

# Closeness and betweenness

Introduction to Network Science

Carlos Castillo

Topic 11



Universitat  
Pompeu Fabra  
*Barcelona*

# Sources

- [Networks, Crowds, and Markets](#) Ch 3.6B
- Barabási 2016 Section 9.3.2
- P. Boldi and S. Vigna: [Axioms for Centrality](#) in Internet Mathematics 2014.
- Esposito and Pesce: [Survey of Centrality](#) 2015.
- C. Castillo: [Other centrality slides](#) 2016

# Types of centrality measure

- Spectral
  - HITS
  - PageRank
- **Non-spectral**
  - Degree
  - Closeness and harmonic closeness
  - Betweenness

# Is u a well-connected person?

- Degree: u has many connections
- Eigenvector: u is connected to the well-connected
- **Closeness:** u is close to many people
  - Average distance from u is small
- **Betweenness:** many connections pass through u
  - Large number of shortest paths pass through u

# Closeness

- Distance between two nodes is  $d(u,v)$
- **Closeness** is the reciprocal of distances

$$\text{closeness}(u) = \frac{1}{\sum_{v \in V, v \neq u} d(u, v)}$$

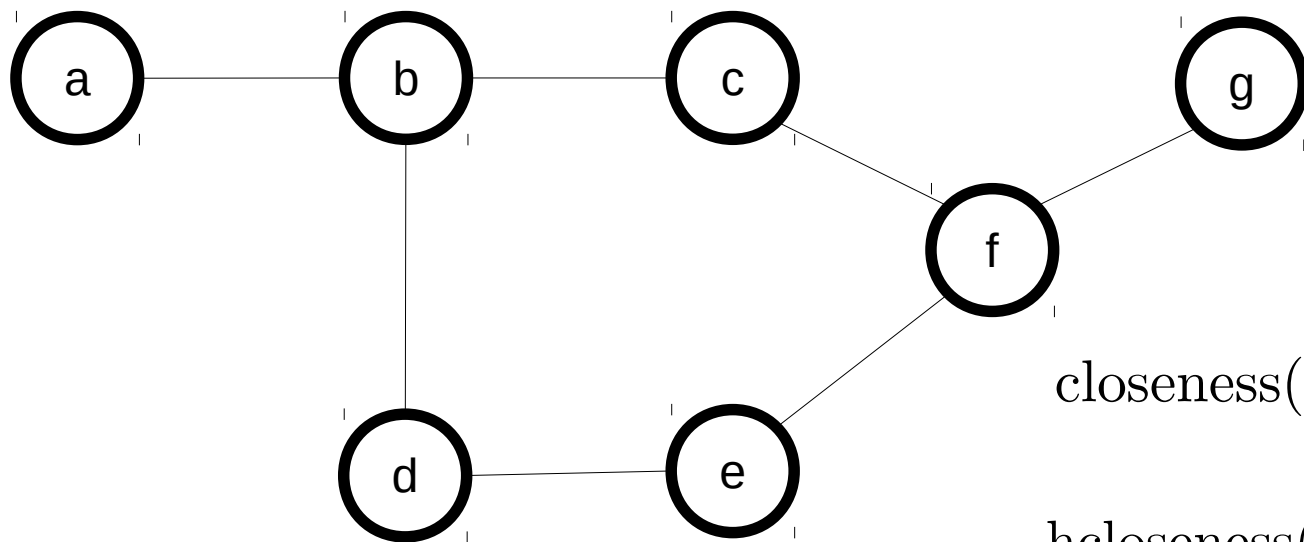
- Some graphs are not connected, in that case  $d(u,v)$  can be  $\infty$ ; setting  $1/\infty = 0$  one can define the **harmonic closeness**:

