

The Generic Search Algorithm

- Recall our generic graph search algorithm:

The Search Algorithm

Put $(*, r)$ into a bag

While the bag is non-empty

 Remove (u, v) from the bag

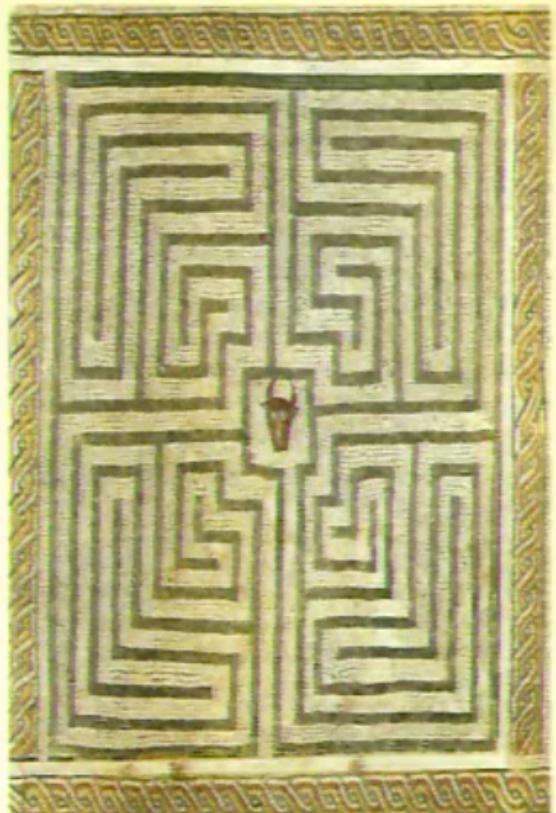
 If v is unmarked

 Mark v

 Set $p(v) \leftarrow u$

 For each arc (v, w)

 Put (v, w) into the bag



Data Structures

- Recall, three choices of **data structure** give fundamental algorithms:

Queue.



Breadth First Search.

Stack.



Depth First Search.

Priority Queue.



Minimum Spanning
Tree Algorithm.

Depth First Search

- Using a Stack (LIFO) data structure produces:

Depth First Search Algorithm

Add $(*, r)$ to a Stack

While the Stack is non-empty

 Remove the first arc (u, v) from the Stack

 If v is unmarked

 Mark v

 Set $p(v) \leftarrow u$

 For each arc (v, w)

 Add (v, w) into the back of the Stack



Depth First Search Trees



- Recall, we proved that the search algorithm produces a **search tree**.

Theorem. Let G be a connected, undirected graph.

Then the predecessor edges form a tree rooted at r .

- However, the structure of the DFS tree is quite different from that of a BFS tree.
- So let's investigate this structure...

Depth First Search Trees

- Recall, we proved that the search algorithm produces a search tree.

