Program Analysis

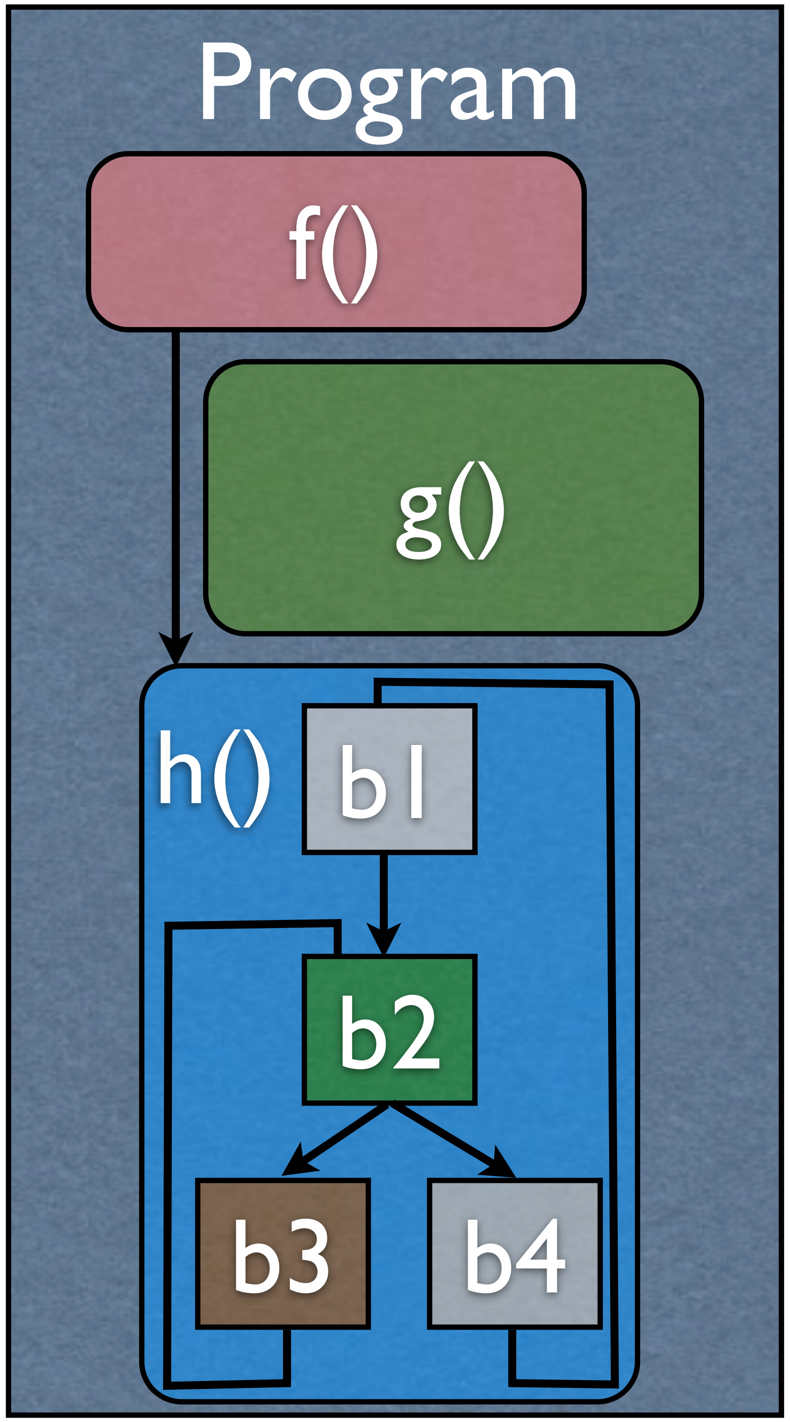
Lecture 10: *Control Flow Analysis II*
Winter term 2011/2012

Prof. Thorsten Holz

Announcements

- Next week (December 22) we will have a lecture, but it is not relevant for the examination
  - Vulnerabilities on document formats
  - Heap spraying
  - We will have a “christmas challenge” with bonus points
- Second part of the exercise after this lecture

Building Blocks



- Program consists of several *procedures*
    - f() calls h()
- Produce consists of several *basic blocks*
    - “Atomic” units of a program

Basic Blocks

- *Basic block* is a sequence of instructions which will always be executed in the given order
- Properties:
  - Basic block has a single entry and single exit
  - Flow of control can only enter at the beginning and leave at the end
  - Only last instruction can be a branch instruction; only first instruction can be target of a branch

Finding BBs

- Identify *leader instructions*, i.e., first instruction of BB:

  1. First instruction of a program is a leader
  2. Any instruction that is the target of a branch is a leader
  3. Any instruction that immediately follows a branch or return instruction is a leader

Example: Finding Leaders

High-Level Language

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

Assembler

```
00401139   mov    [ebp-4], 0
00401140   jmp    loc 40114B
00401142   mov    eax, [ebp-4]
00401145   add    eax, 1
00401148   mov    [ebp-4], eax
0040114B   cmp    [ebp-4], 0Ah
0040114F   jge    loc 401172
00401151   mov    [ebp-8], 0
00401158   call   functionXYZ
0040115D   mov    eax, [ebp-8]
00401160   mov    ecx, [ebp-4]
00401163   mov    edx, [ebp-8]
00401166   add    edx, 1
00401169   mov    [ebp-8], edx
0040116C   cmp    eax, ecx
0040116E   jl     loc 401158
00401170   jmp    loc 401142
00401172   ...
```

Rule 1

High-Level Language	Assembler

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 1: First instruction of a program is a leader

Rule 2

High-Level Language	Assembler

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 2: Any instruction that is the target of a branch is a leader

Rule 2

High-Level Language

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

Assembler

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 2: Any instruction that is the target of a branch is a leader

Rule 2

Systems Security
Ruhr-University Bochum

High-Level Language	Assembler

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 2: Any instruction that is the target of a branch is a leader

Rule 2

High-Level Language

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

Assembler

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 2: Any instruction that is the target of a branch is a leader

Rule 3

High-Level Language

```
int i, j;
for (i = 0; i < 10; i++)
{
  j=0;

  do
  {
    functionXYZ ( ) ;
  } while (j++< i);
}
```

