Distributed

Connections

Outline

- Trees

- Canonical Form
- Distributed Views
- Broadcast

- BFS/DFS

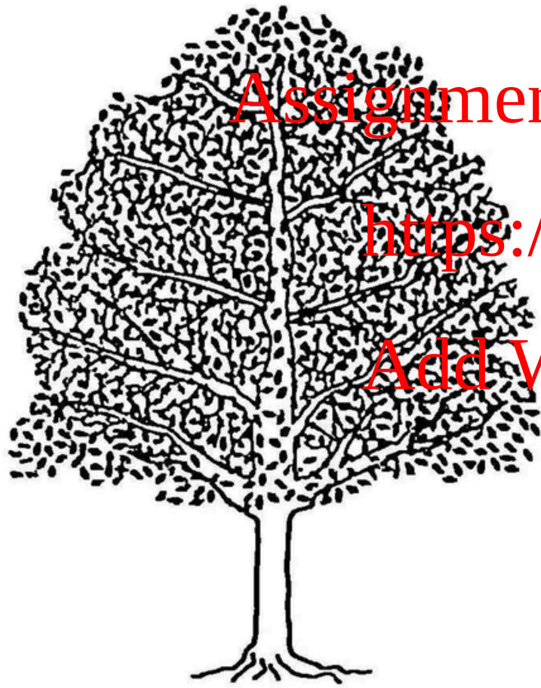
- Flooding

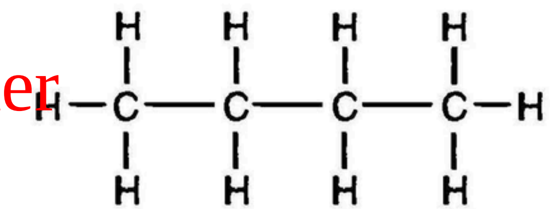
- Convergecast
- Applications

Trees

Trees and Communication

- Trees are everywhere: saplings, rivers, chemical compounds.
  - There is something about their efficiency and economy.



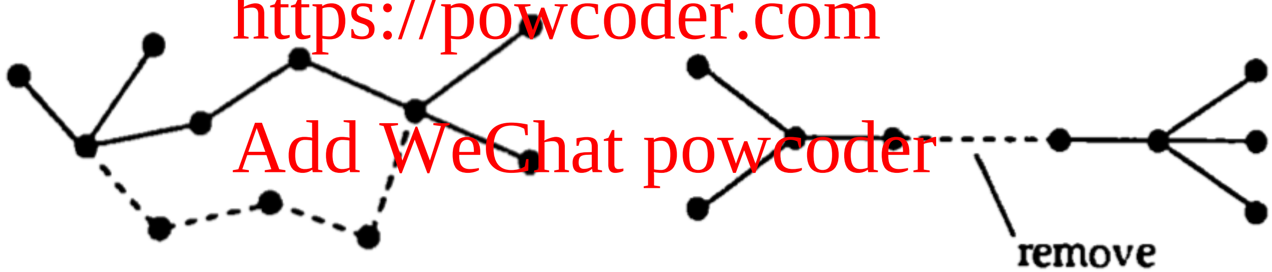


- Trees form a natural communication structure in distributed computing.

Main Concepts on Trees

- A tree is a connected graph that has no cycles.

- Start with the tree of one vertex: we can build up any tree we wish by successively adding a new edge and a new vertex.
  - At each stage, the # of vertices exceeds the number of edges by 1, so every tree with n vertices has exactly $n-1$ edges



Characterization of Trees

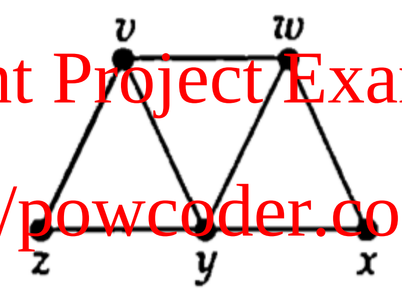
- Let T be a graph with n vertices. Then the following statements are equivalent.

1. T is connected and has no cydes.
2. T has $n-1$ edges and has no cycles.
3. T is connected and has $n-1$ edges.
4. T is connected and the removal of any edge disconnects T.
5. Any two vertices of T are connected by exactly one path.
6. T contains no cycles, but the addition of any new edge creates a cycle.

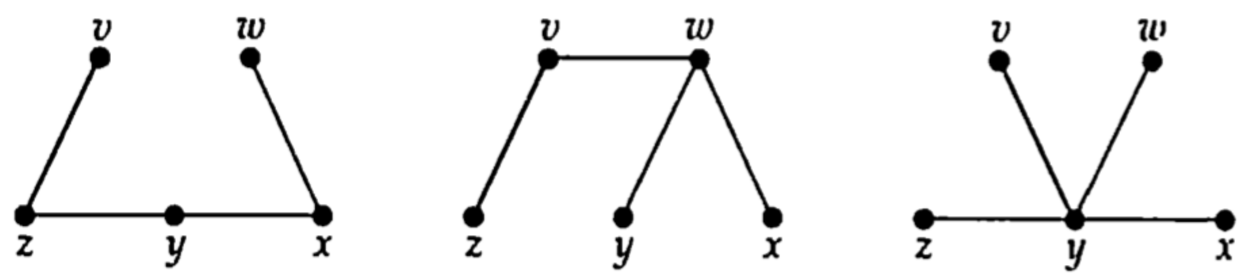
Spanning Trees

- Let G be a connected graph. Then a spanning tree in G is a subgraph of G that includes every vertex and is also a tree.

- A graph . . .



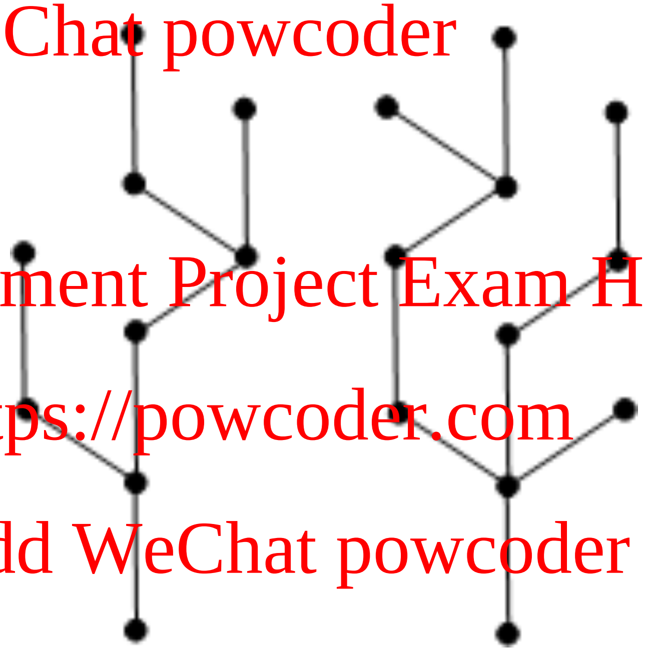
- . . . and possible spanning trees



- Spanning trees emerge naturally in communication.

Forests

- A Forest is a collection of vertex disjoint trees.

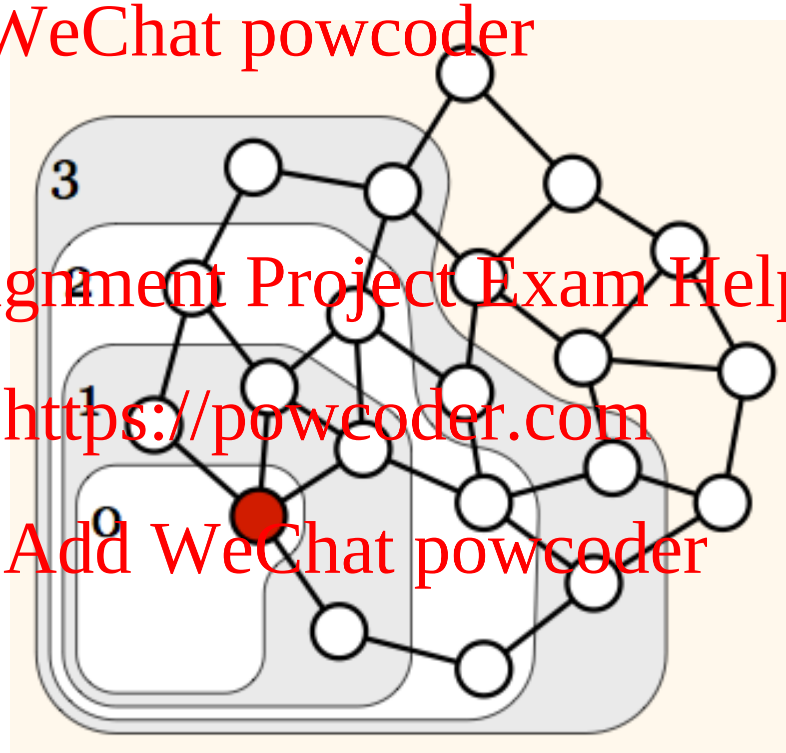


- Forests arise naturally in clustering.
- A Spanning Forest is a collection of vertex disjoint spanning trees.