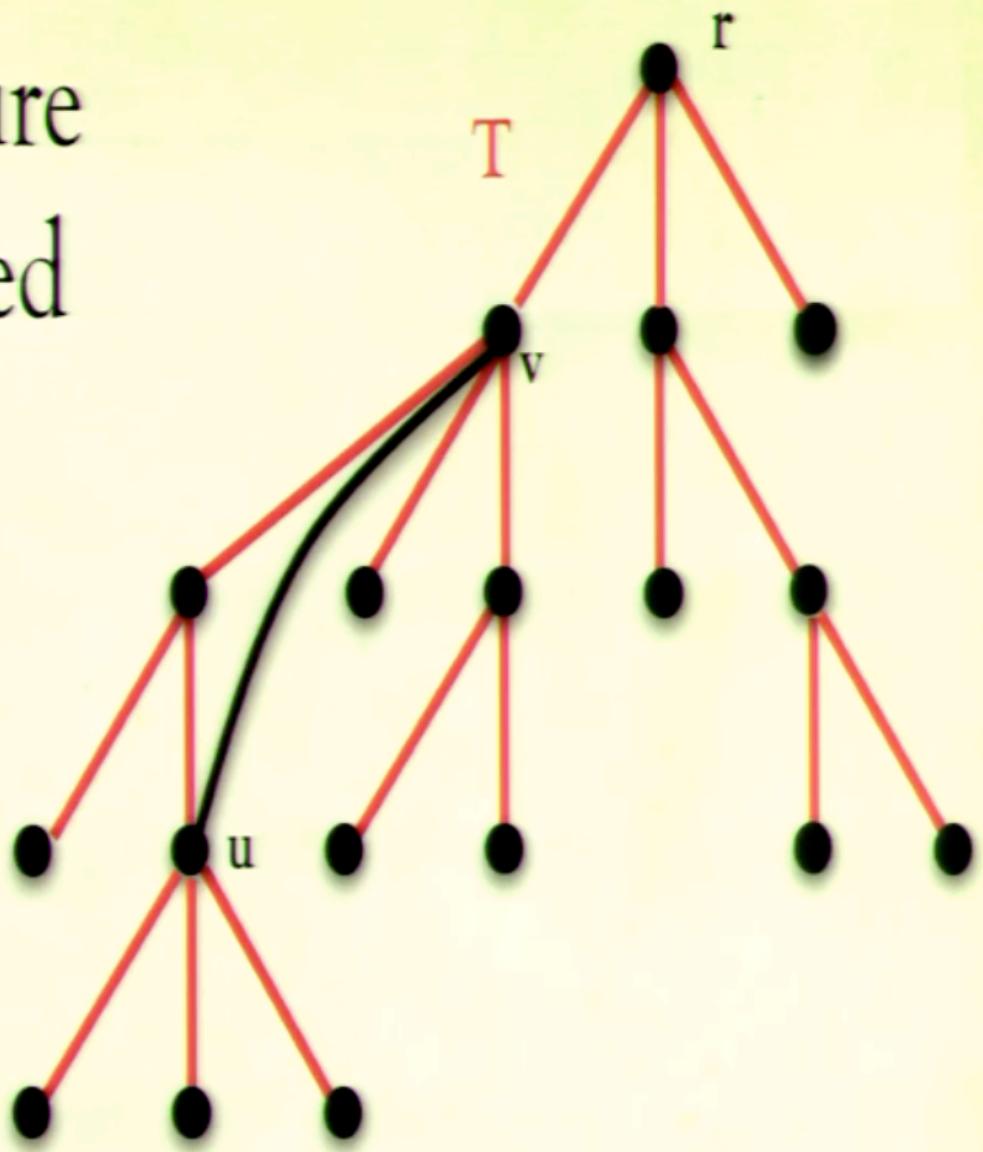


Theorem. Let G be a connected, undirected graph.

Then the predecessor edges form a tree rooted at r .

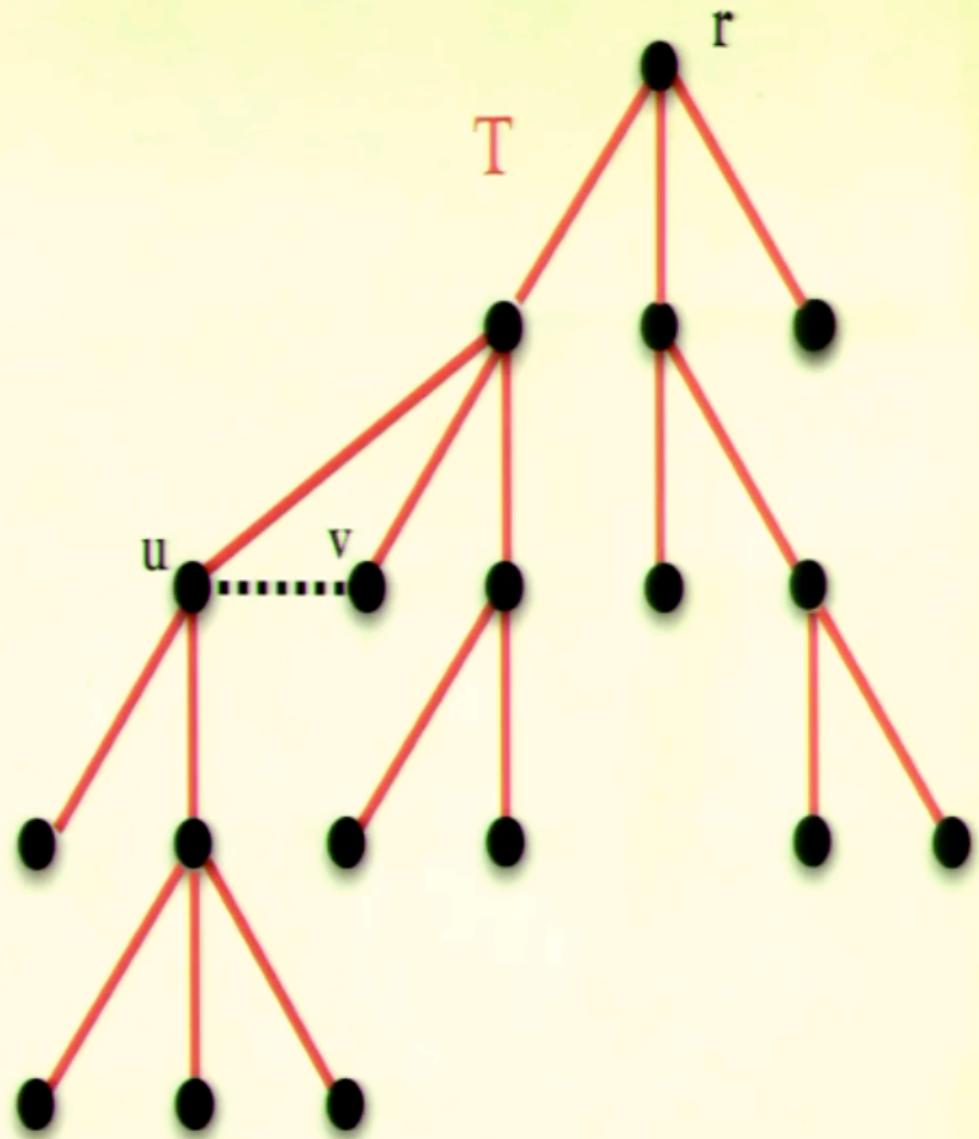
- However, the structure of the DFS tree is quite different from that of a BFS tree.
- So let's investigate this structure...

