Graphs and Social Networks

Finding Communities
Counting Triangles
Estimating Neighborhood Sizes

Cloud and Big Data Summer School, Stockholm, Aug., 2015
Jeffrey D. Ullman

Graphs of Social Networks

1. Facebook "Friends" graph.
   - Undirected graph, over a billion nodes, hundreds of billions of edges.
2. Twitter Followers.
   - Directed graph, three hundred million nodes, hundreds of billions of arcs.
3. Many other examples: telephone calls, emails, Wikipedia articles and editors, coauthorship, etc., etc.

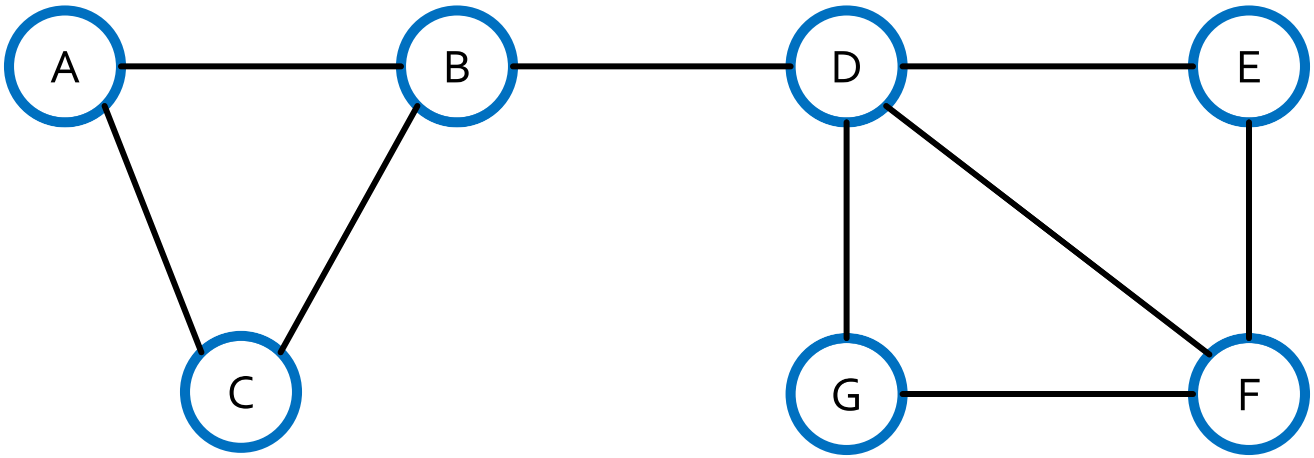
Structure of Social-Network Graphs

- These are not random graphs.
- *Community structure*: if there is an edge (A,B) and an edge (B,C), it is more likely there is an edge (A,C).
- Simple problem: divide a social network into disjoint *communities* (sets of nodes with a relatively high density of edges).
- Harder problem: find overlapping communities.
  - More realistic case.

Betweenness

- Used to divide a graph into reasonable communities.
- Roughly: the betweenness of an edge E is the number of pairs of nodes (A,B) for which the edge lies on the shortest path between A and B.
- More precisely: if there are several shortest paths between A and B, then E is credited with the fraction of those paths on which it appears.
- Edges of high betweenness separate communities.

Example: Betweenness



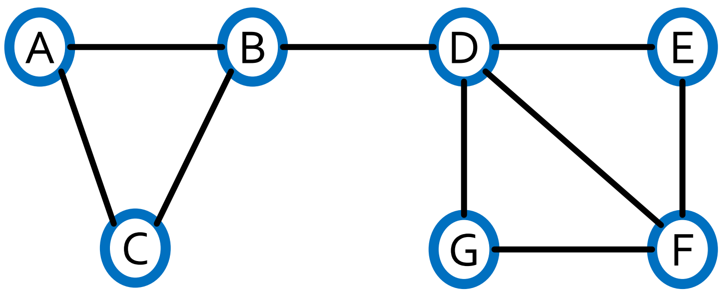
Edge (B,D) has betweenness = 12, since it is on the shortest path from each of {A,B,C} to each of {D,E,F,G}.

Edge (G,F) has betweenness = 1, since it is on no shortest path other than that for its endpoints.

