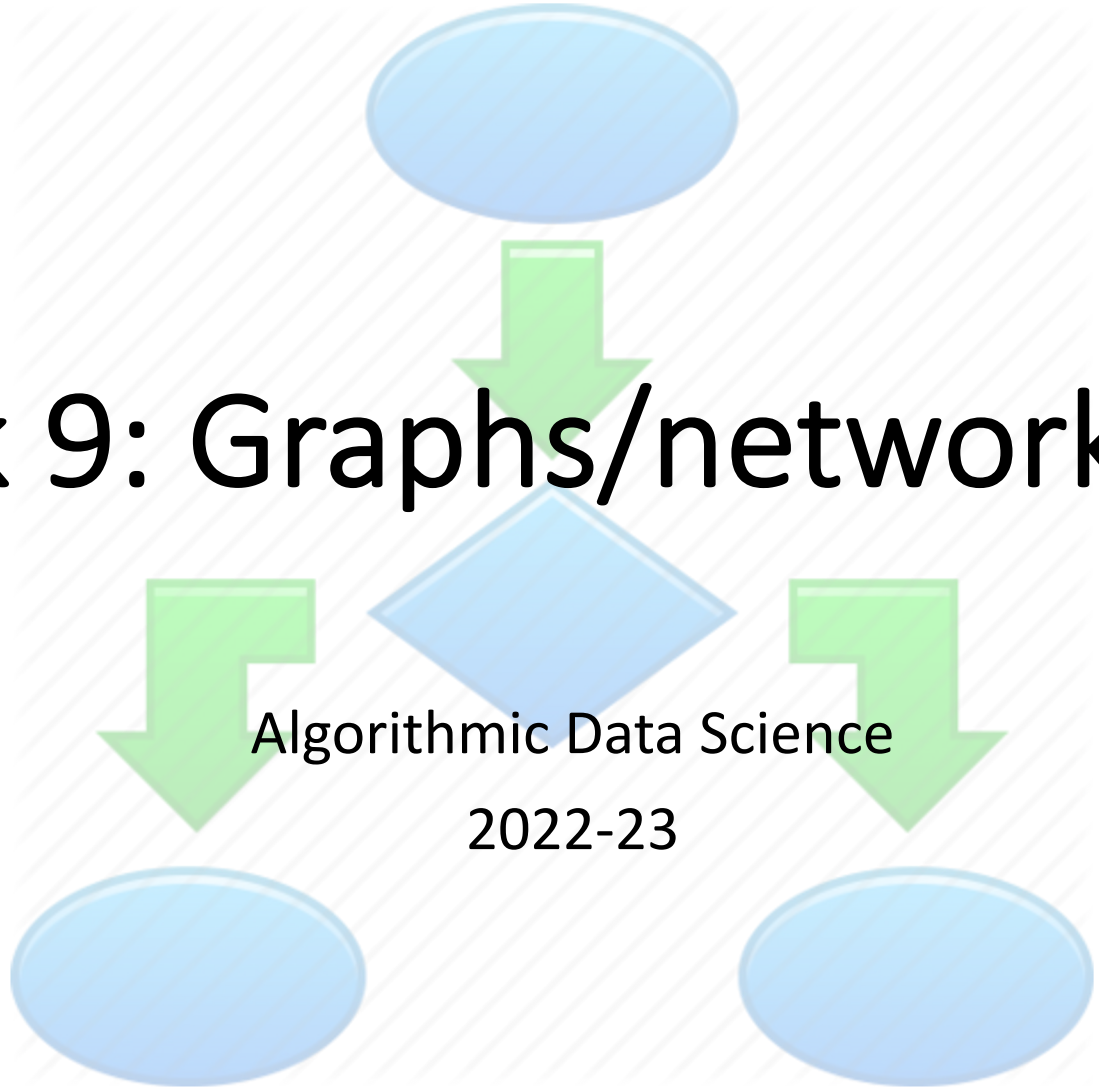
Week 9: Graphs/networks 2



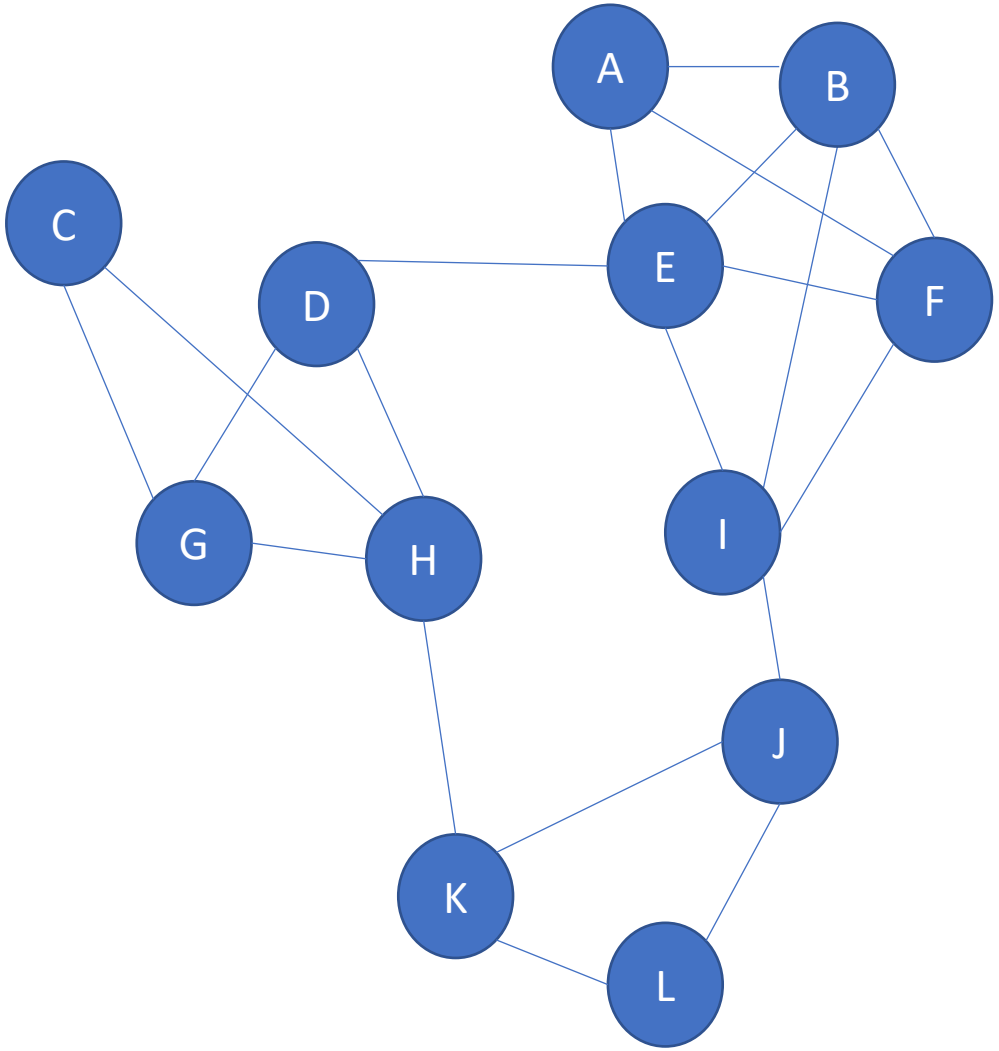
Algorithmic Data Science

2022-23

UNIVERSITY OF SUSSEX

Dr Adam Barrett

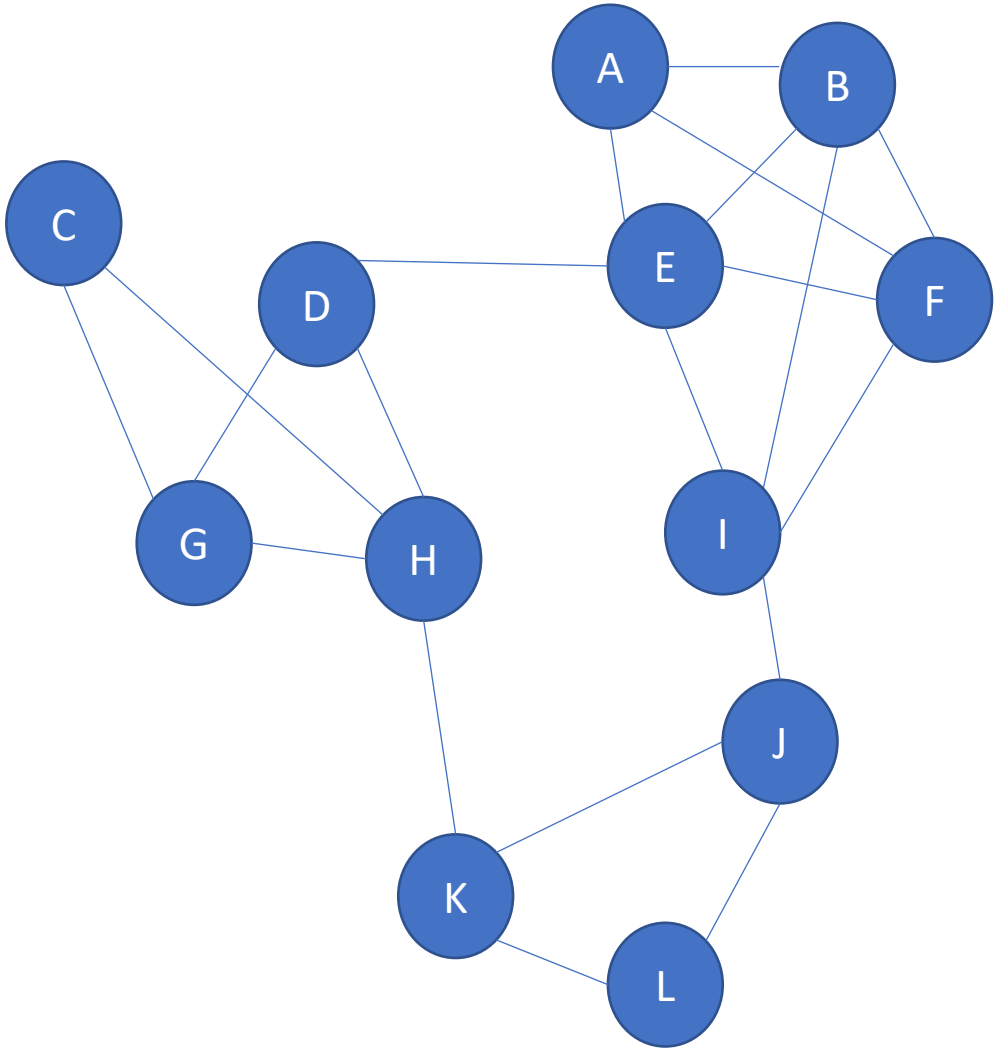
Warm up



Suppose breadth first search is carried out from C. What order are the vertices (nodes) discovered in? (Assume nodes are listed in alphabetical order.)

What if depth first search is carried out?

Warm up



Suppose breadth first search is carried out from C. What order are the vertices (nodes) discovered in?

C,G,H,D,K,E,J,L,A,B,F,I

What if depth first search is carried out?

C,G,D,E,A,B,F,I,J,K,H,L

Week	Who	Topic
1	Barrett	Data structures and data formats
2	Barrett	Algorithmic complexity. Sorting.
3	Barrett	Matrices: Manipulation and computation
4	Barrett	Similarity analysis
5	Rosas	Processes and concurrency
6	Rosas	Distributed computation
7	Barrett	Map/reduce
8	Barrett	Clustering, graphs/networks
9	**Barrett**	**Graphs/networks, PageRank algorithm**
~~10~~	~~Barrett~~	~~Databases~~ **STRIKE**
11		*independent study*

Week 10: No new lab exercises. No lecture because of strike

- AB in lab sessions on Mon 28th and Tues 29th Nov from approx. 9.30-10.30 to answer questions.

- Online support from Joseph Starkey 10am-12noon Mon 28th, and 9am-11am Tues 29th Nov, in module Zoom room.

- Fernando Rosas will also be on Zoom 10am-11am on Tues 29th to answer questions (bring headphones if accessing this from University computers).

Week 11: Independent study, no labs nor lecture.

Tuesday 6th Dec, 10-11: Online helpdesk with Joseph Starkey in module Zoom room.
Weds 7th Dec, 10-11: Q&A in normal lecture theatre (and Zoom).

Second multiple choice quiz assessment

11am on Tues 29th Nov to 5pm on Weds 30th Nov (start it before **4pm on Weds 30th**)

Covers material from lectures 5-9; worth 10% of your mark for the module.

Marks released on the Thursday morning.

Overview

- Current topic:
  - **Graphs / networks**
    - Breadth-first and depth-first searching DONE
    - Mining social media networks TODAY
    - Dijkstra's algorithm for all pairs shortest paths INTRO LAST WEEK, EXAMPLE TODAY

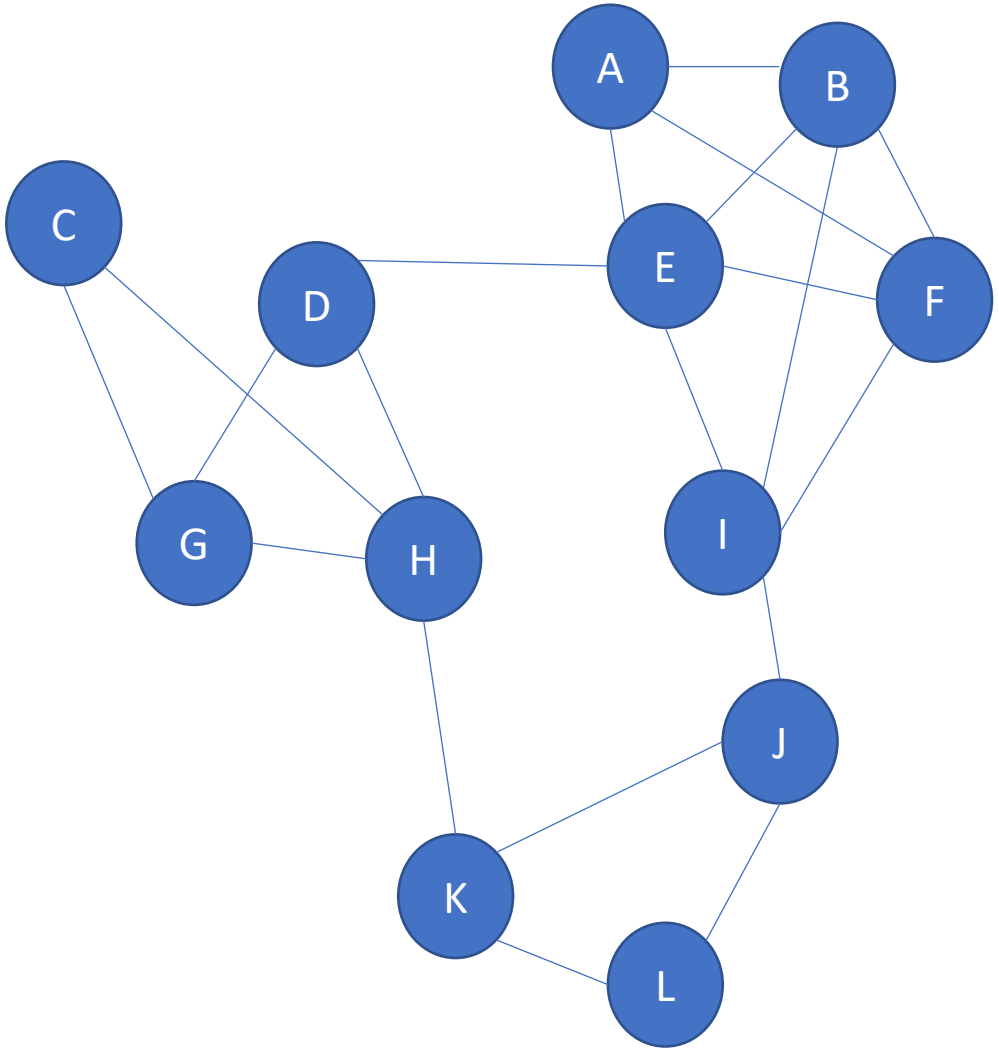
- SECOND PART OF LECTURE: PageRank algorithm

Mining Social Media Networks

- Users of social media networks tend to form internal communities or clusters. These typically correspond to groups of friends at school or groups of researchers interested in the same topic. How can we identify these communities?