Canonical Form

How Do Nodes Build their Knowledge?

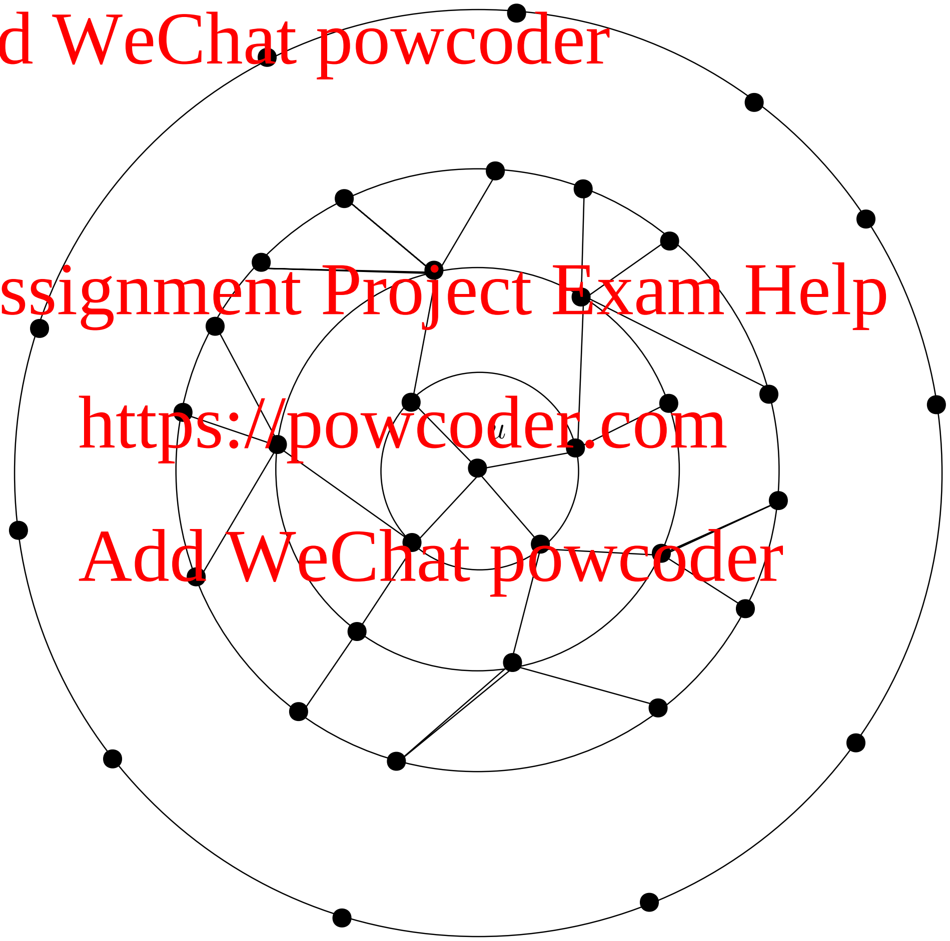
- They learn by exchanging messages in rounds.



- At the same time, different nodes learn different things!

Information Growth and Knowledge Discovery

- Start from node u



- When does the growth stop?

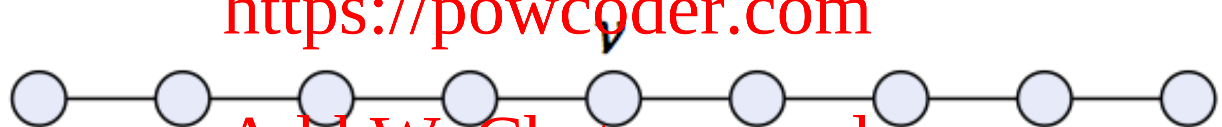
In a Line Graph

- In a typical synchronous distributed algorithm each node executes the following atomic actions

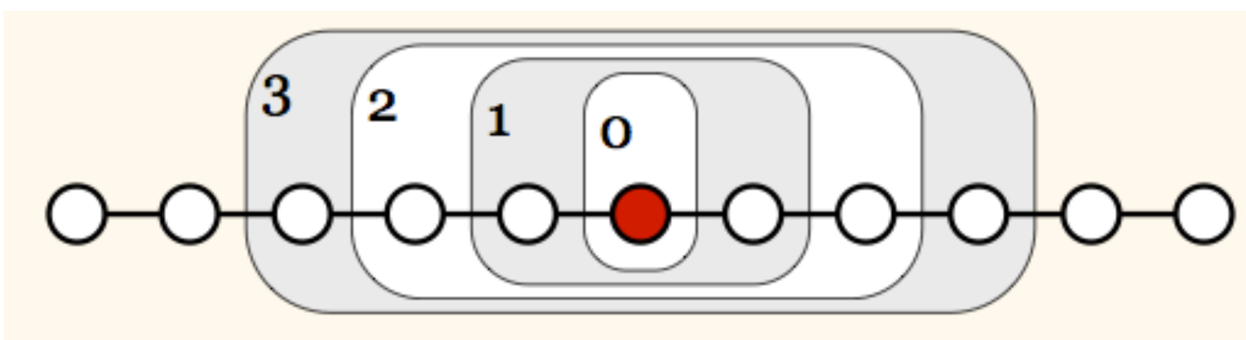
$$\text{send} \to \text{receive} \to \text{process}$$

in synchronous rounds.

- Node v, by exchanging messages in rounds...



- ...receives information about distance $1, 2, 3, \ldots$ nodes.



Canonical Form

- Assume that initially, all nodes only know their own identifier and potentially some additional input.

- Information needs at least r rounds to travel r hops.
- After r rounds, a node v can only learn about other nodes at distance at most r.

- If message size and local computations are not restricted, it is in fact not hard to see, that

  - in r rounds, a node v can learn exactly all the node labels and inputs up to distance r from v.
- This allows us to transform every deterministic r-round synchronous algorithm into a simple canonical form.

Cumulative Messages

- The idea is to "simplify communication" with cumulative messages.

- A typical synchronous distributed algorithm at each node consists of a sequence of executions

$$\text{send} \rightarrow \text{receive} \rightarrow \text{process}$$

in synchronous rounds.

- Often what matters is the source and the destination.

- Can we first do a sequence of r executions "send $\rightarrow$ receive$''$ followed by a single "process$''$ at the end?

- In other words, can we send "cumulative" messages for r rounds and finally do the processing?

Example: Computing the Sum in a Ring

- Consider a ring of n nodes with identifiers ID_i and weights w_i at each node, for $i = 1, 2, \ldots, n$.

- In a typical distributed computation for a node i:

  for r rounds **do**

  1. i sends pair (ID_i, w_i) to $i+1$, and receives pair (ID_{i-1}, w_{i-1}) from $i-1$.

  2. process by adding $w_i + w_{i-1}$.

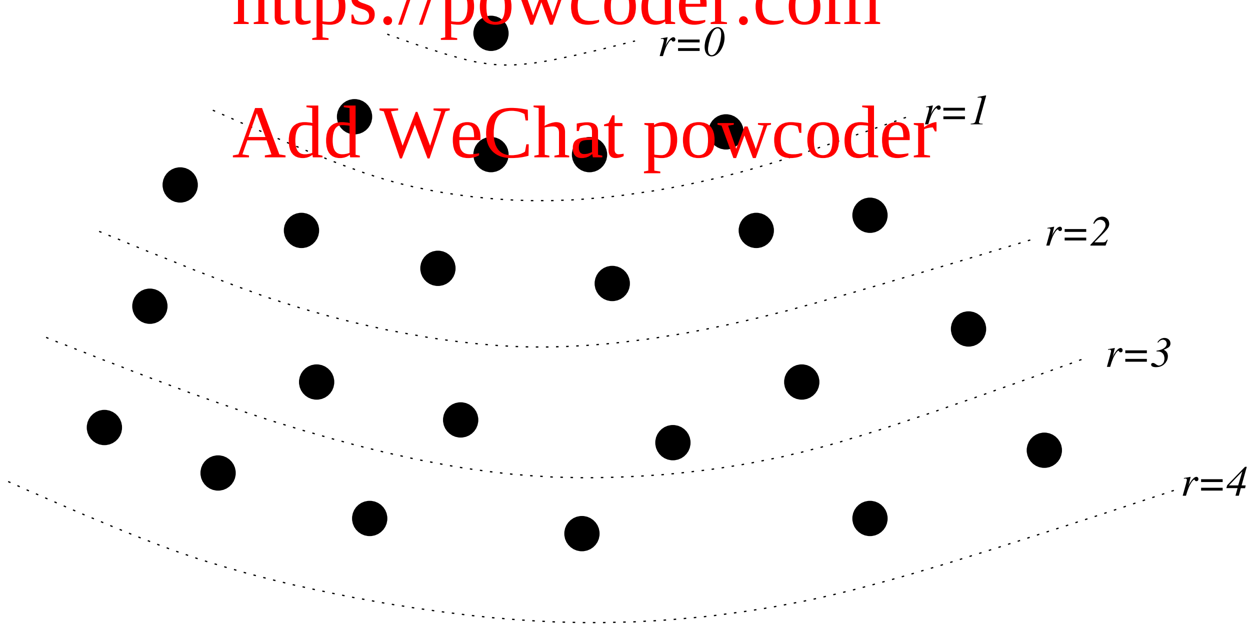
- This can be done in a cumulative manner at i as follows:
  1. i sends $(ID_i, w_i), (ID_{i-1}, w_{i-1}), \ldots, (ID_{i-1}, w_{i-r})$ to $i+1$, receives $(ID_{i-1}, w_{i-1}), (ID_{i-2}, w_{i-2}), \ldots, (ID_{i-r-1}, w_{i-r-1})$ from $i-1$
  2. process by adding $w_{i-1} + w_{i-2} + \cdots + w_{i-r-1}$

Synchronous Algorithm: Canonical Form

- **Synchronous Algorithm Canonical Form**

1. In r rounds: send complete initial state to nodes at distance at most r /* all the communication first */
2. Compute output based on complete information about r-neighborhood /* do all the computation in the end */

- Example: information "moves" in waves!