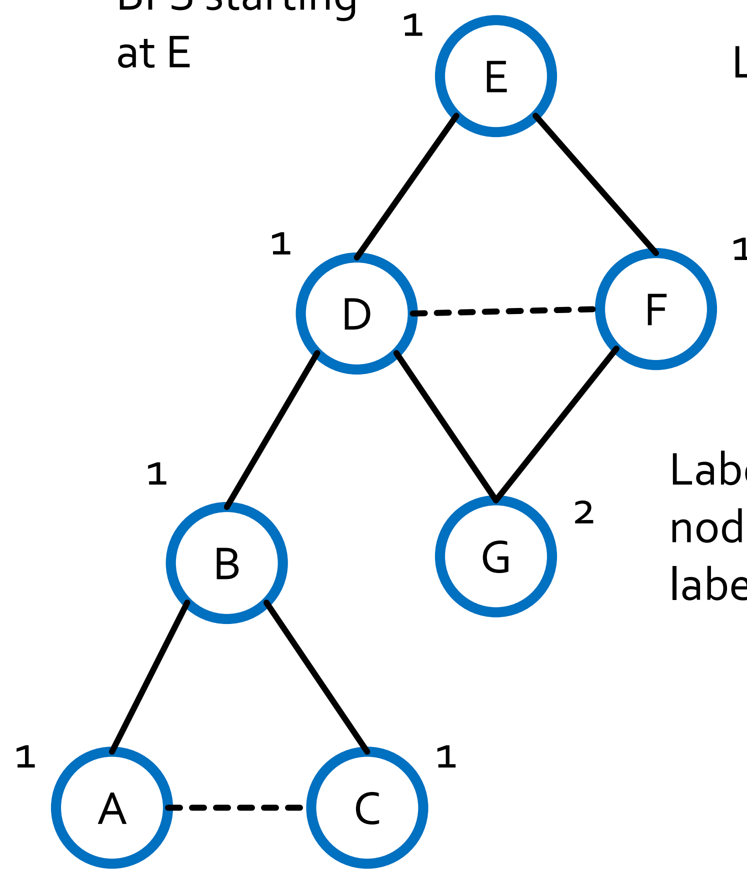
Girvan-Newman Algorithm

1. Perform a breadth-first search from each node of the graph.
2. Label nodes top-down to count the number of shortest paths from the root to that node.
3. Label both nodes and edges bottom-up with the fraction of shortest paths from the root to nodes at or below.
4. The betweenness of an edge is half the sum of the labels of that edge, starting with each node as root.
   - Half to avoid double-counting each edge.