Edge Structure in Undirected Graphs



- Depth First Search partitions the edges of an undirected graph into two types:
 - Tree Edges.** Predecessor edges in the DFS tree T .
 - Back Edges.** Edges where one endpoint is an ancestor of the other endpoint in T .

Cross Edges



- When we perform a DFS, there no crosses of the following type:
Cross Edges. Edges where neither endpoint is an ancestor of the other in T .



Depth First Search (Recursive Description)

- The Depth First Search algorithm can be defined **recursively**:

RecursiveDFS(r)

Mark r

For each edge (r, v)

 If v is unmarked

 Set $p(v) \leftarrow r$

RecursiveDFS(v)

- Using this formulation the aforementioned structure of the graph edges is easy to show...

`RecursiveDFS(u)`

Mark u

For each edge (u, v)

If v is unmarked

Set $p(v) \leftarrow u$

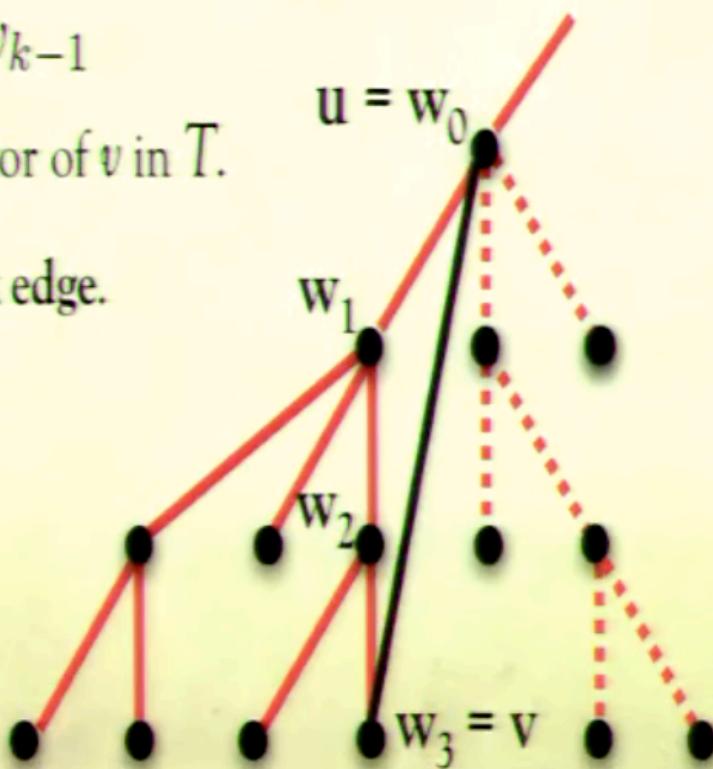
`RecursiveDFS(v)`

Ancestral Edges [cont.]

Proof [cont.]

Case 2: v is marked when `RecursiveDFS(u)` examines (u, v) .

- But v was marked after u , so it was marked during `RecursiveDFS(u)`
- Thus, we have a series of vertices $\{u = w_0, w_1, \dots, w_{\ell-1}, w_\ell = v\}$ where $p(w_k) = w_{k-1}$
 - $\Rightarrow u$ is an ancestor of v in T .
 - $\Rightarrow (u, v)$ is a back edge.



□

On DFS Edges: Tree, Back and Cross



Theorem. Let T be a DFS tree in an undirected graph G . Then, for every edge (u,v) , either u is an ancestor of v in T or v is an ancestor of u .

Corollary. Let T be a DFS tree in an undirected graph G . Then, every non-tree edge is a back edge.

Previsit and Postvisit

- We can add a clock to record when we visit a vertex for the first (*previsit*) and last time (*postvisit*):



$\text{clock} \leftarrow 0$

RecursiveDFS(r)

Mark r

$\text{pre}(r) \leftarrow \text{clock}$

$\text{clock} \leftarrow \text{clock} + 1$

For each edge (r, v)

