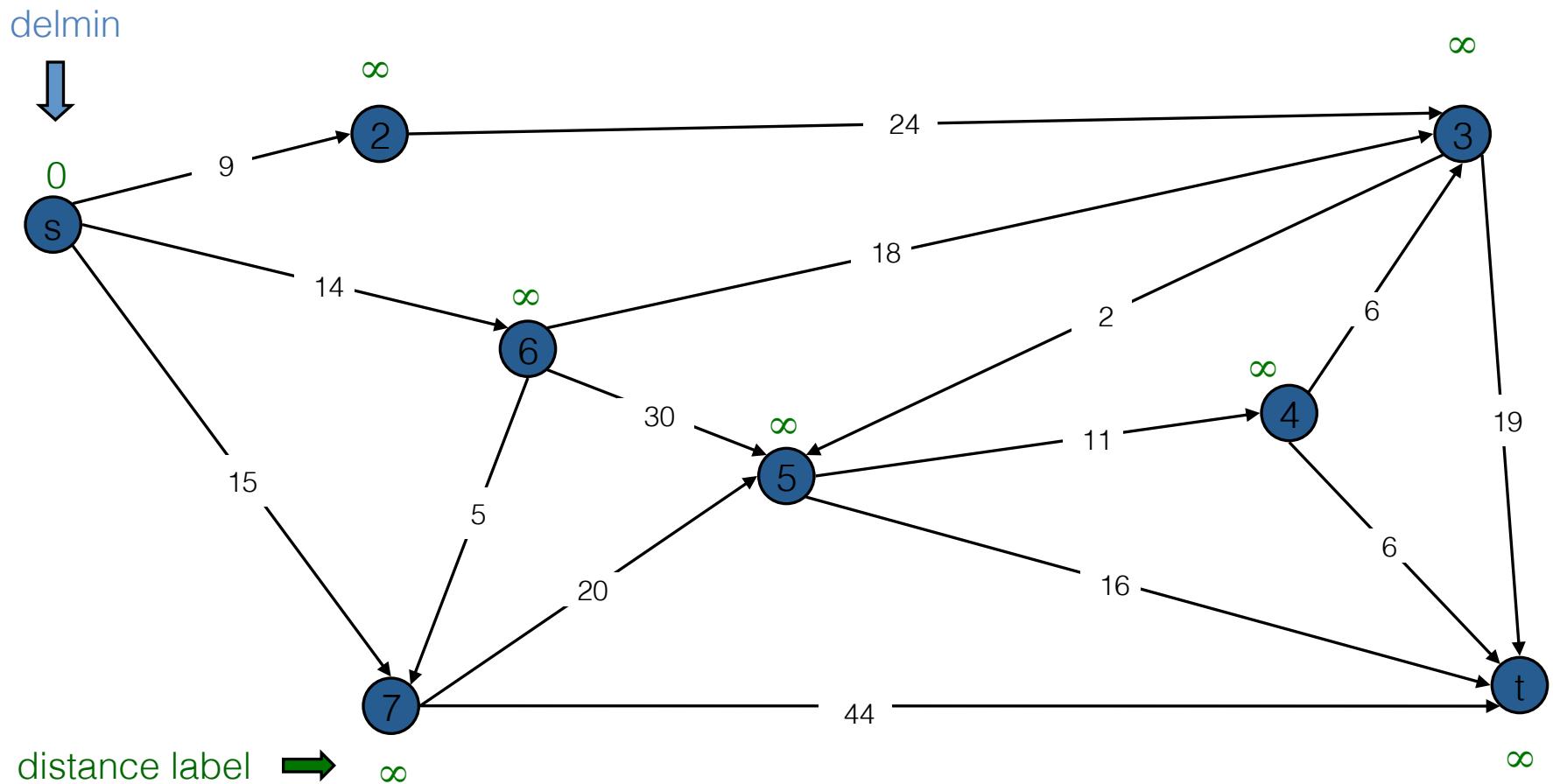
Unvisited = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

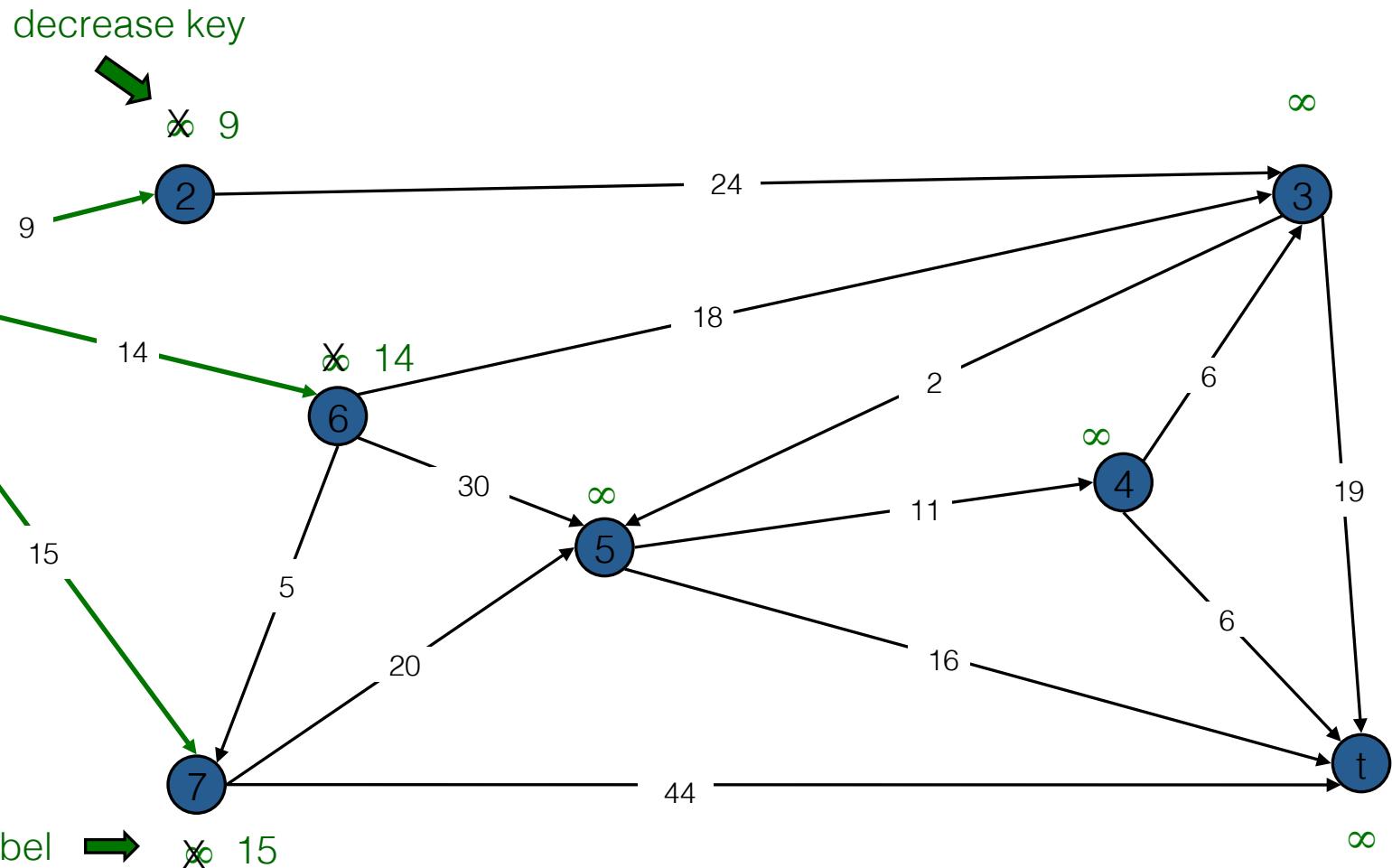
Visited = { }

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



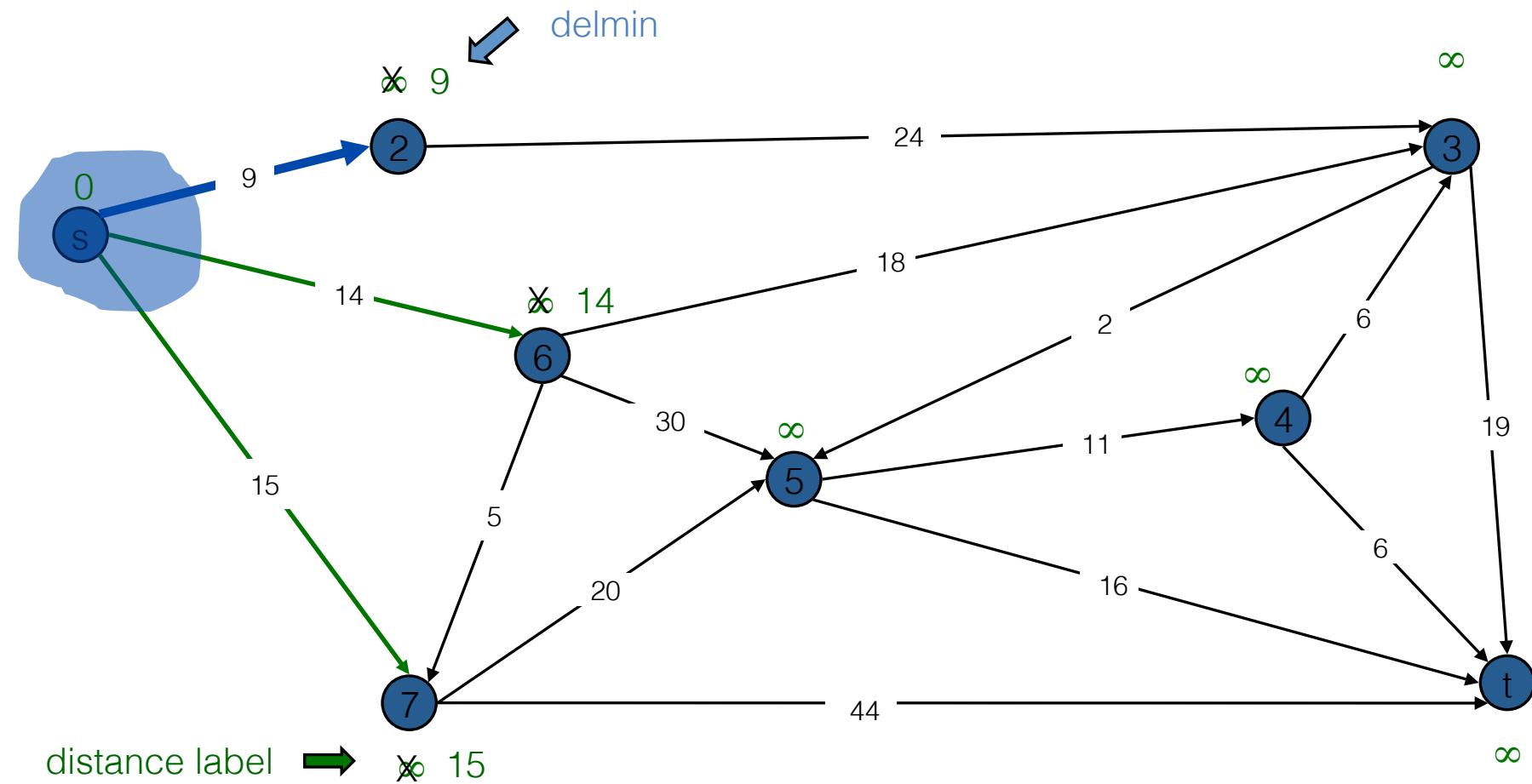
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

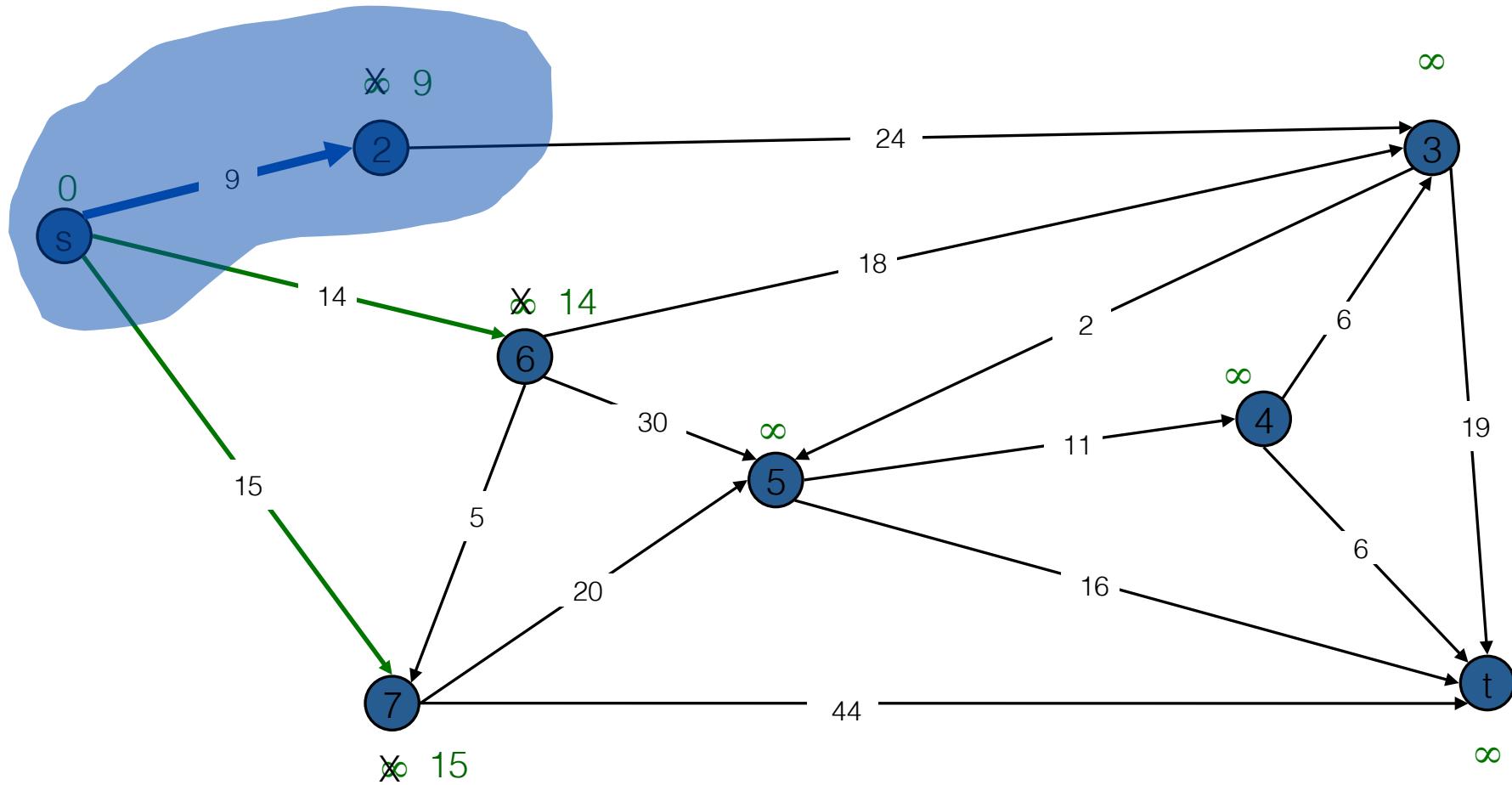
Visited = { s }
Unvisited = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