Main Claim on Canonical Form

- **Theorem 1** *If message size and local computations are not bounded, every deterministic, synchronous r-round algorithm can be transformed into an algorithm having the canonical form (i.e., it is possible to first communicate for r rounds and then do all the computations in the end).*

- Notice the importance of being able to transmit messages of arbitrary size:
  - this size will depend on the number of r rounds, and
  - it can be exponetial in r

- To handle "large size messages" you need "large memory"

Main Argument (1/2)

- Consider an r-round algorithm $\mathcal{A}$. We want to show that $\mathcal{A}$ can be brought to a canonical form.

- First, let the nodes communicate for r rounds.
- Assume that in every round, every node sends its complete state to all of its neighbors.

- By induction, after i rounds, every node knows the initial state of all other nodes at distance at most i.

- Hence, after r rounds, a node v has the combined initial knowledge of all the nodes in its r-neighborhood.
- We want to show that this suffices to simulate locally (at node v) enough of Algorithm $\mathcal{A}$ to compute all the messages that v receives in the r communication rounds of a regular execution of Algorithm $\mathcal{A}$.

Main Argument (2/2)

- We prove the following statement by induction on i.

  Claim. For all nodes at distance at most $r - i + 1$ from v, node v can compute all messages of the first i rounds of a regular execution of $\mathcal{A}$.

- $i = 1$: v knows the initial state of all nodes in the r-neighborhood and can compute all messages of first round.

- Induction Step: from i to $i + 1$.

  – By the induction hypothesis, v can compute the messages of the first i rounds of all nodes in its $(r - i + 1)$-neighborhood.

  – It can therefore compute all messages that are received by nodes in the $(r - i)$-neighborhood in the first i rounds.

  – This is exactly what is needed to compute the messages of round $i + 1$ of nodes in the $(r - i)$-neighborhood.

Issues

- It is straightforward to generalize the canonical form to randomized algorithms:

  – Every node first computes all the random bits it needs throughout the algorithm.

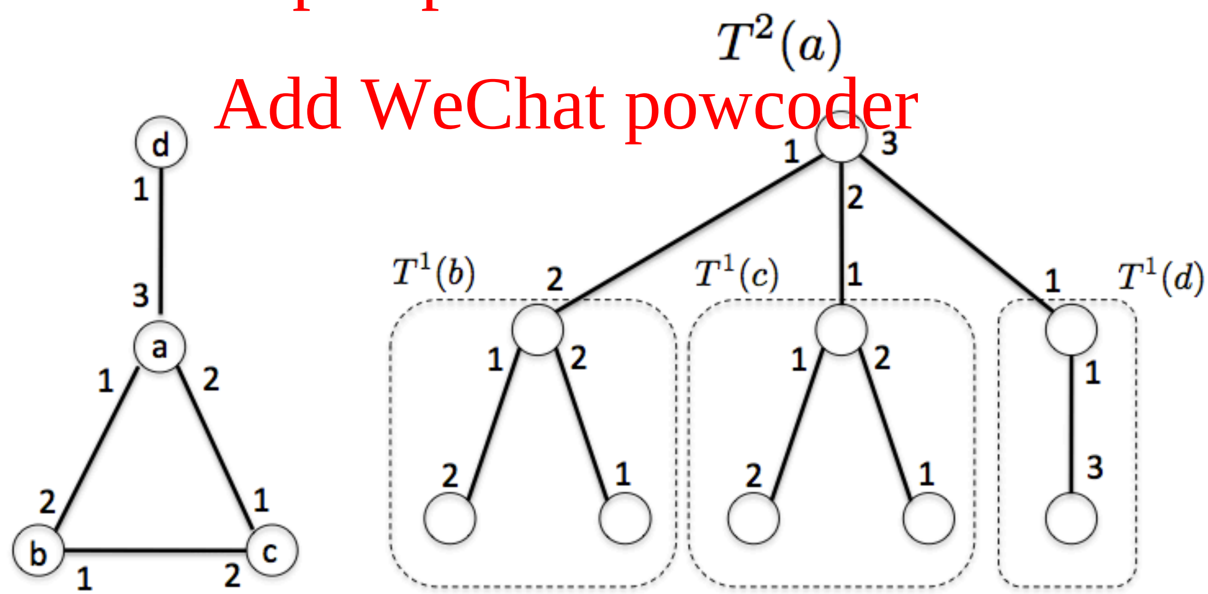
- The random bits are then part of the initial state of a node.

Distributed Views

Views: Undirected Networks

- Each node has port labels and can build a view accumulating its knowledge.

- The view of depth k of a node is a tree containing information on all the walks of length k leaving that node.
- Views contain all the information that nodes could obtain by exchanging messages with their neighbors.



r-View (or r-Hop View or r-Neighborhood)

- Collection of initial states of all nodes in the r-neighborhood of a node v, is called r-hop view (or neighborhood) of v.

  - For a given graph G, it is denoted by

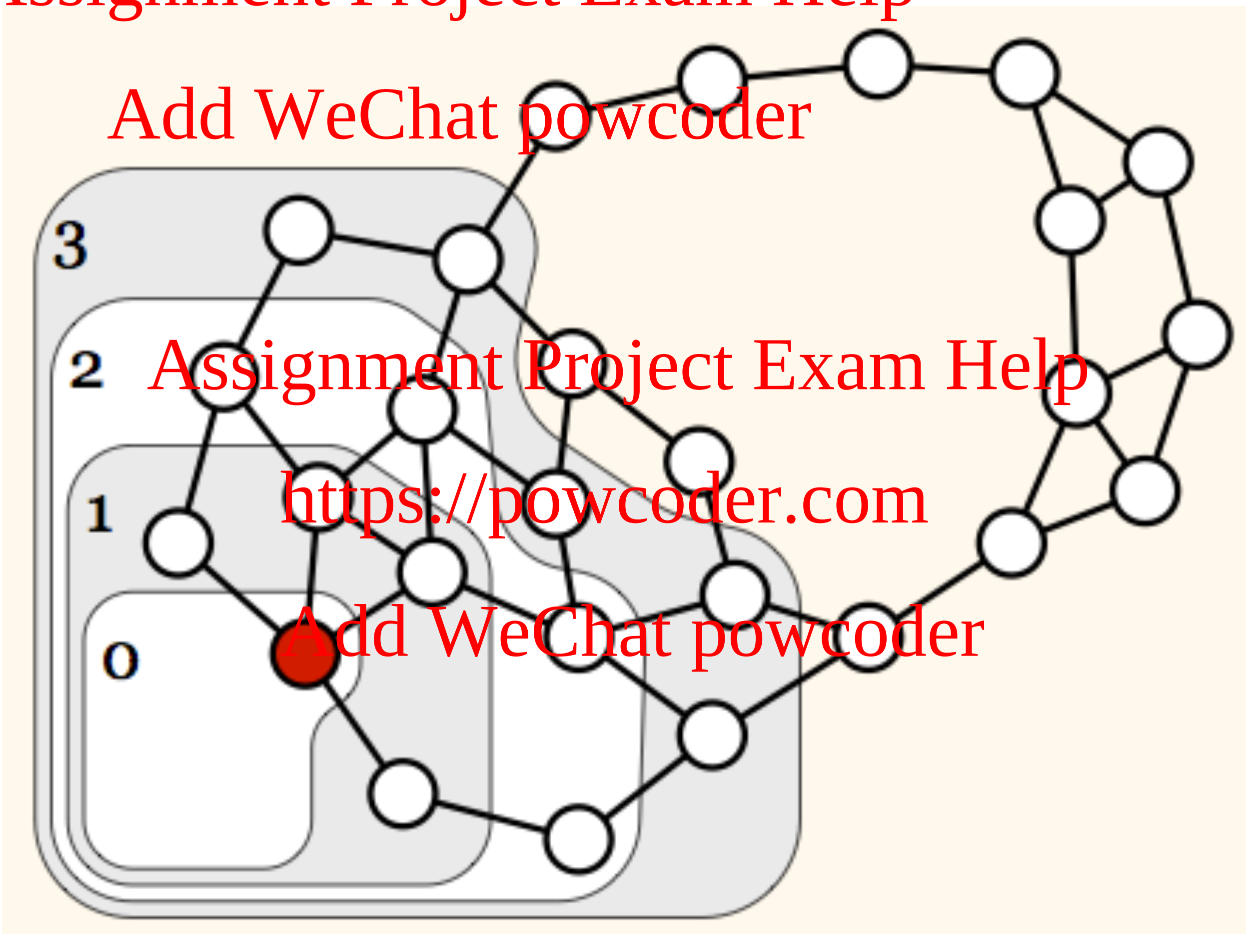
$$V_r^G(v) \text{ or } N_r^G(v).$$

  - We usually omit mention of G (when clear from the context) and denote it by

$$V_r(v) \text{ or } N_r(v).$$

- A view can be enriched as needed by including information:
  - on node states,
  - node topology r hops away from the source v,
  - etc

Example: r-Hop Views for $r = 0, 1, 2, 3$



Issues

- Assume that initially, every node knows its degree, its label (identifier) and potentially some additional input.

