CSE P 501 – Compilers

Loops

Hal Perkins

Autumn 2011

Agenda

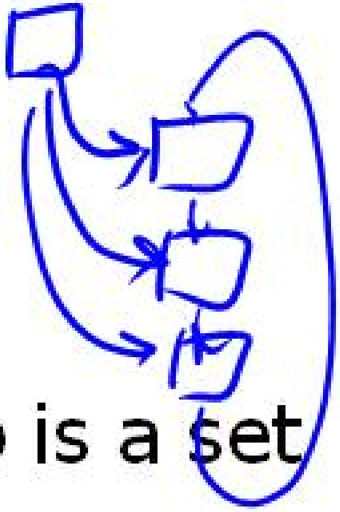
- Loop optimizations
  - Dominators – discovering loops
  - Loop invariant calculations
  - Loop transformations
- A quick look at some memory hierarchy issues

- Largely based on material in Appel ch. 18, 21; similar material in other books

Loops

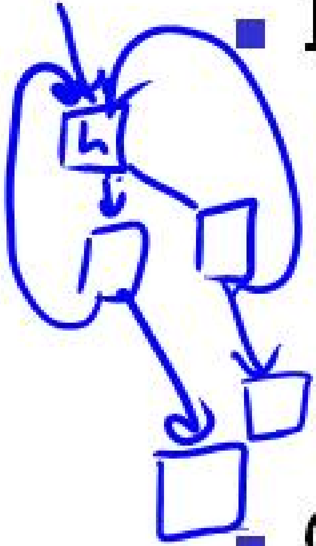
- Much of he execution time of programs is spent here
- ∴ worth considerable effort to make loops go faster
- ∴ want to figure out how to recognize loops and figure out how to “improve” them

What’s a Loop?



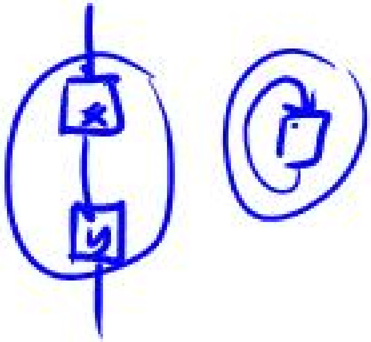
- In a control flow graph, a loop is a set of nodes S such that:
  - S includes a *header node* h
  - From any node in S there is a path of directed edges leading to h
  - There is a path from h to any node in S
  - There is no edge from any node outside S to any node in S other than h