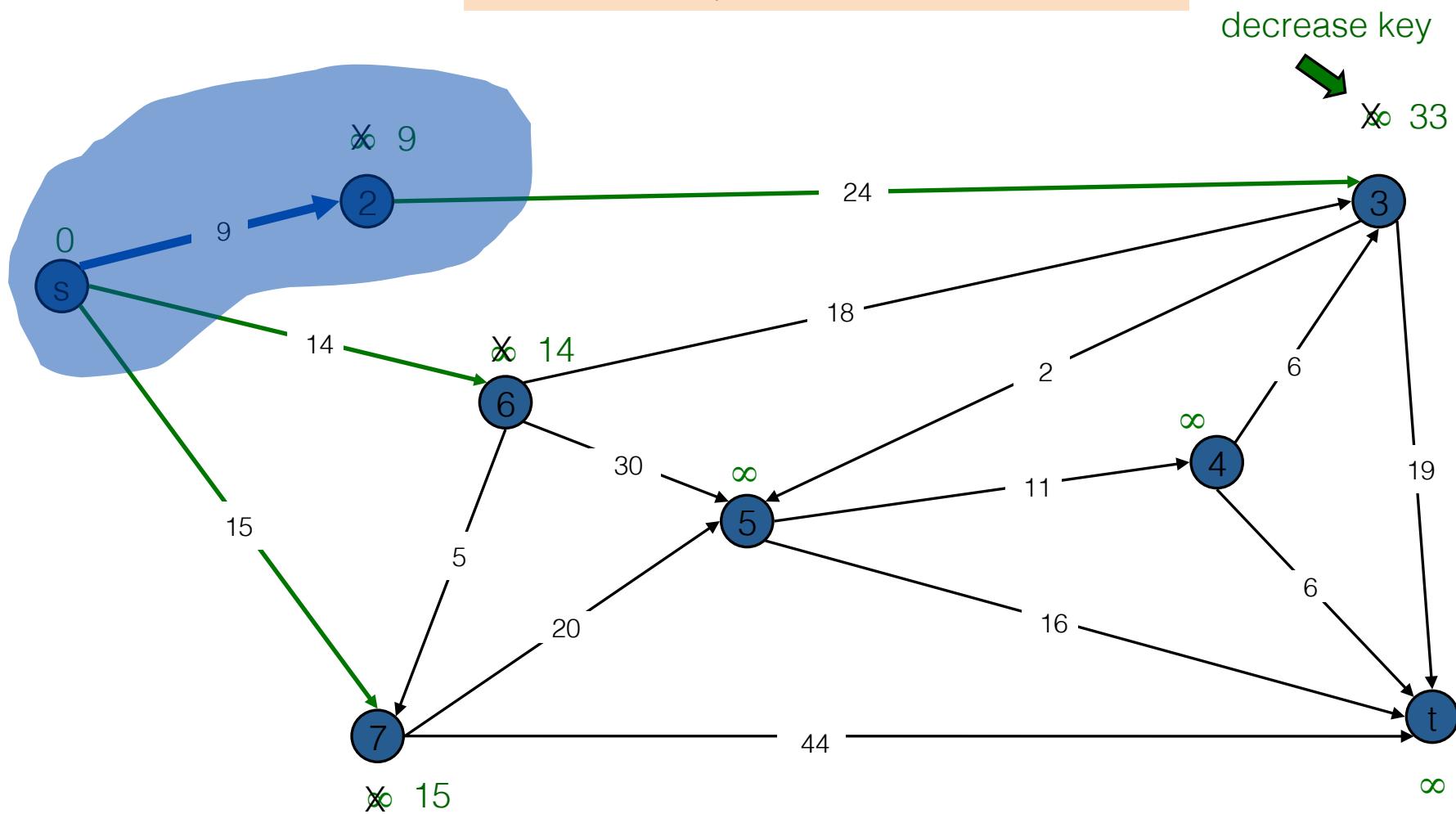
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

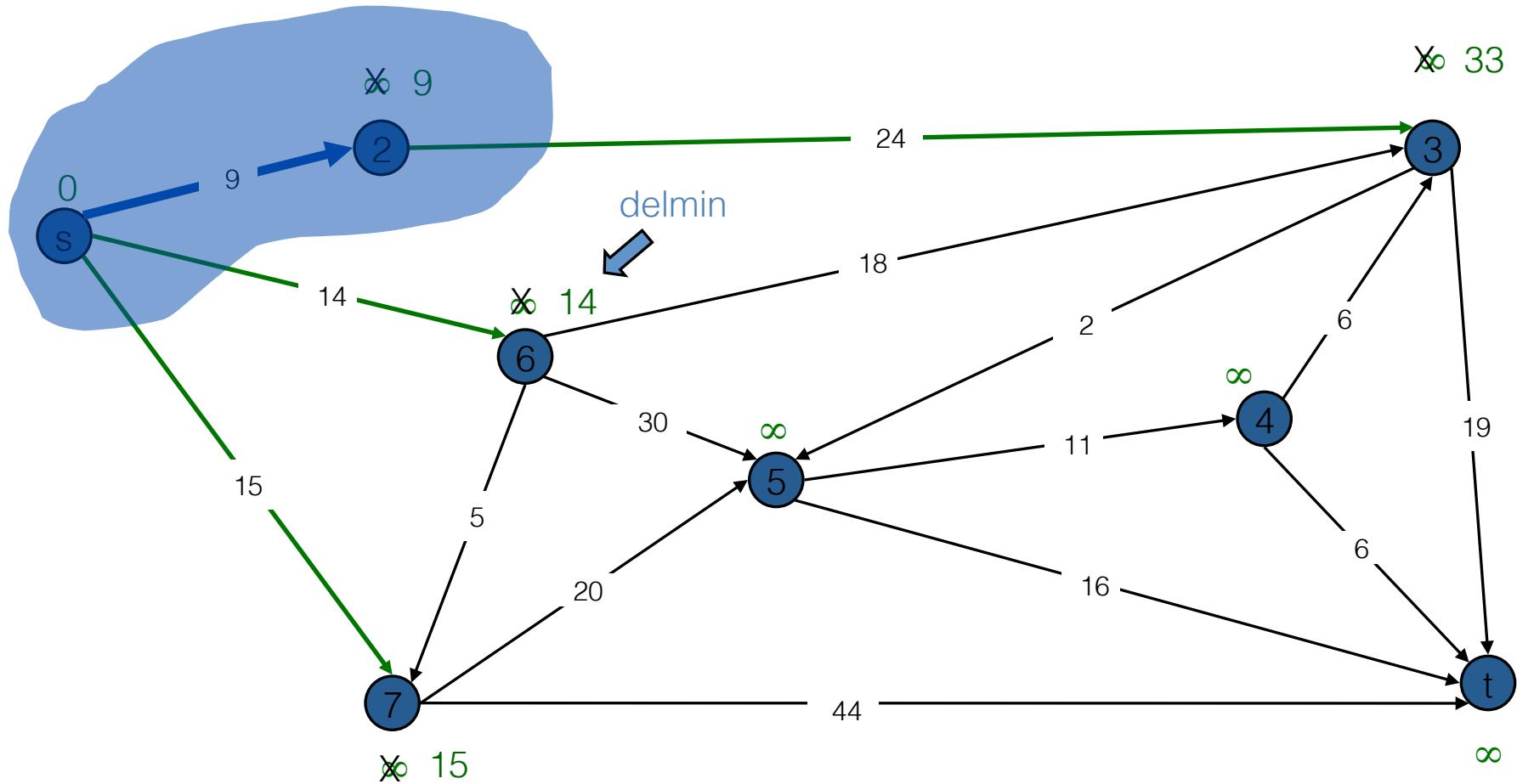
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2 }

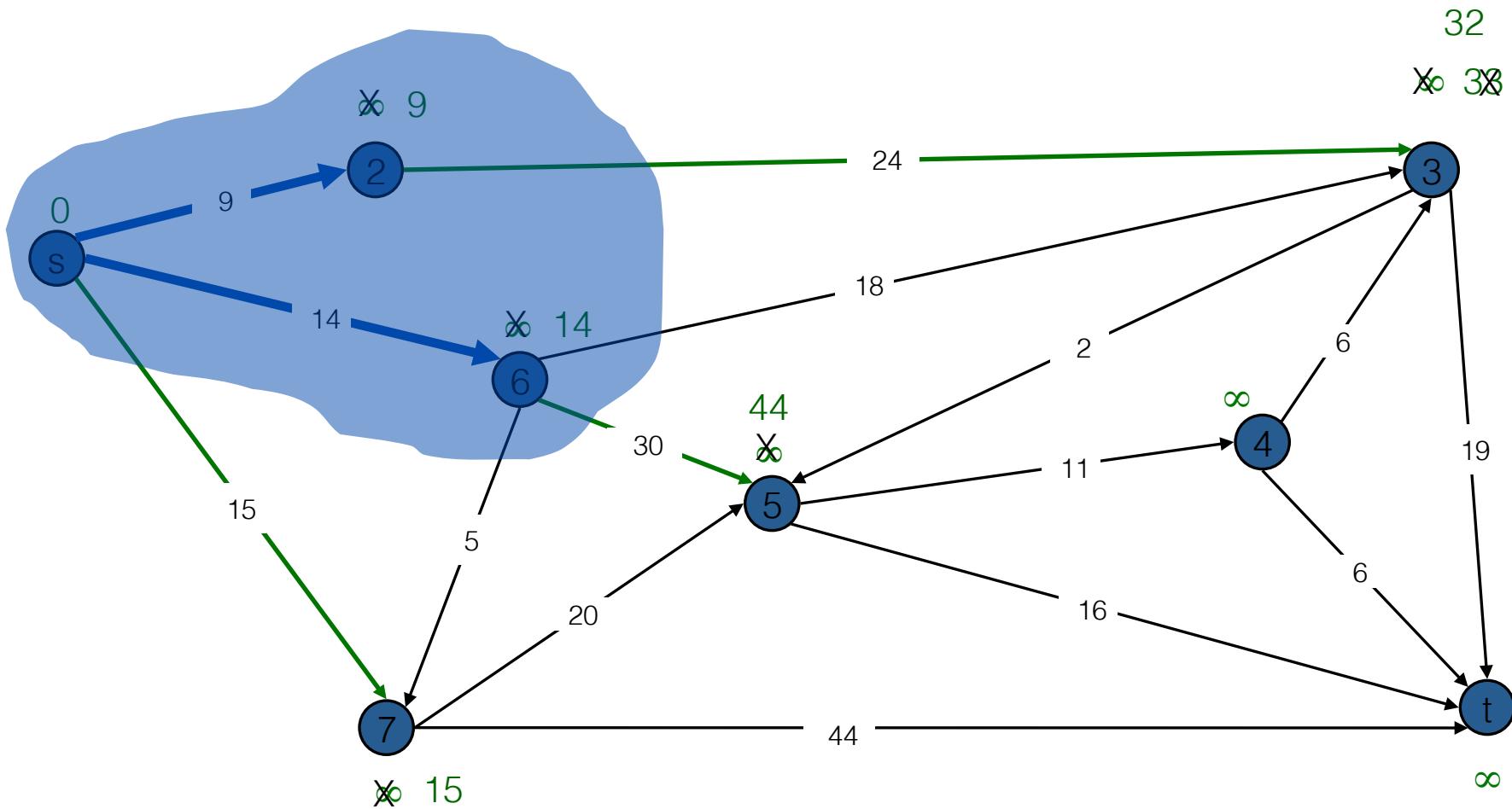
Unvisited = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6 }