$$\text{hcloseness}(u) = \sum_{v \neq u} \frac{1}{d(u, v)}$$

# Try it!

*Compute closeness and harmonic closeness for all the nodes*

*$d(u,v) = 1$  if  $v$  is a neighbor of  $u$*

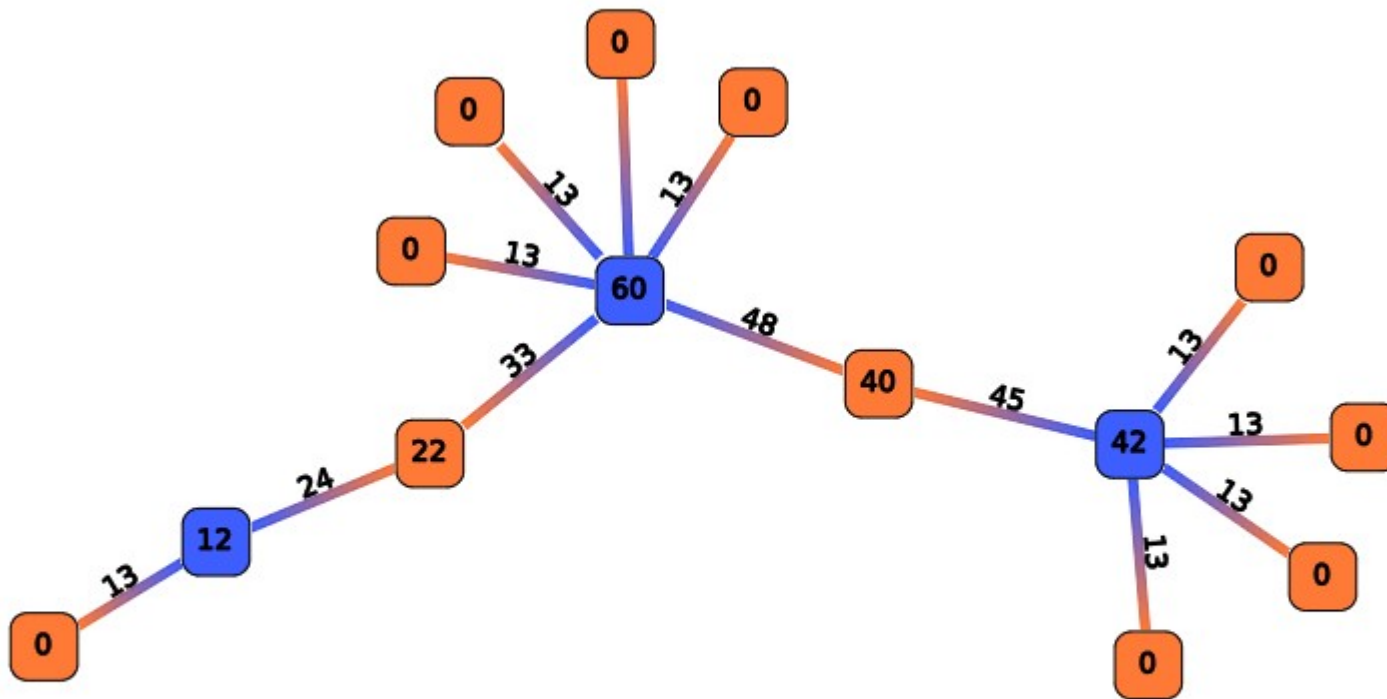