Example: Steps 1 and 2



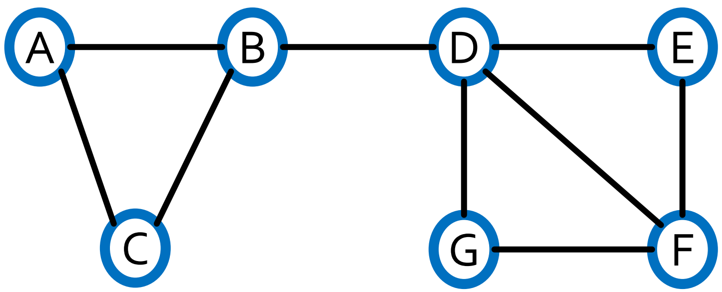
BFS starting at E

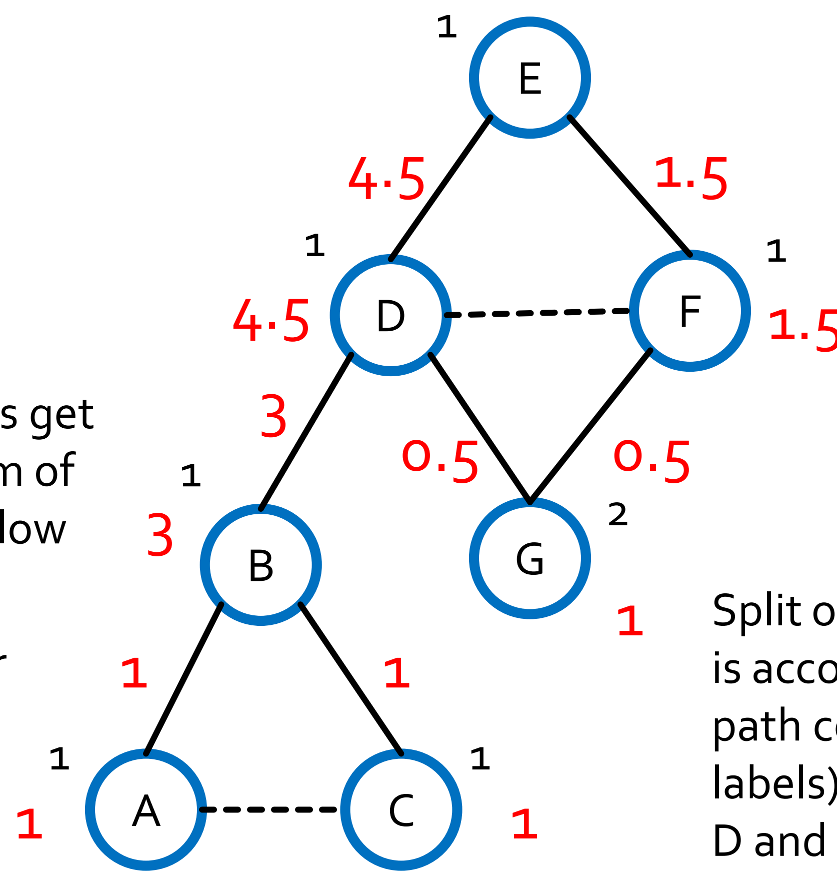


Label of root = 1

Label of other nodes = sum of labels of parents

Example: Step 3





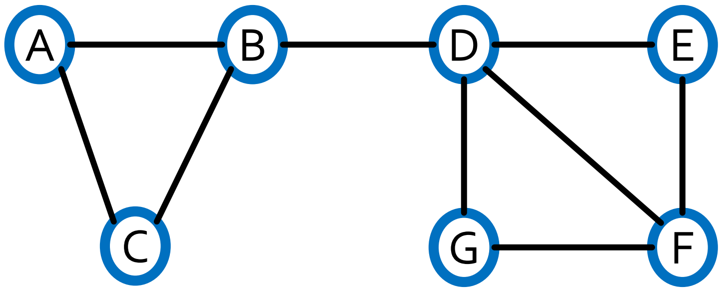
Interior nodes get 1 plus the sum of the edges below

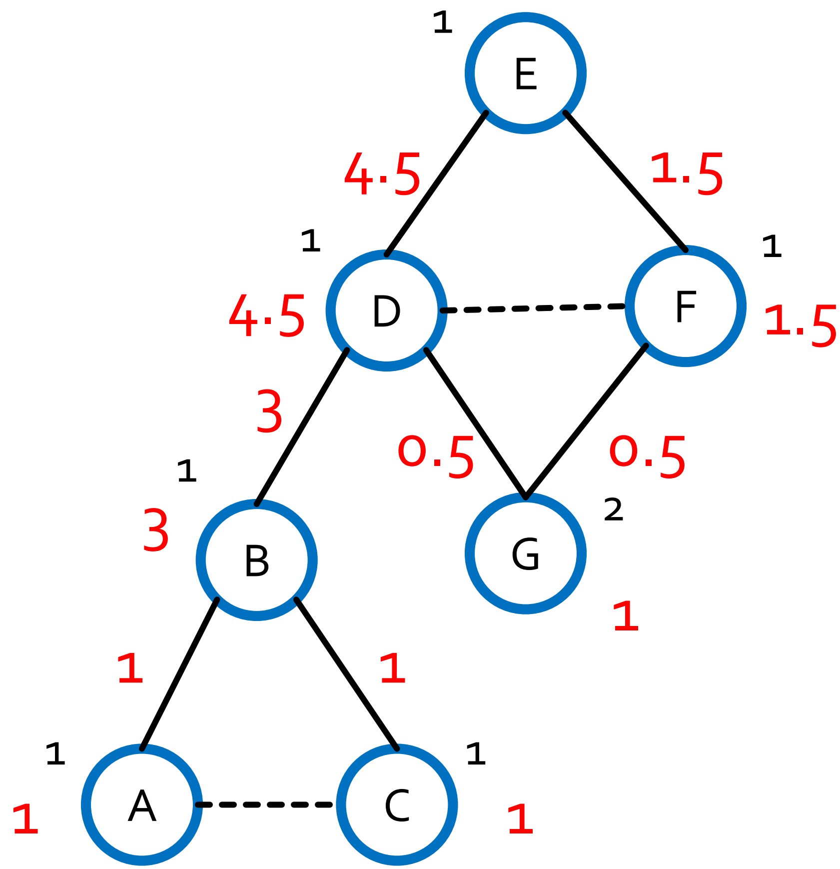
Edges get their share of their children

Leaves get label 1

Split of G's label is according to the path counts (black labels) of its parents D and F.