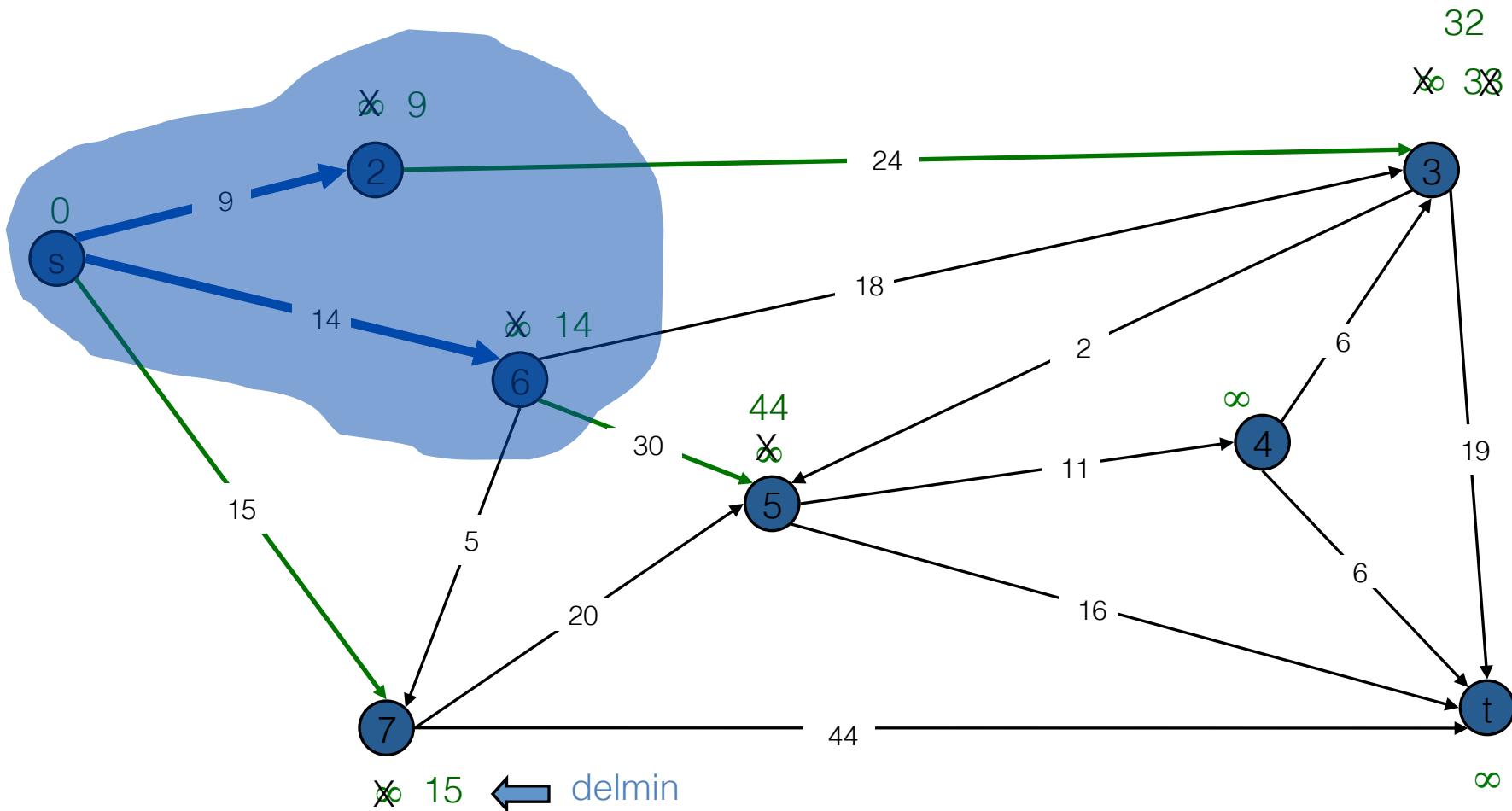
Unvisited = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6 }