If v is unmarked

Set $p(v) \leftarrow r$

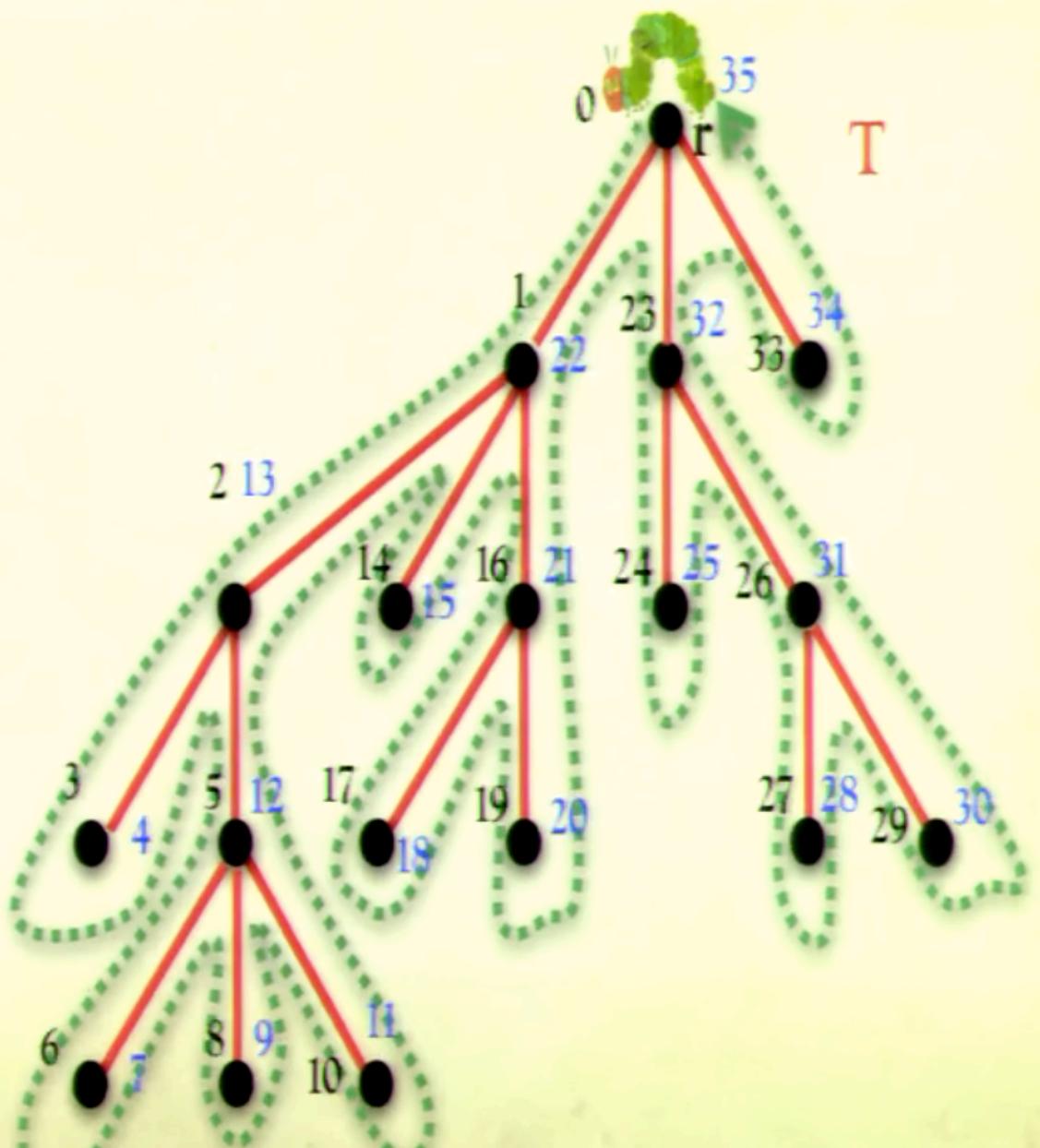
RecursiveDFS(v)

$\text{post}(r) \leftarrow \text{clock}$

$\text{clock} \leftarrow \text{clock} + 1$

- The previsit and postvisit times will highlight more properties of DFS...