Assembler

```
00401139    mov    [ebp-4], 0
00401140    jmp    loc 40114B
00401142    mov    eax, [ebp-4]
00401145    add    eax, 1
00401148    mov    [ebp-4], eax
0040114B    cmp    [ebp-4], 0Ah
0040114F    jge    loc 401172
00401151    mov    [ebp-8], 0
00401158    call   functionXYZ
0040115D    mov    eax, [ebp-8]
00401160    mov    ecx, [ebp-4]
00401163    mov    edx, [ebp-8]
00401166    add    edx, 1
00401169    mov    [ebp-8], edx
0040116C    cmp    eax, ecx
0040116E    jl     loc 401158
00401170    jmp    loc 401142
00401172    ...
```

Rule 3: Any instruction that immediately follows a branch or return instruction is a leader

CFGs

- A *control flow graph* (*CFG*) is a graph representation of the different paths that a program might traverse
  - CFG is directed multigraph
  - Nodes are basic blocks
  - Edges represent flow of control (either branches or fall-through execution)
- No information about data is available in CFG
  - An edge means that a program *may* take the path

Dominators

- A node **a** in a CFG *dominates* a node **b** if every path from the start node to node **b** goes through **a**
- We say that node **a** is a *dominator* of node **b**
- The *dominator set* of node **b**, *dom(b)*, is formed by all nodes that dominate **b**
- By definition, each node dominates itself, therefore, **b** $\in$ dom(**b**)

Domination Relation

- *Definition*: Let G = (N, E, s) denote a flowgraph with set of vertices N, set of edges E, starting node **s**, and let **a** $\in$ N, **b** $\in$ N

1. **a** *dominates* **b**, written **a** $\leq$ **b**, if every path from **s** to **b** contains **a**
2. **a** *properly dominates* **b**, written **a** < **b**, if **a** $\leq$ **b** and **a** $\neq$ **b**

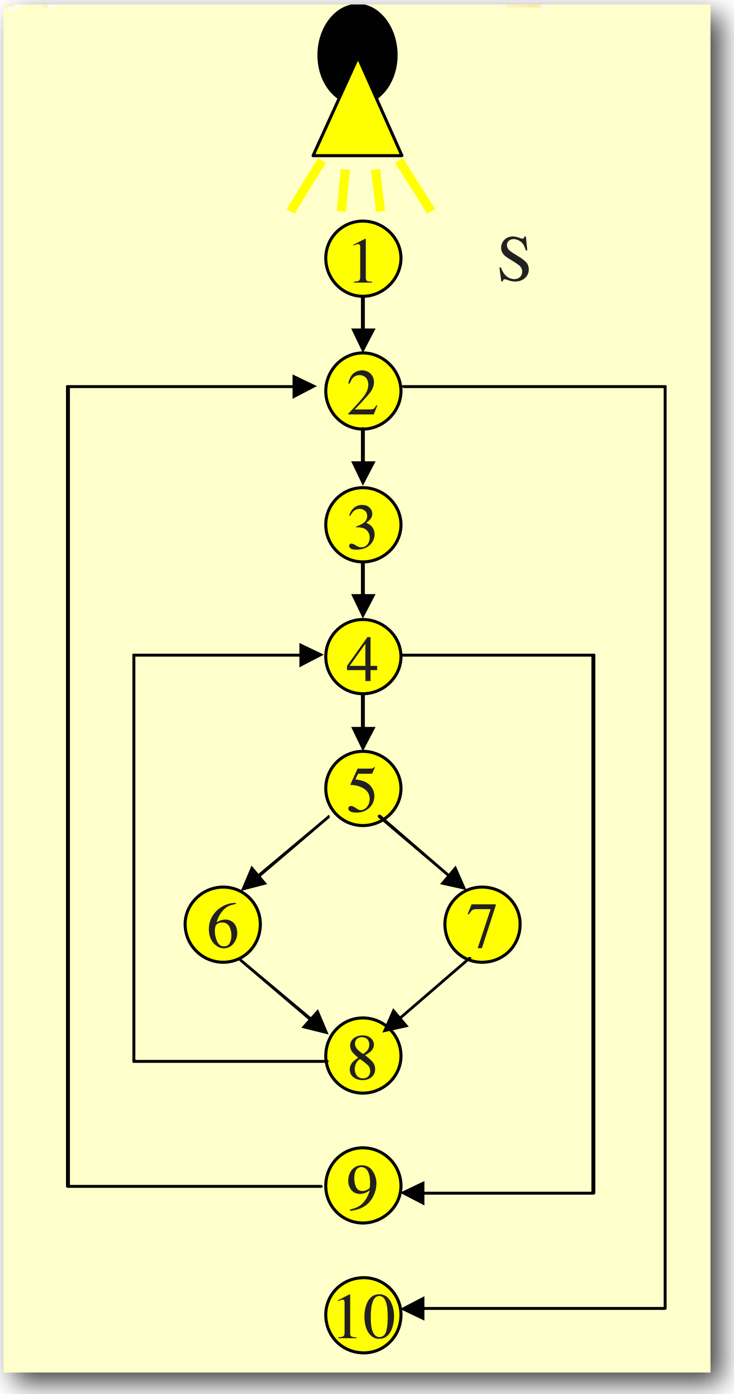
Domination Relation

- *Definition*: Let G = (N, E, s) denote a flowgraph with set of vertices N, set of edges E, starting node **s**, and let **a** $\in$ N, **b** $\in$ N

3. **a** *directly (immediately) dominates* **b**, written **a** $<_d$ **b** if:
   - **a** < **b** and
   - there is no **c** $\in$ N such that **a** < **c** < **b**