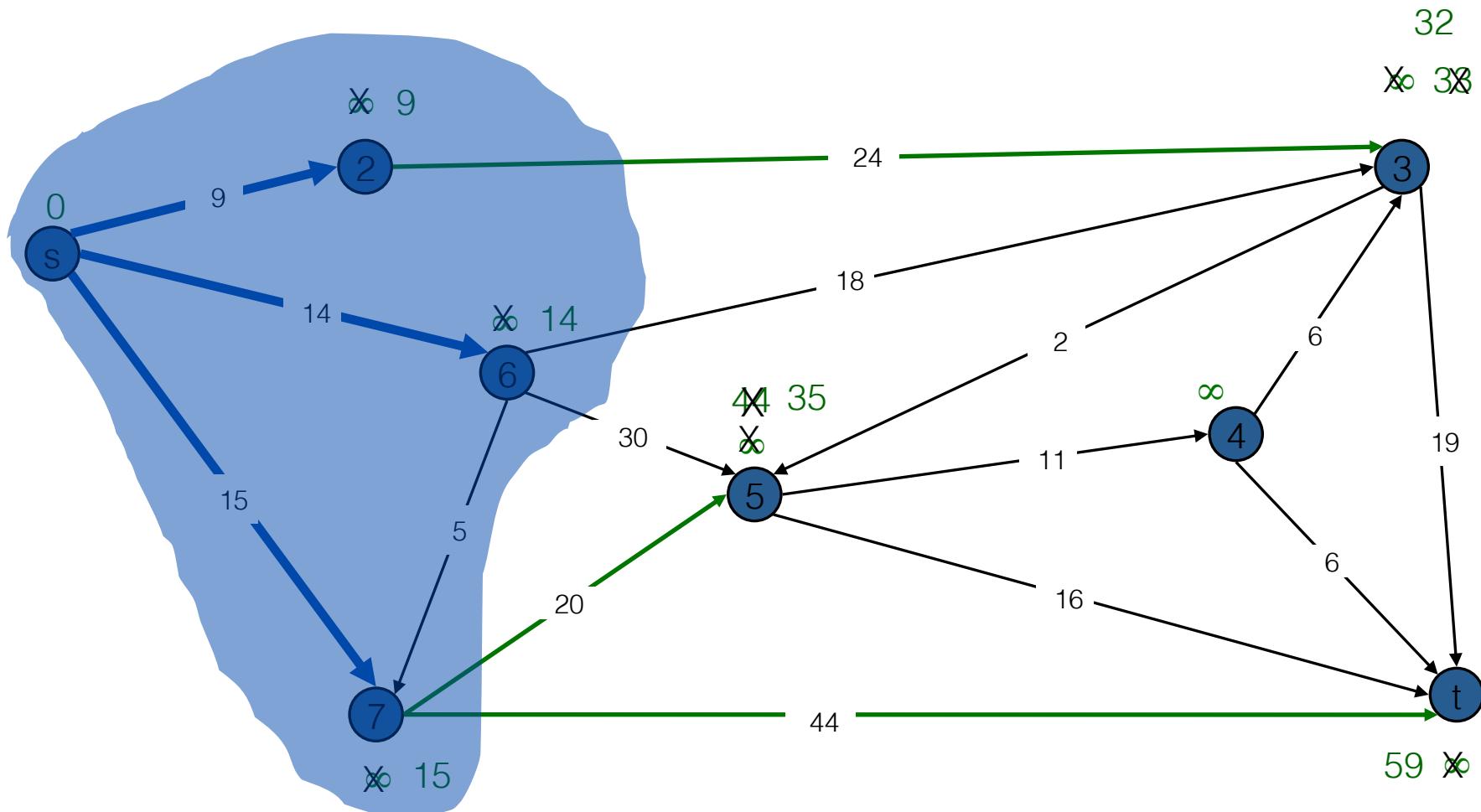
Unvisited = { 3, 4, 5, 7, t }



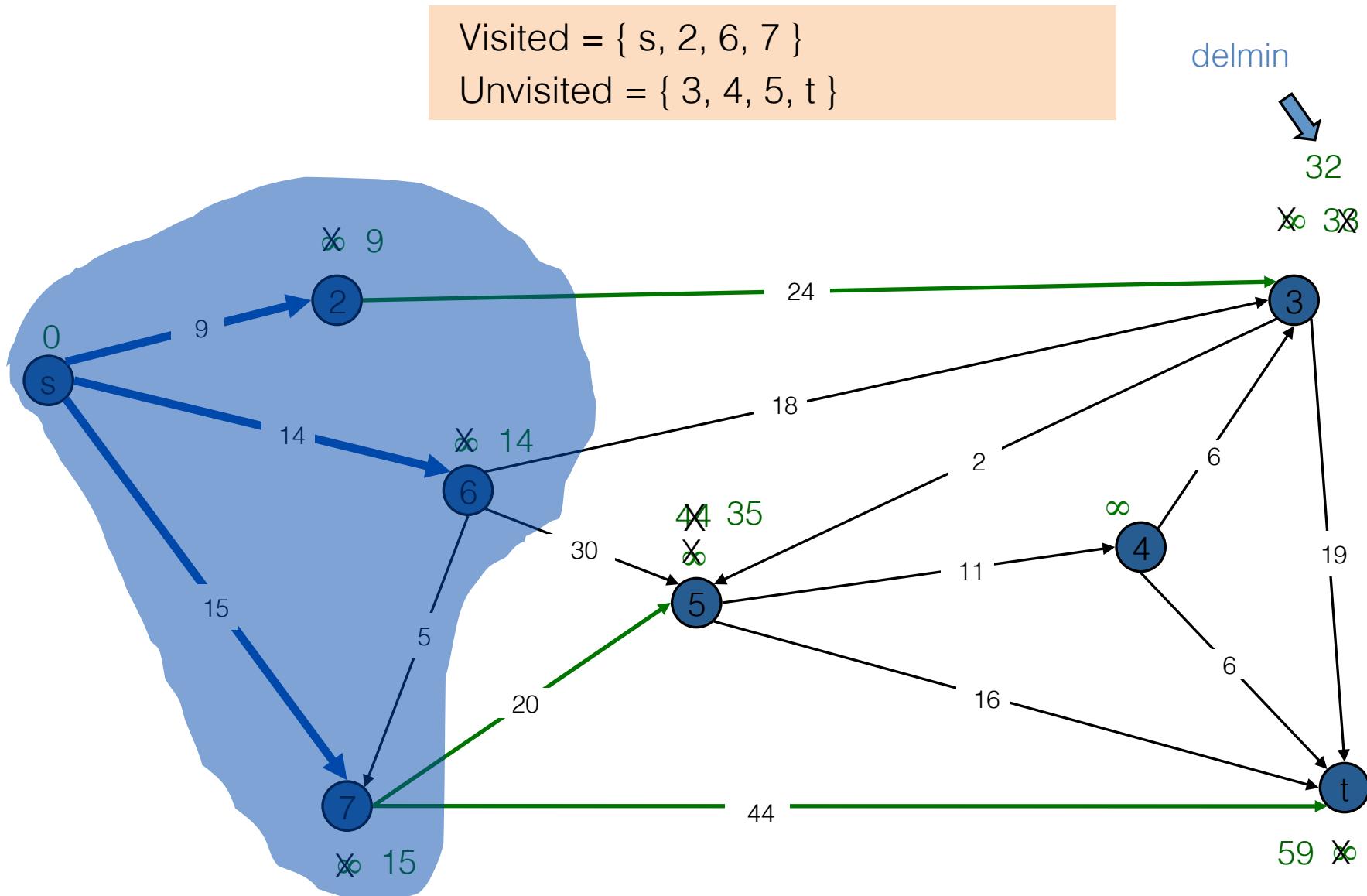
Dijkstra's Shortest Path Algorithm

Visited = { s, 2, 6, 7 }

Unvisited = { 3, 4, 5, t }



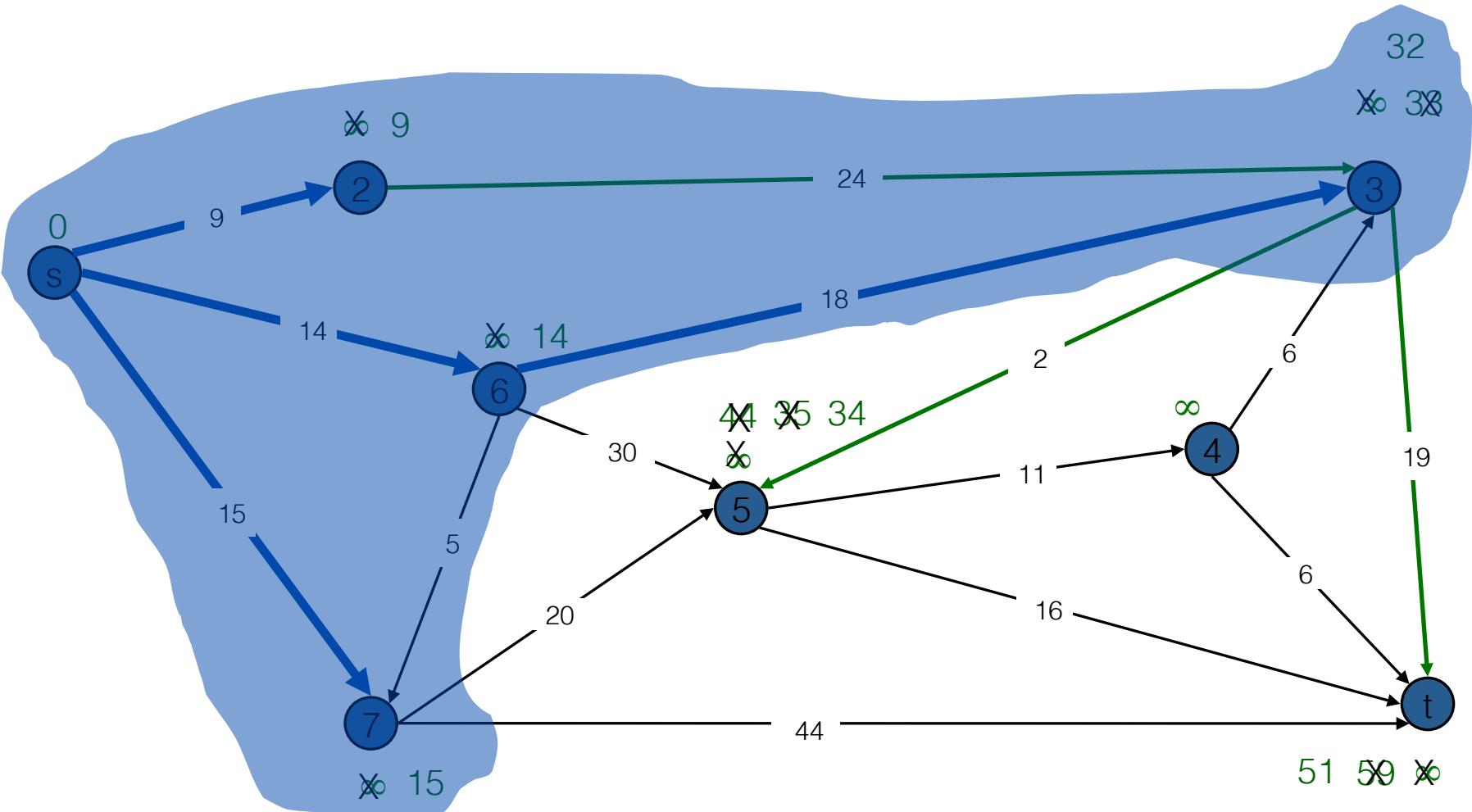
Dijkstra's Shortest Path Algorithm



Dijkstra's Shortest Path Algorithm

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

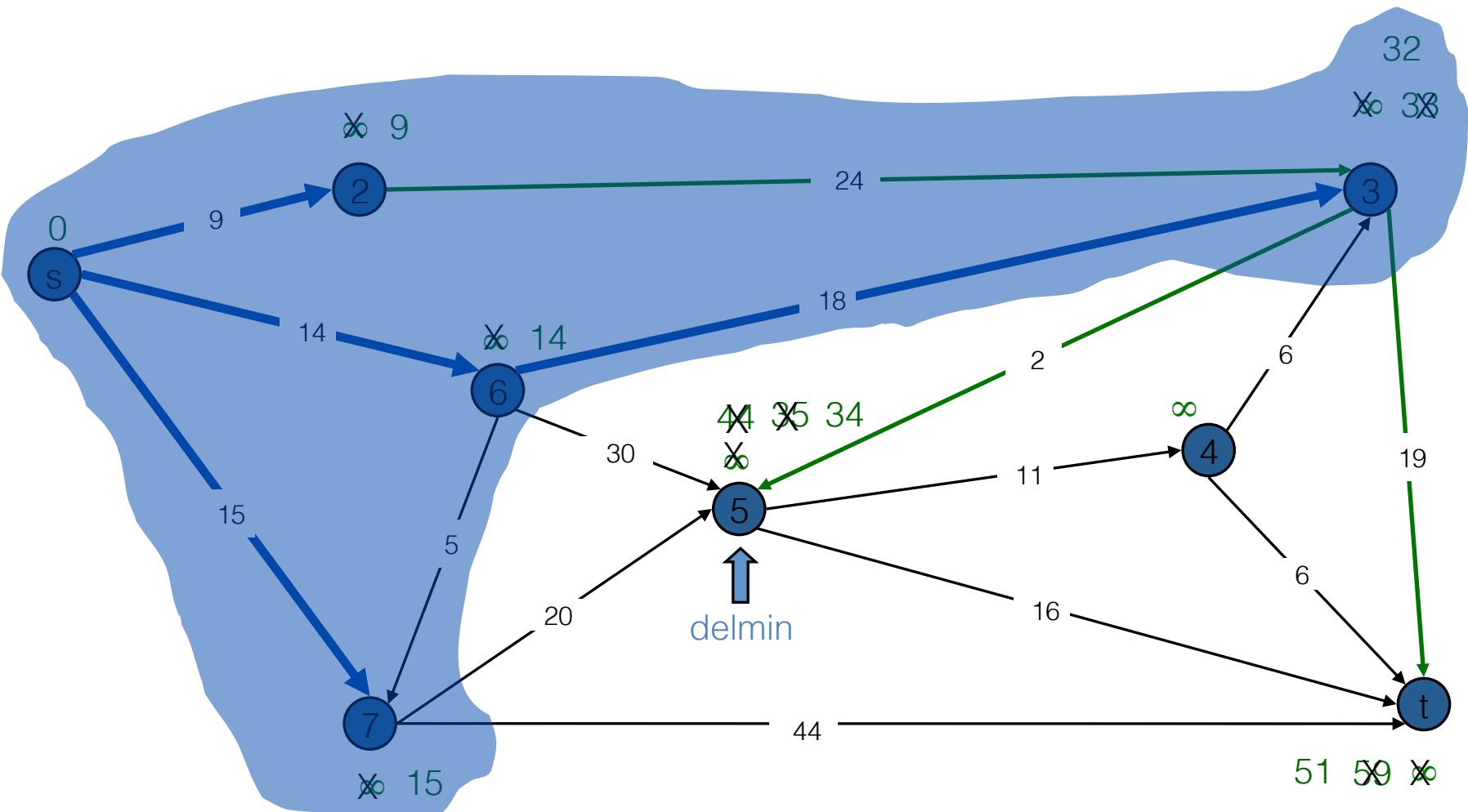
Unvisited = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

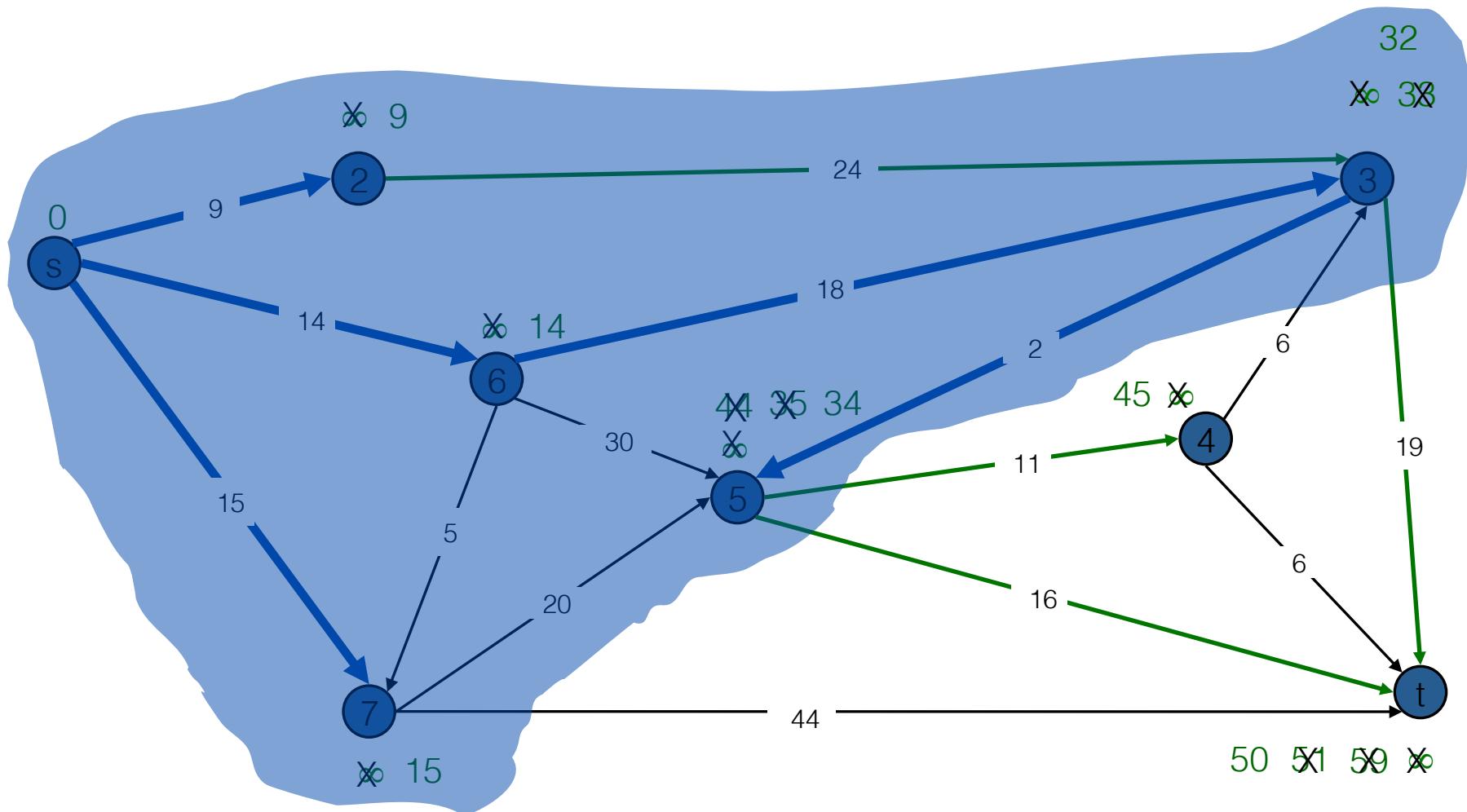
Unvisited = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

