Announcements

• Homework 0 due today
• Homework 1 online, due next week
• Jaibei Han’s office hours will be Monday, Thursday, and Friday from 4-5pm.
  – The syllabus has been updated.
  – Sorry for the inconvenience.
Last Time

- **Graph Definition**
  - Edges connect pairs of vertices

- **Explore/DFS**
  - $O(|V|+|E|)$ runtime
  - $\text{explore}(v)$ discovers all vertices reachable from $v$
**Explore and DFS**

```plaintext
e Expl o re (v)
   v . v is it ed ← true
   For each edge (v, w)
      If not w . v is it ed
         e x p l o re (w)

D F S (G)
   Mark all v ∈ G as unvisited
   For v ∈ G
      If not v . v is it ed, explore(v)
```
Today

• Connected Components
• Pre- and Post- orderings
• [Directed graphs]
Connected Components

- Want to understand which vertices are reachable from which others in graph G.
- \texttt{explore(v)} finds which vertices are reachable from a given vertex.

\textbf{Theorem:} The vertices of a graph G can be partitioned into \textit{connected components} so that v is reachable from w if and only if they are in the same connected component.
Question: Connected Components

How many connected components does the graph below have?

A) 0  
B) 1  
C) 2  
D) 3  
E) 4
Problem: Computing Connected Components

Given a graph G, compute its connected components.

**Easy:** For each v, run \( \text{explore}(v) \) to find vertices reachable from it. Group together into components.

Runtime: \( O(|V|(|V|+|E|)) \).

**Better:** Run \( \text{explore}(v) \) to find the component of v. Repeat on unclassified vertices.
DFS lets us do this!

ConnectedComponents(G)

CCNum ← 0
For v ∈ G
    v.visited ← false
For v ∈ G
    If not v.visited
        CCNum++
        explore(v)
Runtime O(|V|+|E|).

explore(v)

v.visited ← true
v.CC ← CCNum
For each edge (v,w)
    If not w.visited
        explore(w)
Example

CCNum: 4

A - B - C - H - I

D - E - F - G
Discussion about DFS

What does DFS actually do?

• No output.
• Marks all vertices as visited.
• Easier ways to do this.

However, DFS also is a useful way to explore the graph. By *augmenting* the algorithm a bit (like we did with the connected components algorithm), we can learn useful things.
Pre- and Post- Orders

Augment how?

• Keep track of what algorithm does & in what order.

• Have a “clock” and note time whenever:
  – Algorithm visits a new vertex for the first time.
  – Algorithm finishes processing a vertex.

• Record values as $v.pre \text{ and } v.post$. 
Computing Pre- & Post- Orders

ConnectedComponents(G)

\[
\begin{align*}
\text{clock} & \leftarrow 1 \\
\text{For } v \in G \\
& \quad \text{v.visited} \leftarrow \text{false} \\
\text{For } v \in G \\
& \quad \text{If not v.visited} \\
& \quad \quad \text{explore}(v) \\
& \quad \text{v.visited} \leftarrow \text{true} \\
& \quad \text{v.pre} \leftarrow \text{clock} \\
& \quad \text{clock}++ \\
& \quad \text{For each edge } (v,w) \\
& \quad \quad \text{If not w.visited} \\
& \quad \quad \quad \text{explore}(w) \\
& \quad \text{v.post} \leftarrow \text{clock} \\
& \quad \text{clock}++
\end{align*}
\]

Runtime $O(|V|+|E|)$. 
Example
What do these orders tell us?

**Prop:** For vertices $v$, $w$ consider intervals $[v.\text{pre}, v.\text{post}]$ and $[w.\text{pre}, w.\text{post}]$. These intervals:

1. Contain each other if $v$ is an ancestor/descendant of $w$ in the DFS tree.
2. Are disjoint if $v$ and $w$ are cousins in the DFS tree.
3. Never interleave $(v.\text{pre} < w.\text{pre} < v.\text{post} < w.\text{post})$
Proof

- Assume algorithm finds v before w \((v.\text{pre} < w.\text{pre})\)
- If algorithm discovers w after fully processing v:
  - \(v.\text{post} < w.\text{pre}\)
  - Intervals disjoint
  - v and w are cousins
- If algorithm discovers w before fully processing v:
  - Algorithm finishes processing w before it finishes v
  - \(v.\text{pre} < w.\text{pre} < w.\text{post} < v.\text{post}\)
  - Nested intervals
  - v is ancestor of w
Question: Possible Intervals

Which pairs of pre-post intervals are possible for DFS?

A) [1,2] & [3,4]
B) [1,3] & [2,4]
C) [1,4] & [2,3]
D) [1,5] & [2,4]
E) [1,6] & [2,5]
Directed Graphs

Often an edge makes sense both ways, but sometimes streets are one directional.

**Definition:** A directed graph is a graph where each edge has a direction. Goes *from* $v$ *to* $w$.

Draw edges with arrows to denote direction.
Question: Directed Graphs

Which of the following make the most sense as directed rather than undirected graphs:
A) The Internet (links connecting webpages)
B) The Internet (wires connecting servers)
C) Facebook (friendships connecting people)
D) Twitter (followings connecting people)
E) Maps (roads connecting intersections)
DFS on Directed Graphs

• Same code
• Only follow *directed* edges from v to w.
• Runtime still $O(|V|+|E|)$
• $\text{explore}(v)$ discovers all vertices reachable from v following only directed edges.
Directed Acyclic Graphs

- Directed graphs as dependencies
- Linear orderings
- DAGs definition
- Topological sort
Dependency Graphs

A directed graph can be thought of as a graph of dependencies. Where an edge $v \rightarrow w$ means that $v$ should come before $w$.

**Definition:** A topological ordering of a directed graph is an ordering of the vertices so that for each edge $(v, w)$, $v$ comes before $w$ in the ordering.
Question: Existence of Orderings

Does every directed graph have a topological ordering?

A) Yes

B) No
Counterexample