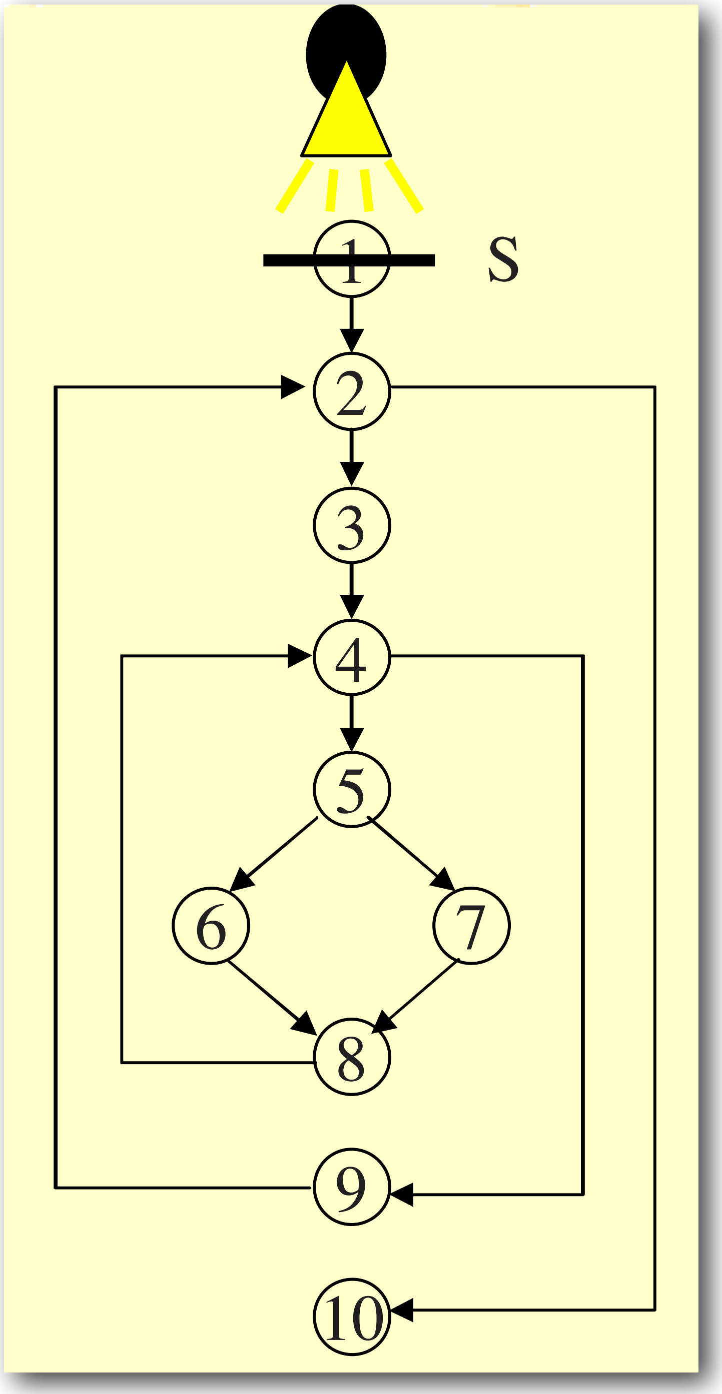
Dominator Intuition

- Imagine a source of light at the start node, and that the edges are optical fibers
- To find which nodes are dominated by a given node **a**, place an opaque barrier at **a** and observe which nodes became dark



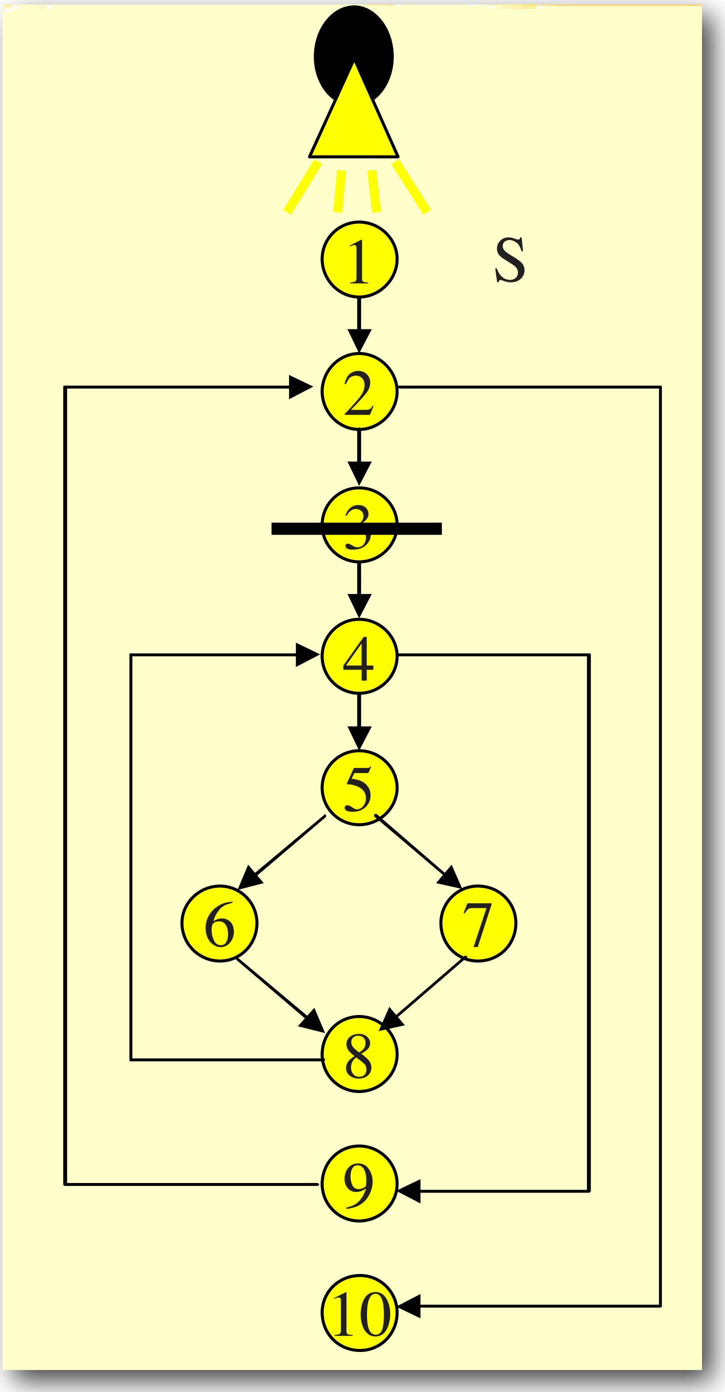
Dominator Intuition

- The start node dominates all nodes in the flowgraph.