Sanity Check





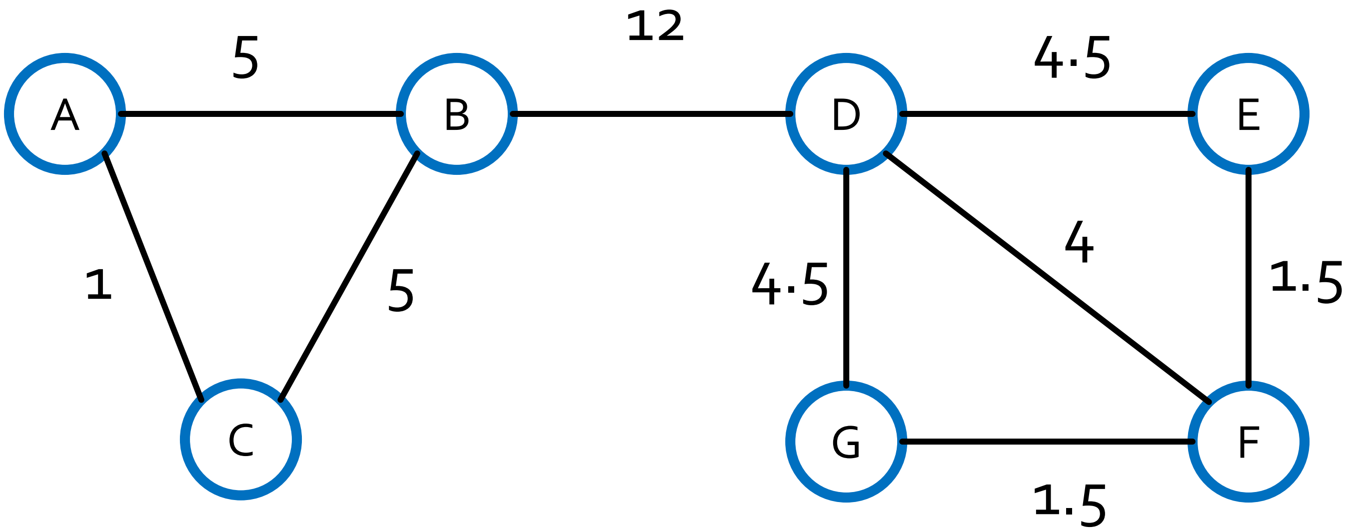
Edge (E,D) has label 4.5.

This edge is on all shortest paths from E to A, B, C, and D.

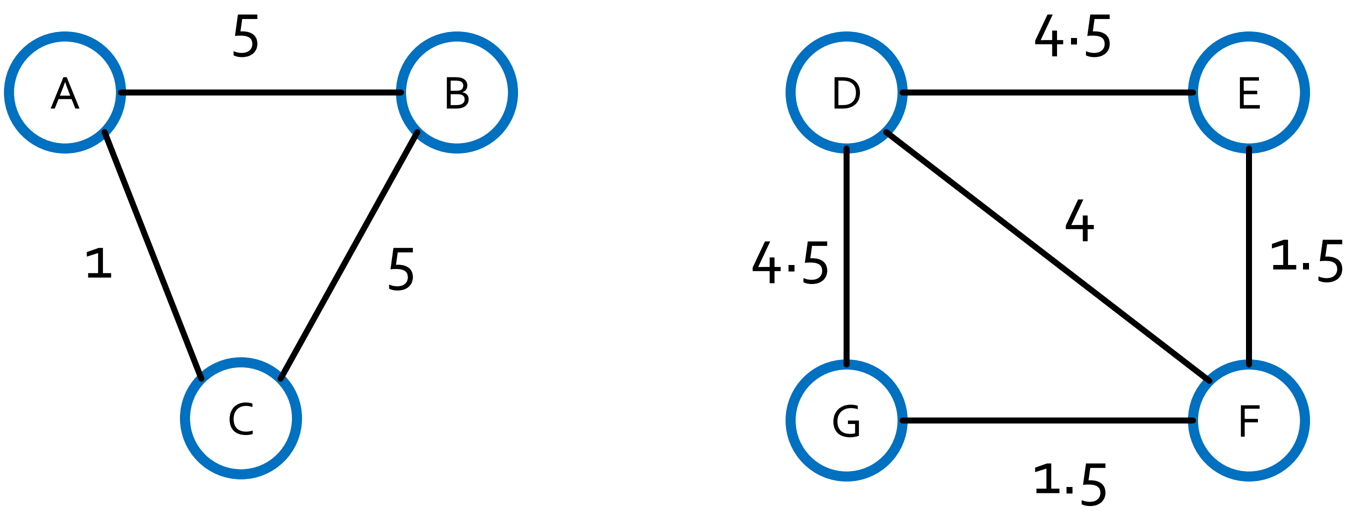
It is also on half the shortest paths from E to G.