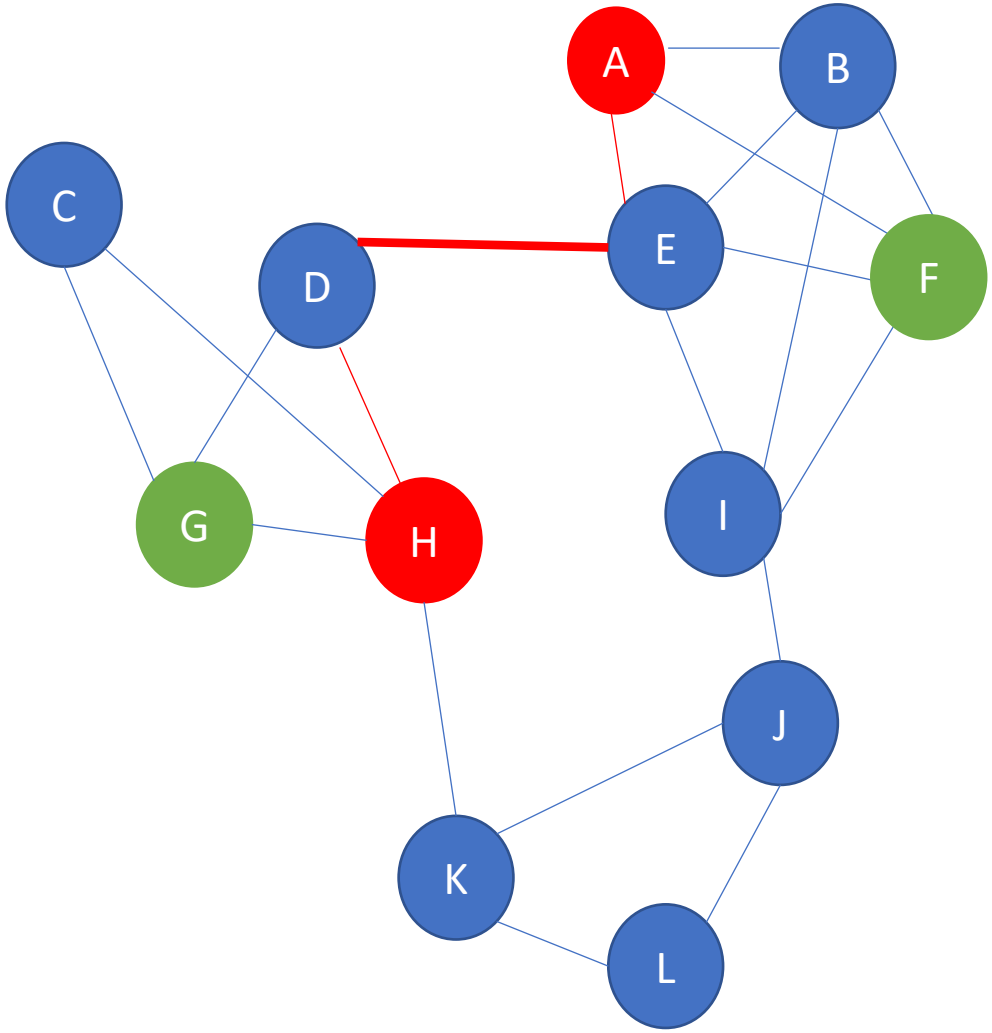
- A social network is a graph. Vertices are users. Edges are *friendships* between them. Some social networks (e.g., Facebook) are undirected graphs whereas others (e.g., Twitter) are directed graphs.

Identifying communities of users



- What communities exist within this social network?
- How does identifying communities differ from similarity analysis?

Betweenness



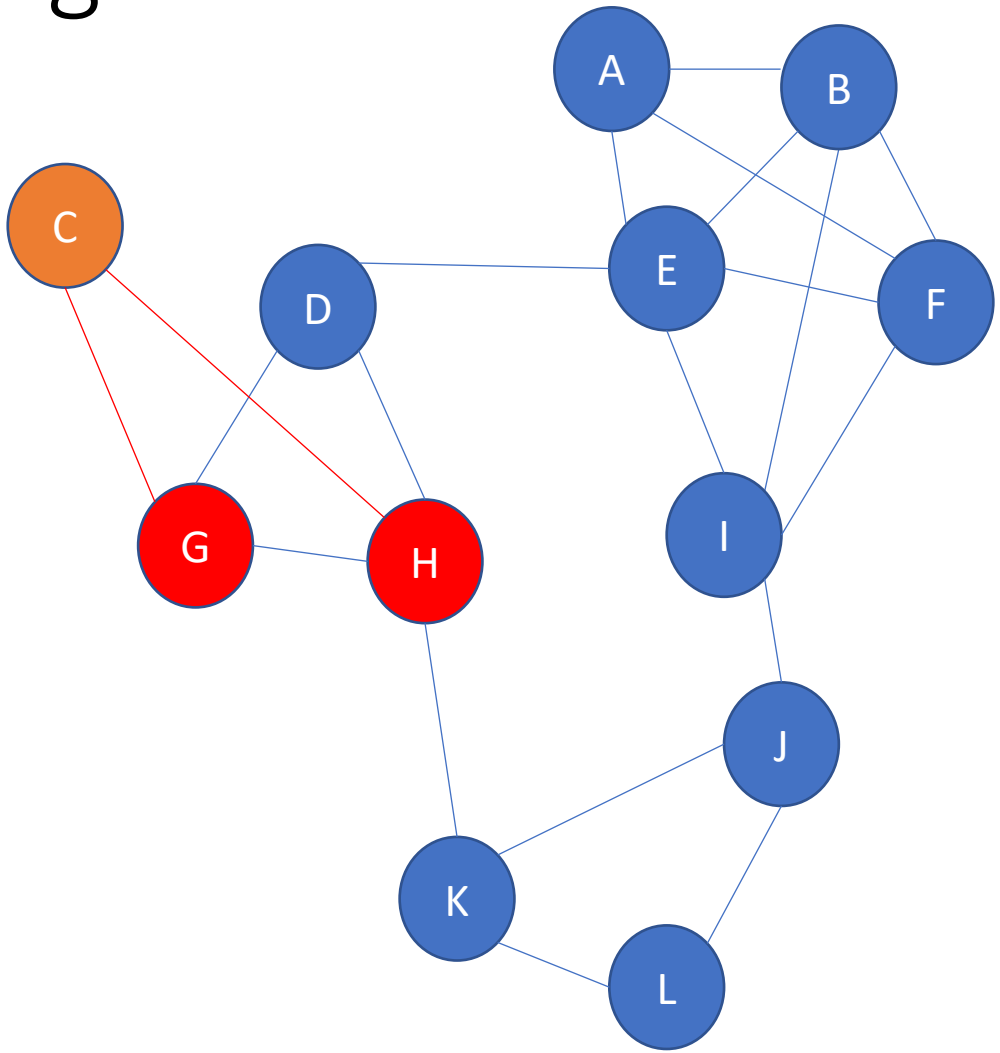
Look at the shortest path from A to H. What does it have in common with the shortest path from F to G?

The *betweenness* of an edge (a,b) is the number of pairs of nodes x and y such that the edge (a,b) lies on the shortest path between x and y.

For example, the edge (D,E) lies on many shortest paths e.g., (D,E), (H,E), (A,D), (G,E), (C,E), (B,D), (F,D), (I,D).....

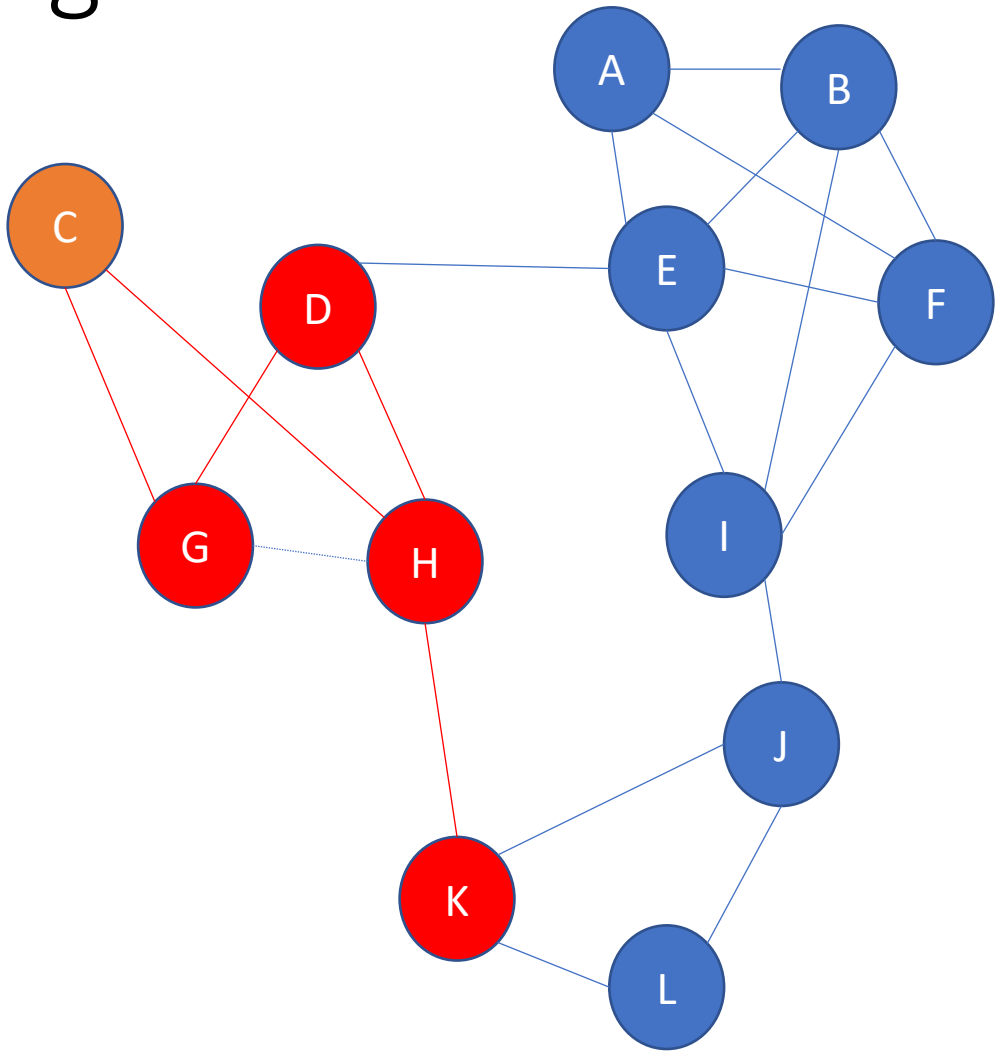
A high betweenness score suggests that an edge runs between 2 different communities.

Calculating betweenness: Girvan-Newman Algorithm



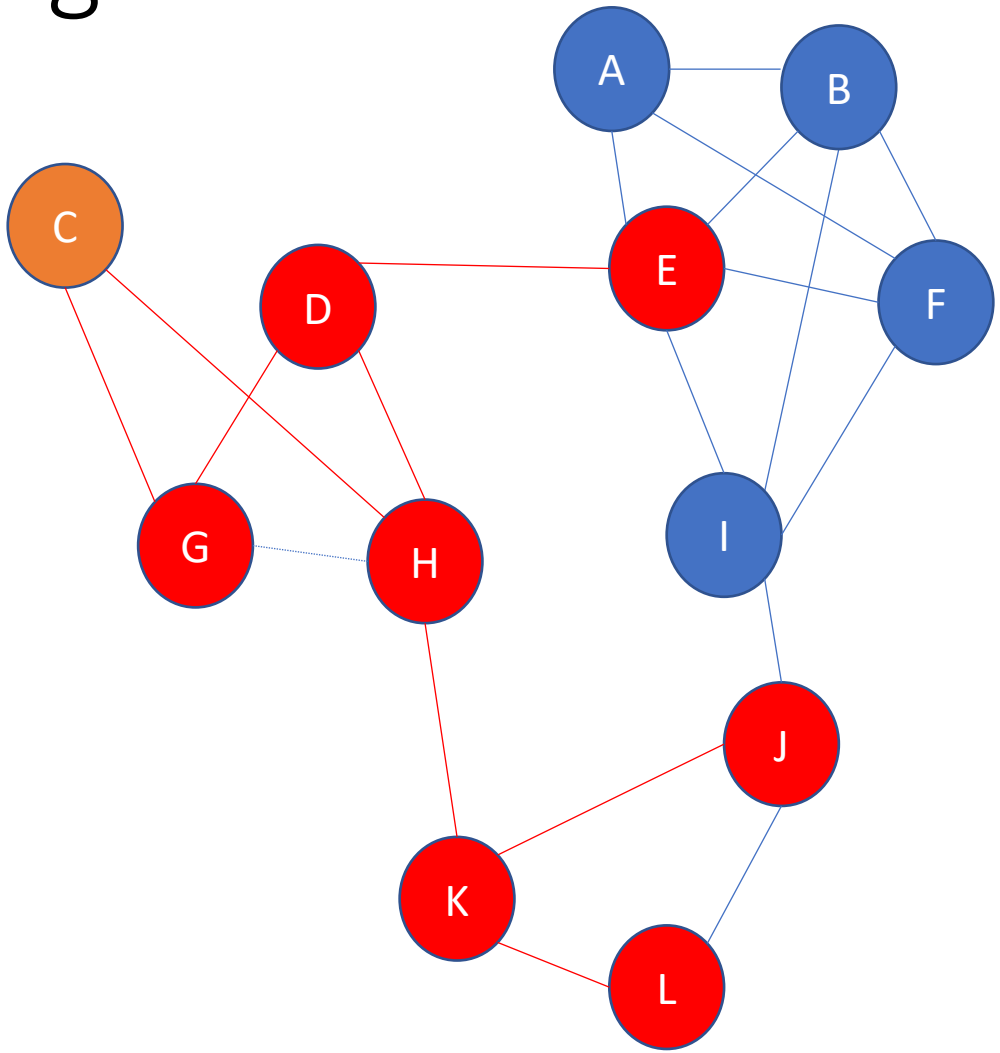
- **Step 1**: Choose a node and perform a breadth-first search from that node

Calculating betweenness: Girvan-Newman Algorithm



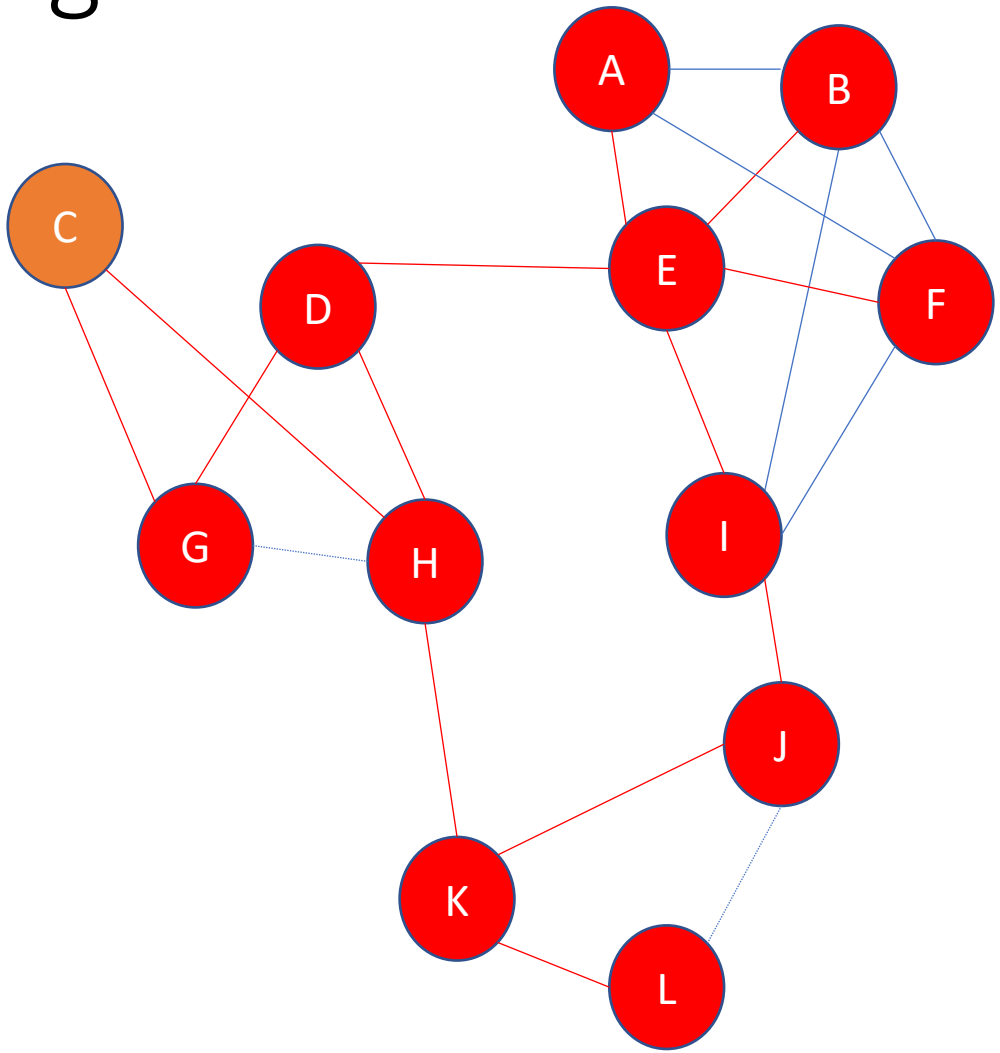
- **Step 1:** Choose a node and perform a breadth-first search from that node