- The r-hop view of a node v then includes
  - the complete topology of the r-neighborhood,
  - possibly edges between nodes at distance r in the subgraph, and
  - the labels and additional inputs of all nodes in the r-neighborhood.

The View as a Function

- **Theorem 2** *A deterministic r-round algorithm A is a function that maps every possible r-hop view to the set of possible outputs.*

- By Theorem 1, we know that we can transform Algorithm A to the canonical form.

- After r communication rounds, every node v knows exactly its r-hop view.

- This information suffices to compute the output of node v.

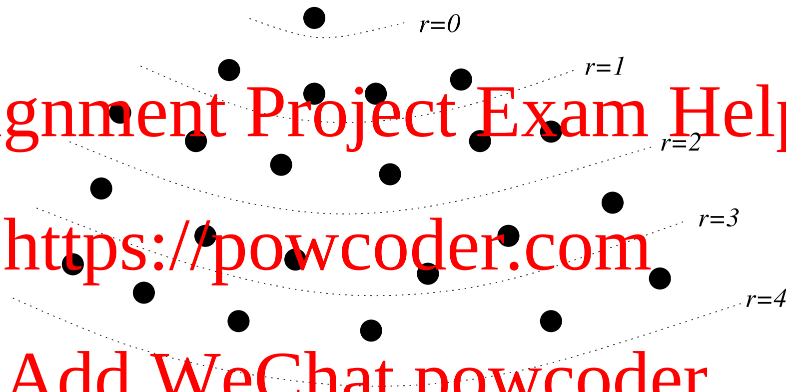
Issues

- Two nodes with equal r-hop views
  - have to compute the same output in every r-round algorithm.

- For coloring algorithms, the only input of a node v is its label.
  - The r-hop view of a node therefore is its labeled r-neighborhood.

Power of Views[a]

- For a graph of n nodes, Norris (1995) proved that if two nodes have the same view of depth $n-1$, then they have the same views for all depths.



- Taking the diameter δ of a graph into account, can improve $n-1$ to

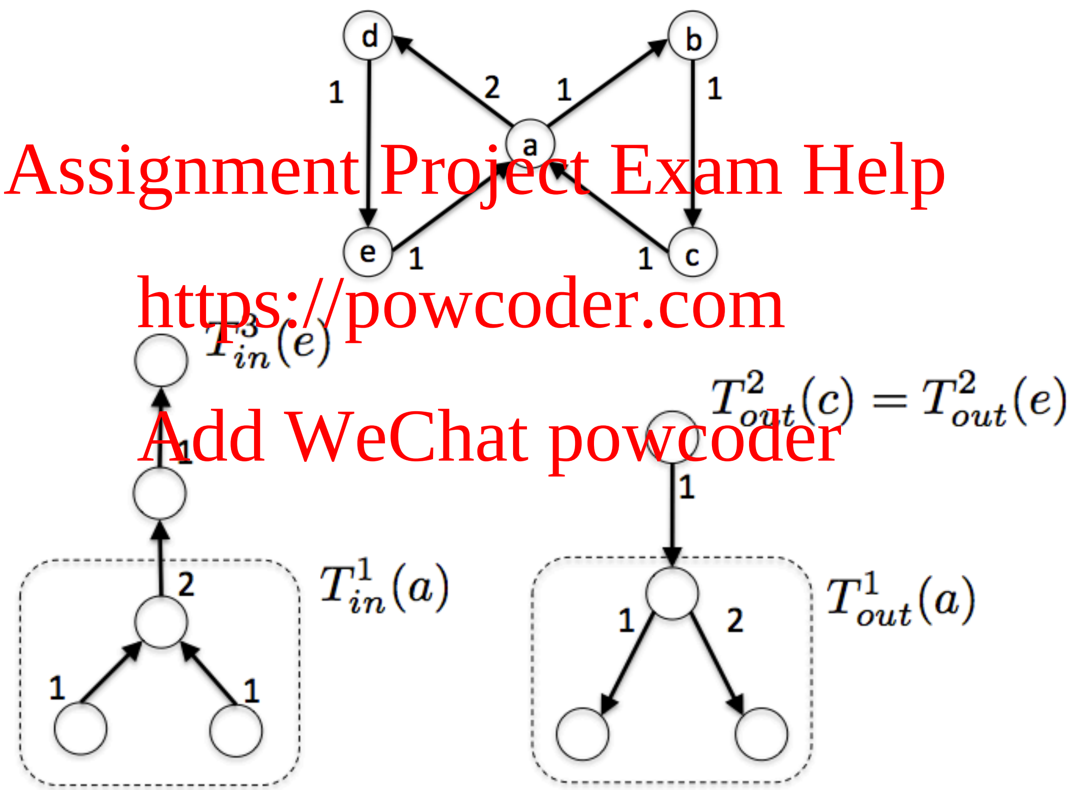
$$O(\delta + \delta \log(n/\delta))$$

  for bidirectional graphs with port numberings

[a] We won't discuss details for these claims.

Views: Directed Networks

- A view can be computed by a node on a network using a distributed deterministic algorithm

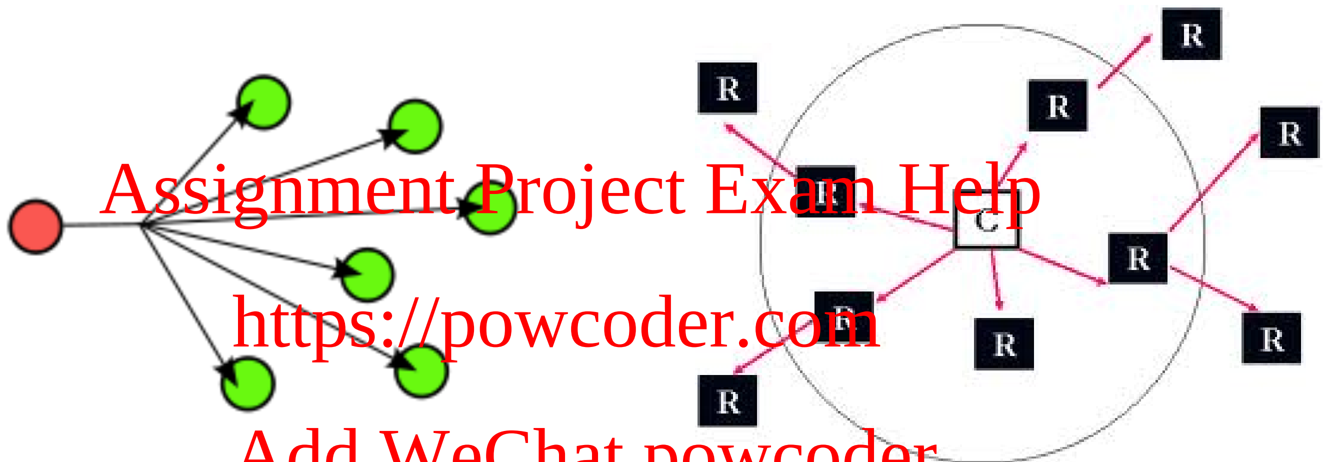


- In directed networks we have "in" and "out" views at a node.

Broadcasting

Broadcasting

- Broadcasting refers to a method of transferring a message to all recipients "at once" in a network.



  – It is initiated by a single processor, the source.
  – The source sends a message to all other nodes in the system.

- In a typical network it may not be possible to send a message "at once" since there might be multiple hops from the source to the rest of the nodes.

Graph Concepts in Broadcasting

- The distance $d(u, v)$ between nodes u and v in an undirected graph G is the number of hops of a minimum length path between u and v.