$$\text{closeness}(u) = \frac{1}{\sum_{v \in V, v \neq u} d(u, v)}$$

$$\text{hcloseness}(u) = \sum_{v \neq u} \frac{1}{d(u, v)}$$

# Node Betweenness

A node has high betweenness if it participates in many shortest-paths

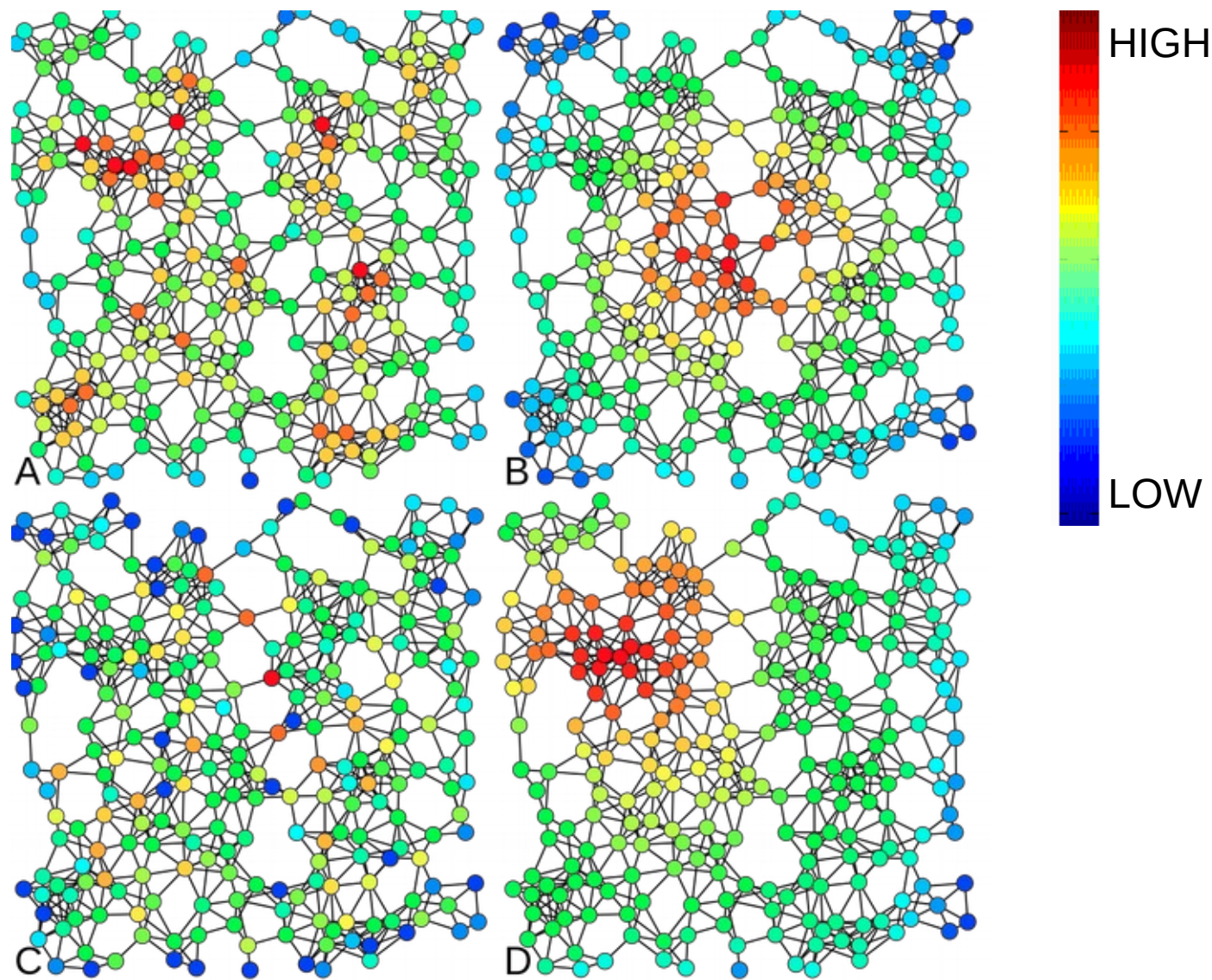


A: Degree