Calculating betweenness: Girvan-Newman Algorithm



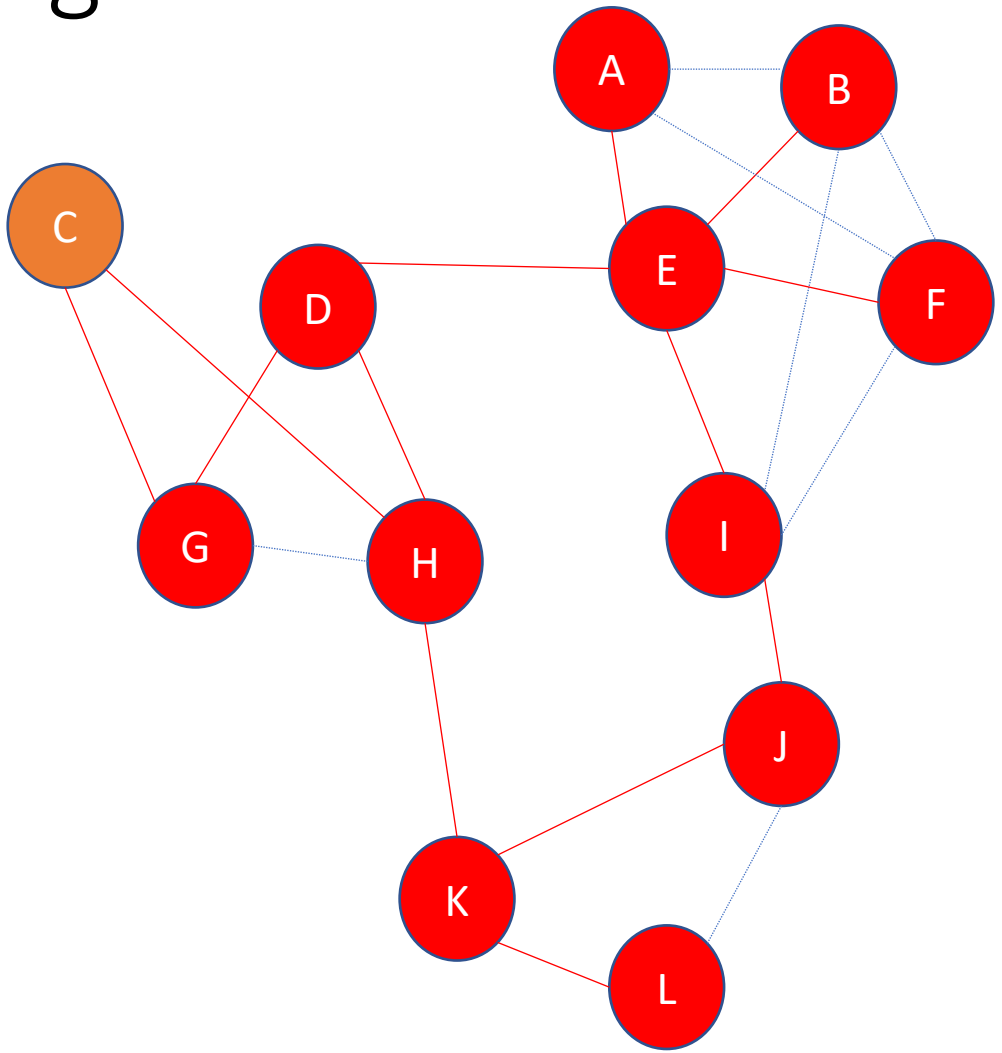
- **Step 1:** Choose a node and perform a breadth-first search from that node

Calculating betweenness: Girvan-Newman Algorithm



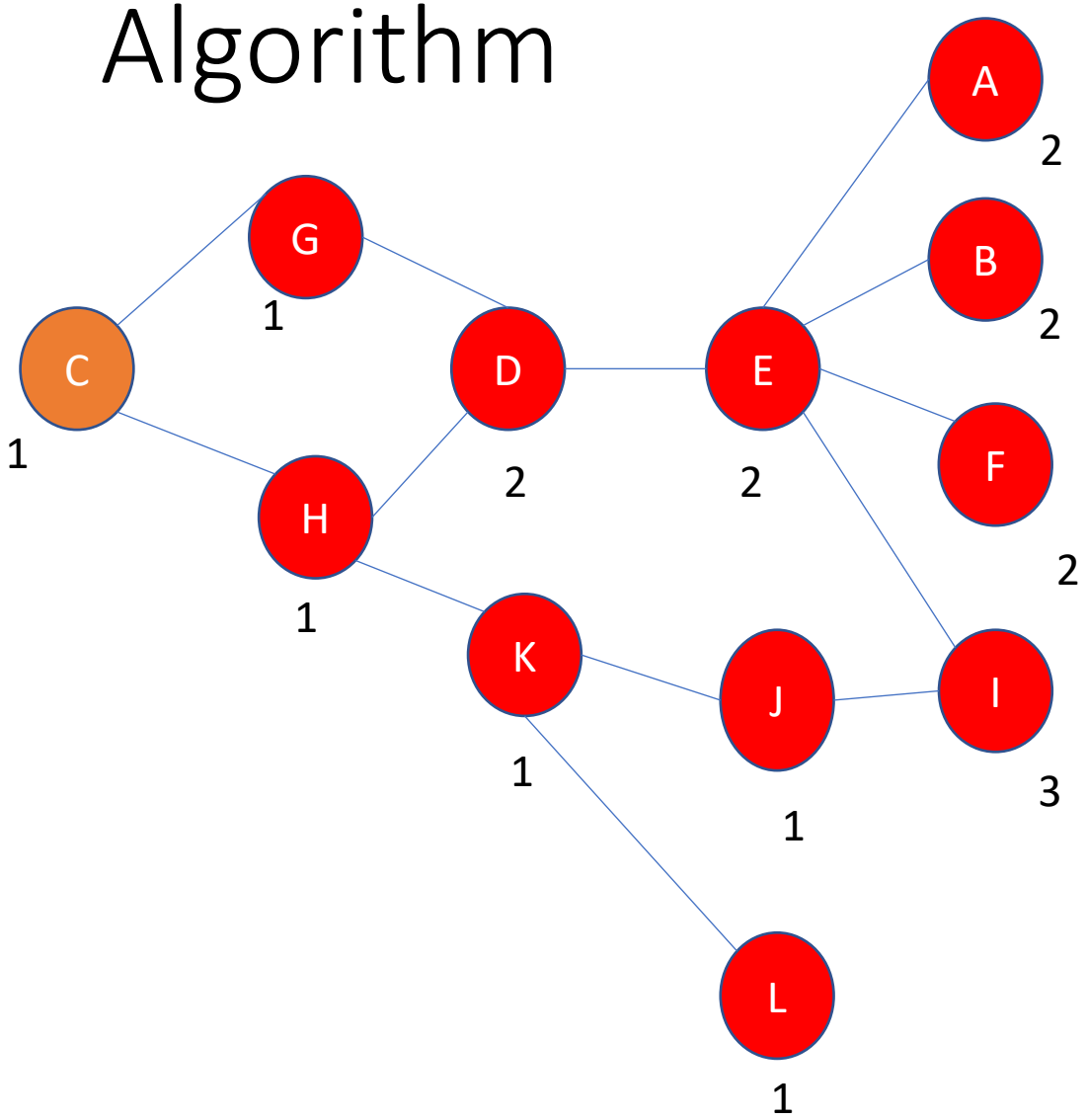
- **Step 1:** Choose a node and perform a breadth-first search from that node

Calculating betweenness: Girvan-Newman Algorithm



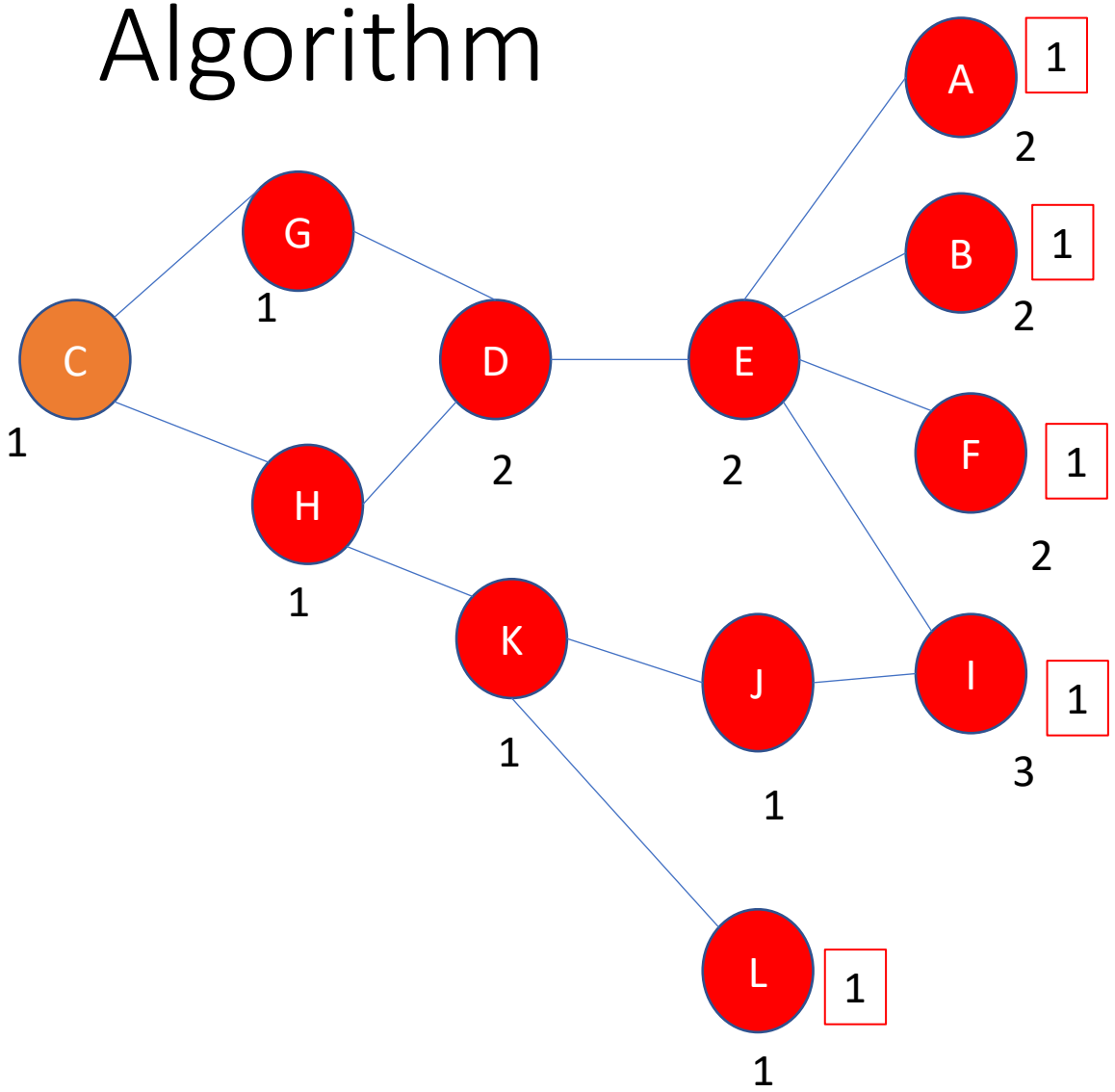
- **Step 1:** Choose a node and perform a breadth-first search from that node

Calculating betweenness: Girvan-Newman Algorithm



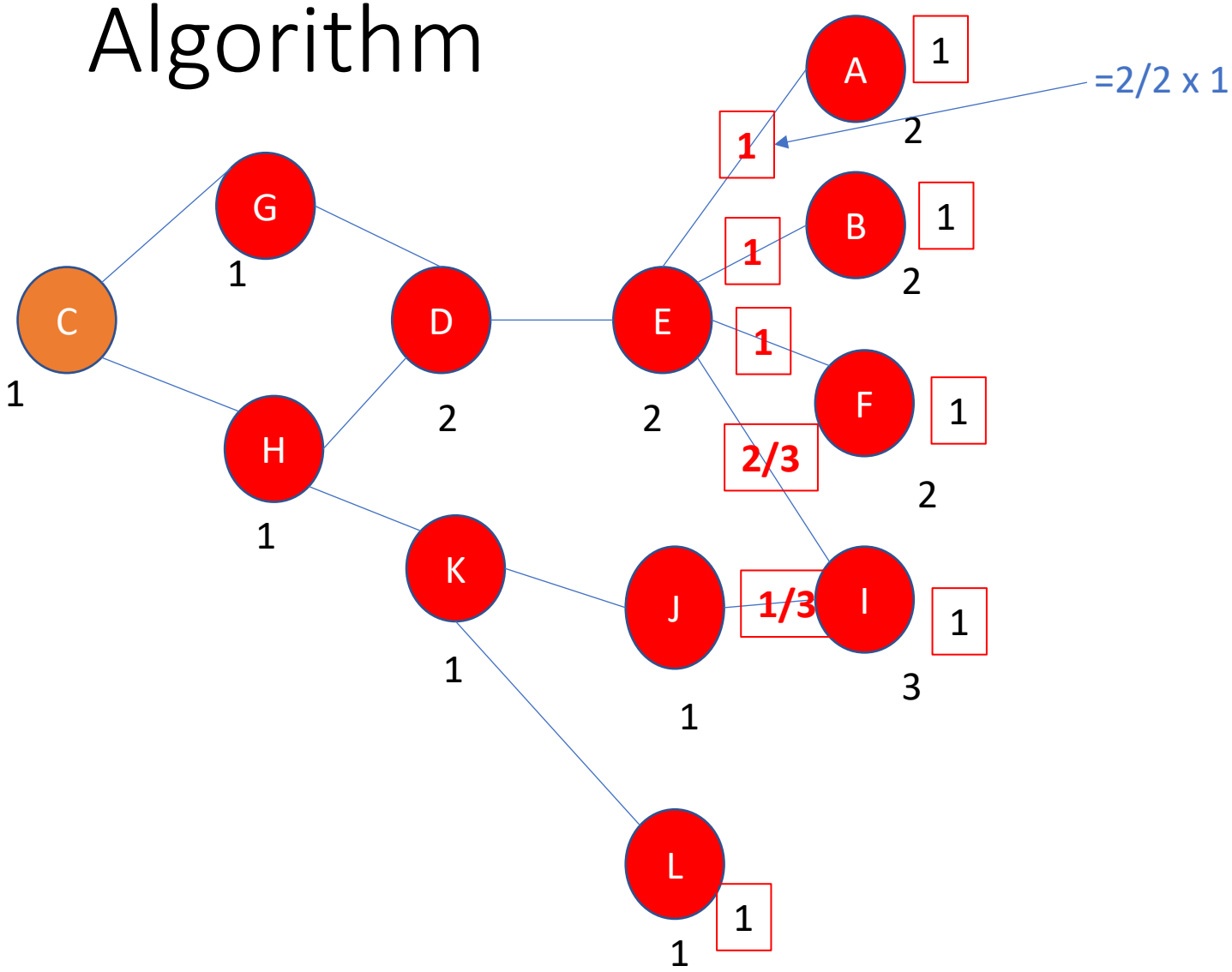
- The **π** data structure gives us the BFS tree rooted at C.
- **Step 2**: Label each node by the number of shortest paths that reach it from the node.
- Start by labelling the root with 1.
- Then from the top down, the label of each node is the sum of its parents