- The radius
  - of a node u is the maximum distance between u and any other node in the graph.

$$R(u) = \max_{v} d(u, v)$$

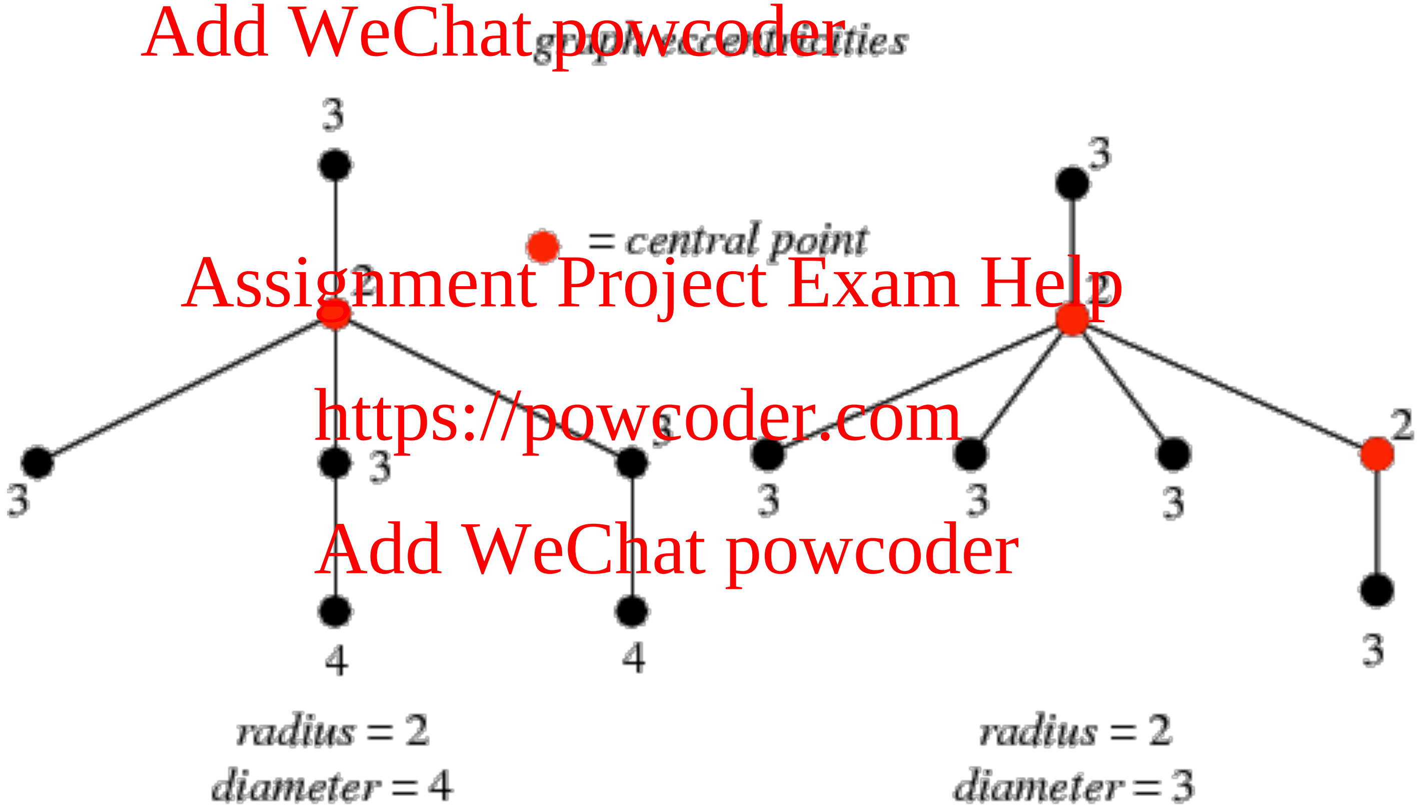
  - of a graph is the minimum radius of any node in the graph.

$$R = \min_{u} R(u)$$

- The radius and diameter of a graph are called graph eccentricities.

Examples: Graph Eccentricities

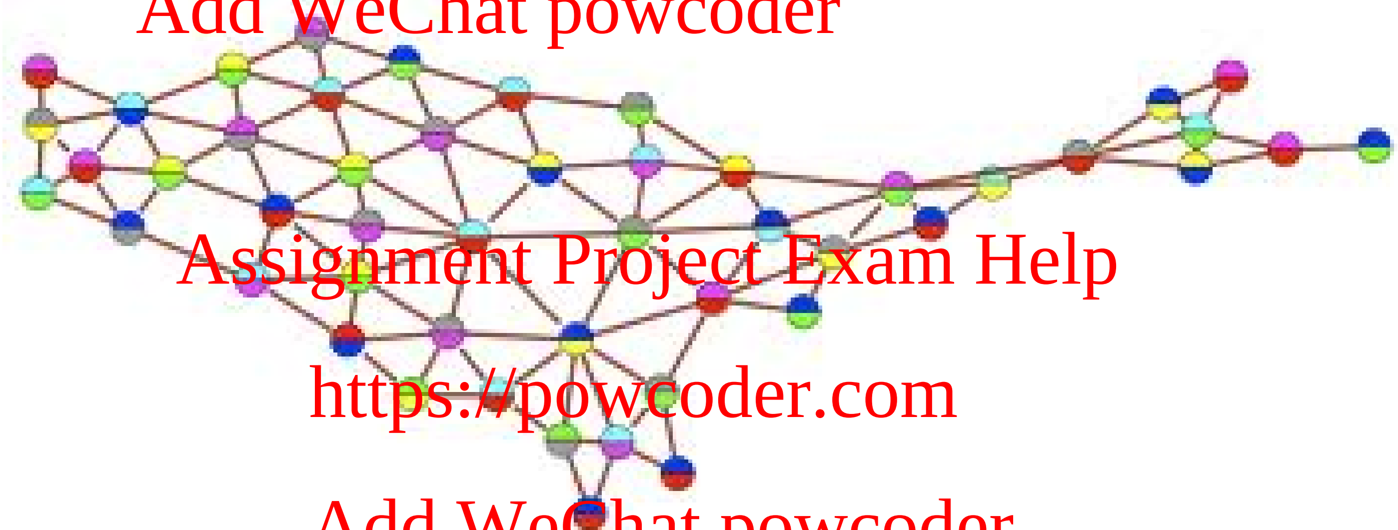
- Distance



- Radius
- Diameter

Examples: Graph Eccentricities

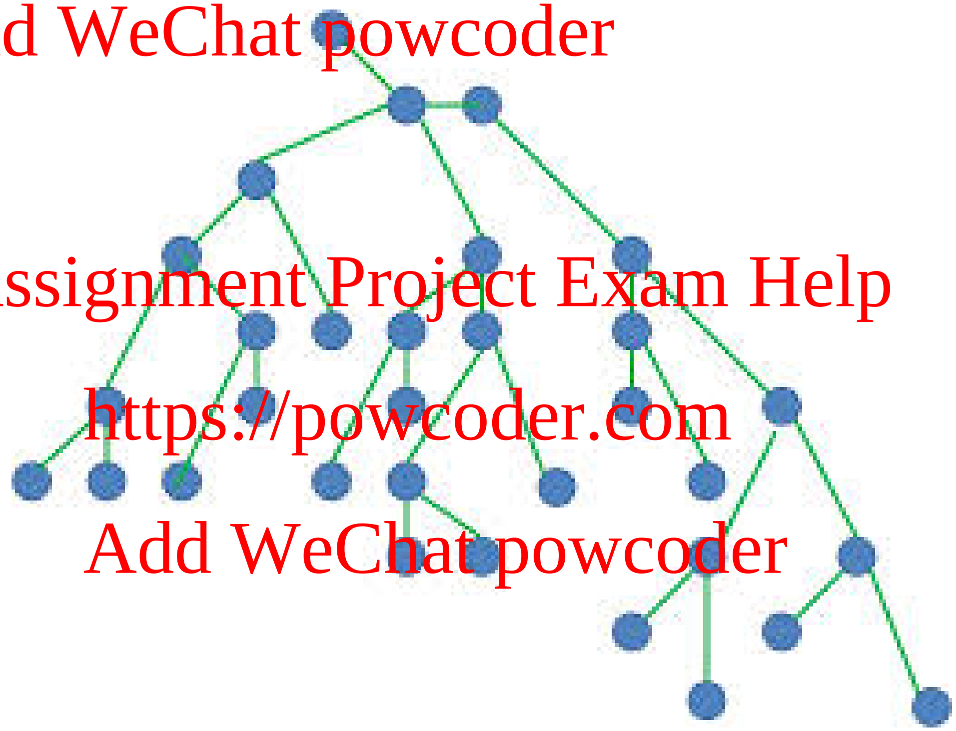
- Radius, Diameter



- There is a close relationship between the radius R and the diameter D of a graph
  - $R \leq D \leq 2R$.

Examples: Graph Eccentricities

- What are the Radius and Diameter?