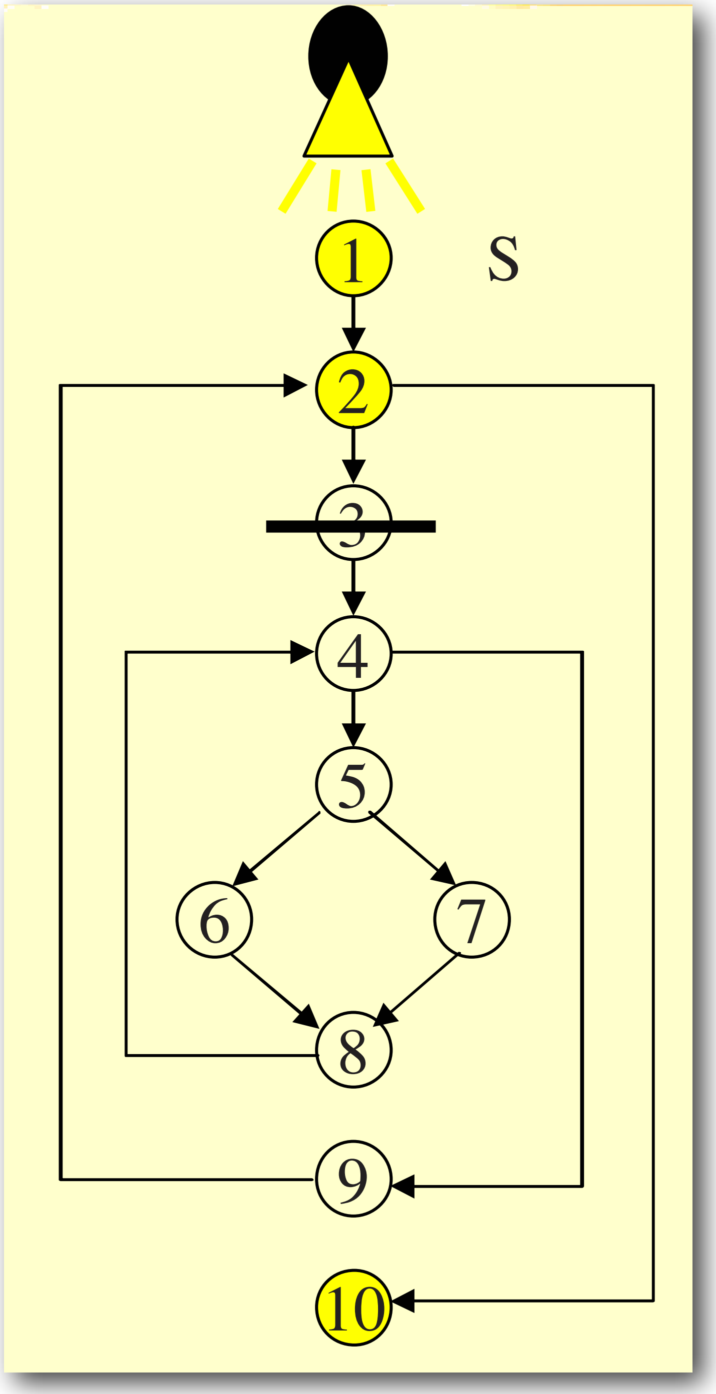
Dominator Intuition

- Which nodes are dominated by node 3?



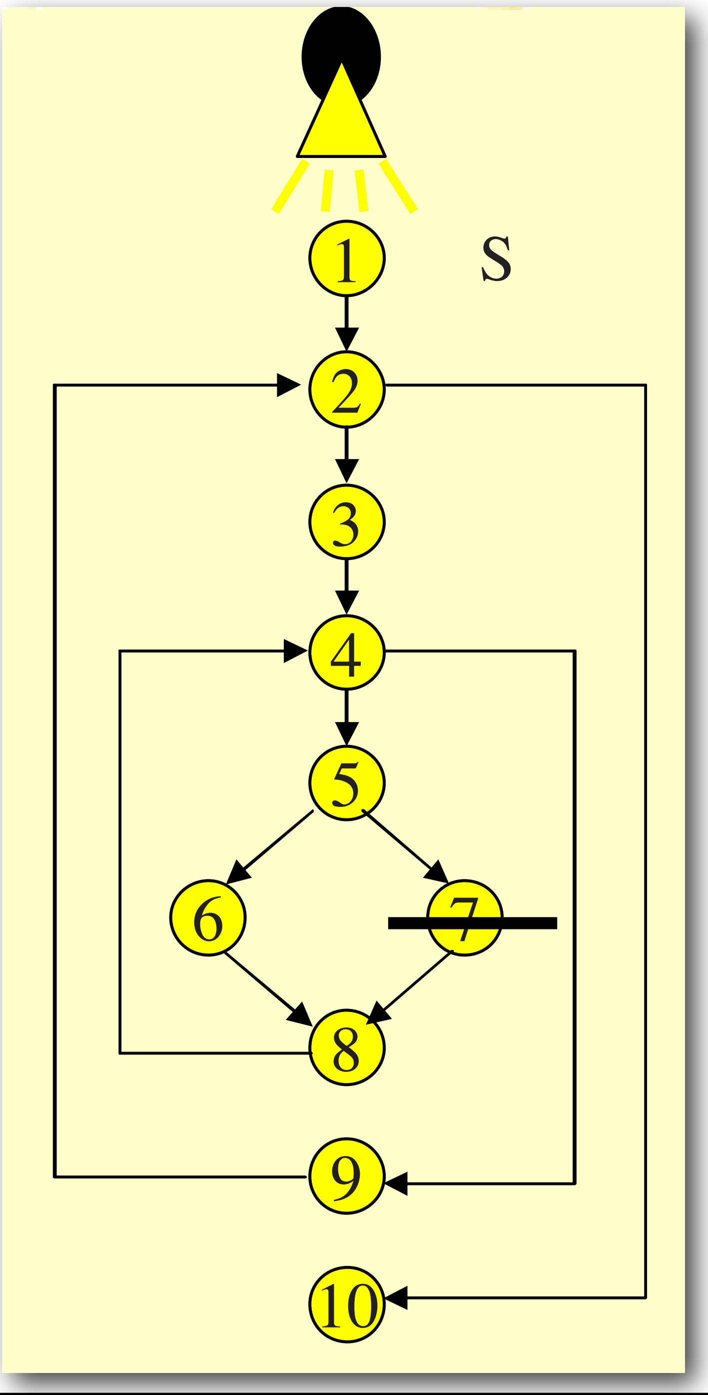
Dominator Intuition

- Which nodes are dominated by node 3?
- Node 3 dominates nodes 3, 4, 5, 6, 7, 8, and 9