Calculating betweenness: Girvan-Newman Algorithm



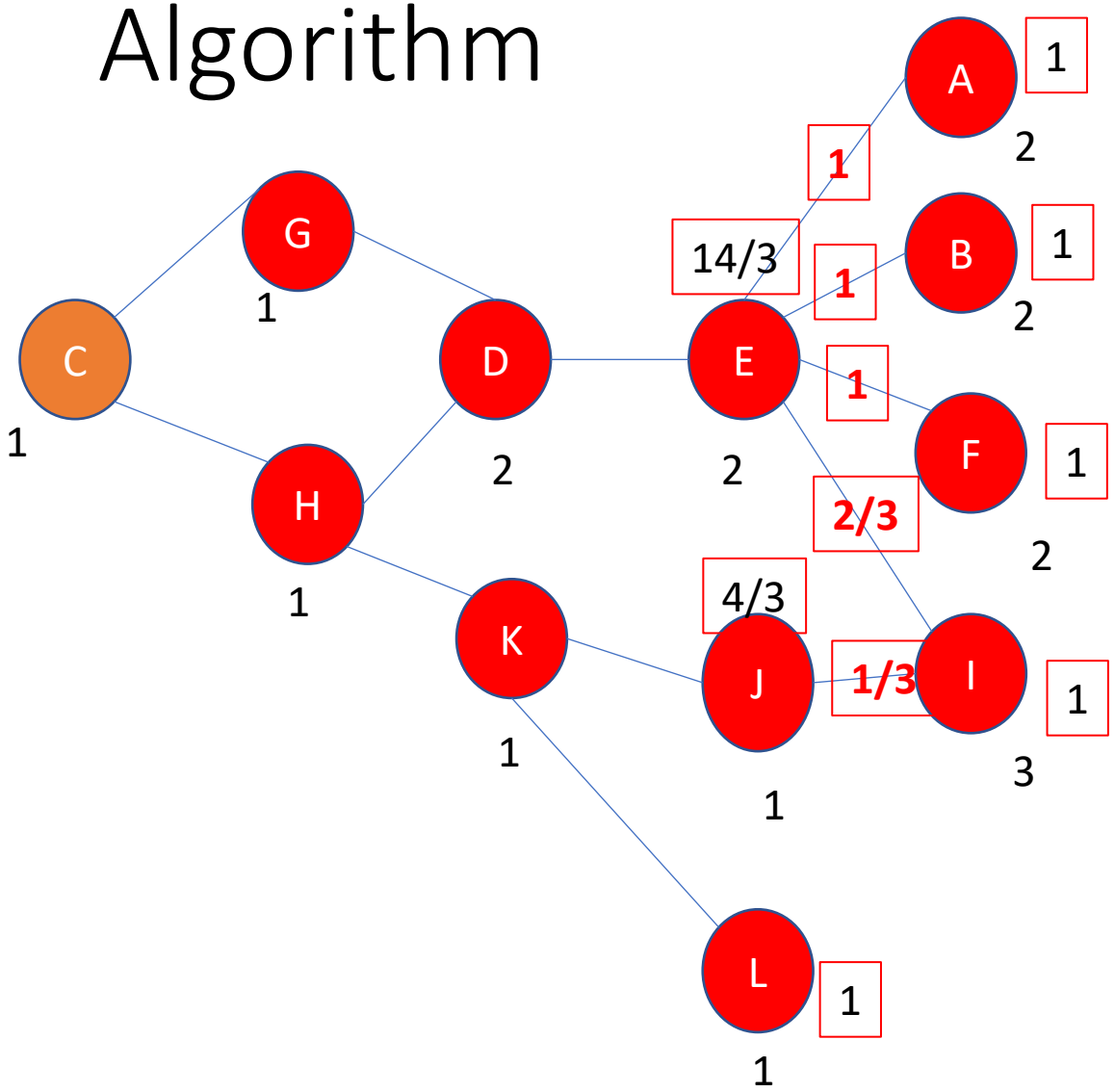
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each leaf is given a credit of 1

Calculating betweenness: Girvan-Newman Algorithm



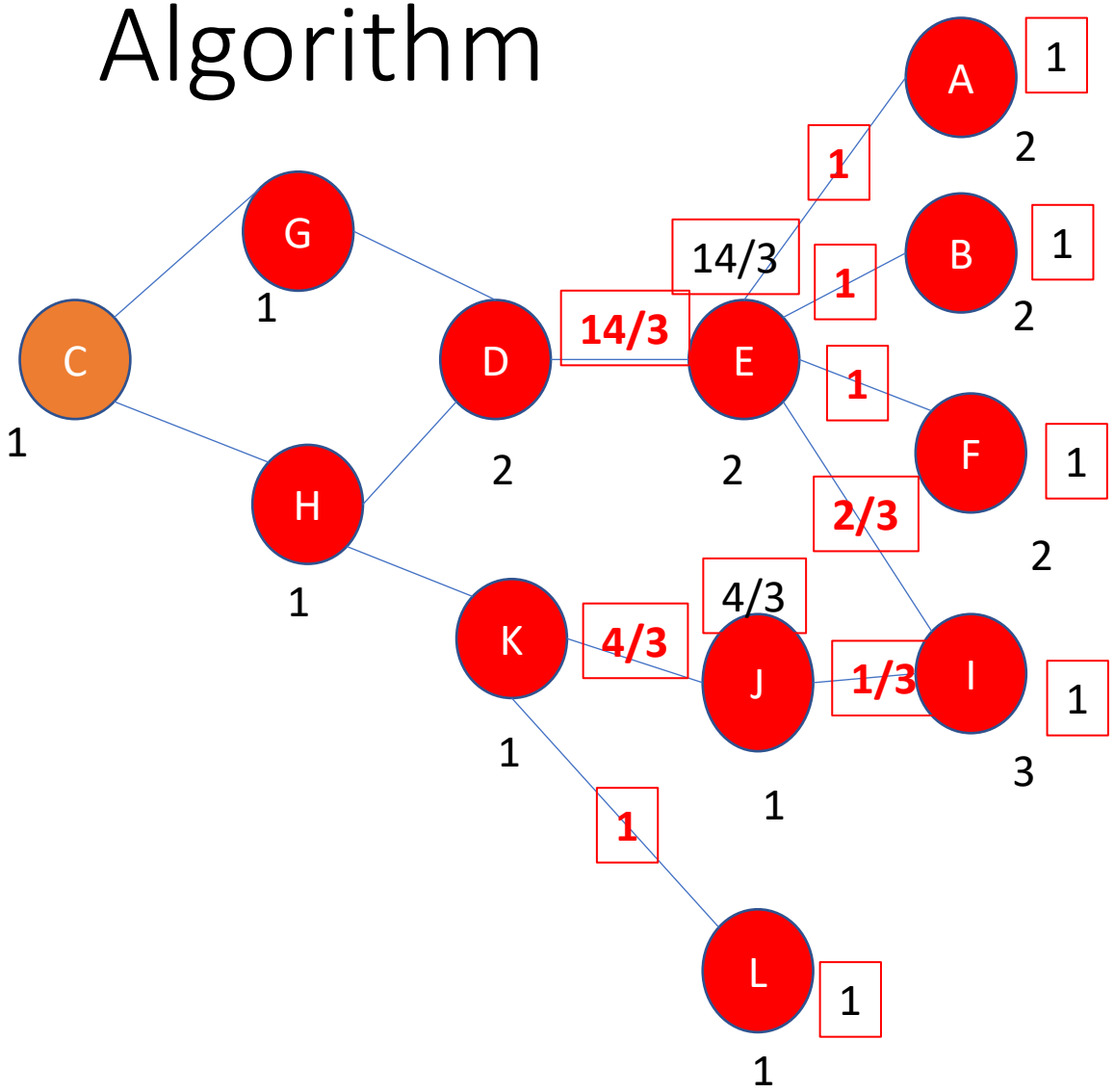
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each edge e entering node Z from above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e

Calculating betweenness: Girvan-Newman Algorithm



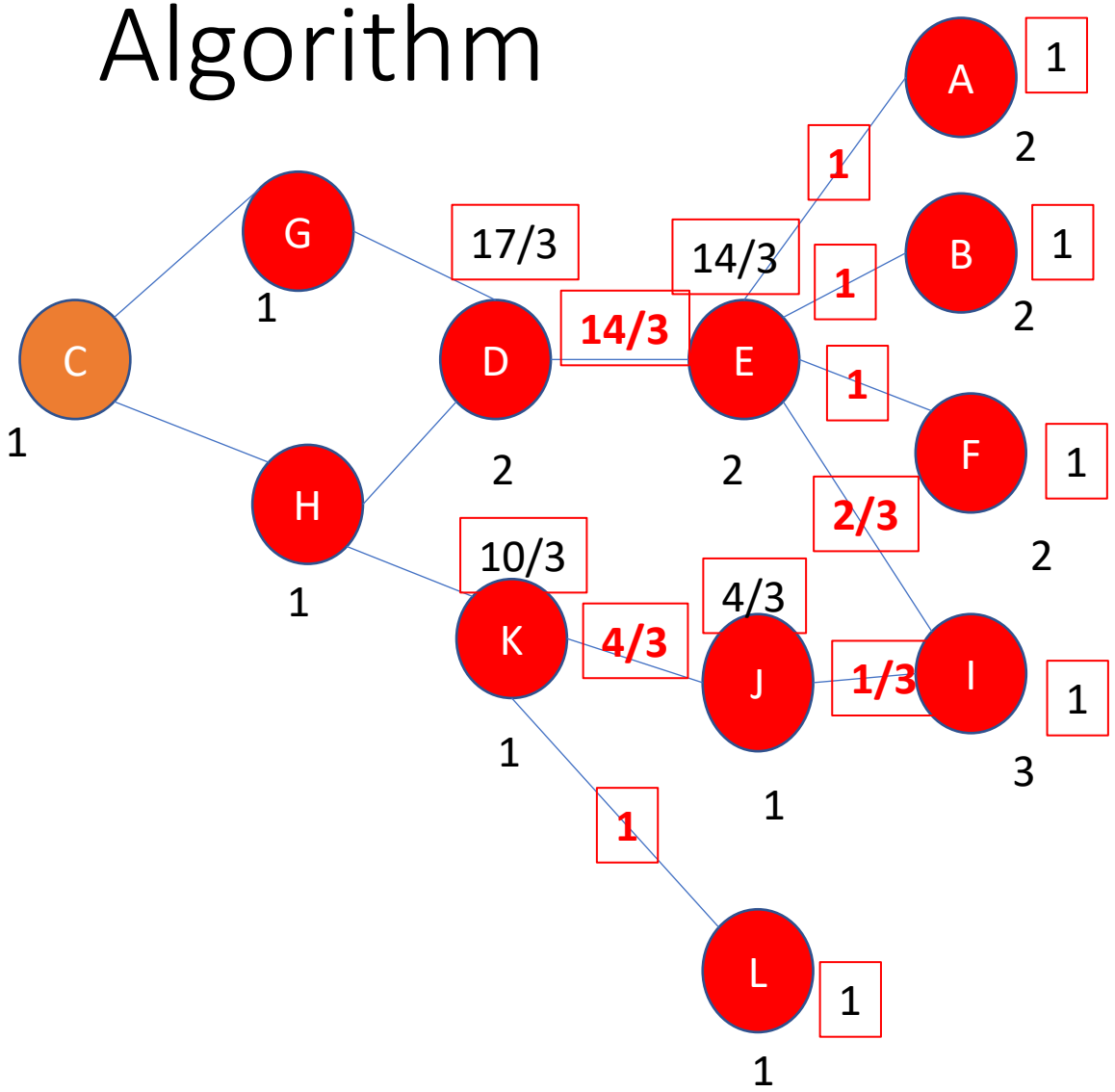
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each node that is not a leaf gets a credit of 1 plus the sum of credits of the edges entering it from below.

Calculating betweenness: Girvan-Newman Algorithm



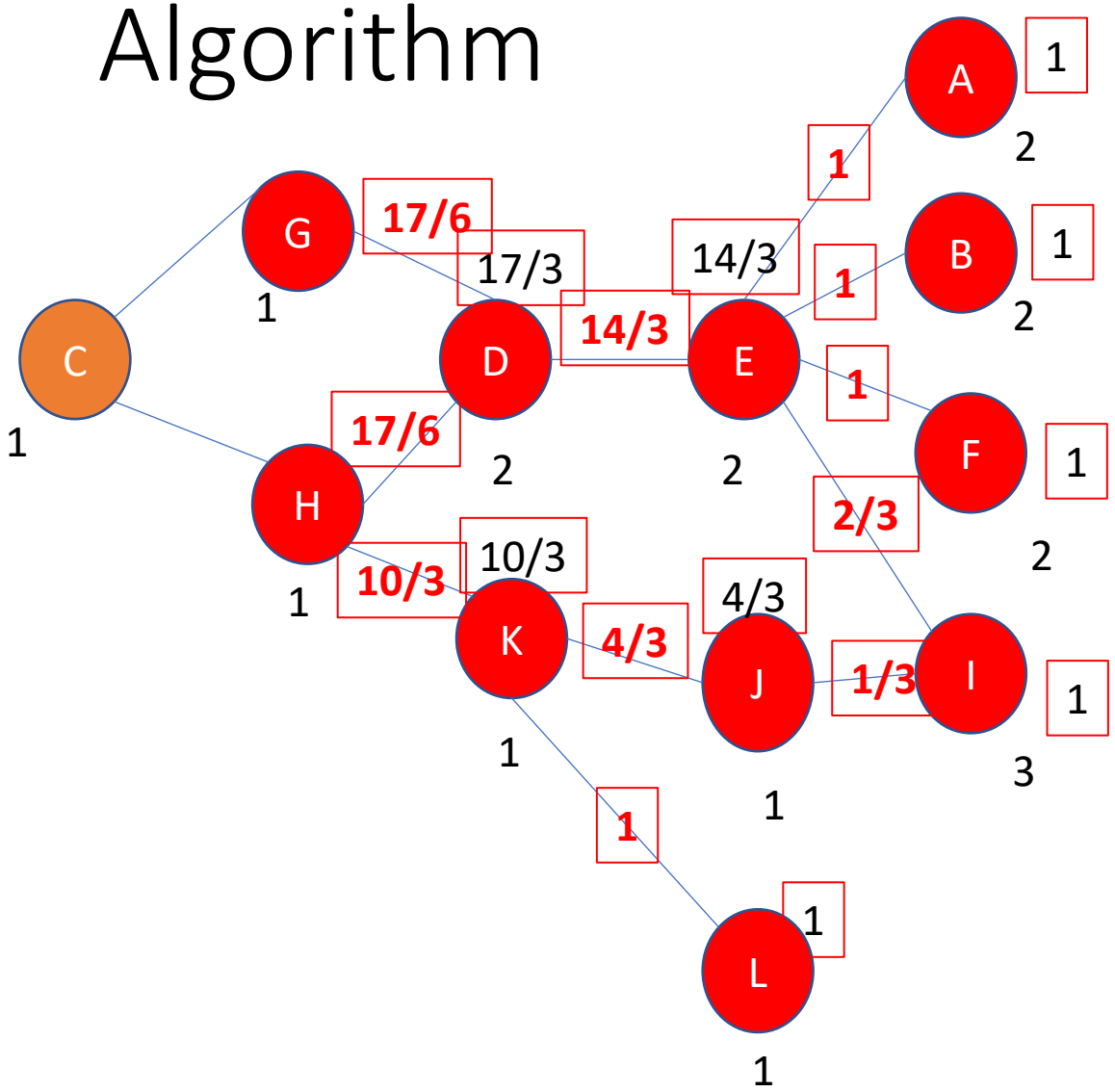
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each edge e entering node Z from above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e

Calculating betweenness: Girvan-Newman Algorithm



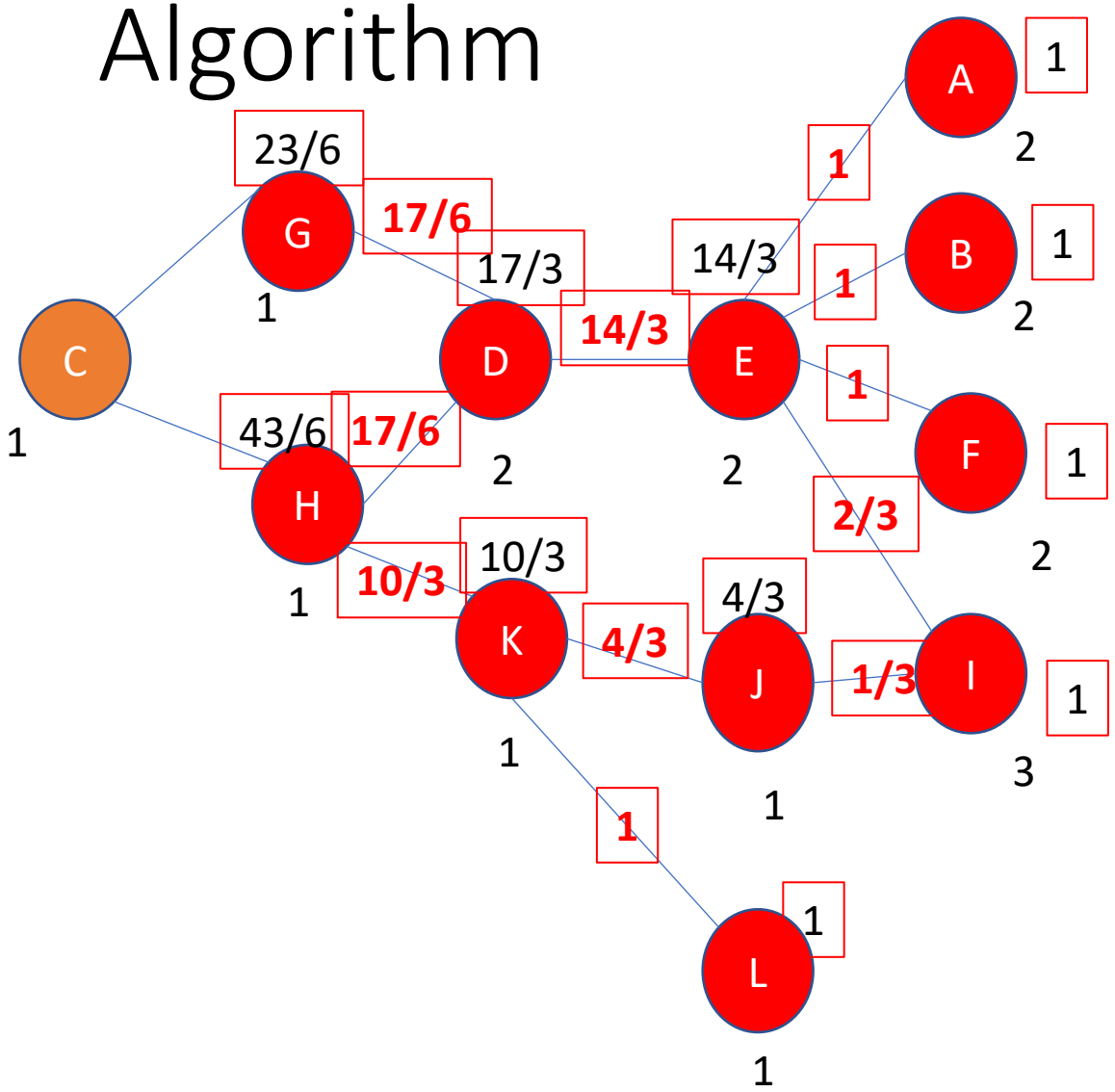
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each node that is not a leaf gets a credit of 1 plus the sum of credits of the edges entering it from below.