Entries and Exits



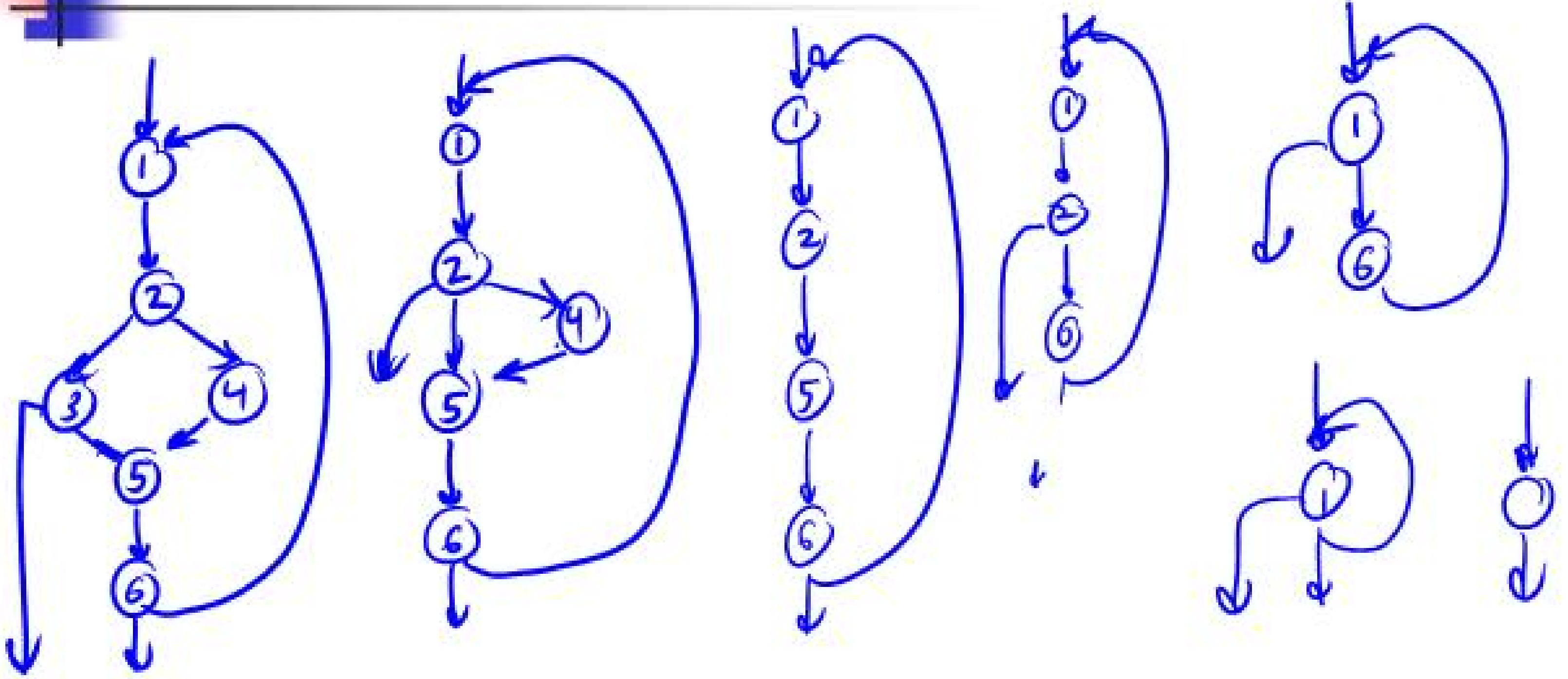
- In a loop
  - An *entry node* is one with some predecessor outside the loop
  - An *exit node* is one that has a successor outside the loop
- Corollary of preceding definitions: A loop may have multiple exit nodes, but only one entry node

Reducible Flow Graphs

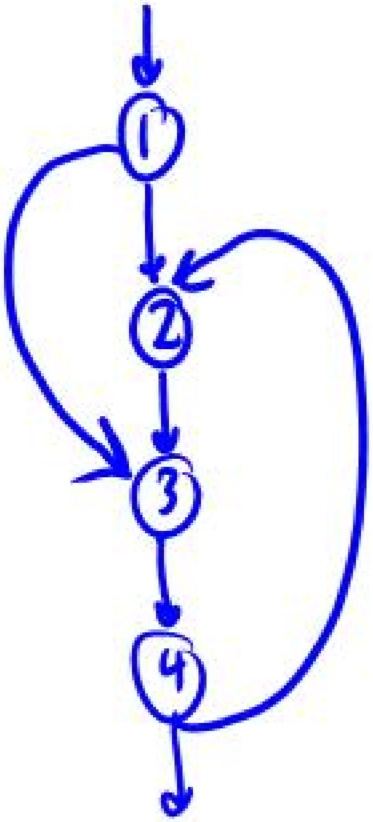


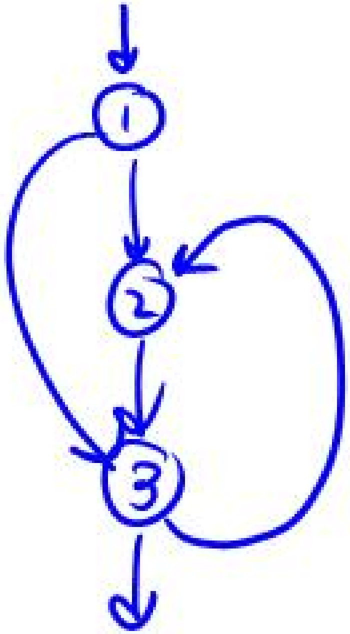
- In a reducible flow graph, any two loops are either nested or disjoint
- Roughly, to discover if a flow graph is reducible, repeatedly delete edges and collapse together pairs of nodes (x,y) where x is the only predecessor of y
- If the graph can be reduced to a single node it is reducible
    - Caution: this is the "powerpoint" version of the definition – see a good compiler book for the careful details