Previsit and Postvisit Times



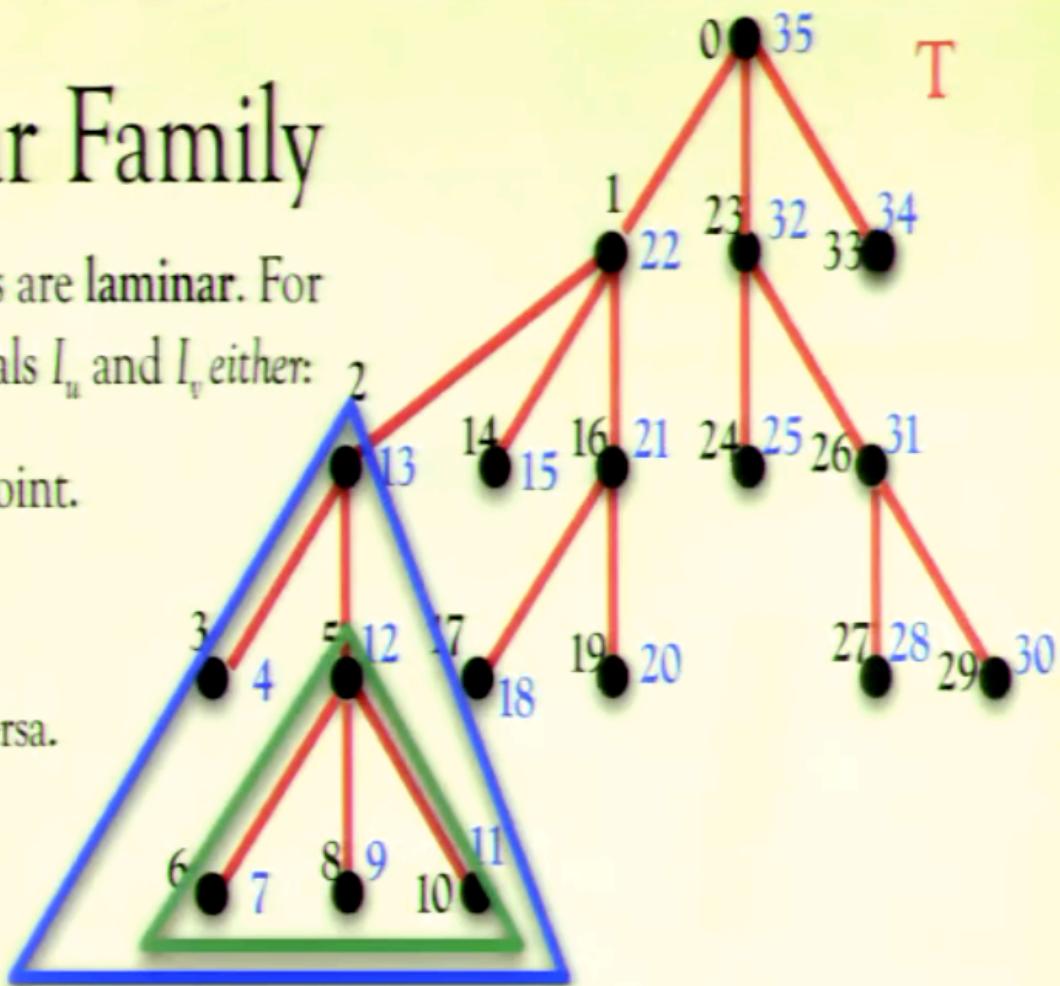
A Laminar Family

- The set of intervals are **laminar**. For each pair of intervals I_u and I_v either:

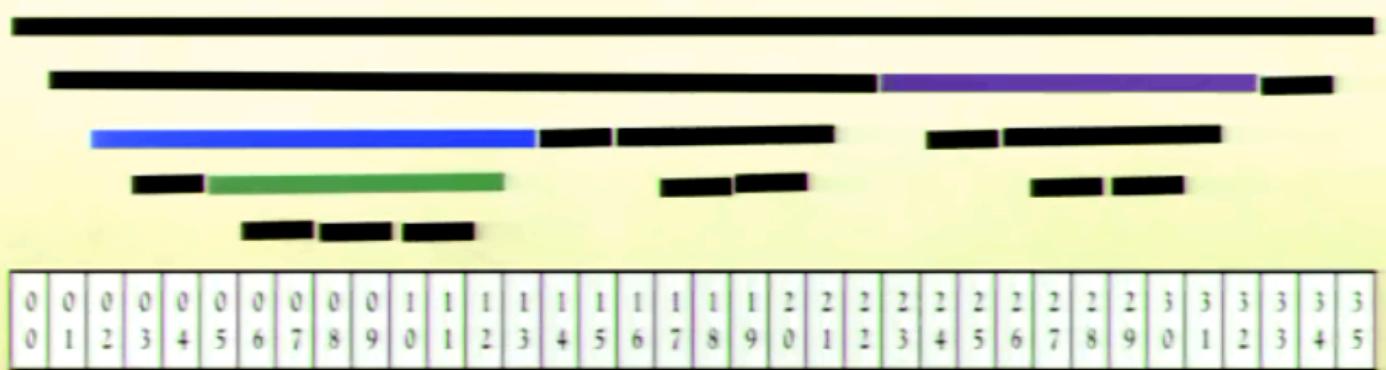
- I_u and I_v are disjoint.

- or*

- $I_u \subseteq I_v$ or vice versa.