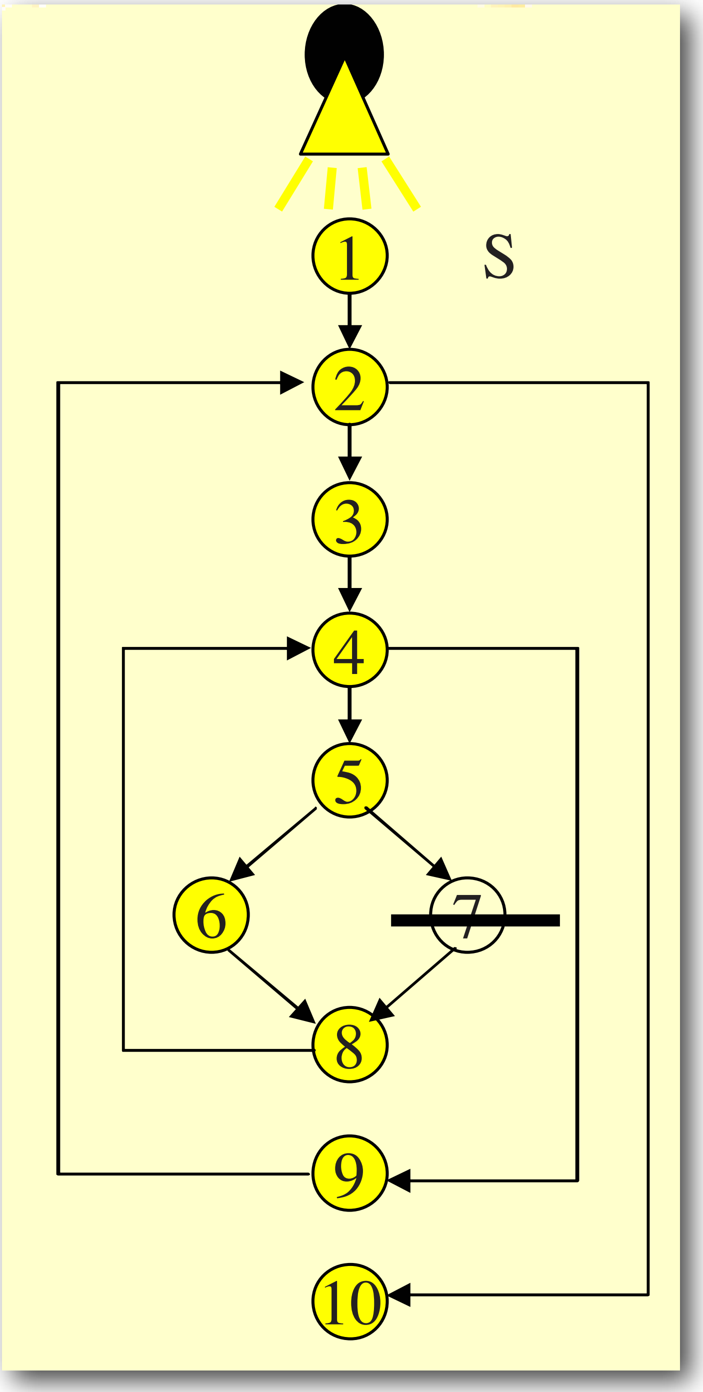
Dominator Intuition

- Which nodes are dominated by node 7?



Dominator Intuition

- Which nodes are dominated by node 7?
- Node 7 only dominates itself



Loop Detection

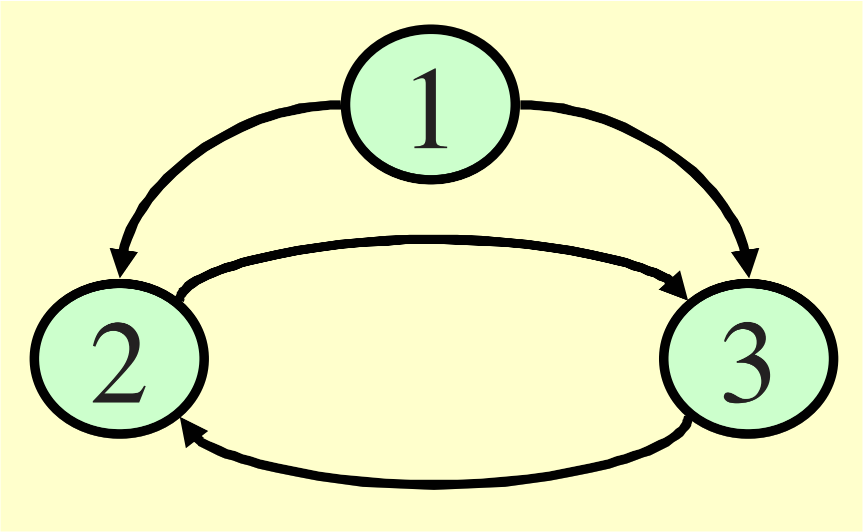
- Loops are defined as repeated execution of the same instructions, typically with different data
- How do we identify loops in a flow graph?
- The goal is to create an uniform treatment for program loops written using different loop structures (e.g. while, for) and loops constructed out of `goto`
- *Basic idea*: Use a general approach based on analyzing graph-theoretical properties of the CFG

Definitions

- A *strongly-connected component* (SCC) of a flowgraph G = (N, E, s) is a subgraph G' = (N', E', s') in which there is a path from each node in N' to every node in N'
- A strongly-connected component G' = (N', E', s') of a flowgraph G = (N, E, s) is a loop with entry s' *if s' dominates all nodes in N'*