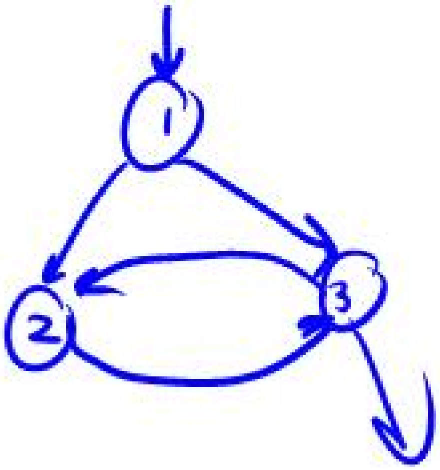
Example: Is this Reducible?



Example: Is this Reducible?







Reducible Flow Graphs in Practice

- Common control-flow constructs yield reducible flow graphs
  - if-then[-else], while, do, for, break(!)
- A C function without goto will always be reducible
- Many dataflow analysis algorithms are very efficient on reducible graphs, but…
- We don't need to assume reducible control-flow graphs to handle loops

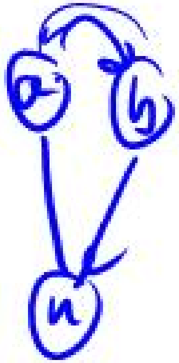
Finding Loops in Flow Graphs

- We use *dominators* for this
- Recall
  - Every control flow graph has a unique start node s0
  - Node x dominates node y if every path from s0 to y must go through x
  - A node x dominates itself

Calculating Dominator Sets

- D[n] is the set of nodes that dominate n
  - $D[\underline{s0}] = \{ \underline{s0} \}$
  - $D[\underline{n}] = \{ \underline{n} \} \cup (\cap_{p \in pred[n]} D\underline{[p]})$
- Set up an iterative analysis as usual to solve this
  - Except initially each D[n] must be all nodes in the graph – updates make these sets smaller if changed

Immediate Dominators

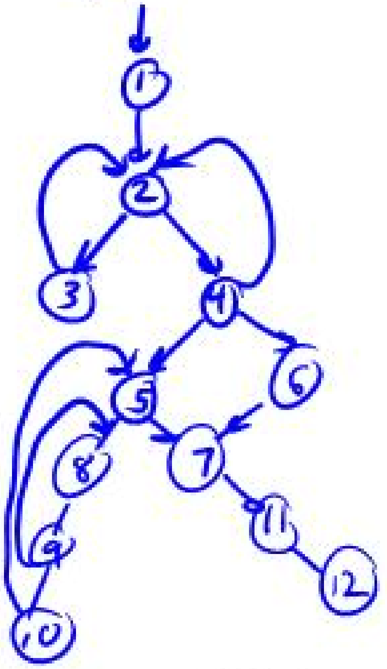


- Every node n has a single *immediate dominator* idom(n)
  - idom(n) differs from n
  - idom(n) dominates n
  - idom(n) does not dominate any other dominator of n
- Fact (er, theorem): If a dominates n and b dominates n, then either a dominates b or b dominates a
  - ∴ idom(n) is unique