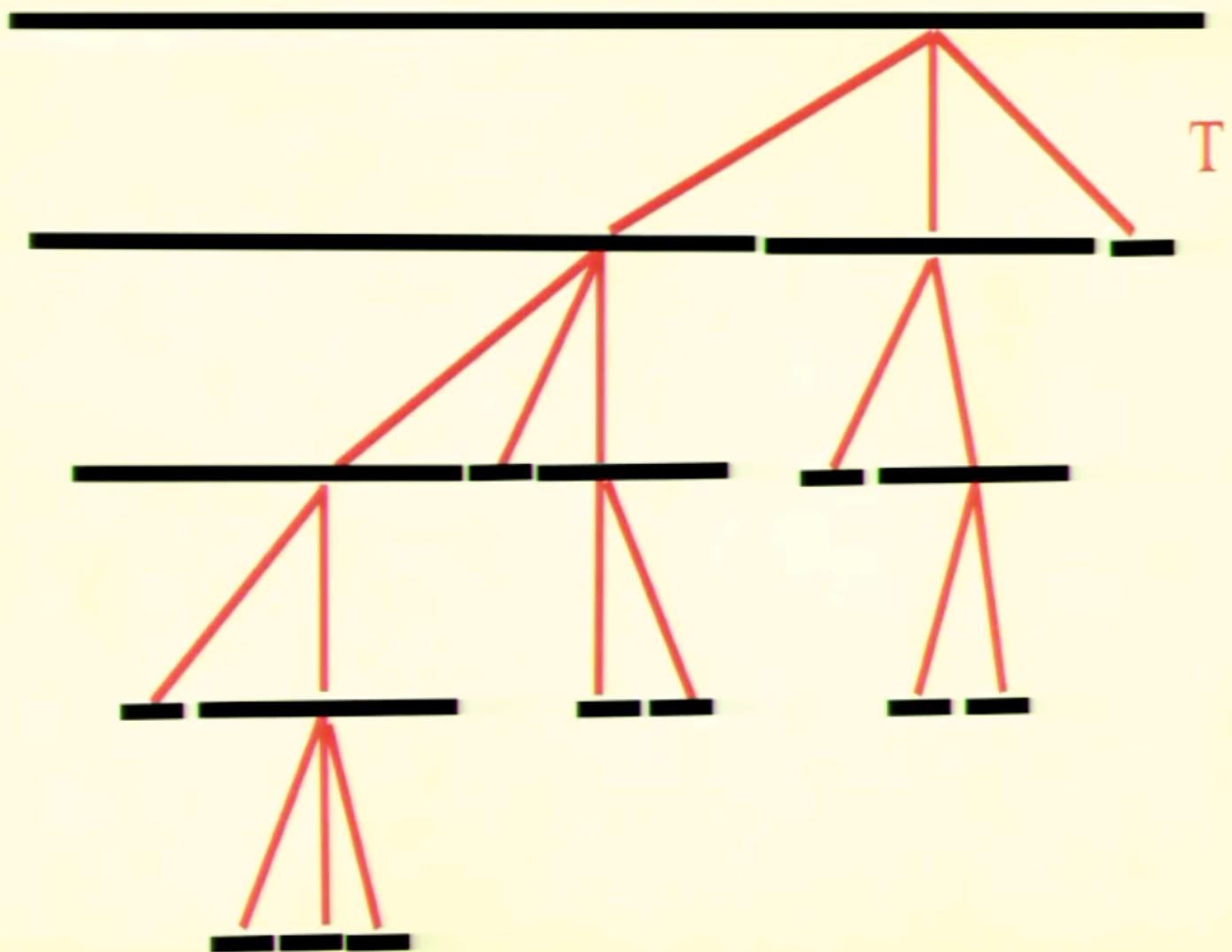


- This follows from the definition of the **worm crawl**, as the intervals correspond to subtrees.



The Tree of Intervals

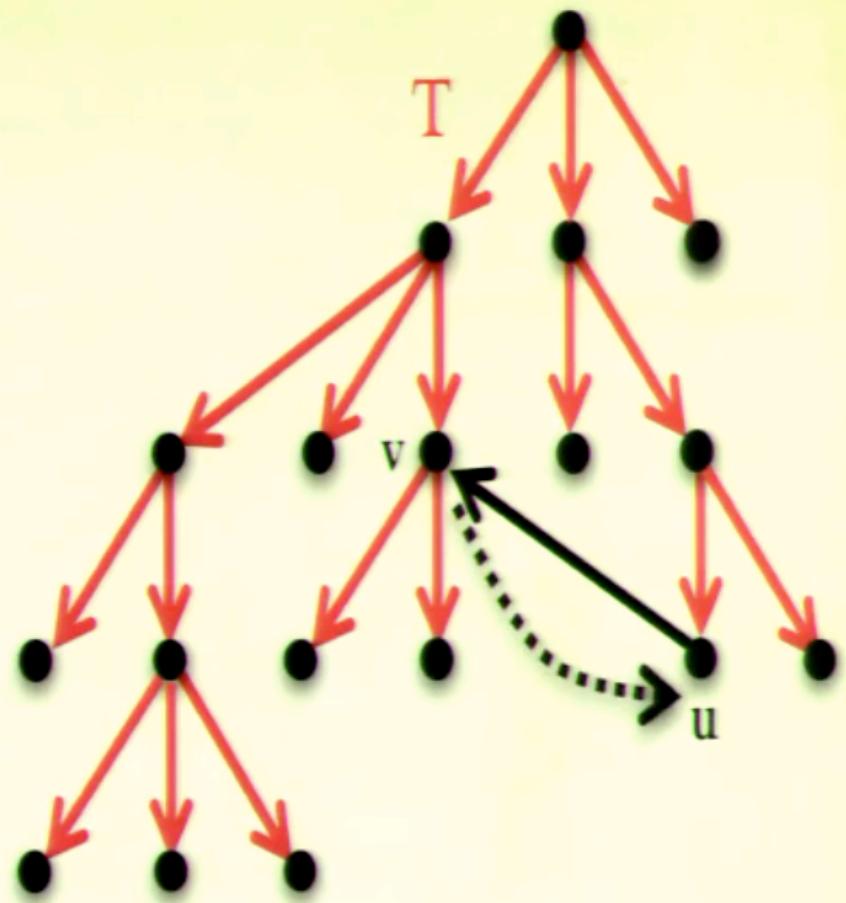
- Add an edge between an interval and the smallest interval that contains it.



- This is exactly the DFS tree!