But on none of the shortest paths from E to F.

Result of G-N Algorithm

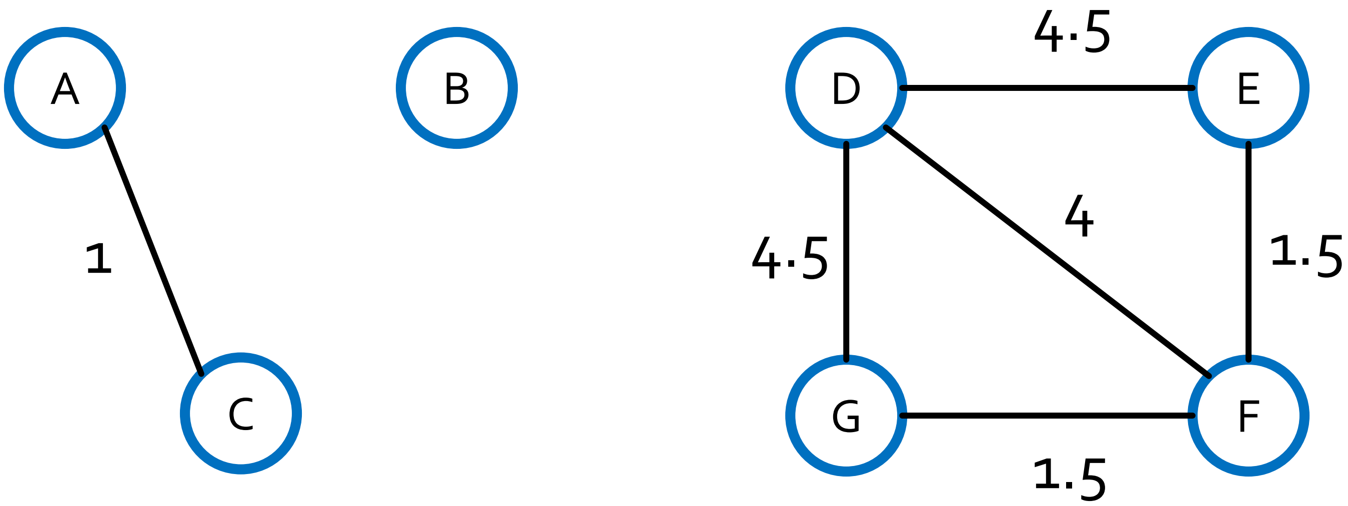


Remove Edge of Highest Betweenness



A sensible partition into communities

Remove Next-Highest Edge(s)



Why are A and C closer than B?
B is a "traitor" to the community,
being connected to D outside the group.

Counting Triangles

- Why Care?
  1. Density of triangles measures maturity of a community.
     - As communities age, their members tend to connect.
  2. The algorithm is actually an example of a recent and powerful theory of optimal join computation.

First Observations

- Let the undirected graph have N nodes and M edges.
  - $N \leq M \leq N^2$.
- One approach: Consider all N-choose-3 sets of nodes, and see if there are edges connecting all 3.
  - An $O(N^3)$ algorithm.
- Another approach: consider all edges e and all nodes u and see if both ends of e have edges to u.
  - An $O(MN)$ algorithm.

Heavy Hitters

- To find a better algorithm, we need to use the concept of a *heavy hitter* – a node with degree at least $\sqrt{M}$.
- Note: there can be no more than $2\sqrt{M}$ heavy hitters, or the sum of the degrees of all nodes exceeds 2M.
- A *heavy-hitter triangle* is one whose three nodes are all heavy hitters.

Finding Heavy-Hitter Triangles

- Consider all triples of heavy hitters and see if there are edges between each pair of the three.
- Takes time $O(M^{1.5})$, since there is a limit of $2\sqrt{M}$ on the number of heavy hitters.

Finding Other Triangles

- At least one node is not a heavy hitter.
- Consider each edge e.
  - If both ends are heavy hitters, ignore.
  - Otherwise, let end node u not be a heavy hitter.
  - For each of the at most $\sqrt{M}$ nodes v connected to u, see whether v is connected to the other end of e.
- Takes time $O(M^{1.5})$.
  - M edges, and at most $\sqrt{M}$ work with each.