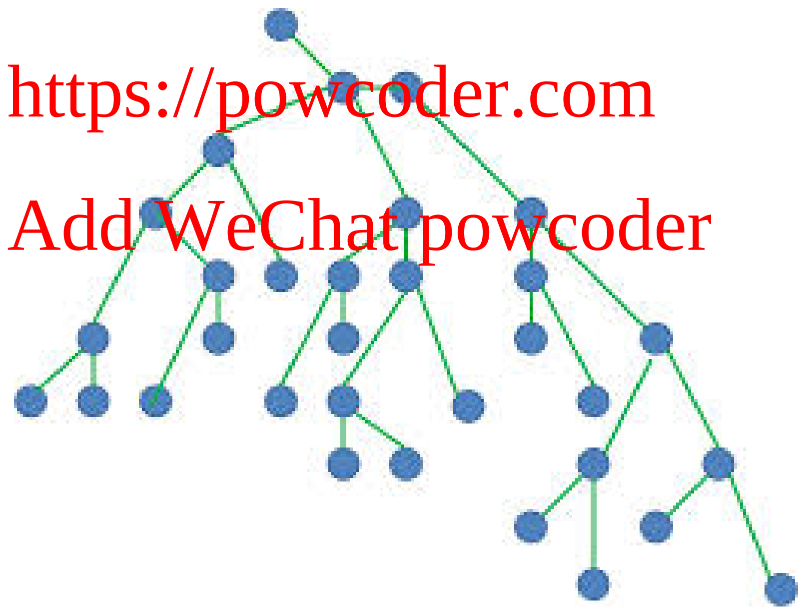


BFS/DFS

BFS Spanning Trees

- Traversal of a graph is performed by visiting all of its vertices in some predefined order.

- **Breadth-First-Search Tree.** A breadth-first-search tree T of a graph G is a spanning tree of G such that for every node of G, the tree path is a minimum hop path to the root.



- Of course a root must be specified!

BFS Algorithm[a]

- **BFS Algorithm:** Input a graph $G = (V, E)$

  Proceed by layers;

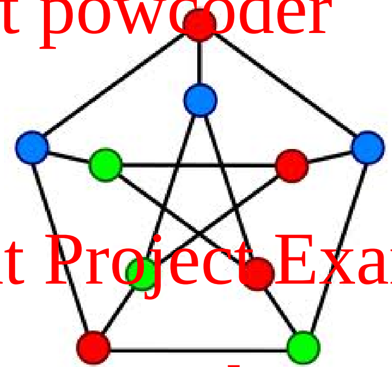
  1. mark the root r;
  2. mark all neighbor vertices that are one hop away from r;
  3. mark new vertices that are one hop away from these neighbors (these are two hops away from r);
  4. and so on.

- It uses a FIFO queue
- It checks whether a vertex has been discovered before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue

[a] Invented in 1945 by Konrad Zuse

BFS Algorithm

- How do you construct a BFS tree from a given graph?



BFS (Distance Computation (1/2))

- It starts by placing the source node s at distance $d(s) = 0$; the distance of all other nodes starts as $d(i) = \infty$.

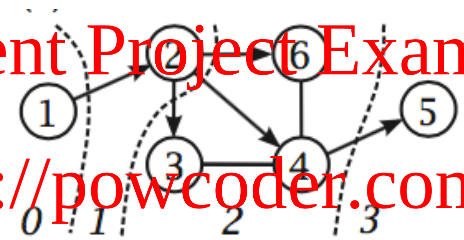
- At the kth step (starting at $k = 0$), all nodes i at distance $d(i) = k$ are examined, and any neighbors j with $d(j) = \infty$ (i.e., not yet discovered) have their distance $d(j)$ set to $k + 1$.

- The process halts when step k finds no such neighbors; $d(j)$ is then the length of the shortest path from s to j, or $d(j) = \infty$ if there is no such path.

BFS (Distance Computation (2/2))

- BFS is the simplest way to search a graph.

- It is suited only for unweighted graphs: ignores edge weights.
- **Example 1:**



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

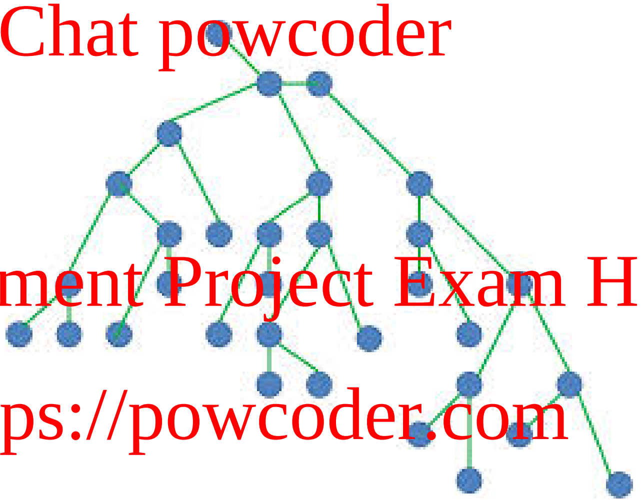
- **Example 2:** In a social network, your friends are at level one and your friends of friends are at level two in a BFS starting at your node.

What is BFS Tree Used for?

- Finding all nodes within one connected component
  - BFS by itself is not enough: some message passing is needed!
- Finding the shortest path between two nodes u and v (with path length measured by number of edges)
  - u and v could be the nodes initiating BFS trees, respectively.
- Testing a graph for bipartiteness
  - Construct a BFS tree from a vertex v and look at all other vertices at odd or even distance from v.
- Doing efficient broadcast
  - from any any node.

DFS Spanning Trees

- For a rooted spanning tree T of a graph G,



let us denote by

- $S(u)$ all the nodes in the subtree of u, and
- $P(u)$ denote all the vertices that exist in a path between u and the root.

DFS Spanning Trees

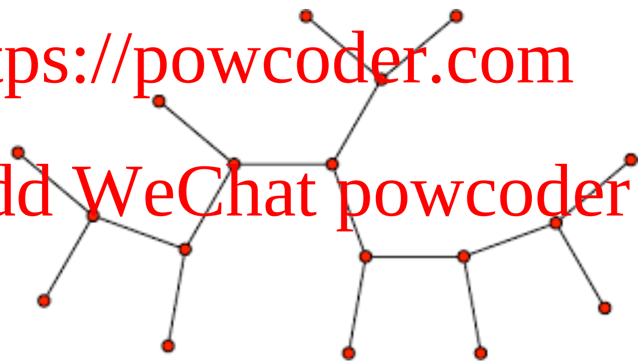
- **DFS Algorithm:** Input a graph $G = (V, E)$

1. Start from a vertex r;
2. visit all possible vertices as far as you can reach;
3. when all vertices are visited, return to the current parent node.

DFS (Depth-First Search) (1/2)

- DFS visits the same nodes as BFS but in a different order.

- If it sees an unvisited node j while examining node i, it fully discovers all unvisited nodes reachable from j and then backtracks to node i to consider the remainder of the nodes adjacent to i.



- It is best described recursively.

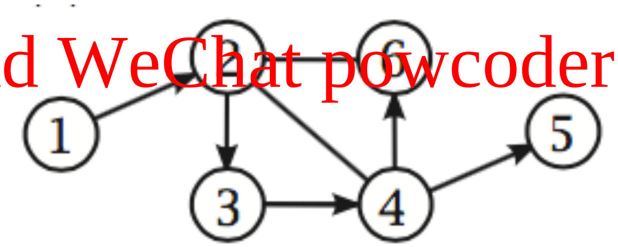
DFS (Depth-First Search) (2/2)

- All nodes start out unvisited.

- DFS(i):

  1. mark i as visited
  2. for all nodes j adjacent to i do:
  3. if node j is not visited $DFS(j)$

- Example



(1 (2 (3 (4 (5 5) (6 6) 4) 3) 2) 1)

What is DFS Tree Used for?

- Finds all of the vertices reachable from a source vertex r in a graph

  - unlike BFS it does not need to search the whole graph.
- Topological sorting.

  - this is because of the way it traverses a directed graph.
- Finding the bridges of a graph.

  - these are edges whose removal disconnects the graph.

- Finding connected components.
  - like BFS.
- Finding strongly connected components.
  - these are maximal "strongly connected components" of a directed graph.

Lower Bounds

- The message complexity of broadcast in an n node graph is at least $n - 1$.

  - This is because every node must receive the message.
  - Which graphs require $n - 1$ message complexity?

- The source's radius is a lower bound for the time complexity.
  - This is because it needs that many hops from a source.

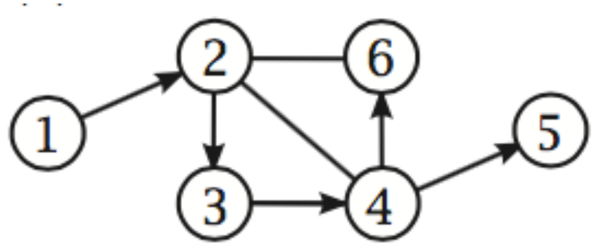
- You can use a pre-computed spanning tree to do broadcast with tight message complexity.

- If the spanning tree is a BFS spanning tree (for a given source), then the time complexity is tight as well.

More on BFS and DFS

- Both BFS and DFS describe a tree; i is the parent of j if the unvisited node j is discovered while examining node i.

- The DFS tree has a rich set of mathematical properties.
  - For example, if "(i" is printed at the start of $DFS(i)$ and "i)" when it finishes (after traversing all its neighbors j), then the result is an expression with properly nested and matching parentheses.
  - The parentheses of two nodes i and j are either nested one within the other, or they are disjoint.



(1 (2 (3 (4 (5 5) (6 6) 4) 3) 2) 1)

Impact of Knowledge: Clean Graphs

- If the graph is stored in adjacency list form, both BFS and DFS take an amount of time that is linear in the size of the graph: $O(|V| + |E|)$, where $|V|$ and $|E|$ are the number of nodes and edges, respectively.