Calculating betweenness: Girvan-Newman Algorithm



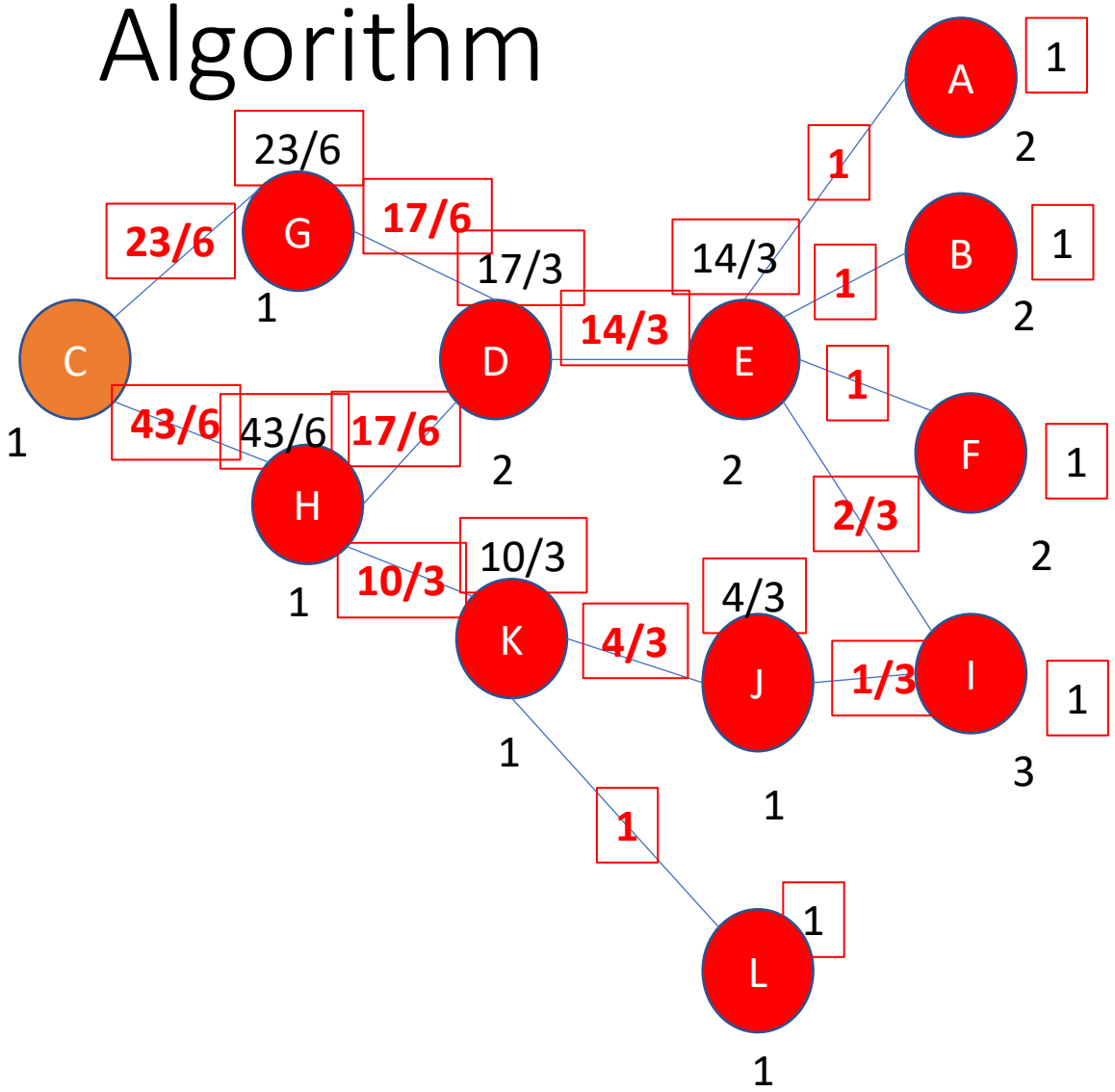
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each edge e entering node Z from above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e

Calculating betweenness: Girvan-Newman Algorithm



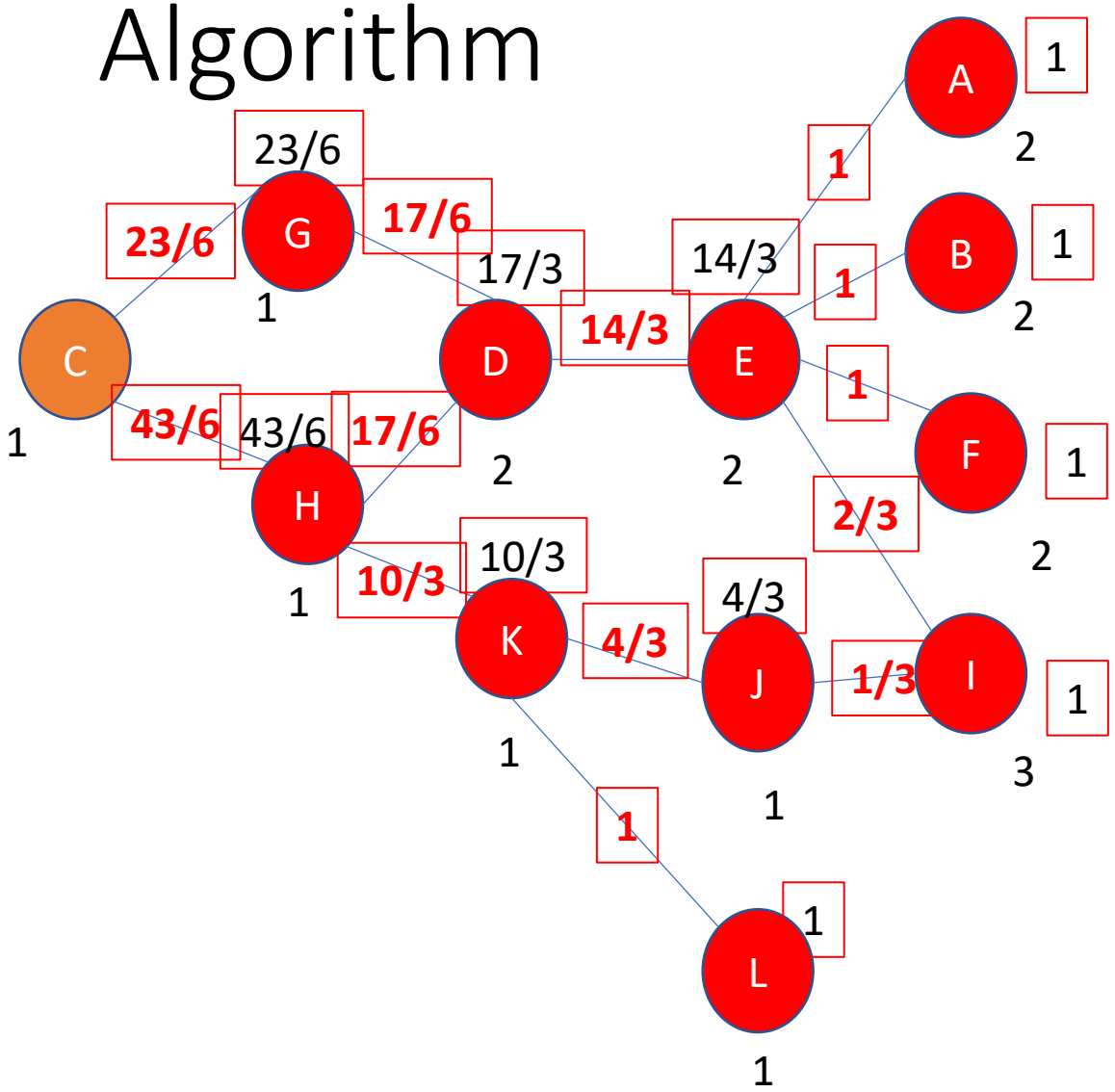
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each node that is not a leaf gets a credit of 1 plus the sum of credits of the edges entering it from below.

Calculating betweenness: Girvan-Newman Algorithm



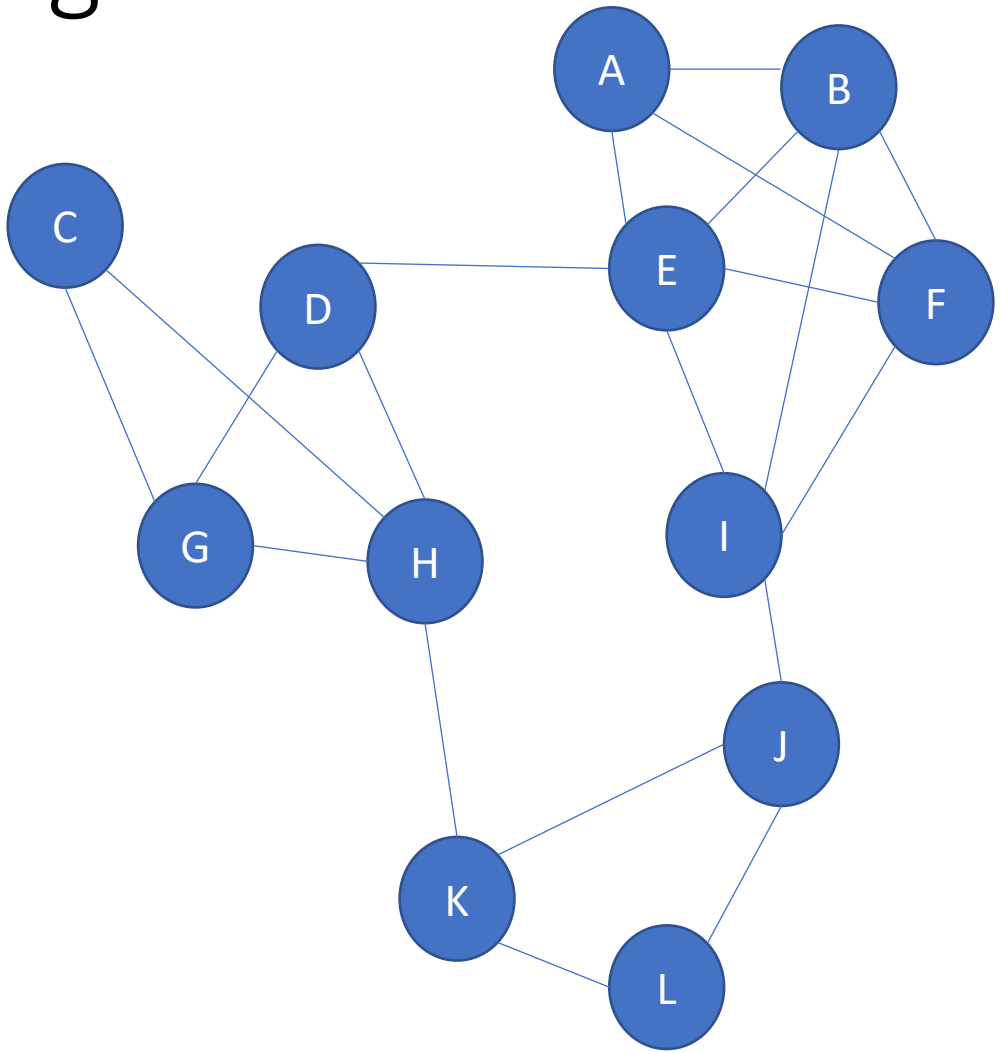
- **Step 3:** Calculate for each edge the sum over all nodes Y of the fraction of shortest paths from the root to Y that go through e.
- Each edge e entering node Z from above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e

Calculating betweenness: Girvan-Newman Algorithm



- **Finally:** Repeat for each node as the root of the BFS tree.
- Sum the betweenness scores for each edge calculated with each node as root
- Divide by 2 (since each shortest path is discovered twice

Calculating betweenness: Girvan-Newman Algorithm

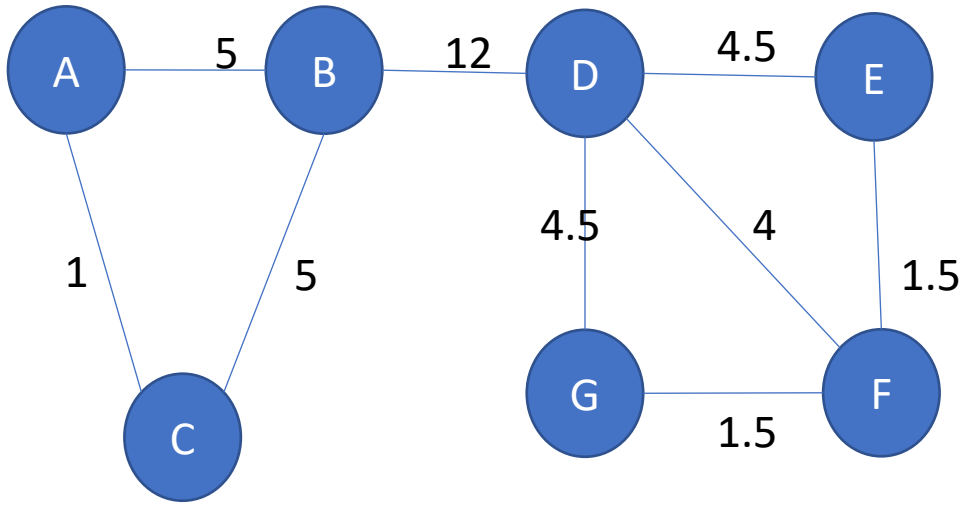


- **Round 2**: Choose a different node and perform a breadth-first search from that node
- We could do this in parallel … one processor for each start node
- Note, however, that each processor needs access to the whole input graph

Analysis of Girvan-Newman Algorithm

- BFS from a single vertex takes O(E) time.
- The labelling steps are also O(E)
- So for V vertices, running time is O(VE)
- In the worst case (for a very dense graph), $E=V^2$ so running time = $O(V^3)$
- This is unfeasible for a large graph (> 1 million users)
- Approximation to betweenness can be found by picking a random set of starting vertices. Running time = $O(E) = O(V^2)$

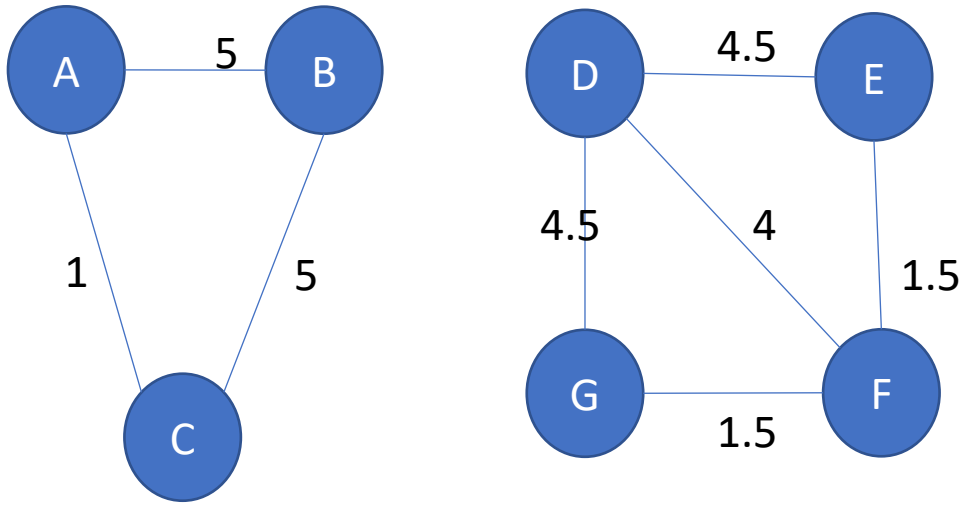
Using betweenness



Given betweenness scores for each edge,

- Remove all edges with betweenness over a certain threshold
- OR remove edges (starting with the highest) until a certain number of clusters are discovered.