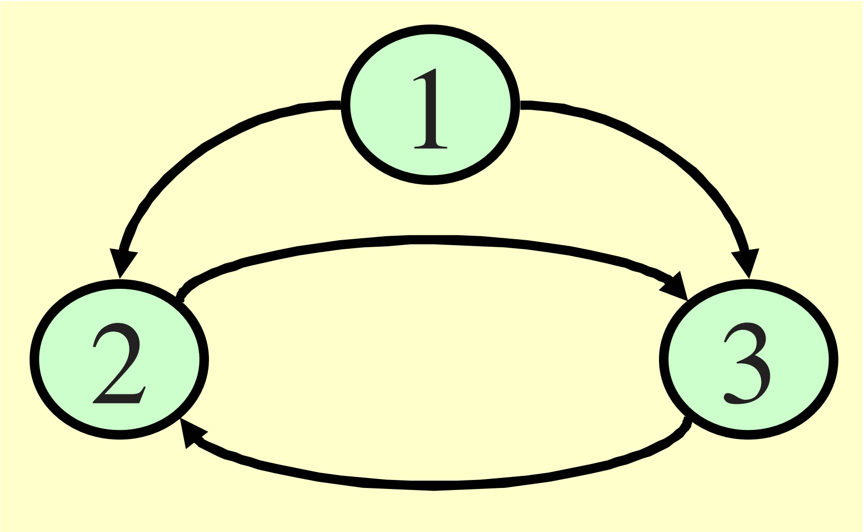
Example

In the flow graph below, do nodes 2 and 3 form a loop?



Example

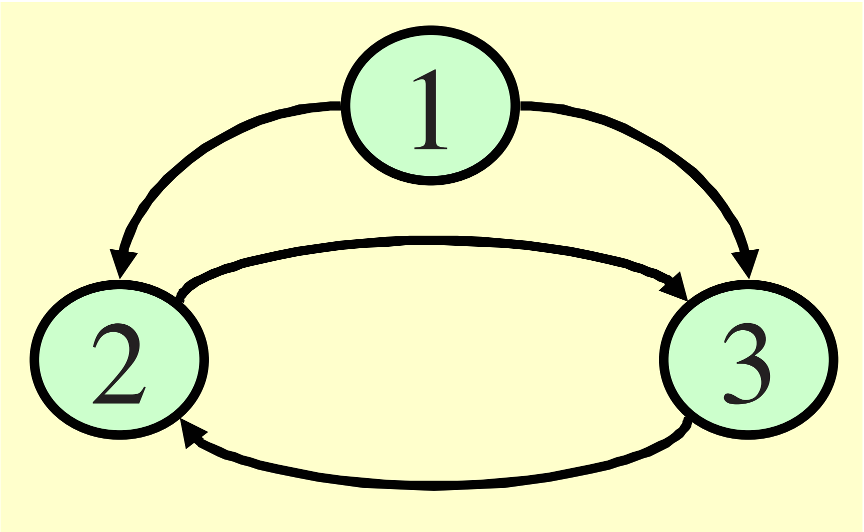
In the flow graph below, do nodes 2 and 3 form a loop?



- Nodes 2 and 3 form a strongly connected component, but they are not a loop. Why?

Example

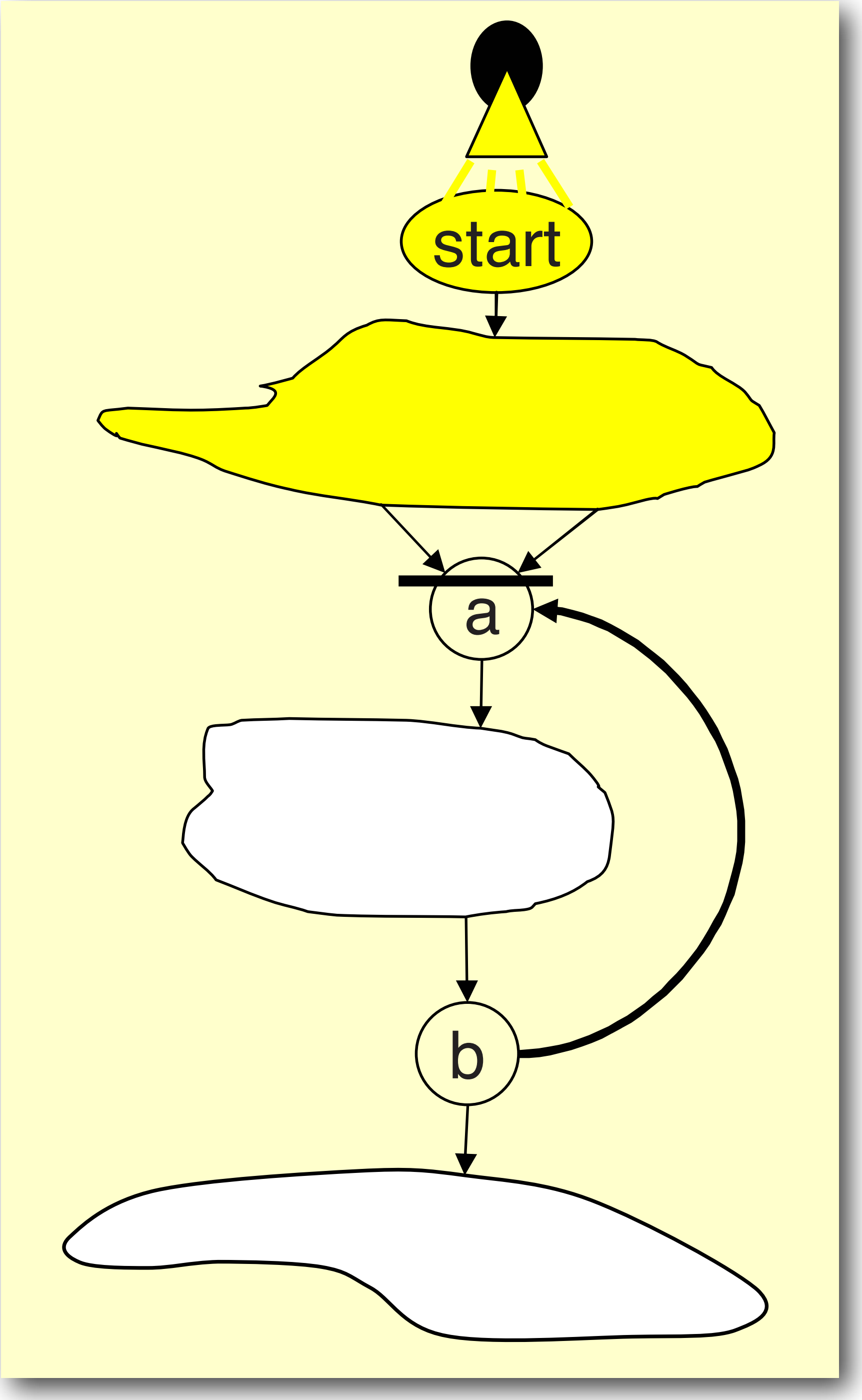
In the flow graph below, do nodes 2 and 3 form a loop?