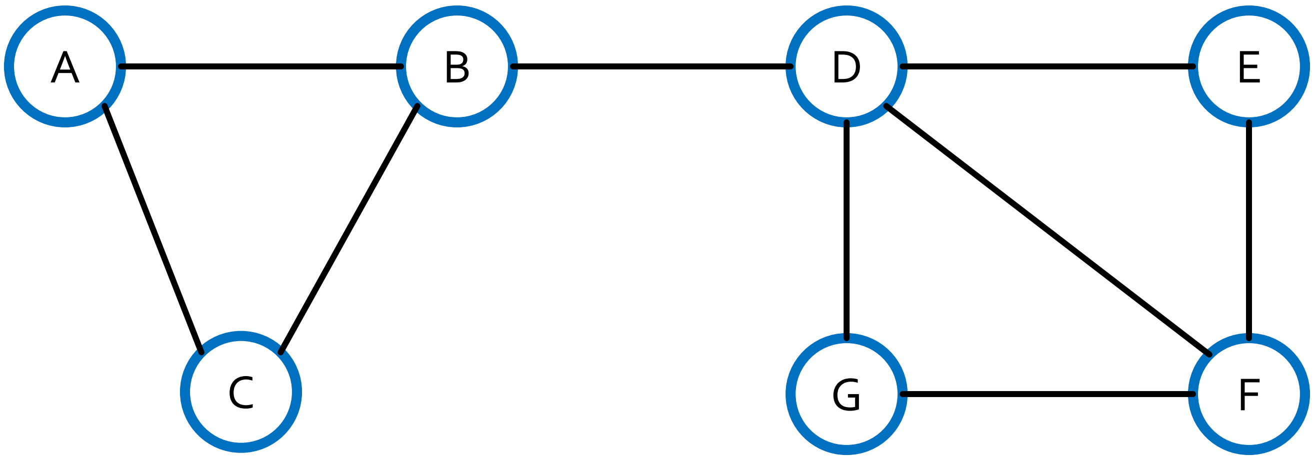
Optimality of This Algorithm

- Both parts take $O(M^{1.5})$ time and together find any triangle in the graph.
- For any N and M, you can find a graph with N nodes, M edges, and $\Omega(M^{1.5})$ triangles, so no algorithm can do significantly better.
- Note that $M^{1.5}$ can never be greater than the running times of the two obvious algorithms with which we began: N^3 and MN.

Neighbors and Neighborhoods

- If there is an edge between nodes u and v, then u is a *neighbor* of v and vice-versa.
- The *neighborhood* of node u at distance d is the set of all nodes v such that there is a path of length at most d from u to v.
  - Denoted n(u,d).
- Notice that if there are N nodes in a graph, then n(u,N-1) = n(u,N) = n(u,N+1) = … = all nodes reachable from u.

Example: Neighborhoods



n(E,0) = {E}; n(E,1) = {D,E,F}; n(E,2) = {B,D,E,F,G};
n(E,3) = {A,B,C,D,E,F,G}.

Why Neighborhoods?

- The sizes of neighborhoods of small distance measure the "influence" a person has in a social network.
  - Note it is the size of the neighborhood, not the exact members of the neighborhood that is important here.

Algorithm for Finding Neighborhoods

- n(u,0) = {u} for every u.
- n(u,d) is the union of n(v, d-1) taken over every neighbor v of u.
- Not really feasible for large graphs, since the neighborhoods get large, and taking the union requires examining the neighborhood of each neighbor.
  - To eliminate duplicates.

Approximate Algorithm for Neighborhood Sizes

- Remember the Flagolet-Martin algorithm for estimating the number of distinct elements in a stream?
- The same idea lets you estimate the number of distinct elements in the union of several sets.
- Pick several hash functions.
- Let h be one of these hash functions.
- For each node u and distance d compute the maximum “tail” length among all nodes in n(u,d), using hash function h.

Approximate Algorithm – (2)

- Remember: if R is the maximum tail length in a set of values, then 2^R is a good estimate of the number of distinct elements in the set.
- Since n(u,d) is the union of all neighbors v of u of n(v,d-1), the maximum tail length of members of n(u,d) is the largest of
  1. The tail length of h(u), and
  2. The maximum tail length for all the members of n(v,d-1) for any neighbor v of u.

Approximate Algorithm – (3)

- Thus, we have a recurrence for the maximum tail length of any neighbor of any node u, using any given hash function h.
- Repeat for some chosen number of hash functions.
- Combine estimates to get an estimate of neighborhood sizes, as for the Flagolet-Martin algorithm.