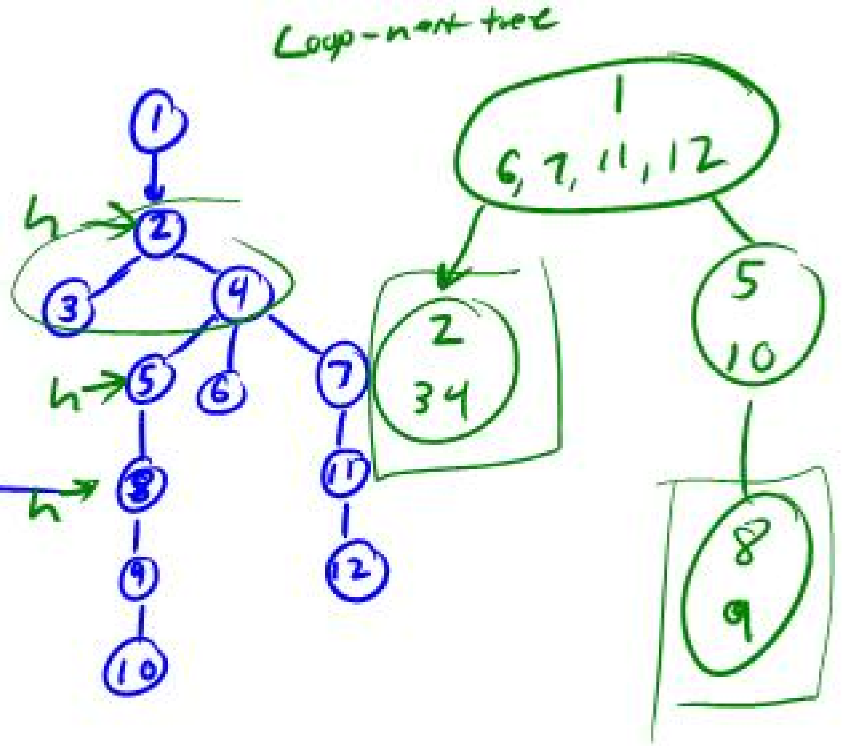
Dominator Tree

- A *dominator tree* is constructed from a flowgraph by drawing an edge form every node in n to idom(n)
    - This will be a tree. Why?

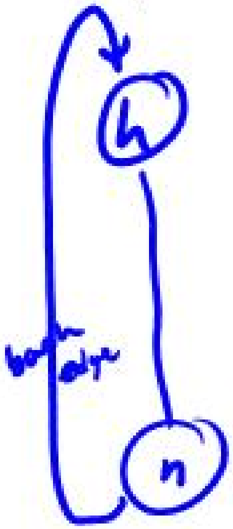
Example



node	dom	idom
1	1	—
2	1,2	1
3	1,2,3	2
4	1,2,4	2
5	1,2,4,5	4
6	1,2,4,6	4
7	1,2,4,7	4
8	1,2,4,5,8	5
9	1,2,4,5,8,9	8
10	1,2,4,5,8,9,10	9
11	1,2,4,7,11	7
12	1,2,4,7,11,12	11

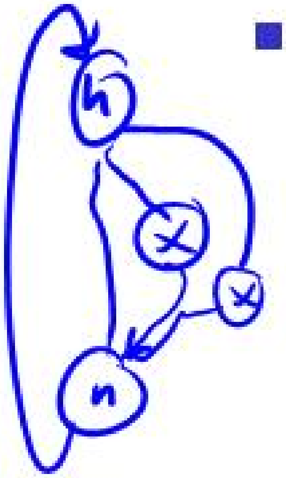


Back Edges & Loops



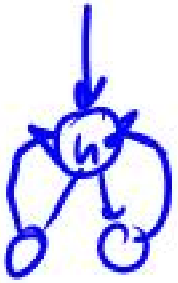
- A flow graph edge from a node n to a node h that dominates n is a *back edge*
- For every back edge there is a corresponding subgraph of the flow graph that is a loop

Natural Loops



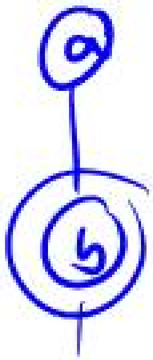
- If h dominates n and n->h is a back edge, then the *natural loop* of that back edge is the set of nodes x such that
  - h dominates x
  - There is a path from x to n not containing h
- h is the *header* of this loop
- Standard loop optimizations can cope with loops whether they are natural or not

Inner Loops



- Inner loops are more important for optimization because most execution time is expected to be spent there
- If two loops share a header, it is hard to tell which one is "inner"
  - Common way to handle this is to merge natural loops with the same header

Inner (nested) loops



- Suppose
  - A and B are loops with headers a and b
  - a ≠ b
  - b is in A
- Then
  - The nodes of B are a proper subset of A
  - B is nested in A, or B is the *inner loop*

Loop-Nest Tree

- Given a flow graph G
  1. ✓ Compute the dominators of G
  2. ✓ Construct the dominator tree
  3. Find the natural loops (thus all loop-header nodes)
  4. For each loop header h, merge all natural loops of h into a single loop: loop[h]
  5. Construct a tree of loop headers s.t. h1 is above h2 if h2 is in loop[h1]