- Nodes 2 and 3 form a strongly connected component, but they are not a loop. Why?
- No node in the subgraph dominates all the other nodes, therefore this subgraph is not a loop

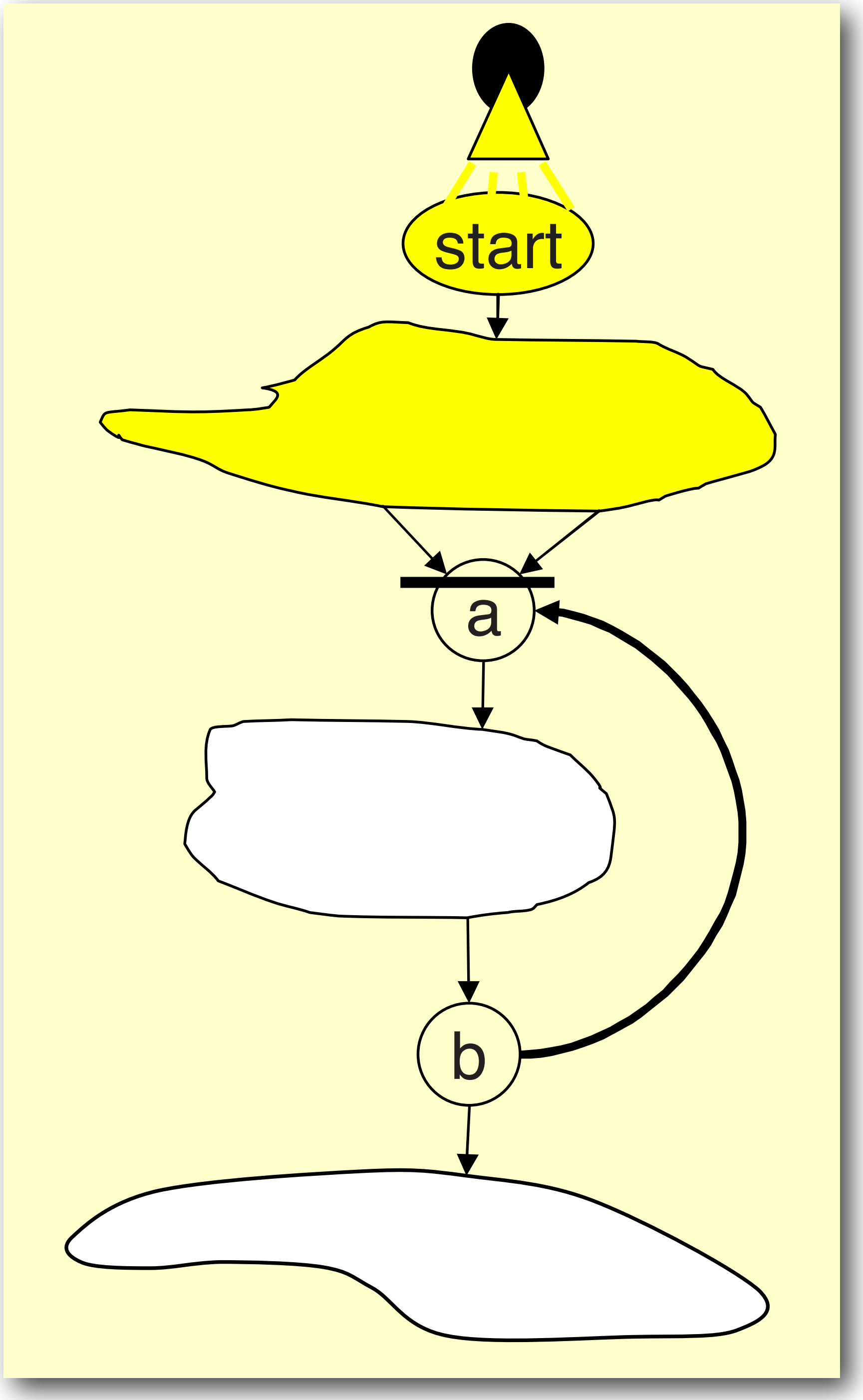
How to Find Loops?

- Look for "back edges"
- An edge (**b**, **a**) of a flowgraph G is a *back edge* if **a** dominates **b**, **a** < **b**