B: Closeness

C: Betweenness

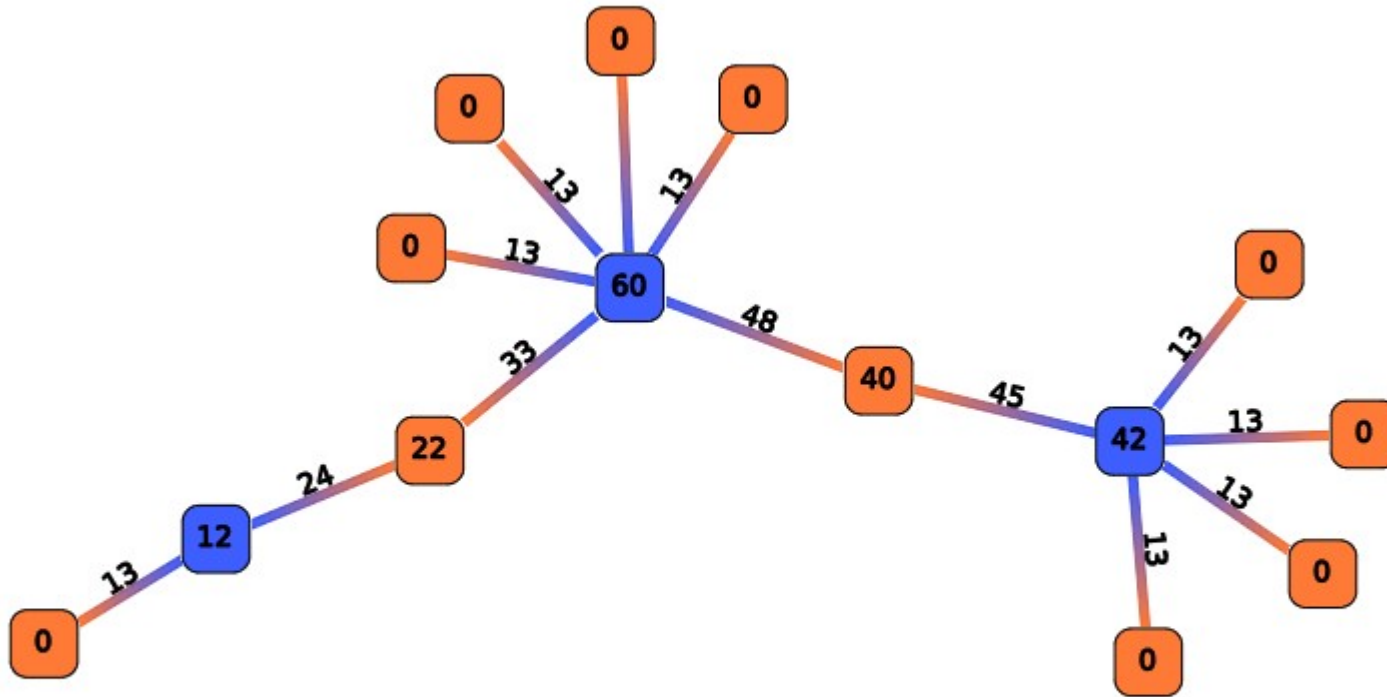
D: PageRank





# Edge Betweenness

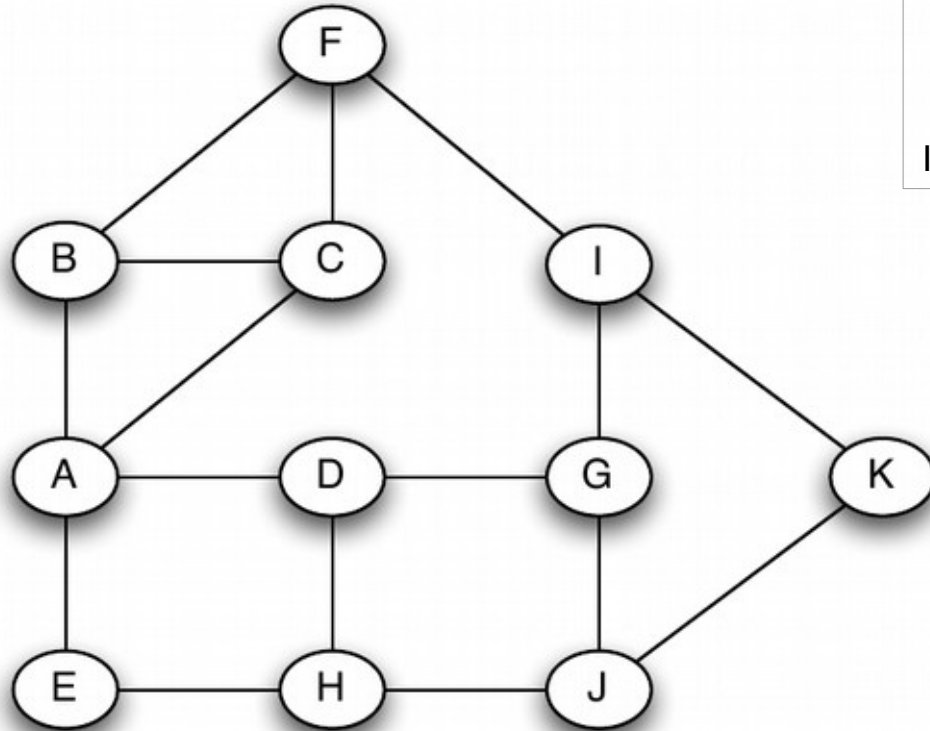
An **edge** has high betweenness if it is part of many shortest-paths ... how to compute this efficiently?