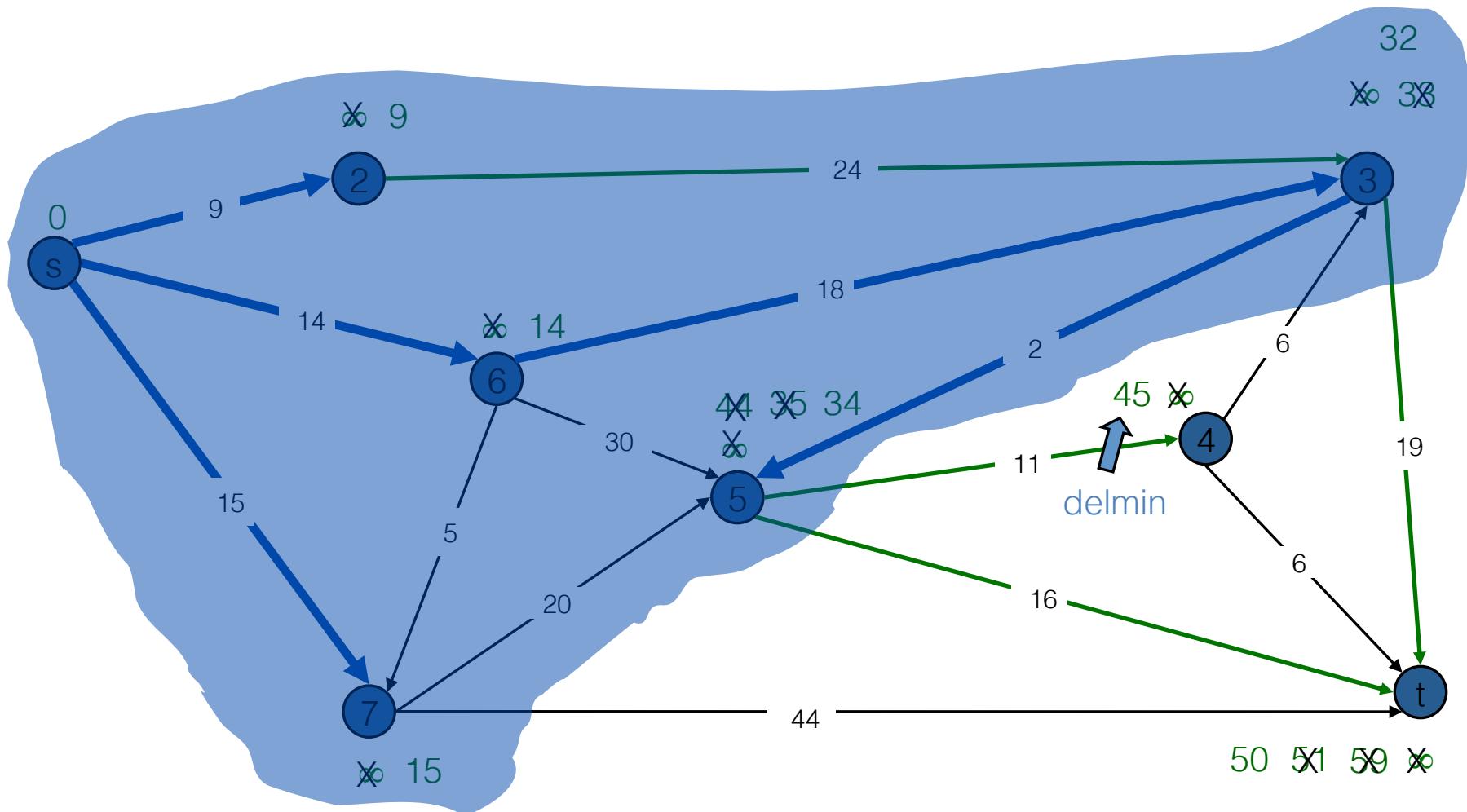
Unvisited = { 4, t }



Dijkstra's Shortest Path Algorithm

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

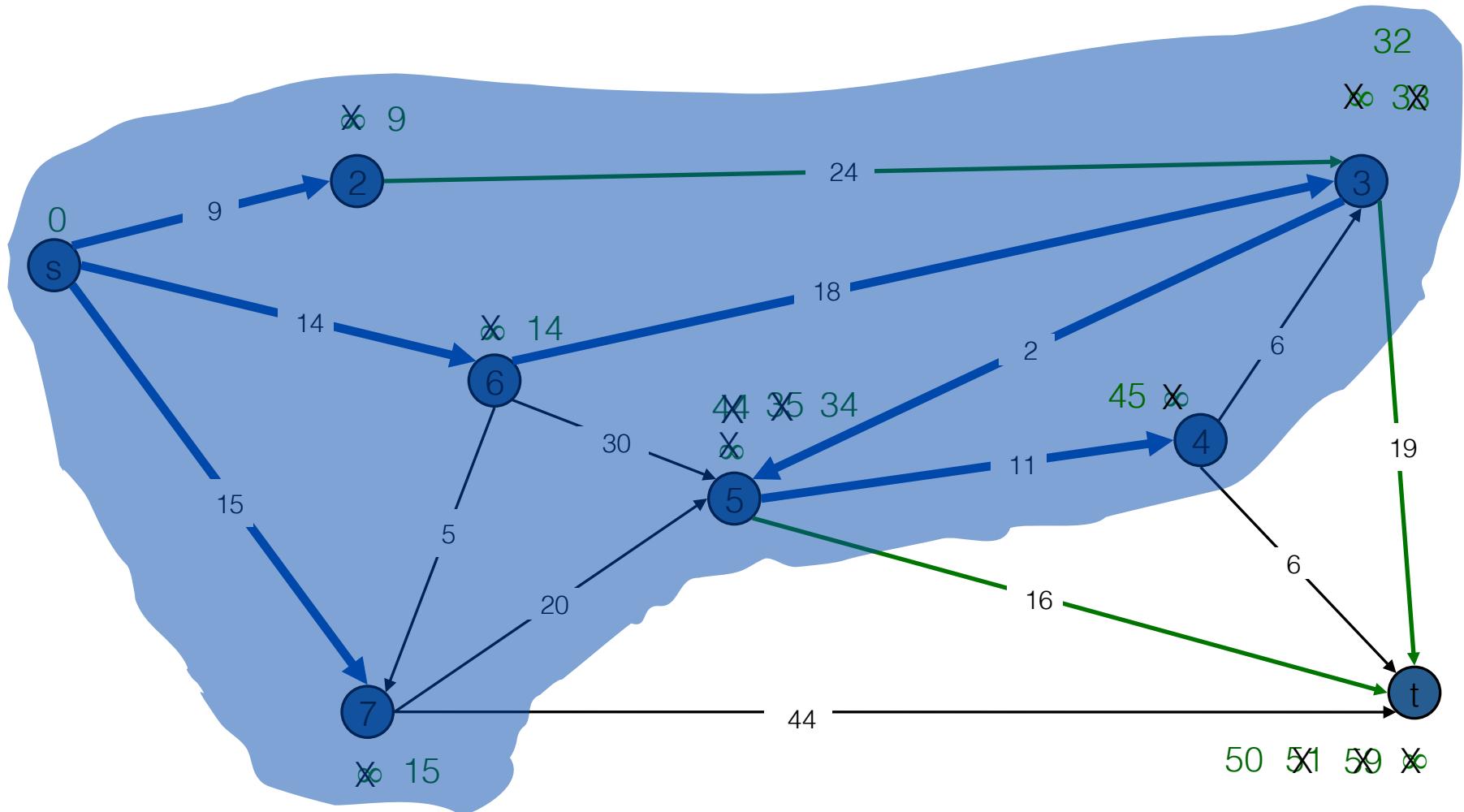
Unvisited = { 4, t }



Dijkstra's Shortest Path Algorithm

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

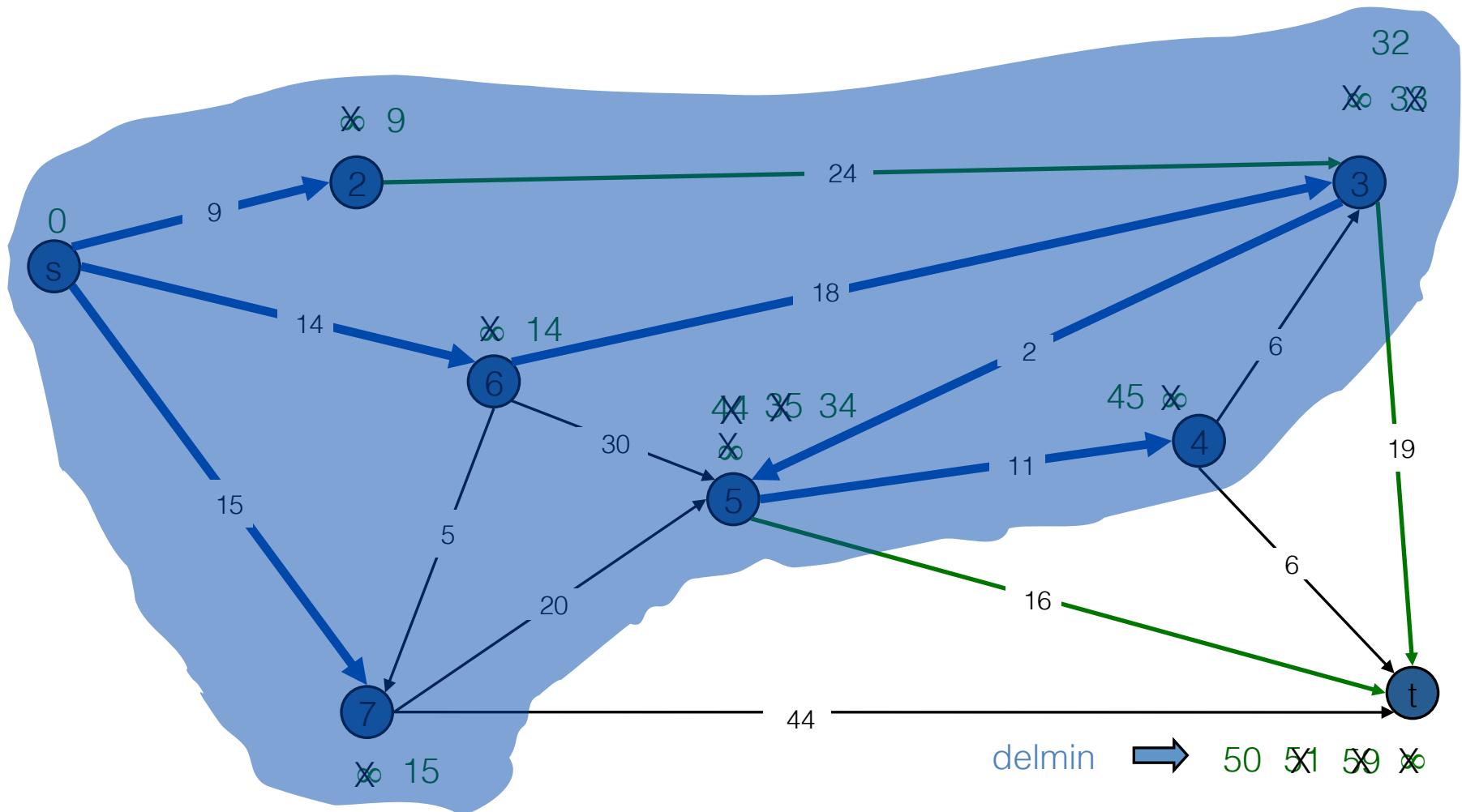
Unvisited = { t }



Dijkstra's Shortest Path Algorithm

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

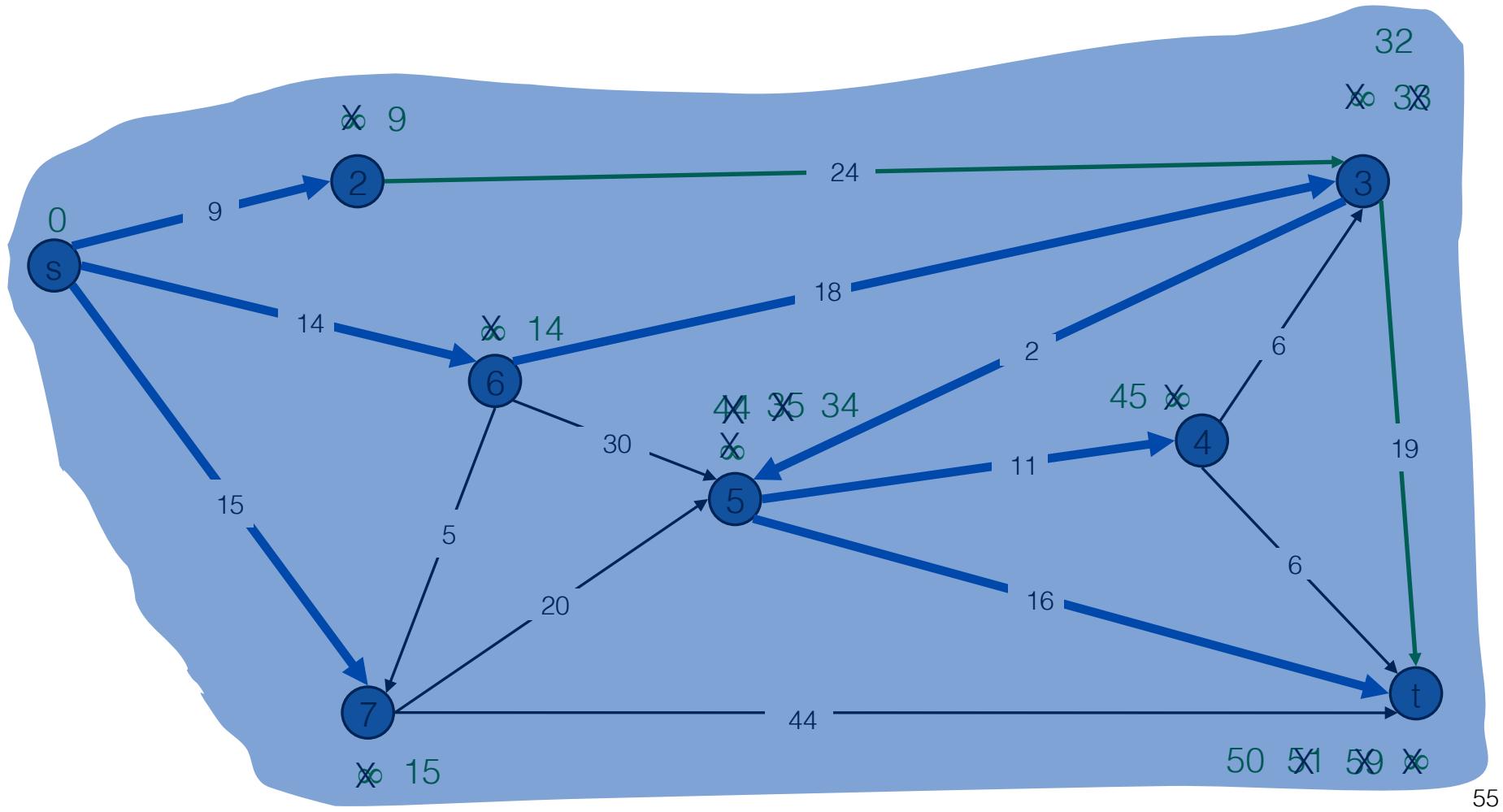
Unvisited = { t }



Dijkstra's Shortest Path Algorithm

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

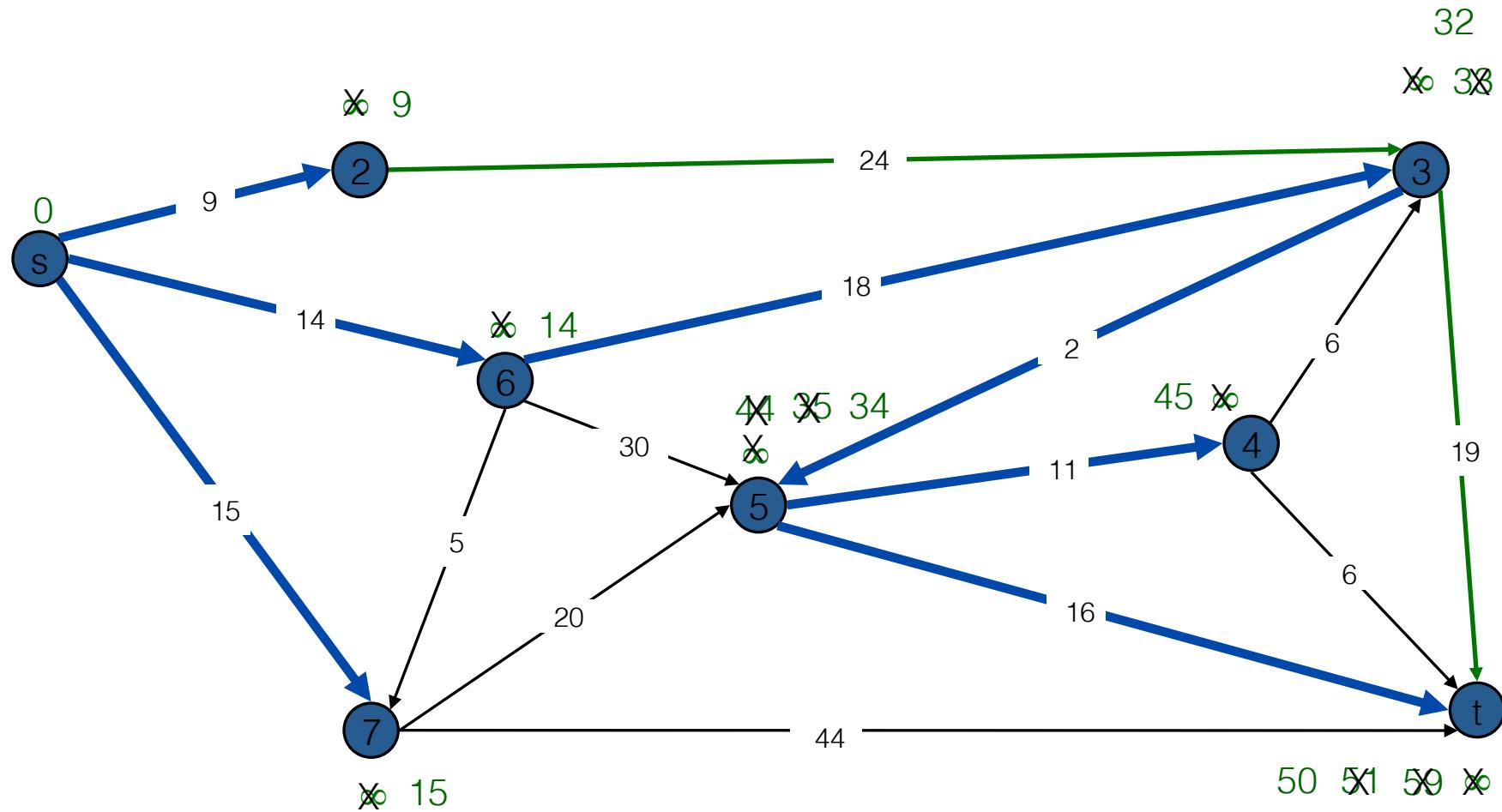
Unvisited = { }



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

$$A = U \Lambda U^T$$

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

Singular Value Decomposition (SVD)