0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

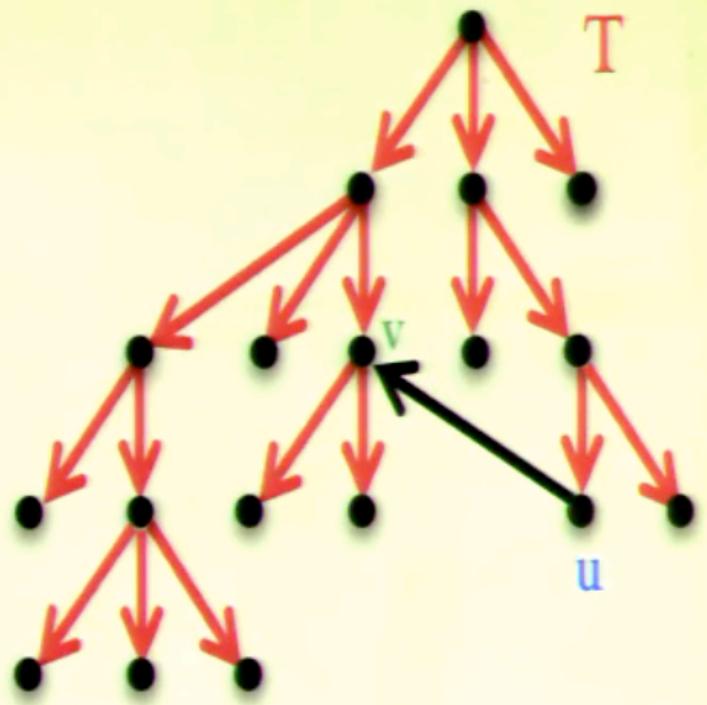
Arc Structure in Directed Graphs



- Depth First Search partitions the edges of an directed graph into four types:
 - Tree Arcs.** Predecessor edges in the DFS tree T .
 - Forward Arcs.** Arcs (u,v) where u is an ancestor of v .
 - Backward Arcs.** Arcs (u,v) where u is descendant of v .
 - Cross Arcs.** Non-Ancestral arcs (u,v) where u is marked after v .

The Intervals for Directed Graphs

- The intervals for an arc (u, v) have the following properties:



Tree Arcs. Predecessor edges in the DFS tree T .



Forward Arcs. Arcs (u, v) where u is an ancestor of v .



Backward Arcs. Arcs (u, v) where u is descendant of v .



Cross Arcs. Non-Ancestral arcs (u, v) where u is marked after v .



Post-Visit Times

- The post-visit times for an arc (u, v) satisfy the following:

Tree Arcs.



$$\text{post}(v) < \text{post}(u)$$

Forward Arcs.



$$\text{post}(v) < \text{post}(u)$$

Backward Arcs.



$$\text{post}(u) < \text{post}(v)$$

Cross Arcs.



$$\text{post}(v) < \text{post}(u)$$

- Observe that $\text{post}(u) < \text{post}(v)$ only for backward arcs.



Directed Acyclic Graphs

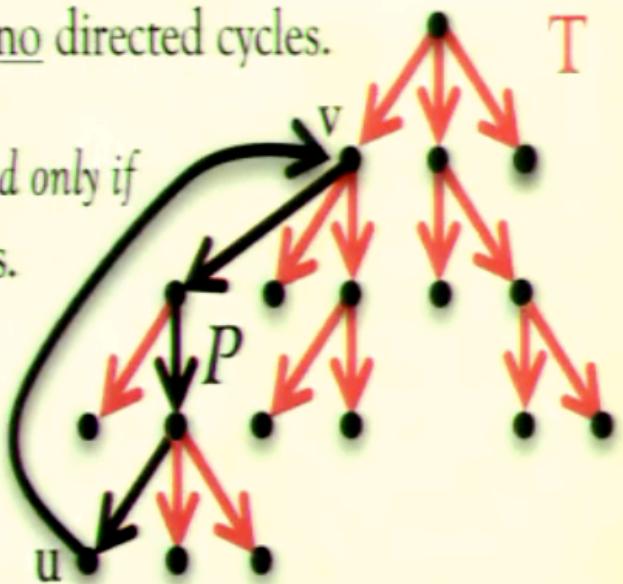
- A directed graph is acyclic if it contains no directed cycles.

Theorem. A directed graph G is acyclic if and only if depth first search produces no backward arcs.

Proof.

(\Rightarrow)

- Let DFS give a backward arc (u, v) .
- By definition, u is a descendant of v in the DFS tree T .
- So there is a path $P = \{v = v_0, v_1, \dots, v_k = u\}$ in T .
 $\implies P \cup (u, v)$ is a directed cycle in G .



Theorem. A directed graph is acyclic if and only if depth first search produces no backward arcs.

Proof [cont.]

(\Leftarrow)

- Assume DFS gives no backward arcs.
- Suppose there is a directed cycle $C = \{v_0, v_1, \dots, v_k, v_0\}$ in T .
- As there are no backward arcs we have that:

$$post(v_0) > post(v_1) > post(v_2) \dots > post(v_k) > post(v_0)$$

- Thus we obtain the contradiction that $post(v_0) > post(v_0)$.
- So G is acyclic.



Corollary. There is a linear time algorithm to test whether or not a directed graph is acyclic

Proof.