- Knowledge can affect the message complexity!

- Call a graph (network) *clean* if the nodes do not know the topology of the graph.

- If the nodes do not know the topology of the graph (i.e., for a clean network) then the number of edges is a lower bound for the broadcast message complexity.
  - If you do not try every edge, you might miss a whole part of the graph behind it.

Flooding

Outline

- Flooding

- FloodMaxID
- OptFloodMax

Flooding

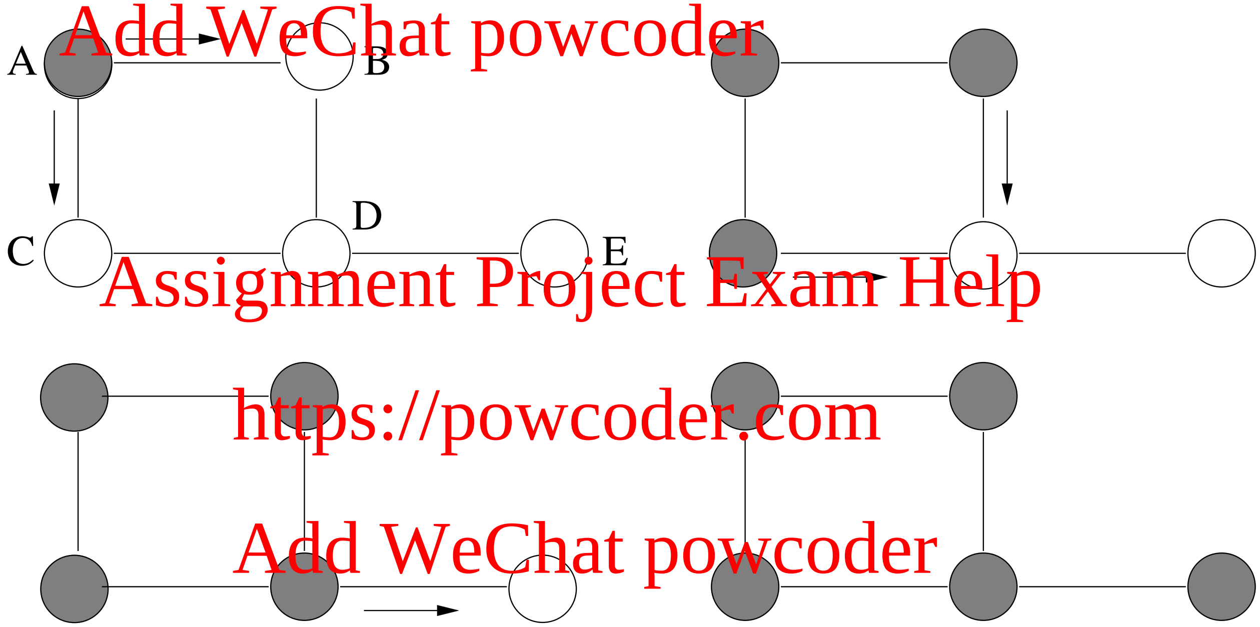
- Used by nodes to identify themselves

- **Flooding Algorithm**

1. The source (root) sends the message to all neighbors.
2. Each other node v upon receiving the message the first time forwards the message to all (other) neighbors.
3. Upon later receiving the message again (over other edges), a node can discard the message.

Flooding Example

- Let A be the initiating node:



- Note that node D receives two messages.

Flooding and Trees

- If node v receives the message first from node u, then node v calls node u parent.

  – Parent relation defines a spanning tree T (nodes receiving more than one message keep message only from one initiator).

  – If flooding algorithm is executed in a synchronous system, then T is a BFS spanning tree (with respect to the root).

- Let $R(s)$ be the radius of the source s in the network.

  – In asynchronous systems the flooding algorithm terminates after $R(s)$ time units.

  – However, the spanning tree constructed may not be a BFS spanning tree.

FloodMaxID

- We give a simple algorithm that causes both leaders and non-leaders to identify themselves.

- The algorithm
  - requires that the processes know the diameter of the network;
  - floods the maximum ID throughout the network,
    - ∗ so we call it the FloodMaxID algorithm.

- The algorithm makes leader election possible in a general network.

Flood MaxID

- **FloodMaxID Algorithm**

  1. Every process maintains a record of the maximum ID it has seen so far (initially its own).
  2. At each round, each process propagates this maximum on all of its outgoing edges.
  3. After D (diameter) rounds, if the maximum value seen is the process's own ID, the process elects itself the leader; otherwise, it is a non-leader.

- FloodMax elects the process with the maximum ID.

Analysis of FloodMax

- Define $i_{\max}$ to be the index of the process with the maximum ID, and $u_{\max}$ to be that user ID.

- **Theorem 3** *In the FloodMax algorithm, process $i_{\max}$ outputs leader and each other process outputs non-leader, within diameter rounds.*

- **Main Claim** After diameter rounds,

  – $status_{i_{\max}} = leader$ and

  – $status_j = non - leader$, for every $j \neq i_{\max}$.

- The key to the proof of this Claim is the fact that
  – after r rounds, the maximum ID has reached every process that is within distance r of $i_{\max}$, as measured along directed paths in G.

FloodMax

- The FloodMax algorithm does not extend directly to the asynchronous setting, because there are no rounds in the asynchronous model.

- However, it is possible to simulate the rounds asynchronously.
  - We simply require each process that sends a round r message to tag that message with its round number r.
  - The recipient waits to receive round r messages from all its neighbors before performing its round r transition.

- By simulating diameter rounds, the algorithm can terminate correctly.

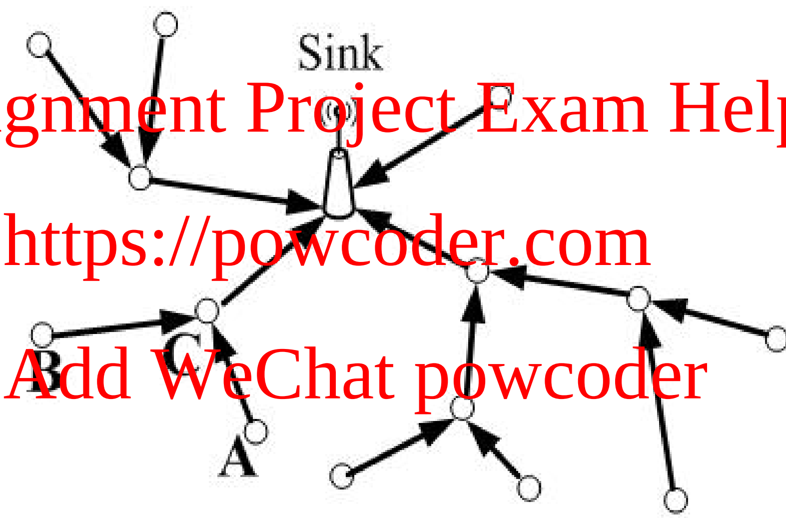
OptFloodMax Algorithm

- There is a simple improvement that can be used to decrease the communication complexity in many cases, although it does not decrease the order of magnitude in the worst case.
- Namely, processes can send their current max user ID values only when they first learn about them, not at every round.

Convergecast

Convergecast

- Convergecast is reversed broadcast:
  - Instead of a root sending a message to all other nodes, all other nodes send information to a root.



- Convergecast is useful for input collection.

Echo Algorithm

Echo Algorithm

- **Requirement:** This algorithm is initiated at the leaves.
  1. A leaf sends a message to its parent.
  2. If an inner node has received a message from each child, it sends a message to the parent.

Complexity Issues: Broadcast and Convergecast (1/2)

- The echo algorithm is paired with the flooding algorithm, which is used to let the leaves know that they should start the echo process; this is known as flooding/echo.

- One can use convergecast for termination detection.
  - If a root wants to know whether all nodes in the system have finished some task, it initiates a flooding/echo;
    - the message in the echo algorithm then means "This subtree has finished the task."

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

- Message complexity of the echo algorithm is $n - 1$,
  - but together with flooding it is $O(m)$, where $m = |E|$ is the number of edges in the graph.

Complexity Issues: Broadcast and Convergecast (2/2)

- The time complexity of the echo algorithm is determined by the depth of the spanning tree (i.e., the radius of the root within the tree) generated by the flooding algorithm.

- The flooding/echo algorithm can do much more than collecting acknowledgements from subtrees.
  - For instance, one can use it to compute the number of nodes in the system, or the maximum ID (for leader election), or the sum of all values stored in the system.

- By combining results one can compute even fancier aggregations, e.g., with the number of nodes and the sum one can compute the average. With the average one can compute the standard deviation. And so on . . .

Application to Leader Election

- Asynchronous broadcast and convergecast can be used to solve the leader election problem in arbitrary graphs
  - without any distinguished source node and
  - without the processes having any knowledge of the number of nodes or the diameter of the network.
- The processes need to have user IDs.

Basic Leader Election Algorithm

- Every node can initiate
  - first a broadcast, and
  - next a convergecast

  in order to discover the maximum user ID in the network.
- The node that finds that the maximum is equal to its own ID elects itself as leader.
- This algorithm uses $O(n|E|)$ messages.