# Algorithm [Brandes, Newman]

- For every node  $u$  in  $V$ 
  - Layer the graph performing a BFS from  $u$
  - For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
    - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
  - For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by reverse layer
    - Score to distribute = 1 + score from children
    - Add score to parent edges in proportion to  $s(v)$
- In the end divide all edge scores by two

# Example

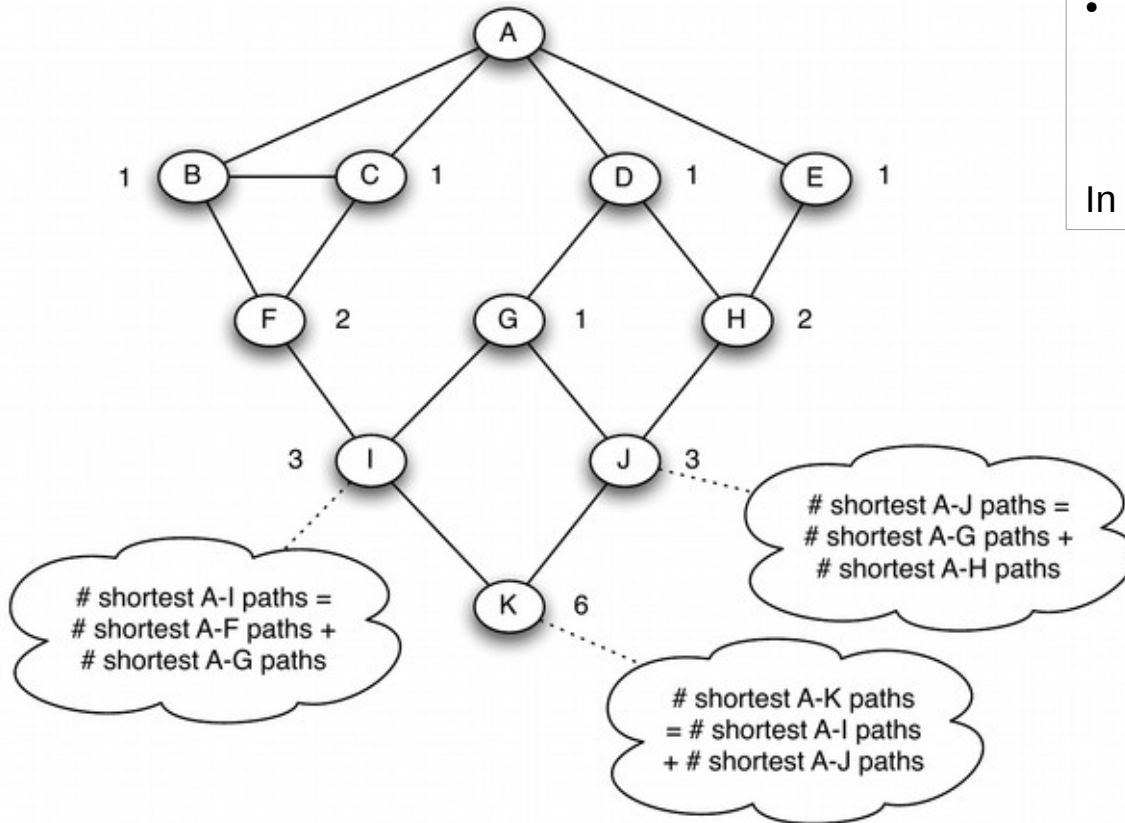


For every node  $u$  in  $V$

- Layer the graph performing a BFS from  $u$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
  - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by reverse layer
  - Score to distribute =  $1 + \text{score from children}$
  - Add score to distribute to parent edges in proportion to  $s(v)$

In the end divide all edge scores by two

# Example



For every node  $u$  in  $V$

- Layer the graph performing a BFS from  $u$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
  - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by reverse layer
  - Score to distribute =  $1 + \text{score from children}$
  - Add score to distribute to parent edges in proportion to  $s(v)$