Using betweenness

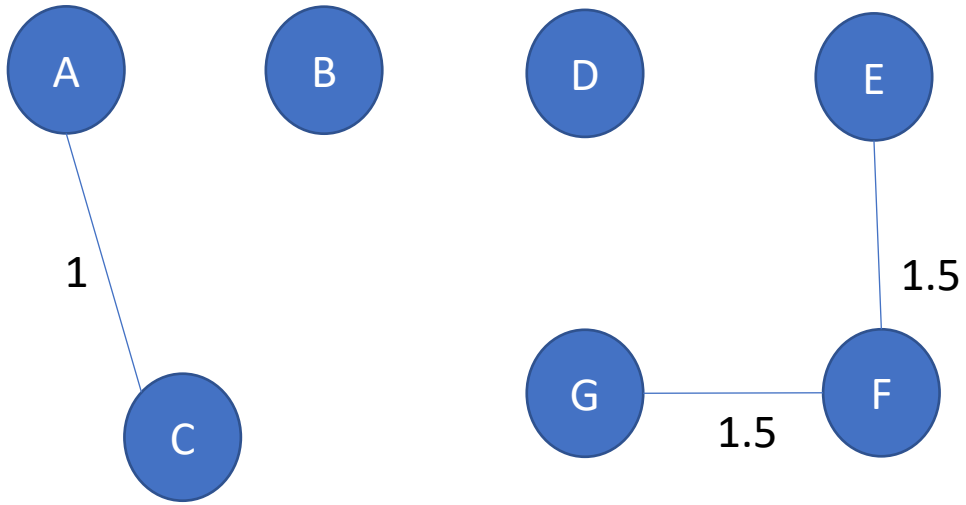


Here we have removed edges where betweenness > 10

Given betweenness scores for each edge,

- Remove all edges with betweenness over a certain threshold
- OR remove edges (starting with the highest) until a certain number of clusters are discovered.

Using betweenness



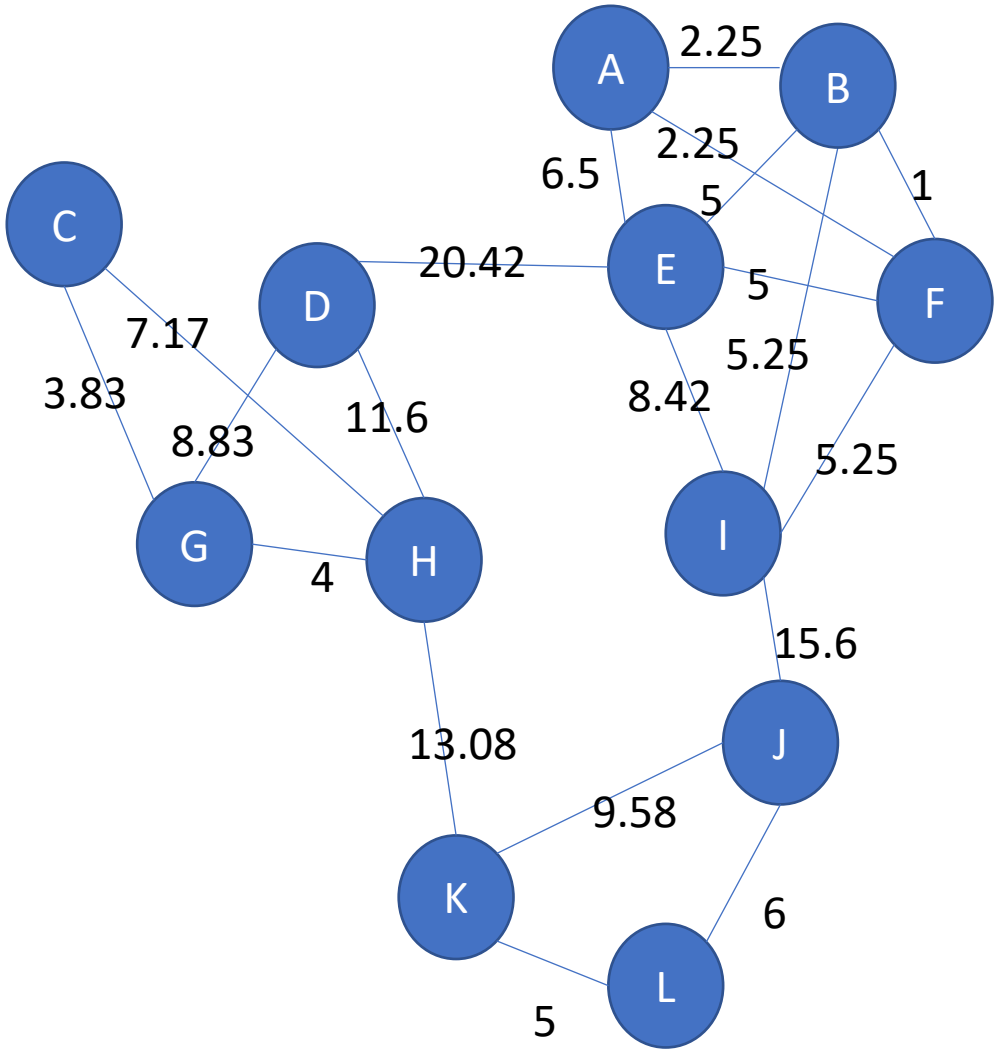
Here we have removed edges where betweenness > 3.5

Given betweenness scores for each edge,

- Remove all edges with betweenness over a certain threshold
- OR remove edges (starting with the highest) until a certain number of clusters are discovered.

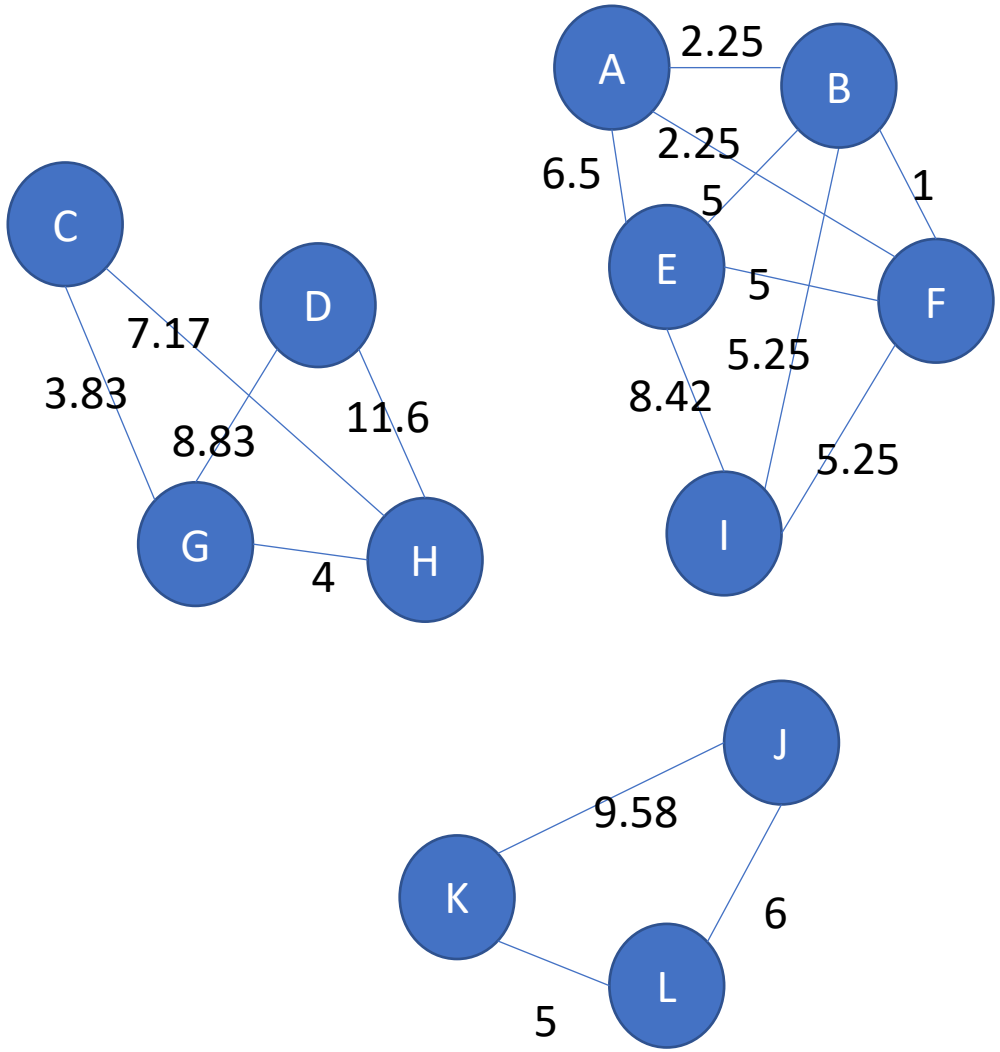
Why are B and D considered outside of their communities?

Using betweenness



- Use the betweenness scores to separate this graph into 3 clusters.

Using betweenness



Overview

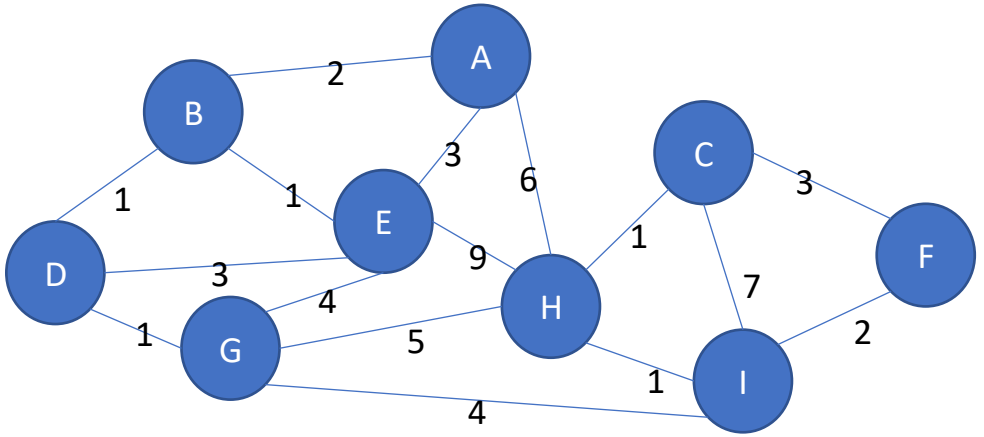
- Current topic:
  - **Graphs / networks**
    - Breadth-first and depth-first searching DONE
    - Mining social media networks DONE
    - Dijkstra's algorithm for all pairs shortest paths INTRO LAST WEEK, EXAMPLE TODAY

- SECOND PART OF LECTURE: PageRank algorithm

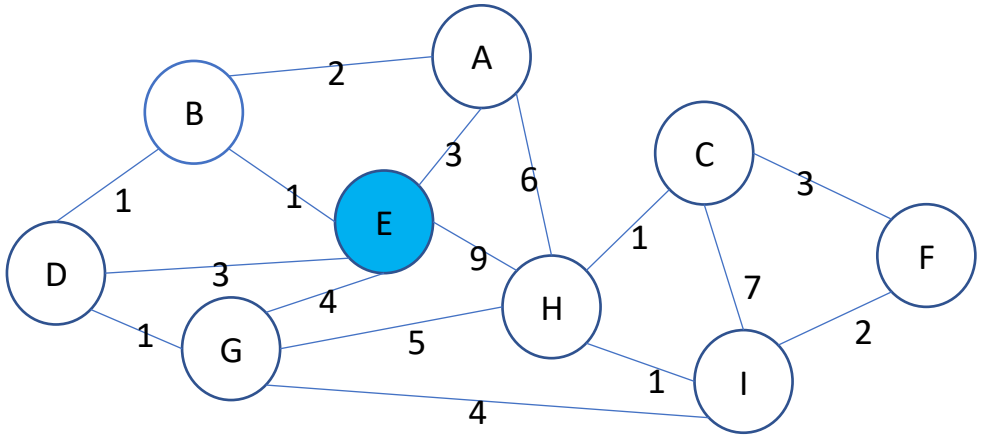
Dijkstra's algorithm: example

1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.

2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.



3. Does Dijkstra's Algorithm work for directed graphs?

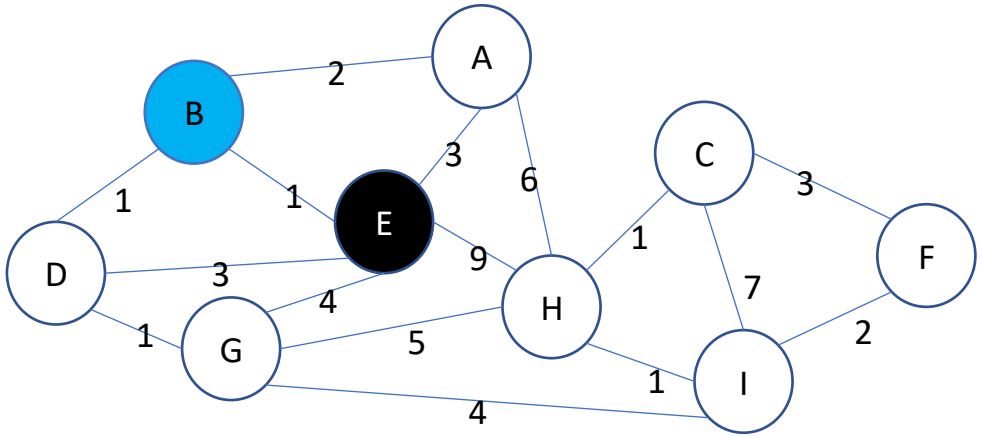


1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed:
Queue: E, B, A, D, G, H

Distances:
A: 3,
B: 1,
C: inf,
D: 3,
E: 0,
F: inf,
G: 4,
H: 9,
I: inf

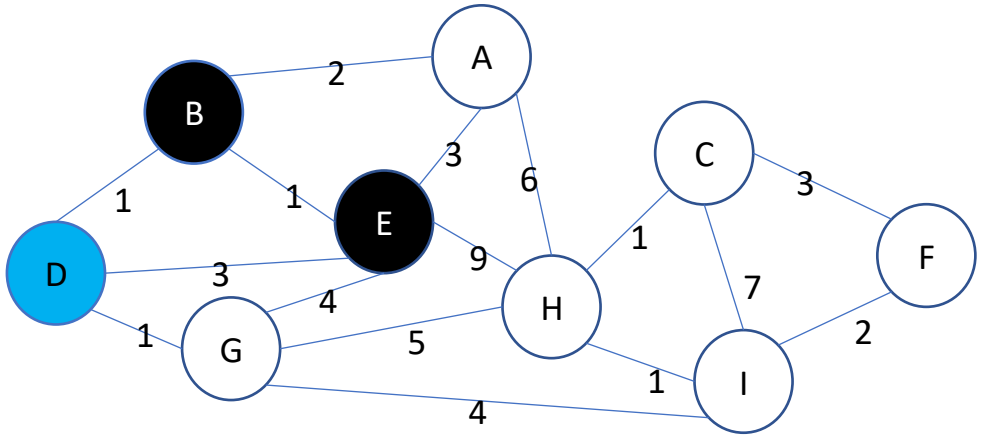
Predecessors:
A: E,
B: E,
C: null ,
D: E,
E: null,
F: null,
G: E,
H: E,
I: null