(Directed Graph) BFS and Applications

Construction of BFS

- How do we perform breadth-first search (BFS) in a network based on an arbitrary strongly connected directed graph having a distinguished source node?

- We consider how to establish a breadth-first spanning tree for the (di)-graph.

- Motivation for constructing such a tree comes from the desire to have a convenient structure to use as a basis for broadcast communication.

- The BFS tree minimizes the maximum communication time from the process at the distinguished node to all other processes in the network
  - To do this run BFS from each node of the graph and compare values obtained ateach node.

Construction of BFS

- We suppose that the network is strongly connected and that there is a distinguished source node i_0.

- The algorithm is supposed to output the structure of a breadth-first directed spanning tree of the network graph, rooted at i_0.

- The output should appear in a distributed fashion: each process other than i_0 should have a parent component that gets set to indicate the node that is its parent in the tree.

- As usual, processes only communicate over directed edges.
- Processes are assumed to have user IDs but to have no knowledge of the size or diameter of the network.

SynchBFS

- The basic idea for this algorithm is the same as for the standard sequential breadth-first search algorithm.

- **SynchBFS Algorithm**

1. At any point during execution, there is some set of processes that is *marked*: initially just i_0.
2. Process i_0 sends out a search message at round 1, to all of its outgoing neighbors.
3. At any round, if an unmarked process receives a search message, it marks itself and chooses one of the processes from which the search has arrived as its parent.
4. At the first round after a process gets marked, it sends a search message to all of its outgoing neighbors.

Analysis of SynchBFS

- We can prove the invariant that
  - after r rounds, every process at distance d from i_0 in the graph, $1 \leq d \leq r$, has its parent pointer defined; moreover, each such pointer points to a node at distance $d-1$ from i_0.

  This invariant can, as usual, be proved by induction on the number of rounds.
- The time complexity is at most diameter rounds.
- The number of messages is just $|E|$
  - a search message is transmitted exactly once on each directed edge.

Applications of BFS

- Message Broadcast

- Global computation
- Electing a leader
- Computing the diameter

Message Broadcast: Piggybacking

- The SynchBFS algorithm can easily be augmented to implement message broadcast.

- If a process has a message M that it wants to communicate to all of the processes in the network,
  - it can simply initiate an execution of SynchBFS with itself as the root, piggybacking message M on the search message it sends in round 1.

- Other processes continue to piggyback M on all their search messages as well.
  - Since the tree eventually spans all the nodes, message M is eventually delivered to all the processes.

Global Computation

- This means
  - Collection of information from throughout the network or,
  - more generally, the computation of a function based on distributed inputs.

- For example,
  - consider the problem in which each process has a nonnegative integer input value and we want to find the sum of all the inputs in the network.
  - Using a BFS tree, this can be done easily (and efficiently) as follows.

Global Computation

- Starting from the leaves, "fan in" the results in a convergecast procedure, as follows:

  1. Each leaf sends its value to its parent;
  2. each parent waits until it gets the values from all its children, adds them to its own input value, and then sends the sum to its own parent.

- The sum calculated by the root of the BFS tree is the final answer.

Electing a Leader

- Using SynchBFS, an algorithm can be designed to elect a leader in a network with IDs, even when the processes have no knowledge of n or diameter.

  1. Namely, all the processes can initiate breadth-first searches in parallel.
  2. Each process i uses the tree thereby constructed and the global computation procedure just described to determine the maximum ID of any process in the network.
  3. The process with the maximum ID then declares itself to be the leader, and all others announce that they are not the leader.

- If the graph is undirected, the time is $O(diameter)$ and the number of messages is $O(n|E|)$.

Computing the Diameter

- The diameter of the network can be computed by having all processes initiate breadth-first searches in parallel.

  1. Each process i uses the tree thereby constructed to determine $\max - dist$, defined to be the maximum distance from i to any other process in the network.

Assignment Project Exam Help

  2. Each process i then reuses its breadth-first tree for a global computation to discover the maximum of the $\max - dist$ values.

https://powcoder.com
Add WeChat powcoder

- If the graph is undirected, the time is $O(diam)$ and the number of messages is $O(n|E|)$, where $diam$ is the diameter of the graph.
- The diameter thus computed could be used, for example, in the leader-election algorithm FloodMax.

Exercises[a]

1. Explain why every tree is a bipartite graph.

2. Let T be a graph with n vertices. Then the following statements are equivalent.

   (a) T is connected and has no cydes.

   (b) T has $n-1$ edges and has no cycles.

   (c) T is connected and has $n-1$ edges.

   (d) T is connected and the removal of any edge disconnects T.

   (e) Any two vertices of T are connected by exactly one path.

   (f) T contains no cycles, but the addition of any new edge creates a cycle.

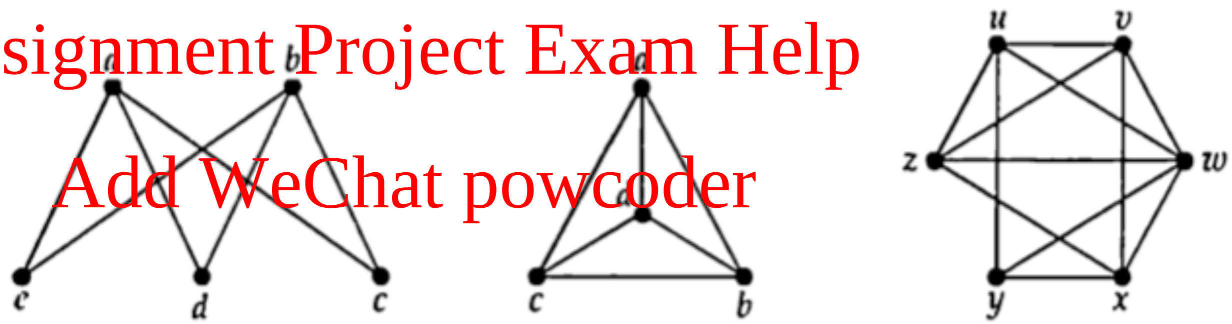
3. Give an algorithm to compute the diameter and radius of a tree.

4. Determine the size of a message which propagates for r hops in

[a]Not to hand in!

"Synchronous Algorithm Canonical Form". More specifically, consider a complete binary rooted tree with height n. Label the ports at an interior node as L, R (for the Left and Right siblings at a node), and P for its parent. Do the same in an analogous manner for the root and the leaves. For each $r \leq n$ and each node v construct the r-view at this node.

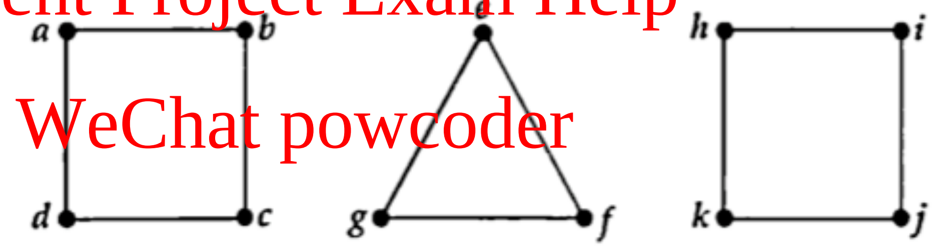
5. A connected graph is Hamiltonian if there is a cycle, that includes every vertex exactly once (such a cycle is called Hamiltonian). A connected graph is semi-Hamiltonian if there is a path (but not a cycle) that includes every vertex exactly once (such a path is called semi-Hamiltonian). Determine which of the following graphs are semi-Hamiltonian, and write down a corresponding semi-Hamiltonian path where possible:



6. A forest is a graph (not necessarily connected), each of whose components is a tree.
   (a) Let G be a forest with 11 vertices and k components. How many edges does G have?
   (b) Construct a forest with 12 vertices and 9 edges.
   (c) Is it true that every forest with k components has at least $2k$ vertices of degree 1?

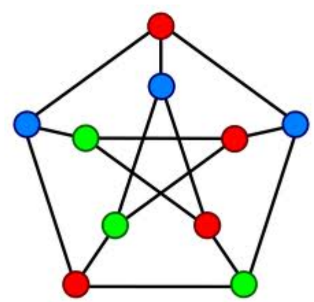
7. A spanning forest in a graph G (not necessarily connected) is obtained by constructing a spanning tree for each component of G.

(a) Find a spanning forest for the following graph.



(b) Let G be a graph, and let F be a subgraph of G. If F is a forest which includes all vertices of G, is F necessarily a spanning forest of G?

8. Find three spanning trees in the Petersen graph (depicted below):



9. Prove that trees and forests are bipartite graphs.

10. Prove that, in a bipartite graph, every cycle has an even number of edges. Conversely, prove that, if every cycle of a graph has an even number of edges, then the graph is bipartite.
    Hint: Consider a connected graph G. Choose a vertex v in G and consider those vertices whose minimum distance from v is even and those whose minimum distance from v is odd. To which vertices are the "odd" vertices adjacent? To which vertices are the "even" vertices adjacent?