- Run DFS and then check if any arc is a backward arc.



Topological Orderings

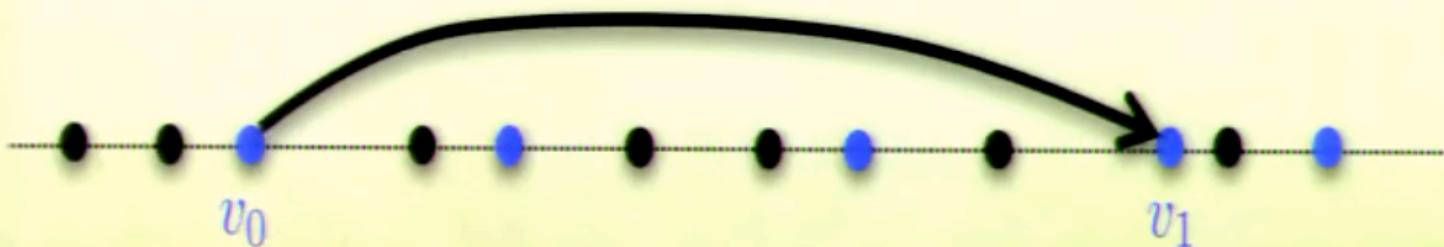
- A directed graph has a **topological ordering** (*linear ordering*) if the vertices can be horizontally ordered such that every arc is from right to left.

Theorem. A directed graph G has a topological ordering if and only if depth first search produces no backward arcs.

Proof.

(\Rightarrow)

- If DFS produces a **backward arc** then G contains a cycle C .
- Let the cycle be $C = \{v_0, v_1, \dots, v_k, v_0\}$ where wlog v_0 is the **leftmost** vertex of the cycle in the order.



\implies The arc (v_0, v_1) goes from **left to right**.

Proof [cont.]

(\Leftarrow)

- Assume DFS gives no backward arcs.
- Then for every arc (u, v) we have :

$$post(u) > post(v)$$

- Thus, if we place each vertex v at coordinate $post(v)$ on the x-axis then every arc goes from right to left.
- Thus G has a topological ordering.



Corollary. There is a linear time algorithm to test whether or not a directed graph has a topological ordering.

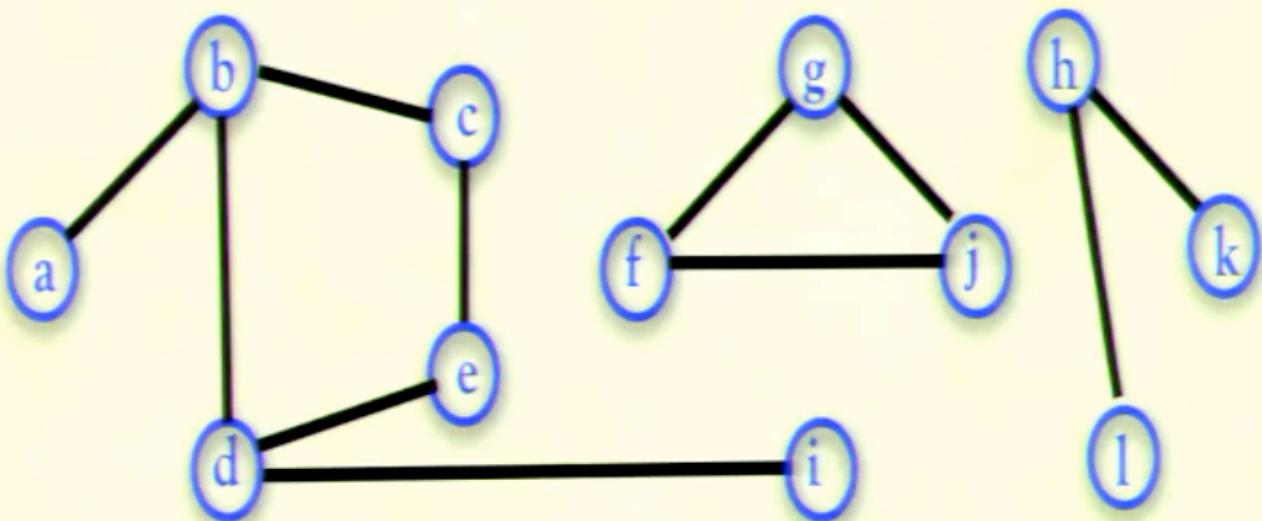


Corollary. A directed graph is acyclic if and only if it has a topological ordering.



Disconnected Graphs

- So far, for undirected graphs, we have focussed on **connected** graphs.
- For disconnected undirected graphs, a DFS search will find the entire component containing the root vertex.



- Thus, after we have searched one component, we must run DFS from a **new root node** to find the next component.

Strongly Connected Components

- In a directed graph, a subgraph $S \subseteq V$ is strongly connected if for every pair $u, v \in S$ there is a directed path from u to v and vice versa.
- Surprisingly, DFS can be used to decompose a directed graph into its strongly connected components extremely quickly...

Theorem. There is a linear time algorithm to find the strongly connected components of a directed graph.

Proof.

- See the Dasgupta or Erickson texts.