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

$$A = U \Sigma V^T$$

- U is the $m \times m$ orthogonal matrix that contains the left-singular vectors of A
 - V is the $n \times n$ orthogonal matrix that contains the right-singular vector of A
 - Σ is the $m \times n$ 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 $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r)$

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

Low Rank Approximation with SVD

- How to approximate matrix $A_{m \times n}$ with a rank- k matrix ($k \ll n$)?
 - **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 (a $k \times k$ diagonal matrix) – k largest singular values

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

The Frobenius norm $\| A - A_k \|_F$ is minimized

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

Basics in Spectral Graph Theory

Spectral Graph Theory

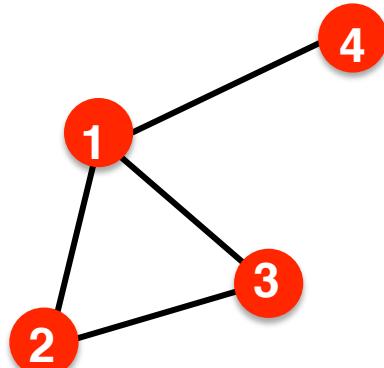
- **Spectral graph theory:** study the properties of a graph based on matrices that are associated with the graph
 - Spectral properties: eigenvalues and eigenvectors
- **Adjacency matrix** and **Laplacian matrix**
- We have already seen the basic properties of the adjacency matrix of a graph G
 - In fact, it's an $n \times n$ matrix A
 - Symmetric, if G is undirected
 - Non symmetric, if G is directed

Laplacian Matrix

- Let $G = (V, E)$ be a graph. Then, the **Laplacian matrix** is defined as

$$\mathbf{L}_{ij} = \begin{cases} k_i & \text{if } i = j \\ -1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Diagonal degree matrix D , where $D_{ii} = k_i$ (node degrees)



$$\mathbf{L} = \mathbf{D} - \mathbf{A} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

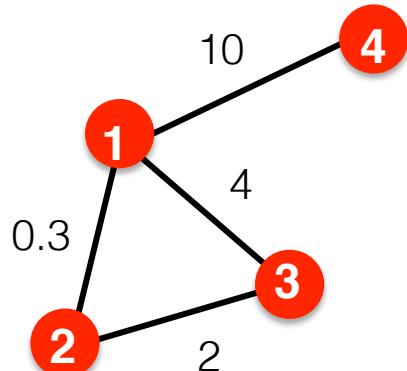
Not so useful spectrum
(largest eigenvalues)

Weighted Laplacian Matrix

- Let $G = (V, E)$ be a weighted graph

$$\mathbf{L}_{ij} = \begin{cases} k_i = \sum_j w_{ij} & \text{if } i = j \\ -w_{ij} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Diagonal degree matrix D , where $D_{ii} = k_i$ (node degrees)



$$\mathbf{L} = \begin{bmatrix} 14.3 & -0.3 & -4 & -10 \\ -0.3 & 2.3 & -2 & 0 \\ -4 & -2 & 6 & 0 \\ -10 & 0 & 0 & 10 \end{bmatrix}$$

Normalized Laplacian Matrix

- Let $G = (V, E)$ be a graph. The **normalized Laplacian matrix** is defined as

$$\mathbf{L}^n_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{\sqrt{k_i k_j}} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

For L : $0 = \lambda_0 \leq \lambda_i \leq 2k^{max}$

For L^n : $0 = \lambda_0 \leq \lambda_i \leq 2$

$$\begin{aligned}\mathbf{L}^n &= \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} \\ &= \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}\end{aligned}$$

Properties of the Laplacian Matrix L^n

- Properties of spectrum $\lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}$ of Laplacian L^n
 - Symmetric and positive semi-definite ($\lambda_i \geq 0$, for all i). Also, $\lambda_0 = 0$
 - If the second smallest eigenvalue $\lambda_1 \neq 0$, then G is connected
 - If L^n has K zero eigenvalues, G has K connected components
 - The smallest non-zero eigenvalue is called spectral gap
(the corresponding eigenvector is used in spectral clustering)

More on graph matrices:
Transition matrix

(lecture on random walks and centrality)

How do we measure and characterize a network?

More properties

Key Network Properties

Degree distribution: $P(k)$

Path length: h

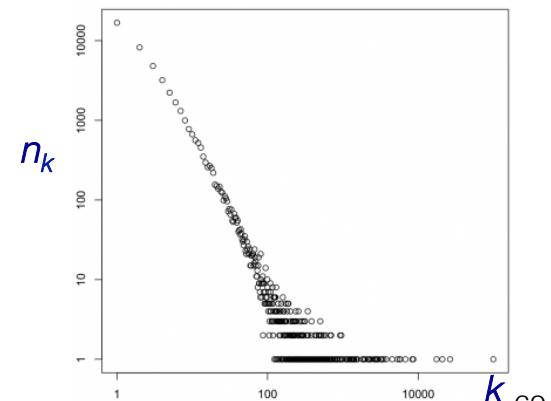
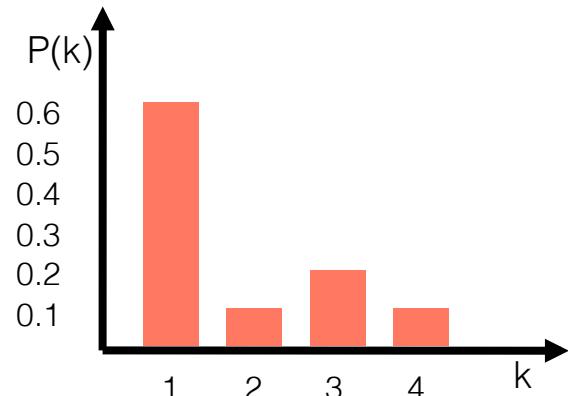
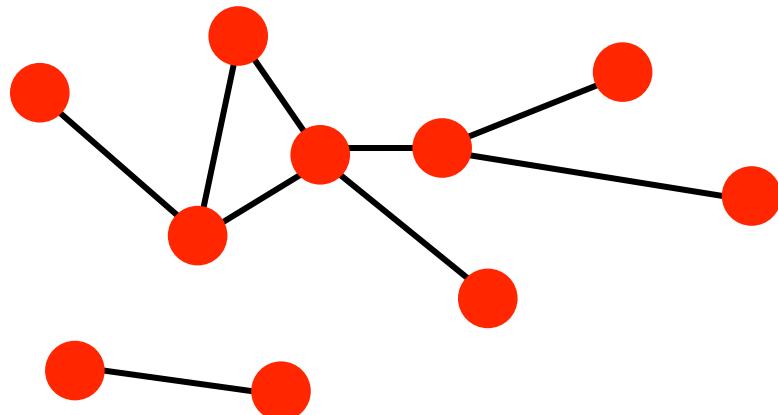
Clustering coefficient: C

Degree Distribution

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

$$n_k = \# \text{ nodes with degree } k$$

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



Number of Paths

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

- Then, element \mathbf{A}_{uv}^h 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 , $\mathbf{A}_{uv}=1$ else $\mathbf{A}_{uv}=0$

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

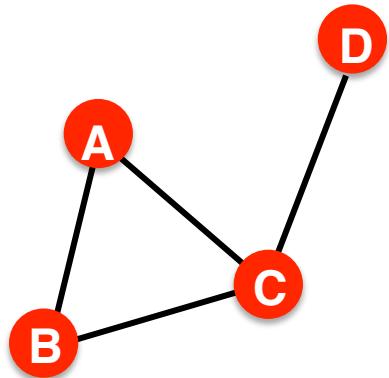
$$H_{uv}^{(2)} = \sum_{k=1}^n \mathbf{A}_{uk} \mathbf{A}_{kv} = \mathbf{A}_{uv}^2$$

- # of paths of length h : If there is a path of length h between u and v then $\mathbf{A}_{uk} \dots \mathbf{A}_{kv}=1$ else $\mathbf{A}_{uk} \dots \mathbf{A}_{kv}=0$

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

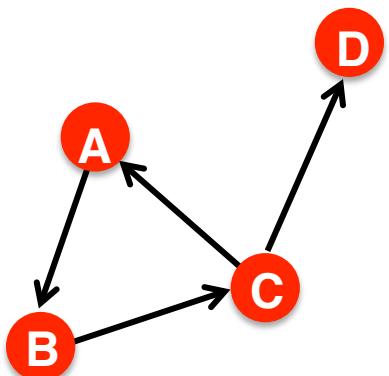
(holds for both directed and undirected graphs)

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)

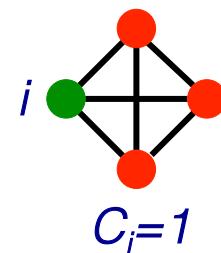
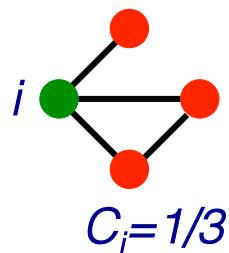
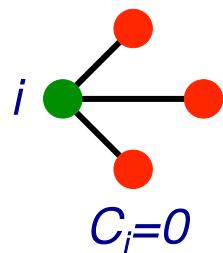
$$\bar{h} = \frac{1}{n(n-1)} \sum_{i,j \neq i} h_{ij} \quad \text{where } h_{ij} \text{ is the distance from node } i \text{ to node } j$$

Clustering Coefficient (1/2)

- 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



Average clustering coefficient:

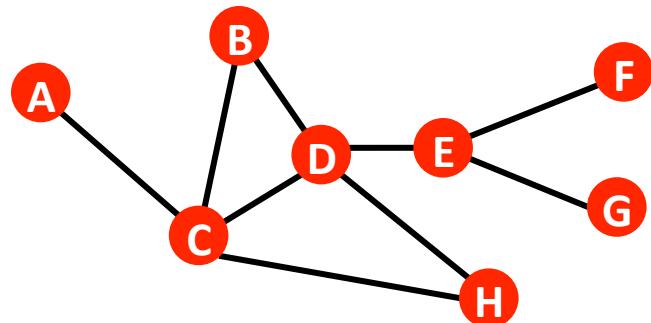
$$C = \frac{1}{|V|} \sum_{i=1}^{|V|} C_i$$

Clustering Coefficient (2/2)

- 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



$$k_B=2, e_B=1, C_B=2/2 = 1$$
$$k_D=4, e_D=2, C_D=4/12 = 1/3$$

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:

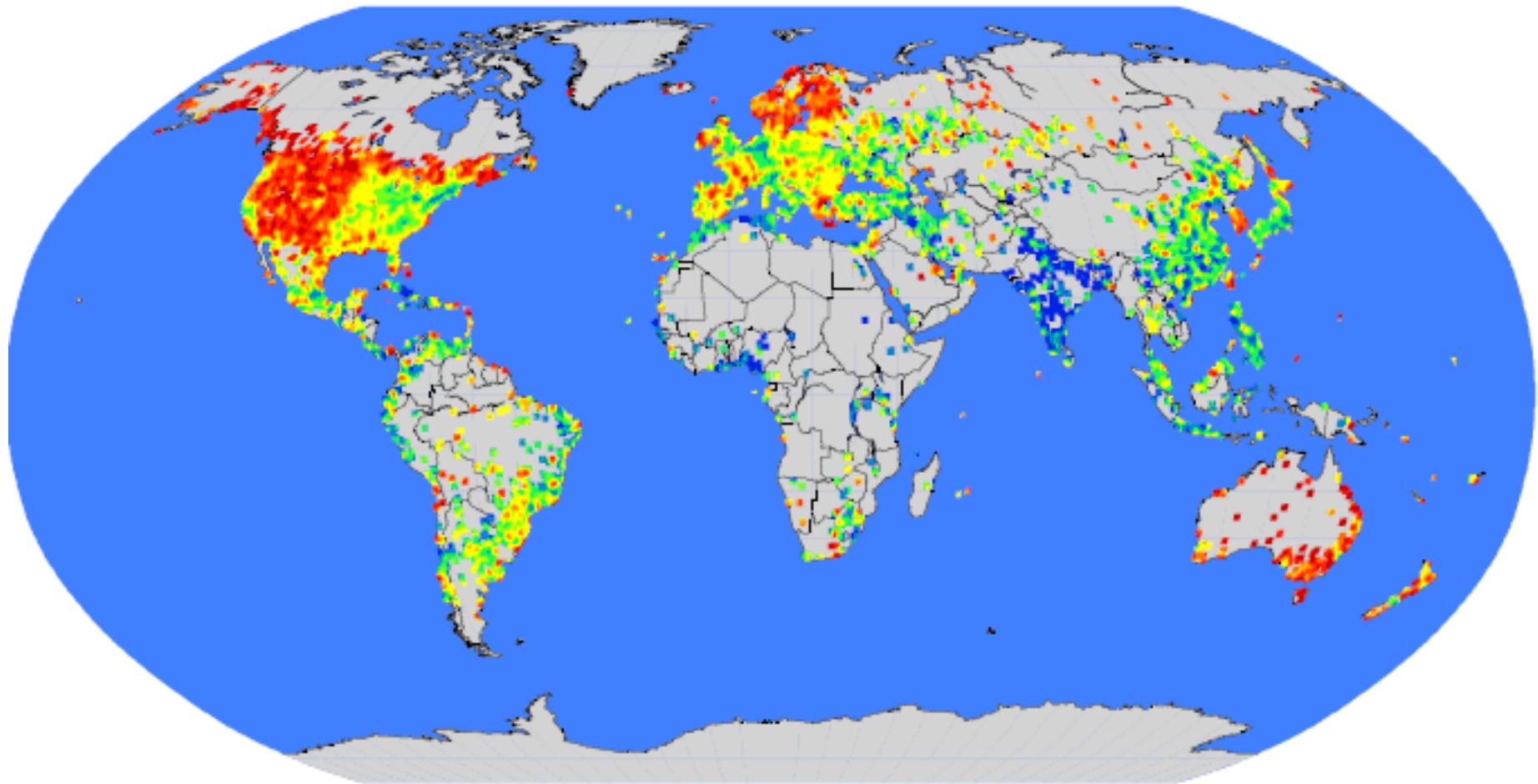
The MSN messenger case

The MSN Messenger

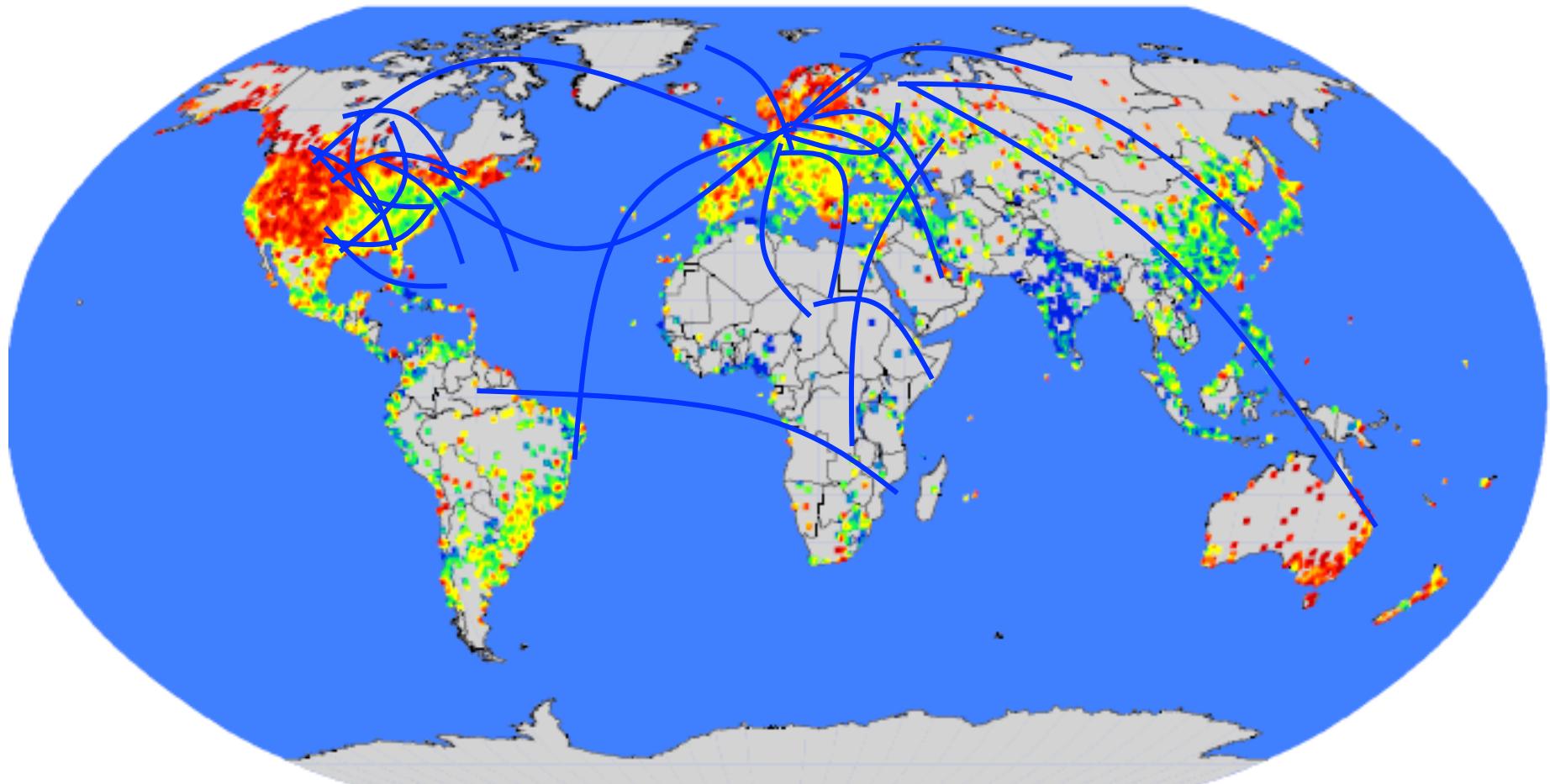


- **MSN Messenger** activity in June 2006:
 - 245 million users logged in
 - 180 million users engaged in conversations
 - More than 30 billion conversations
 - More than 255 billion exchanged messages
 - Now called *Windows Live Messenger*

