CSE 421 Algorithms
Richard Anderson
Winter 2009
Lecture 6

Announcements
• Monday, January 19 – Holiday
• Reading
  – 4.1 – 4.3, Important material

Announcement

Lecture Summary
Bipartite Graphs and Two Coloring
• Algorithm
  – Run BFS
  – Color odd layers red, even layers blue
  – If no edges between the same layer, the graph is bipartite
  – If edge between two vertices of the same layer, then there is an odd cycle, and the graph is not bipartite
• Theorem
  – A graph is bipartite if and only if it has no odd cycles

Graph Search
• Data structure for next vertex to visit determines search order

Breadth First Search
• All edges go between vertices on the same layer or adjacent layers

Depth First Search
• Each edge goes between vertices on the same branch
• No cross edges
Connected Components

• Undirected Graphs

Computing Connected Components in $O(n+m)$ time

• A search algorithm from a vertex $v$ can find all vertices in $v$’s component
• While there is an unvisited vertex $v$, search from $v$ to find a new component

Directed Graphs

• A Strongly Connected Component is a subset of the vertices with paths between every pair of vertices.

Identify the Strongly Connected Components

Strongly connected components can be found in $O(n+m)$ time

• But it’s tricky!
• Simpler problem: given a vertex $v$, compute the vertices in $v$’s scc in $O(n+m)$ time

Topological Sort

• Given a set of tasks with precedence constraints, find a linear order of the tasks
Find a topological order for the following graph

If a graph has a cycle, there is no topological sort

- Consider the first vertex on the cycle in the topological sort
- It must have an incoming edge

Lemma: If a graph is acyclic, it has a vertex with in degree 0

- Proof:
  - Pick a vertex \( v_1 \), if it has in-degree 0 then done
  - If not, let \( (v_2, v_1) \) be an edge, if \( v_2 \) has in-degree 0 then done
  - If not, let \( (v_3, v_2) \) be an edge . . .
  - If this process continues for more than \( n \) steps, we have a repeated vertex, so we have a cycle

Topological Sort Algorithm

While there exists a vertex \( v \) with in-degree 0
- Output vertex \( v \)
- Delete the vertex \( v \) and all out going edges

Details for \( O(n+m) \) implementation

- Maintain a list of vertices of in-degree 0
- Each vertex keeps track of its in-degree
- Update in-degrees and list when edges are removed
- \( m \) edge removals at \( O(1) \) cost each

Large Graphs

- Examples of large (real world graphs)
- What would you compute?
- What are the programming considerations?