Speeding up DFA

- Proper ordering of nodes of a flow graph speeds up the iterative algorithms: **depth-first ordering**.
- “Normal” flow graphs have a surprising property — **reducibility** — that simplifies several matters.
- Outcome: few iterations “normally” needed.

Acknowledgement

Rest of the slides based on the material at http://infolab.stanford.edu/~ullman/dragon/w06/w06.html
Depth-First Search

- Start at entry.
- If you can follow an edge to an unvisited node, do so.
- If not, backtrack to your *parent* (node from which you were visited).

Depth-First Spanning Tree (DFST)

- Root = *Entry*.
- Tree edges are the edges along which we first visit the node at the head.

DFST Example

1
2
3
4
5

Depth-First Node Order

- The reverse of the order in which a DFS *retreats* from the nodes.
- Alternatively, reverse of postorder traversal of the tree.
DF Order Example

Four Kind of Edges

1. Tree edges.
2. Forward edges: node to proper descendant.
3. Retreating edges: node to ancestor.
4. Cross edges: between two node, neither of which is an ancestor of the other.

A Little Magic

◮ Of these edges, only retreating edges go from high to low in DF order.
◮ Most surprising: all cross edges go right to left in the DFST.
  ▶ Assuming we add children of any node from the left.

Example: Non-Tree Edges
Roadmap

- “Normal” flow graphs are “reducible.”
- “Dominators” needed to explain reducibility.
- In reducible flow graphs, loops are well defined, retreating edges are unique (and called “back” edges).
- Leads to relationship between DF order and efficient iterative algorithm.

Example: Dominators

<table>
<thead>
<tr>
<th>Node</th>
<th>Dominators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1, 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>1, 4</td>
</tr>
<tr>
<td>5</td>
<td>1, 5</td>
</tr>
</tbody>
</table>

Common Dominator Cases

- The test of a while loop dominates all blocks in the loop body.
- The test of an if-then-else dominates all blocks in either branch.

Dominators

- Node $d$ dominates node $n$ if every path from the Entry to $n$ goes through $d$.
- Quick observations:
  - Every node dominates itself.
  - The entry dominates every node.
Back Edges

- An edge is a **back edge** if its head dominates its tail.
- **Theorem:** Every back edge is a retreating edge in every DFST of every flow graph.
  - Proof? Discuss/Exercise
  - Converse almost always true, but not always.

Example: Back Edges

<table>
<thead>
<tr>
<th>Node</th>
<th>Dominators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1, 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>1, 4</td>
</tr>
<tr>
<td>5</td>
<td>1, 5</td>
</tr>
</tbody>
</table>

Reducible Flow Graphs

- A flow graph is **reducible** if every retreating edge in any DFST for that flow graph is a back edge.
- **Testing reducibility:** Take any DFST for the flow graph, remove the back edges, and check that the result is acyclic.

Example: Remove Back Edges

Remaining graph is acyclic.
Why Reducibility?

- **Folk theorem**: All flow graphs in practice are reducible.
- **Fact**: If you use only while-loops, for-loops, repeat-loops, if-then(-else), break, and continue, then your flow graph is reducible.

Example: Nonreducible Graph

![Nonreducible Graph Diagram]

In any DFST, one of these edges will be a retreating edge.

Why Care About Back/Retreating Edges?

- Proper ordering of nodes during iterative algorithm assures number of passes limited by the number of “nested” back edges.
- Depth of nested loops upper-bounds the number of nested back edges.

DF Order and Retreating Edges

- Suppose that for a RD analysis, we visit nodes during each iteration in DF order.
- The fact that a definition $d$ reaches a block will propagate in one pass along any increasing sequence of blocks.
- When $d$ arrives along a retreating edge, it is too late to propagate $d$ from OUT to IN.
Example: DF Order

Node 2 generates definition $d$. Other nodes “empty” w.r.t. $d$. Does $d$ reach node 4?

Example: Depth = 2

Similarly . . .

Depth of a Flow Graph

- The **depth** of a flow graph is the greatest number of retreating edges along any acyclic path.
- For RD, if we use DF order to visit nodes, we converge in depth+2 passes.
  - Depth+1 passes to follow that number of increasing segments.
  - 1 more pass to realize we converged.
- AE also works in depth+2 passes.
- Unavailability propagates along retreat-free node sequences in one pass.
- So does LV if we use reverse of DF order.
  - A use propagates backward along paths that do not use a retreating edge in one pass.
In General . . .

◮ The depth+2 bound works for any monotone bit-vector framework, as long as information only needs to propagate along acyclic paths.
  ◮ Example: if a definition reaches a point, it does so along an acyclic path.

Why Depth+2 is Good?

◮ Normal control-flow constructs produce reducible flow graphs with the number of back edges at most the nesting depth of loops.
  ◮ Nesting depth tends to be small.

Example: Nested Loops

3 nested while loops.
depth = 3.

3 nested do-while loops.
depth = 1.

Natural Loops

◮ The natural loop of a back edge $a \rightarrow b$ is $\{b\}$ plus the set of nodes that can reach $a$ without going through $b$.
◮ Theorem: two natural loops are either disjoint, identical, or nested.
◮ Proof: Discuss/Exercise
Example: Natural Loops

1
4
5
2
3
1
4
5
2
3

Natural loop 3 → 2
Natural loop 5 → 1

Reading Assignment

- New Dragon Book (Aho, Lam, Sethi, Ullman)
- Chapter 9