Communication: Geography

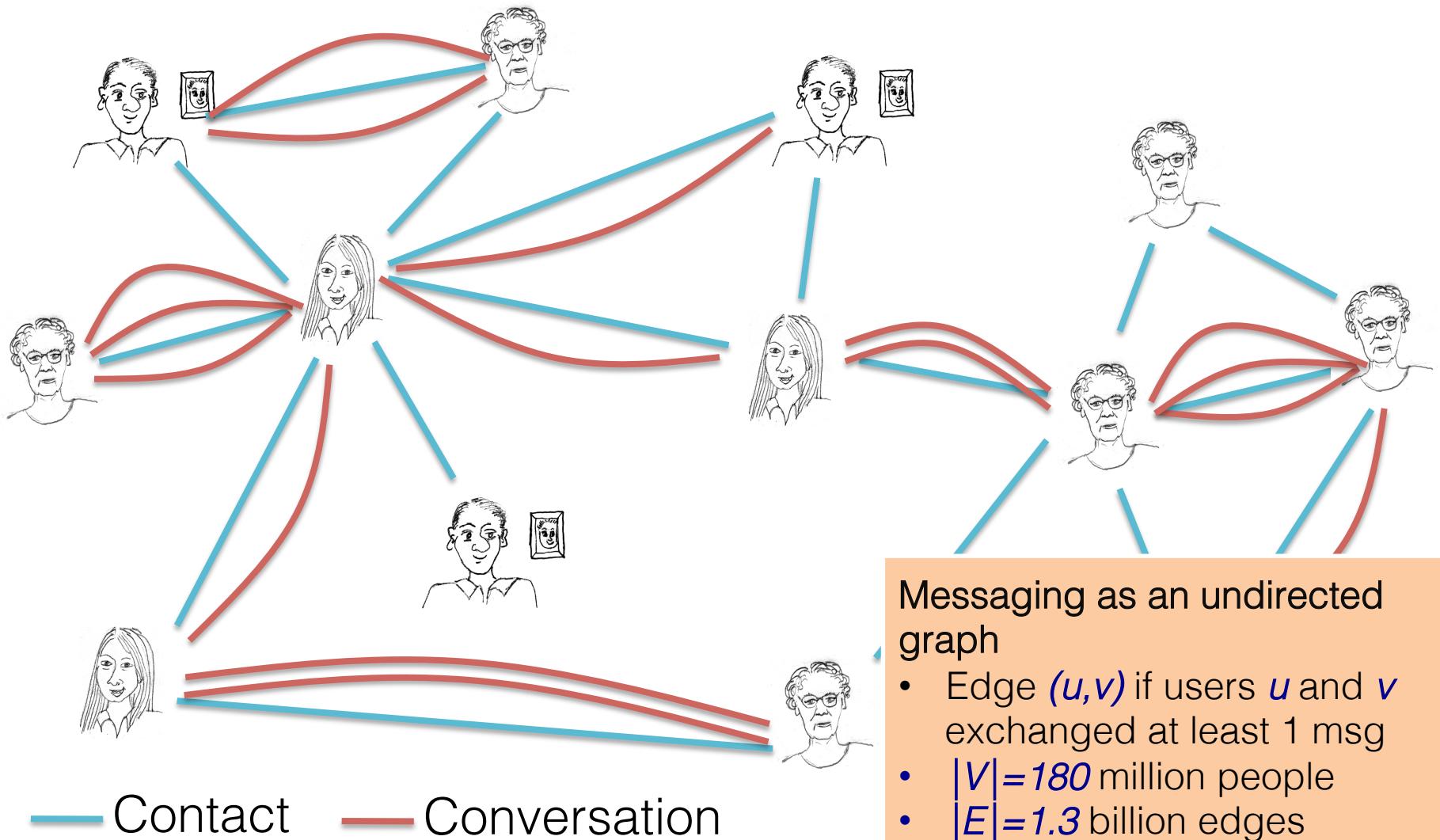


Communication network

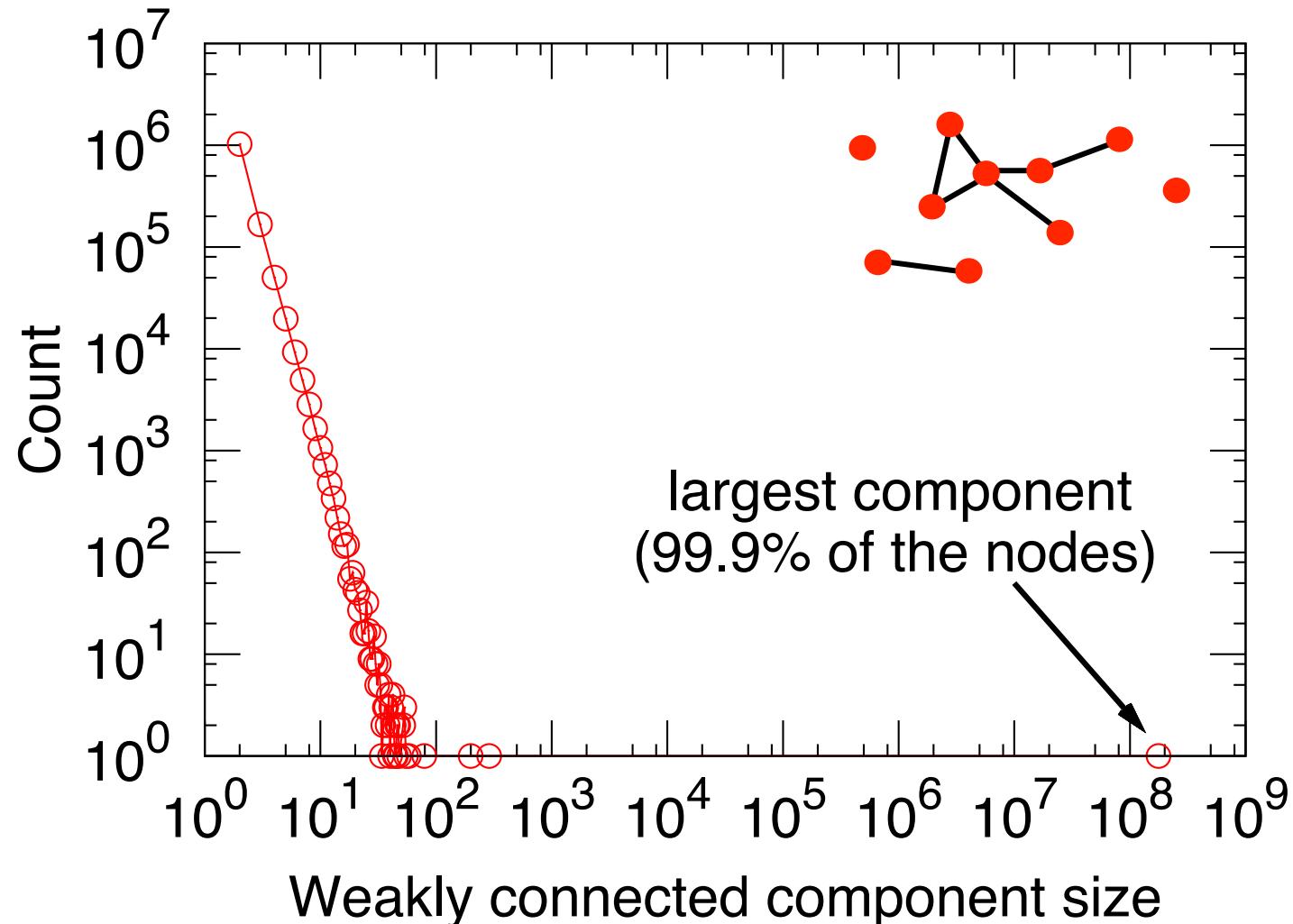


Network: 180M people, 1.3B edges

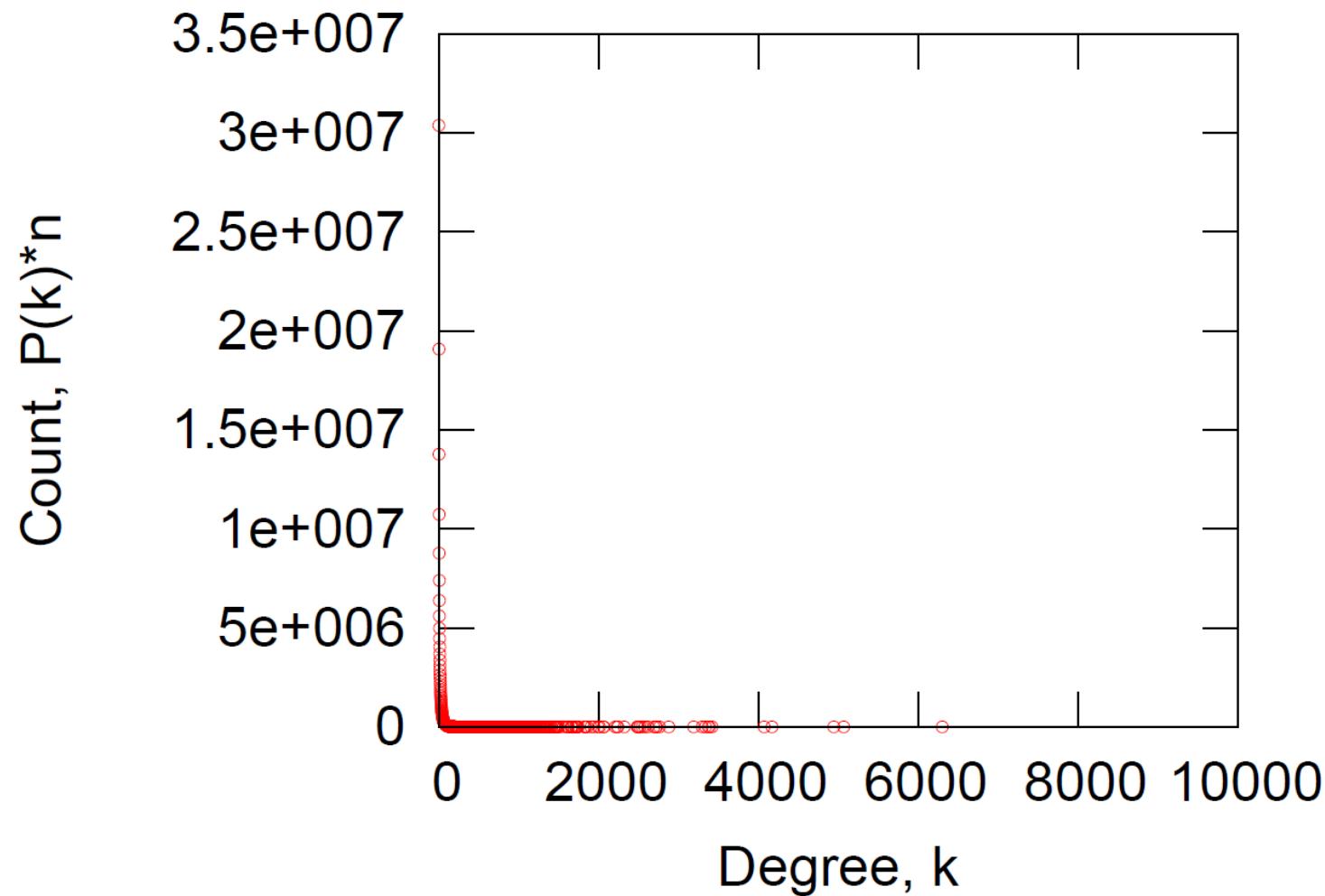
Messaging as a Multigraph



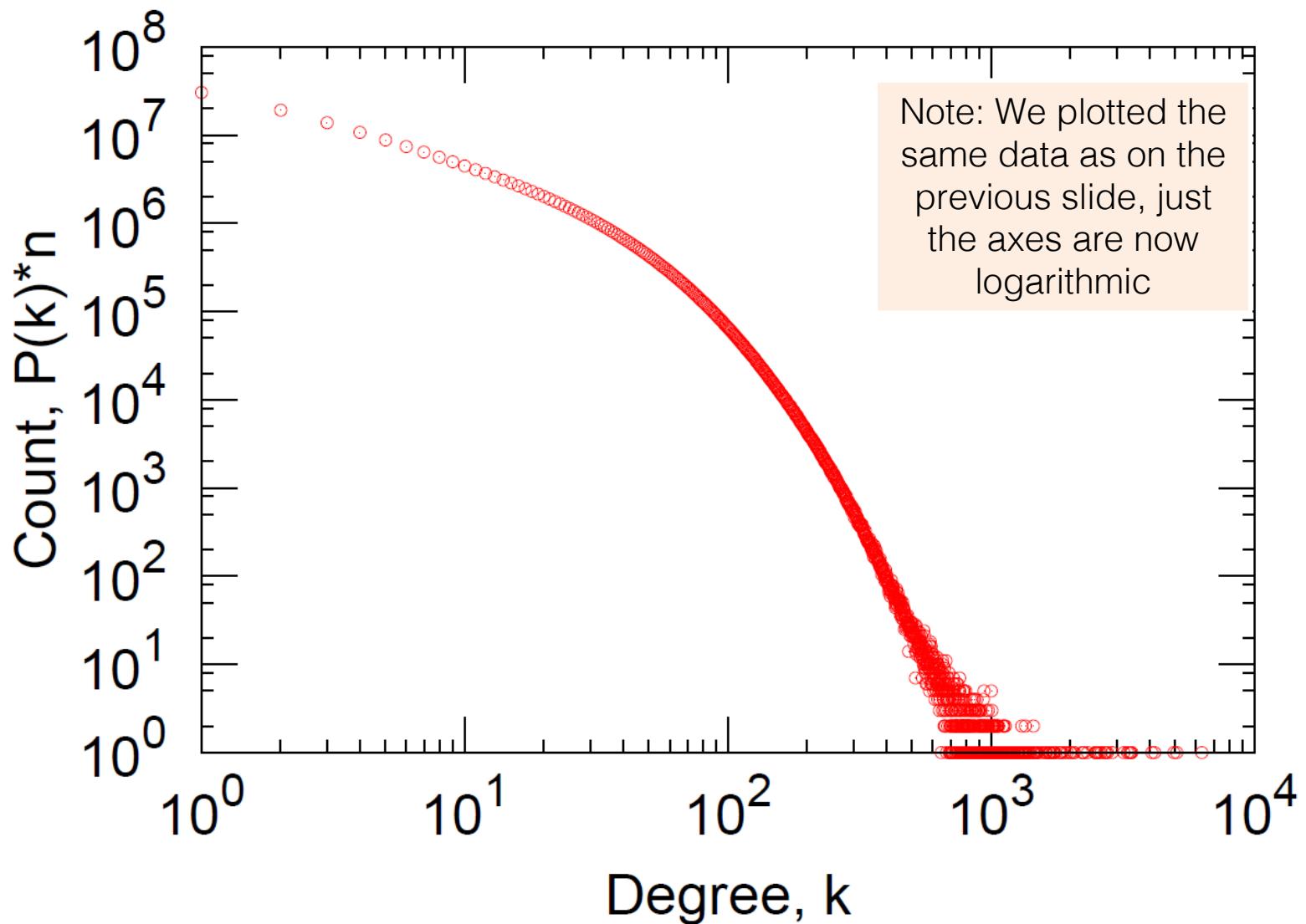
MSN: Connectivity



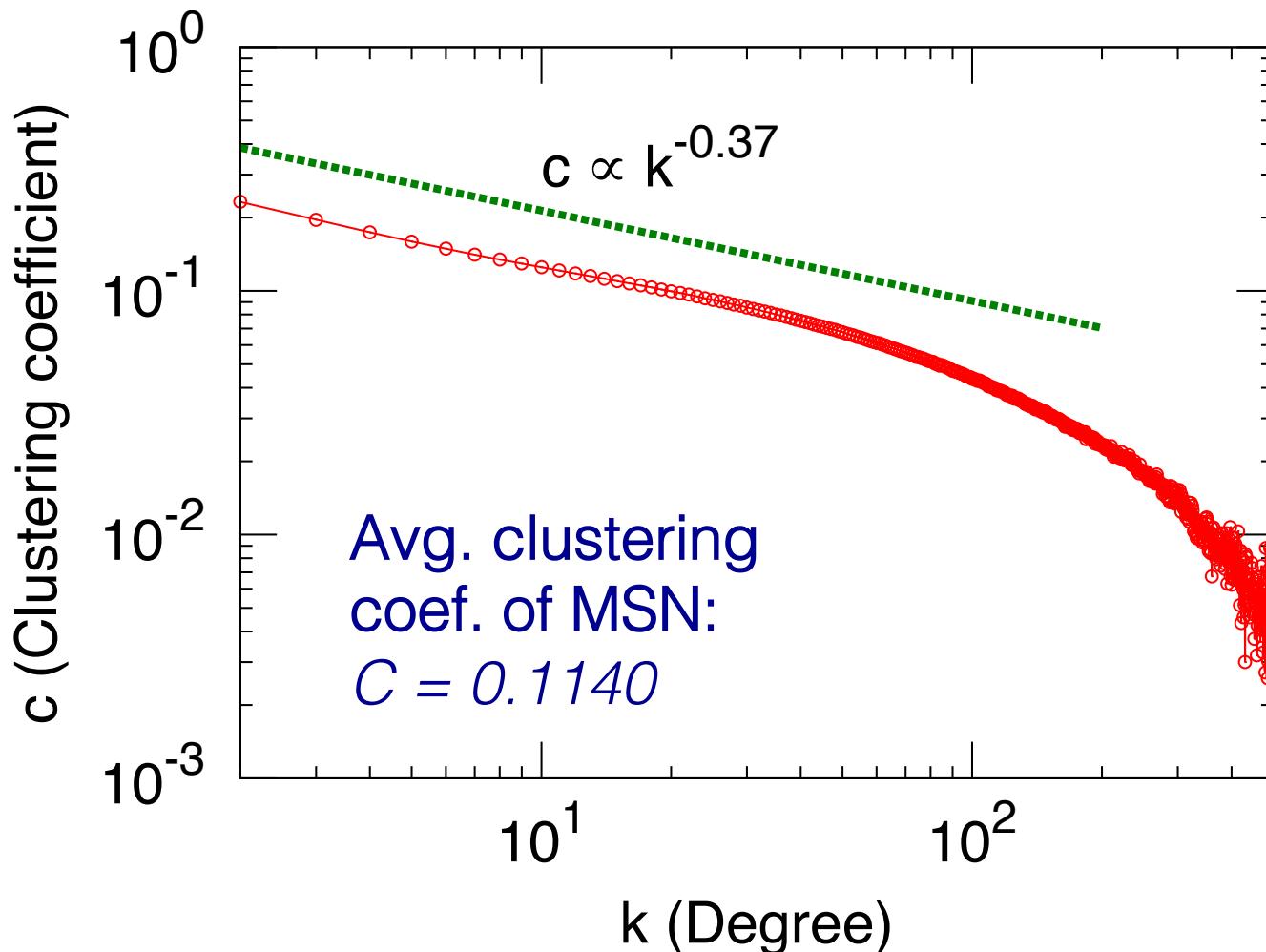
MSN: Degree Distribution



MSN: Log-Log Degree Distribution

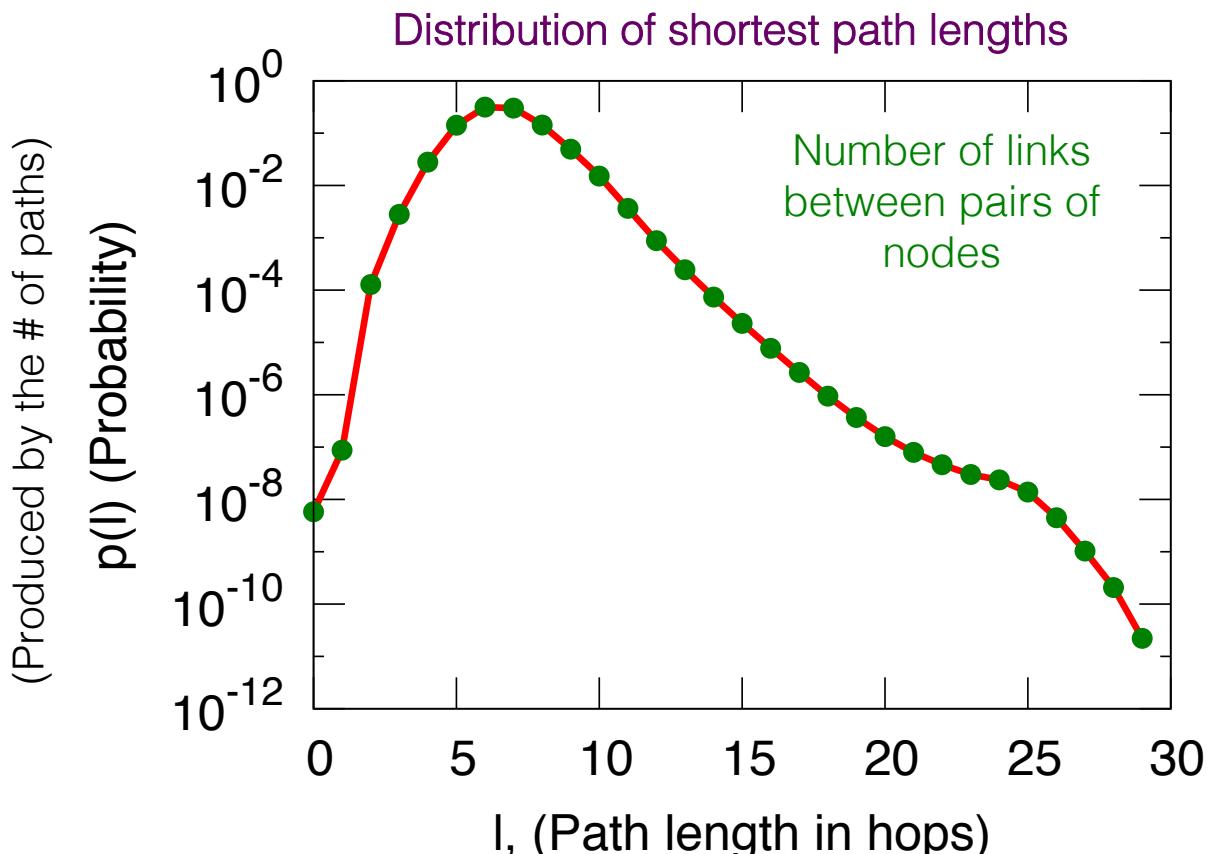


MSN: Clustering



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

MSN: Diameter



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

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

nodes as we do BFS out of a random node

MSN: Key Network Properties

Degree distribution: *Heavily skewed*
avg. degree = 14.4

Path length: 6.6

Clustering coefficient: 0.11

Are these values “expected”?

Are they “surprising”?

To answer this we need a model

Next lecture

Next Lecture

- Can we explain those properties using network models?
 - The random graph model and its properties
 - Small-world phenomenon
- Power-law degree distribution
 - Preferential Attachment model

Thank You!