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E
Queue: B, D, A, G, H

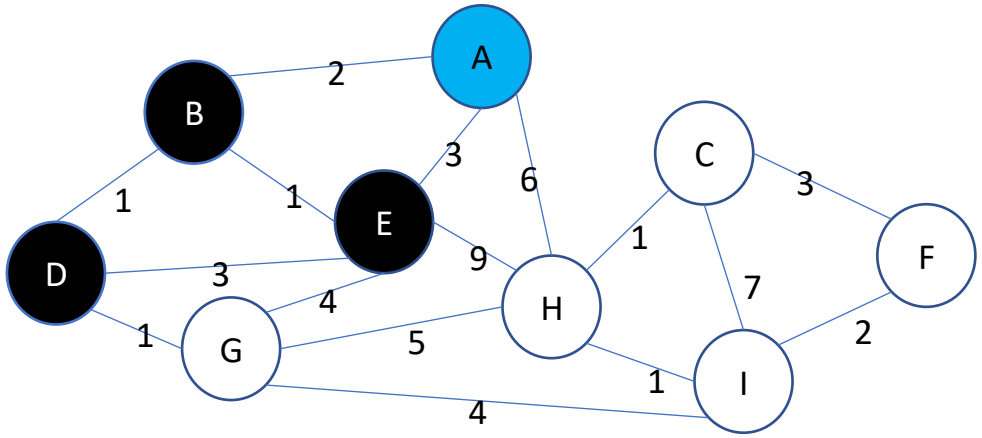
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: inf,	C: null ,
D: 2,	D: B,
E: 0,	E: null,
F: inf,	F: null,
G: 4,	G: E,
H: 9,	H: E,
I: inf	I: null



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

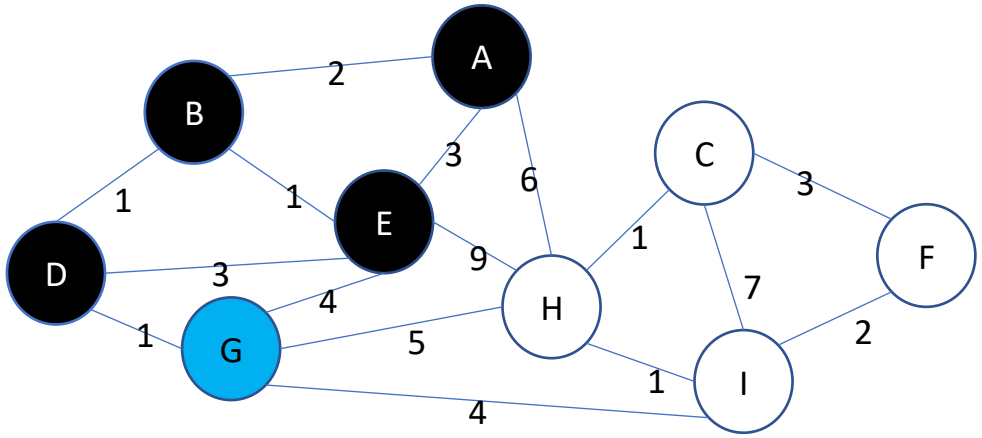
Completed: E, B
Queue: D, A, G, H

Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: inf,	C: null ,
D: 2,	D: B,
E: 0,	E: null,
F: inf,	F: null,
G: 3,	G: D,
H: 9,	H: E,
I: inf	I: null



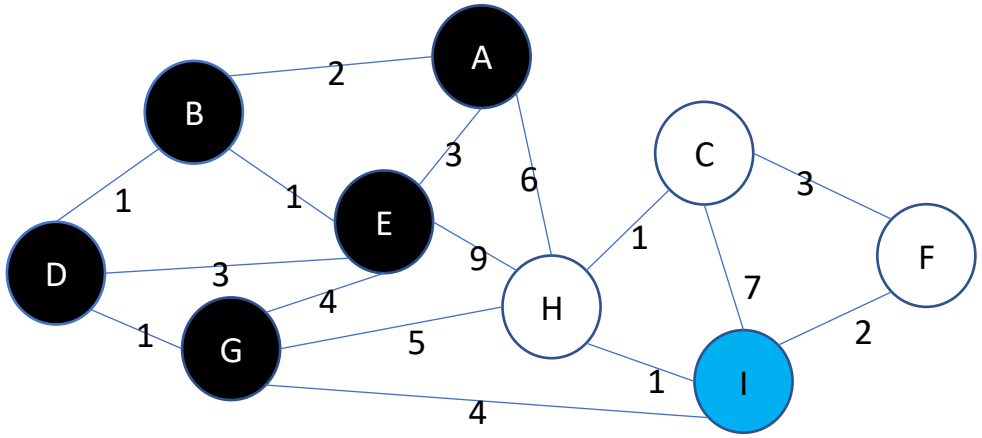
Completed: E, B, D
Queue: A, G, H

Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: inf,	C: null ,
D: 2,	D: B,
E: 0,	E: null,
F: inf,	F: null,
G: 3,	G: D,
H: 9,	H: E,
I: inf	I: null



Completed: E, B, D, A
Queue: G, I, H

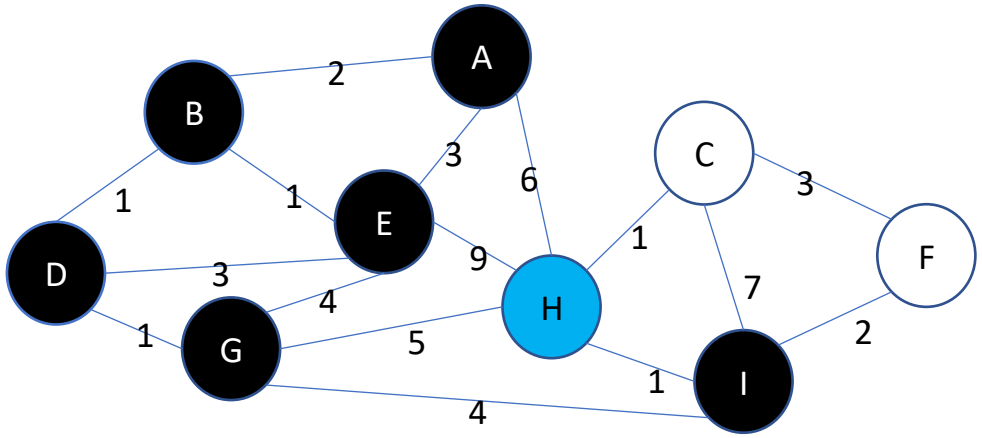
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: inf,	C: null ,
D: 2,	D: B,
E: 0,	E: null,
F: inf,	F: null,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E, B, D, A, G
Queue: I, H, F, C

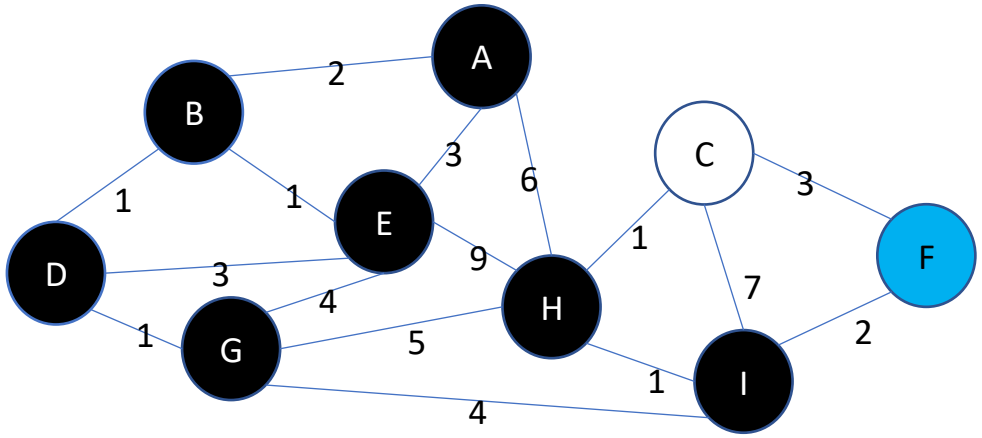
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: 14,	C: I ,
D: 2,	D: B,
E: 0,	E: null,
F: 9,	F: I,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E, B, D, A, G, I
Queue: H, F, C

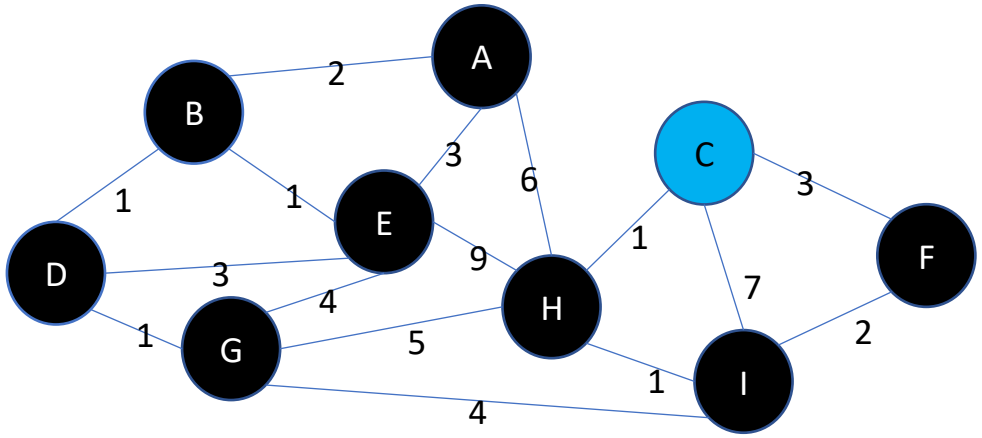
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: 9,	C: H ,
D: 2,	D: B,
E: 0,	E: null,
F: 9,	F: I,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E, B, D, A, G, I, H
Queue: F, C

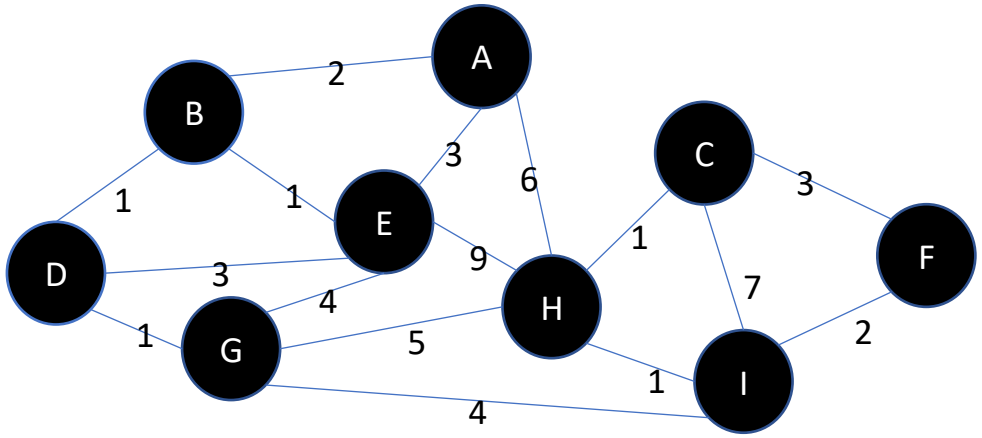
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: 9,	C: H ,
D: 2,	D: B,
E: 0,	E: null,
F: 9,	F: I,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E, B, D, A, G, I, H, F
Queue: C

Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: 9,	C: H ,
D: 2,	D: B,
E: 0,	E: null,
F: 9,	F: I,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G



1. Use Dijkstra's Algorithm to find the shortest paths from vertex E to every other vertex in the graph below.
2. Write down the lengths of the shortest paths, the predecessor for each vertex on the shortest path and the order in which the vertices are completed / finalised.
3. Does Dijkstra's Algorithm work for directed graphs?

Completed: E, B, D, A, G, I, H, F, C
Queue:

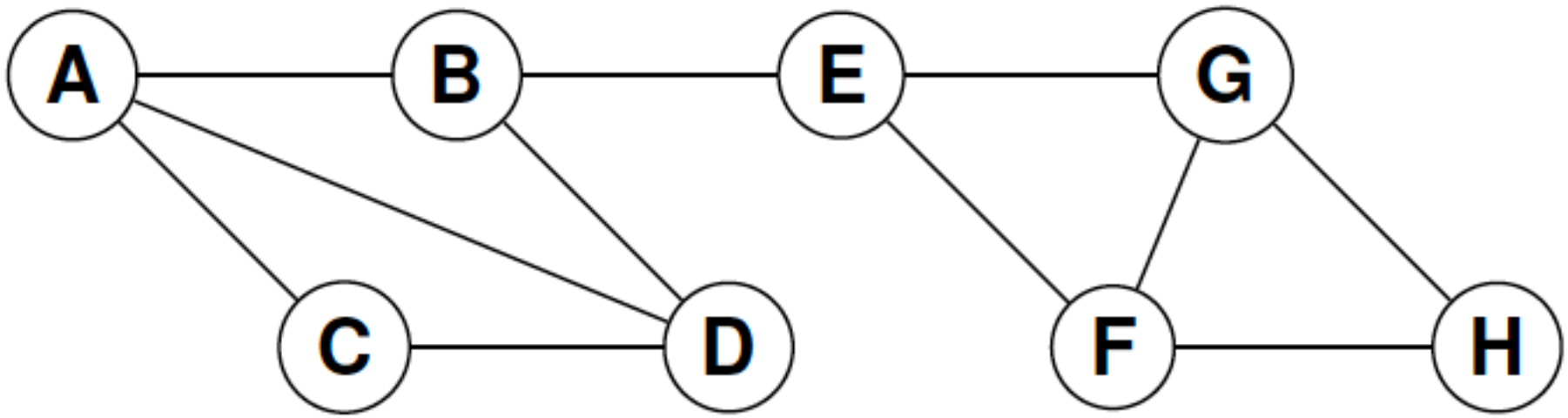
Distances:	Predecessors:
A: 3,	A: E,
B: 1,	B: E,
C: 9,	C: H ,
D: 2,	D: B,
E: 0,	E: null,
F: 9,	F: I,
G: 3,	G: D,
H: 8,	H: G,
I: 7	I: G

Summary of graphs/networks

- Breadth-first and depth-first searching
- Dijkstra's algorithm for all pairs shortest paths
- Mining social media networks: betweenness and the Girvan-Newman algorithm

- Suggested reading:
  - LRU(2014): Mining Massive Datasets (Chapters 7 and 10)
  - CLRS(1990): Intro to Algorithms (Chapters 5, 23, 25)

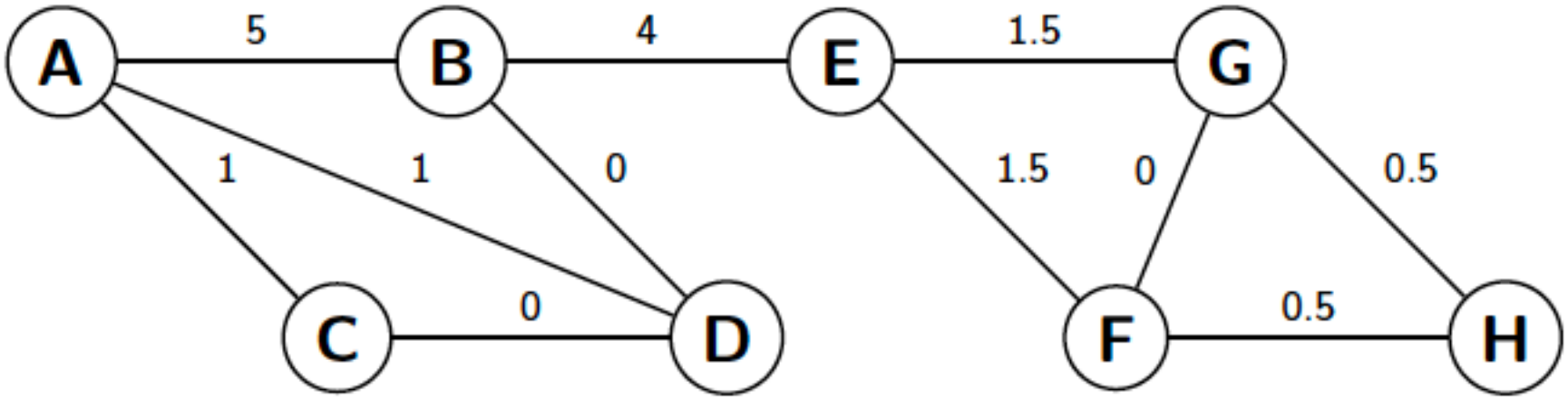
Exercise on betweenness



In order to find the *betweenness* of each edge, the *Girvan-Newman* algorithm is applied to each node. Apply a single round of the *Girvan-Newman* algorithm, starting at node A. State the resulting scores on each edge after this single round.

What is the **total** betweenness score for the edge (B,E)? (Without actually applying 8 rounds of the GN algorithm?) Explain your answer.

Exercise - Solution



- Total betweenness score of (B,E) is 16. It is on the shortest path from any of A, B, C, D to any of E, F, G, H, so 4x4=16.