In the end divide all edge scores by two

All nodes in layer 1 get  $s(v)=1$

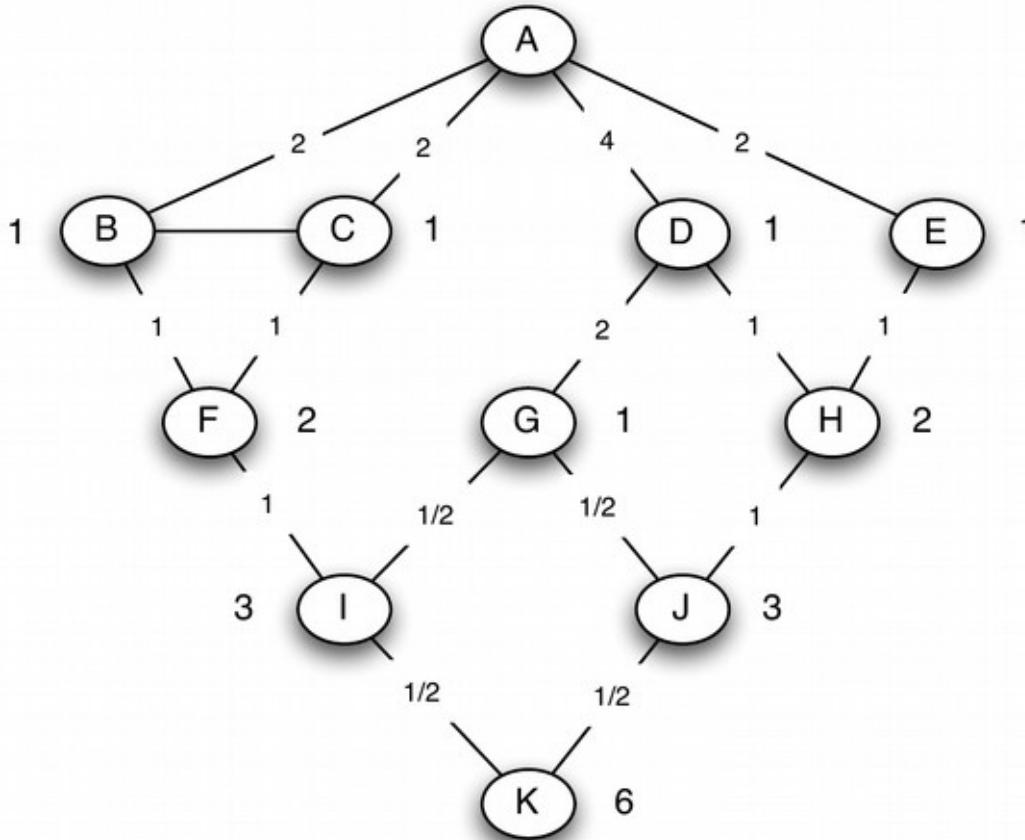
Remaining nodes: simply add  $s(\cdot)$  of their parents

# Example

For every node  $u$  in  $V$

- Layer the graph performing a BFS from  $u$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
  - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
- **For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by rev. layer**
  - **Score to distribute = 1 + score from children**
  - **Add score to distribute to parent edges in proportion to  $s(v)$**

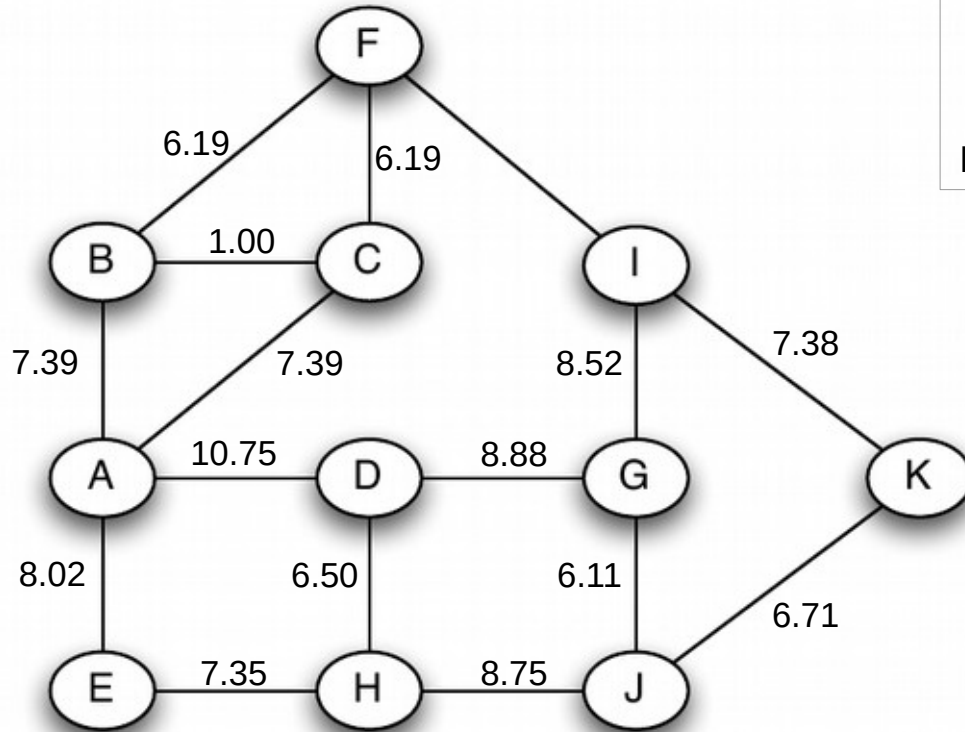
In the end divide all edge scores by two



Nodes without children distribute a score of 1

Other nodes distribute 1 + whatever they receive from their children

# Example



For every node  $u$  in  $V$

- **Layer the graph performing a BFS from  $u$**
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
  - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by reverse layer
  - Score to distribute =  $1 + \text{score from children}$
  - Add score to distribute to parent edges in proportion to  $s(v)$

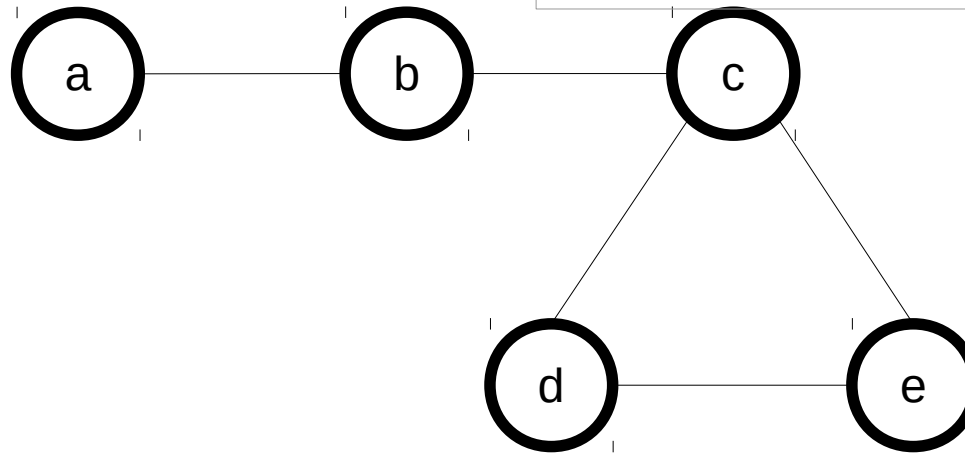
In the end divide all edge scores by two

Computed using NetworkX  
(edge betweenness)

# Try it!

*Try to compute it by inspection first*

*Then use the algorithm;  
you should get the same results*



For every node  $u$  in  $V$

- Layer the graph performing a BFS from  $u$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by layer
  - Assign to  $v$  a number  $s(v)$  indicating how many shortest paths from  $u$  arrive to  $v$
- For every node  $v$  in  $V$ ,  $v \neq u$ , sorted by reverse layer
  - Score to distribute =  $1 + \text{score from children}$
  - Add score to distribute to parent edges in proportion to  $s(v)$

In the end divide all edge scores by two

# Application: the Girvan-Newman algorithm (more on this later)

- Repeat:
  - Compute edge betweenness
  - Remove edge with larger betweenness