Overview

- Current topic:
  - **Graphs / networks**
    - Breadth-first and depth-first searching DONE
    - Mining social media networks DONE
    - Dijkstra's algorithm for all pairs shortest paths DONE

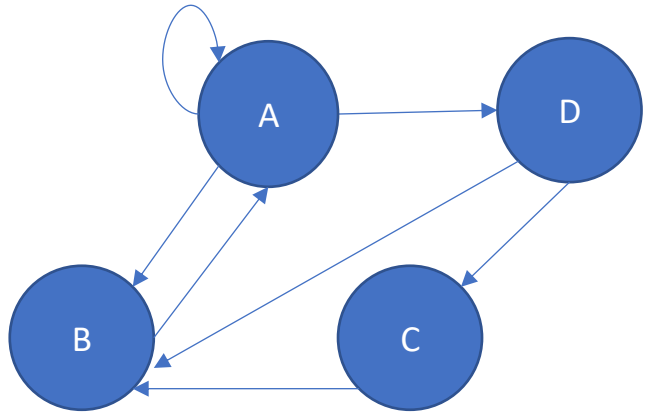
- SECOND PART OF LECTURE: PageRank algorithm

The PageRank Algorithm

- Developed by Stanford PhD students Larry Page and Sergei Brin in 1996
  - Page and Brin went on to found Google in 1998
- Designed to combat *term spam*, which rendered early search engines virtually useless
- It simulates where Web surfers would tend to congregate if:
  - They start at a random web page
  - They follow randomly chosen outlinks from their current page
  - This process is repeated many times

The Idealised PageRank Algorithm

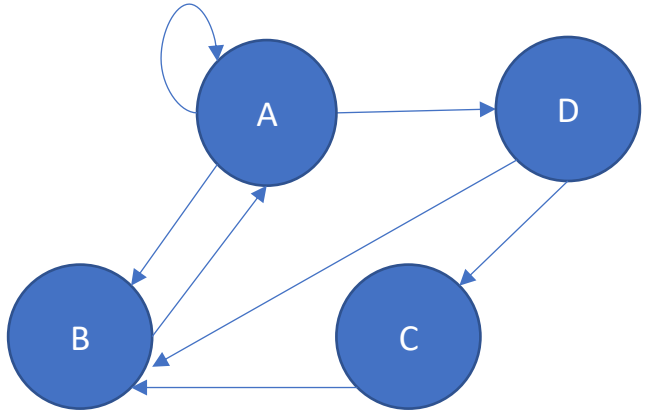
- Ranks pages on the web by their perceived importance
- Pages are considered more important if they have more links TO them from other more important pages ….
- Imagine a random surfer on a web with 4 pages. If he is truly random, then there is a uniform probability of him starting anywhere. The probability of where he goes next depends on the number of outlinks from a page



1. Write down the initial probability distribution, which specifies where a random surfer is at t_0 i.e., before any transitions are made.

2. Write down the transition matrix for a random surfer in this graph

The Idealised PageRank algorithm: initialisation



The transition matrix for a random surfer in this network is:

$$T = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix}$$

At time 0, t_0:
P(A) = P(B) = P(C) = P(D) = 1/4

$$u = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$$

At time t_n:

$P(A_n | A_{n-1}) = 1/3$ $\quad$ $P(A_n | B_{n-1}) = 1$
$P(B_n | A_{n-1}) = 1/3$ $\quad$ $P(B_n | B_{n-1}) = 0$
$P(C_n | A_{n-1}) = 0$ $\quad$ $P(C_n | B_{n-1}) = 0$
$P(D_n | A_{n-1}) = 1/3$ $\quad$ $P(D_n | B_{n-1}) = 0$

Markov iteration – makes the assumption that the transition matrix does not change over time, the probability of where the surfer goes next depends ONLY on where it currently is (not on where it has been previously).

The Idealised PageRank Algorithm

Transition matrix: T

At t_1:

$$\begin{pmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{pmatrix} = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 8/24 \\ 11/24 \\ 3/24 \\ 2/24 \end{pmatrix}$$

At t_n:

$$u_n = \begin{pmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{pmatrix} = T^n \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$$

This tells us where the random surfer is likely to be after n steps.

Will this converge?
Is there some *v* such that:

$$v = Tv$$

Condition for convergence

The distribution of the surfer will approach a limiting distribution v that satisfies $v = Tv$, provided:

The graph is ***strongly connected***; that is, it is possible to get from any node to any other node.

Finding the limiting distribution

$$v = Tv$$

- The limiting distribution v is an eigenvector of T, where the associated eigenvalue $\lambda = 1$
- T is called a **stochastic matrix –** all of its columns add up to 1. $\lambda = 1$ is the principal (largest) eigenvalue of T, and **v is the principal eigenvector of T**

- How do we find an eigenvector? E.g.,

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Finding the limiting distribution by Gaussian elimination

- We have a system of equations to solve:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

$$\longrightarrow \quad a = \frac{a}{3} + b$$

$$b = \frac{a}{3} + c + \frac{d}{2}$$

$$c = \frac{d}{2}$$

$$d = \frac{a}{3}$$

Finding an eigenvector by Gaussian elimination

- Gaussian elimination performs operations on rows of the matrix to turn it into an easier system of equations to solve e.g., take the 3rd row away from the 2nd row

- If it can be converted into an upper triangular matrix (or a lower triangular matrix), then the system can be solved straightforwardly using back substitution (or forward substitution) – **see LUP decomposition, Lecture 3**.

- Time to run Gaussian elimination is cubic in the number of equations, i.e. $O(n^3)$:
  - LU(P) decomposition is $O(n^3)$.
  - After doing LU(P), solving the equations using the straightforward backward/forward substitution is $O(n^2)$.

- Is Gaussian elimination feasible in realistic examples where there are tens or hundreds of billions of nodes?

From lecture 3: using the LU(P) decomposition

- We know *Ax = b* and *PA = LU*
  - *PAx = Pb*
  - *LUx = Pb*
  - Let *y = Ux*
  - Solve *Ly = Pb* using forward substitution
  - Solve *Ux = y* using backward substitution

e.g.

$$Ly = Pb: \begin{pmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 3 & 2 & 4 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

$\rightarrow y_1 = 1,\ \ 2y_1 - y_2$=3, $\ \ 3y_1 + 2y_2 + 4y_3 = 2$

Which is easy.

$$\mathrm{U}x = y: \begin{pmatrix} 2 & 1 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1/4 \end{pmatrix}$$

$\rightarrow 6x_3 = 1/4,\ \ x_2 + 5x_3$=-1, $\ \ 2x_1 + x_2 + 3x_3 = 1$

Also easy!

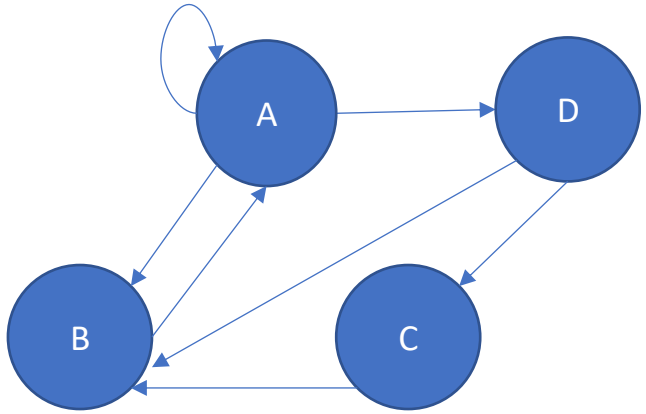
Finding an eigenvector using Markov Process Iteration

- It is more efficient to perform the **Markov Process** iteration:

$$v = u_n = \begin{pmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{pmatrix} = T^n u_0$$

- Start with the initial vector and multiply by T some number of times until the vector shows little change at each round.
- Matrix-vector multiplication can be done in $O(n^2)$
- T tends to be very sparse so multiplications can be done much faster in practice
- Even for the Web itself, we would expect convergence in 50-75 iterations
- So much better than $O(n^3)$ for Gaussian elimination

Finding the limiting distribution



The eigenvector for the transition matrix of this graph is:

$$v = \begin{pmatrix} 6 \\ 4 \\ 1 \\ 2 \end{pmatrix} \longrightarrow v = \frac{1}{13}\begin{pmatrix} 6 \\ 4 \\ 1 \\ 2 \end{pmatrix}$$

Does the relative importance of each dimension match our intuitions about the perceived importance of each web page?

Condition for convergence

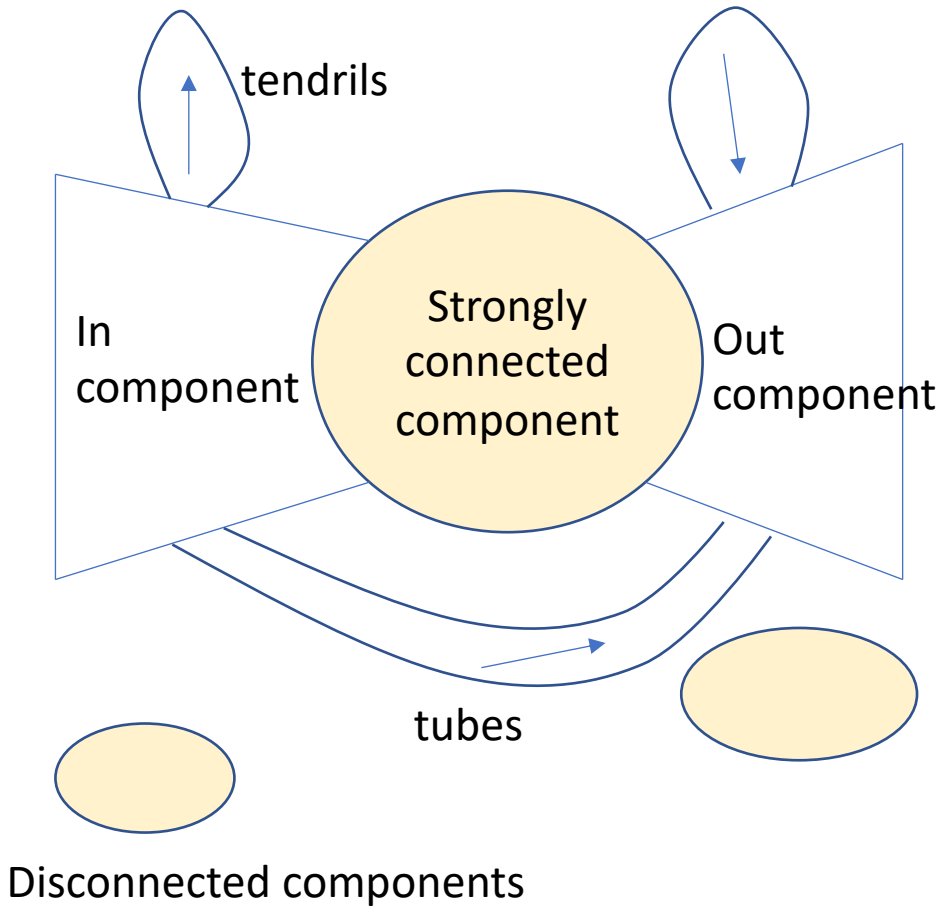
The distribution of the surfer will approach a limiting distribution v that satisfies $v = Tv$, provided:

The graph is ***strongly connected***; that is, it is possible to get from any node to any other node.

- In particular there must be ***no dead ends***: nodes that have no arcs out

How likely is it for these conditions to be met in practice?

Structure of the Web



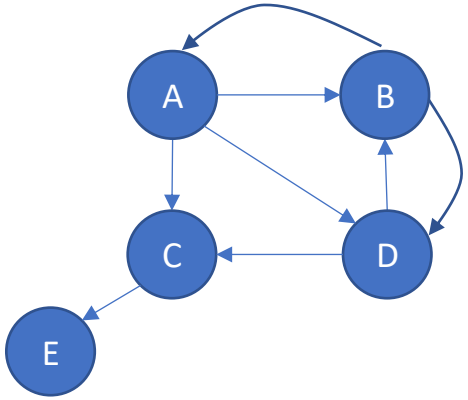
- There is a large strongly connected component (SCC) but also:-
  - The in-component: pages that can reach the SCC but are not reachable from the SCC
  - The out-component: pages reachable from the SCC but unable to reach the SCC
  - Tendrils: out of the in-component or into the out-component
  - Tubes: pages which are reachable from the in-component and which reach the out-component whilst bypassing the SCC
  - Disconnected components: unable to reach and unreachable from the other large components

Dead ends and Spider Traps

- The out-component of the Web is a problem for PageRank because:
  - surfers starting in the SCC or the in-component will eventually go into the out-component
  - when a surfer enters the out-component they can never leave
  - no page in the SCC or the in-component ends up with any probability of a surfer being there
  - falsely conclude that nothing in the SCC or the in-component is of any importance

- Need to avoid:-
  - dead ends: pages with no links out
  - spider traps: collections of pages which link to each other but have no links to other pages

Avoiding dead ends



$$T = \begin{pmatrix} 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

E is a dead end

T is actually **substochastic** rather than stochastic because the sum of one of the columns is less than 1.

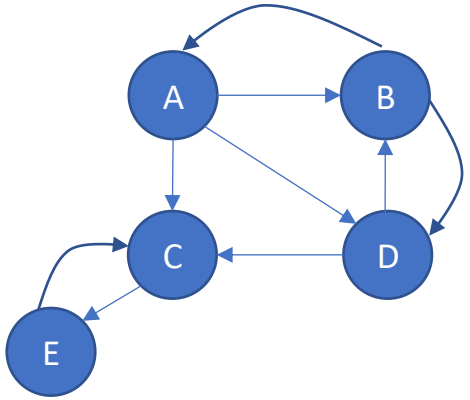
Method 1:

- Recursively delete dead ends. In other words, identify dead-ends (because its column total is 0) and drop them and their incoming edges from the graph. Repeat until no more dead ends are found

Method 2:

- Modify the process by which random surfers are assumed to move about the web using a process called **taxation,** which also solves the problem of spider traps

Spider Traps



$$T = \begin{pmatrix} 0 & 1/2 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 1/2 & 1 \\ 1/3 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

- Spider traps are collections of pages, which once a spider enters them, there is no way out.
- They occur both intentionally and unintentionally on the Web

Modified PageRank

- uses a process called **taxation**
- allows a random surfer a small probability of teleporting to another page
- overcomes the problems of dead ends and spider traps

Taxation

- Modifies the calculation of PageRank by allowing each random surfer a small probability of teleporting to another random page rather than following an out-link on the current page

$$\boldsymbol{u_n} = \beta \boldsymbol{T u_{n-1}} + (1-\beta)\mathbf{e}/n$$

- $\boldsymbol{\beta}$ is chosen constant (usually in the range 0.8-0.9)
- $\boldsymbol{e}$ is a vector of all 1s (1 for each vertex in the graph)
- n is the number of vertices in the graph

- Taxation limits the effect of spider traps (although they still accumulate more importance than they deserve)

Efficient Computation of PageRank

- Matrix-vector multiplication can be performed using MapReduce:
  - Map() -> multiply each individual matrix element by the appropriate vector element and yield the key-value pair $(i, m_{ij}v_j)$
  - Reduce() -> sum all of the values associated with a given key and yield the key-value pair (i, x_i)
- Use vertical striping of matrix and horizontal striping of vector to break the input into chunks which fit in main memory
- Use combiners to combine the results for chunks at the map nodes before transmitting over the network to the reduce nodes
- Represent sparse matrices efficiently (using a list of the non-zero entries and their values) – will be linear in the number of nonzero entries (number of edges) rather than quadratic in the size of the matrix (number of vertices). Then use algorithms that make use of the sparsity.

You do it! Find the limiting distribution

- Find an eigenvector of the transition matrix T, with eigenvalue 1

$$T = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix}$$

Finding the limiting distribution

- Find an eigenvector of the transition matrix T, with eigenvalue 1

$$T = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix}$$

$v = Tv$ gives

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

So we have:

(i) $a = \frac{a}{3} + b$

(ii) $b = \frac{a}{3} + c + \frac{d}{2}$

(iii) $c = \frac{d}{2}$ (iv) $d = \frac{a}{3}$

Choose **a=3**

(iv) → d=1 (iii) $c = \frac{1}{2}$

(i) $3 = 1 + b$ b=2

(ii) $2 = 1 + \frac{1}{2} + \frac{1}{2}$ ✓

$v = \begin{pmatrix} 3 \\ 2 \\ 1/2 \\ 1 \end{pmatrix}$

Overview

- **Graphs / networks**
  - Breadth-first and depth-first searching
  - Mining social media networks
  - Dijkstra’s algorithm for all pairs shortest paths

- **PageRank algorithm**