Natural Loops

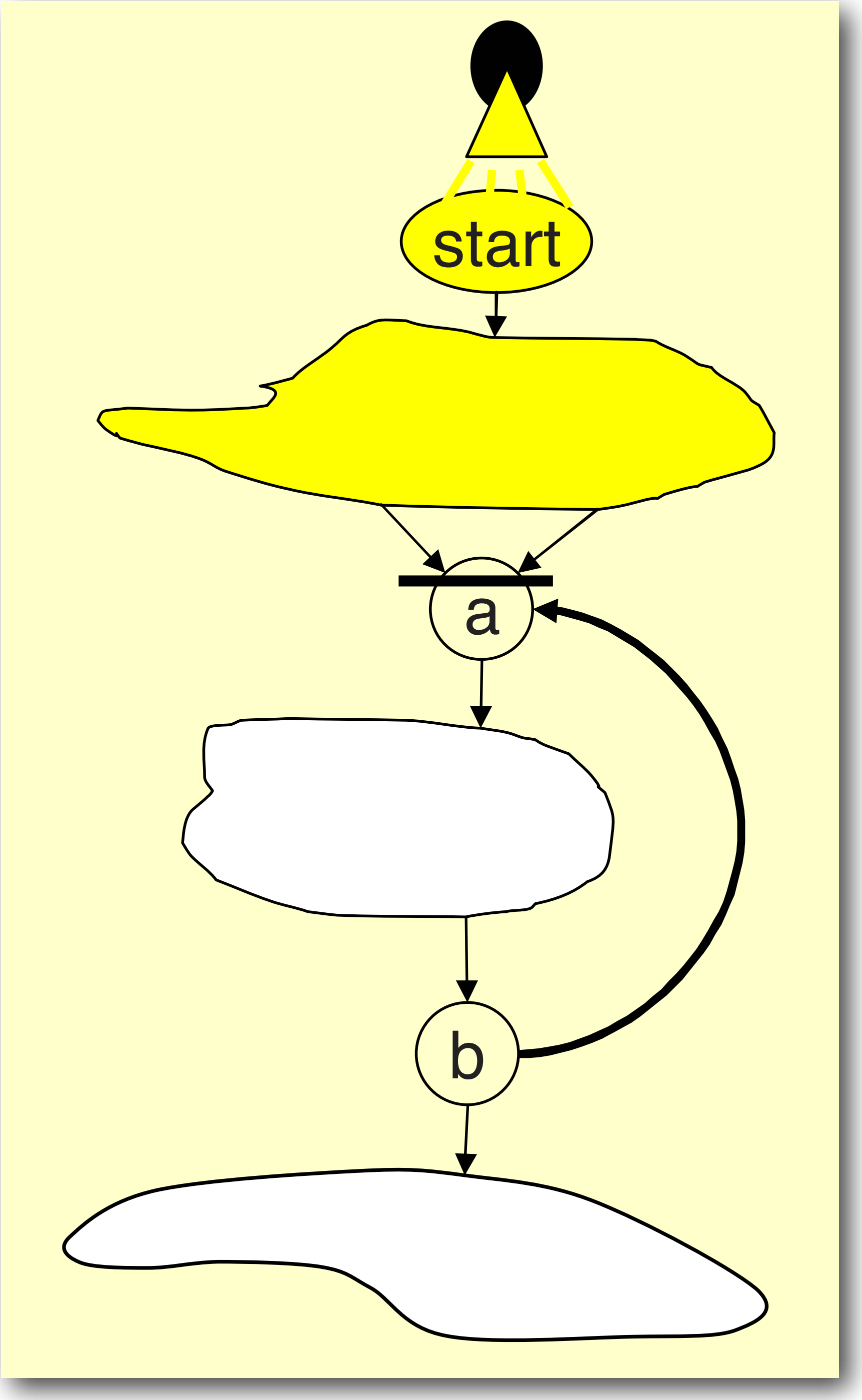
Given a back edge (**b**, **a**), a *natural loop* associated with (**b**, **a**) with entry in node **a** is the subgraph formed by **a** plus all nodes that can reach **b** without going through **a**



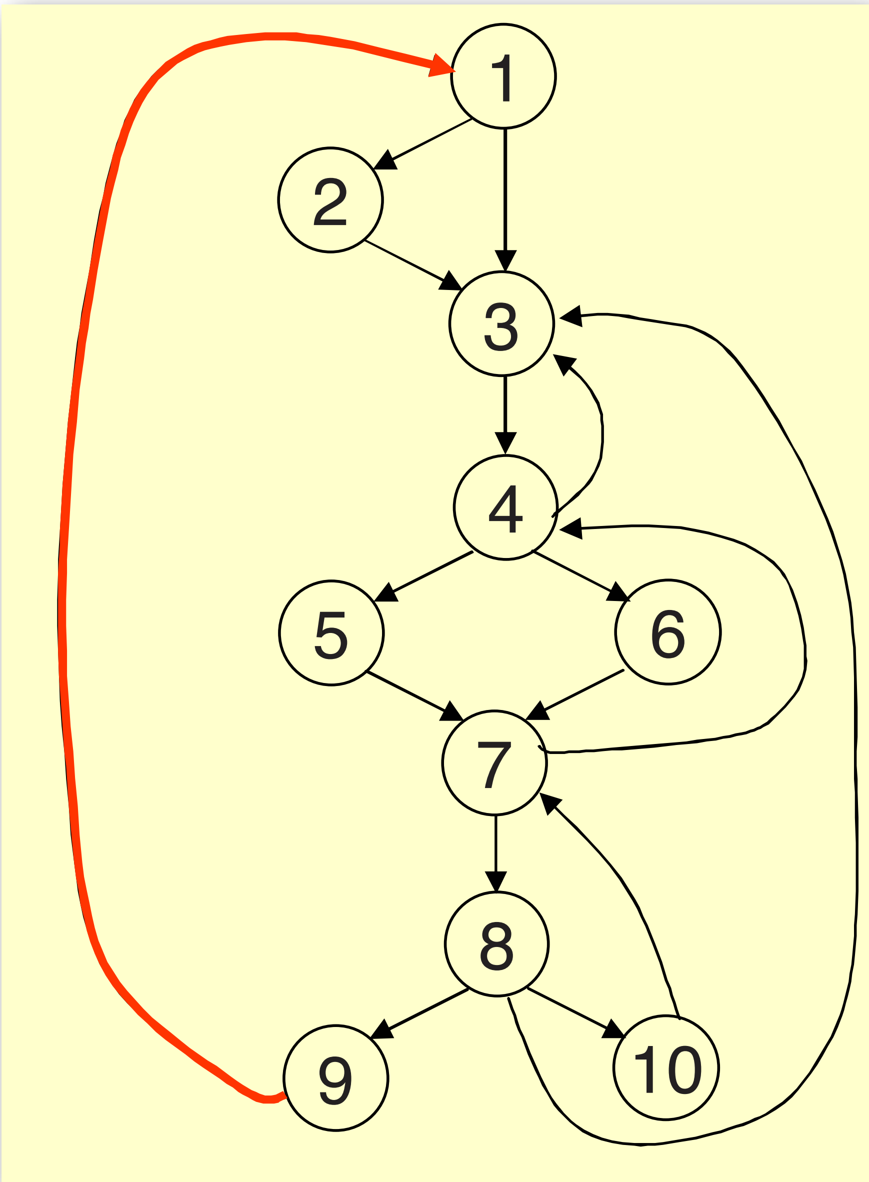
Natural Loops

One way to find natural loops is:

1. find a back edge (**b**, **a**)
2. find the nodes that are dominated by **a**
3. look for nodes that can reach **b** among the nodes dominated by **a**



Example

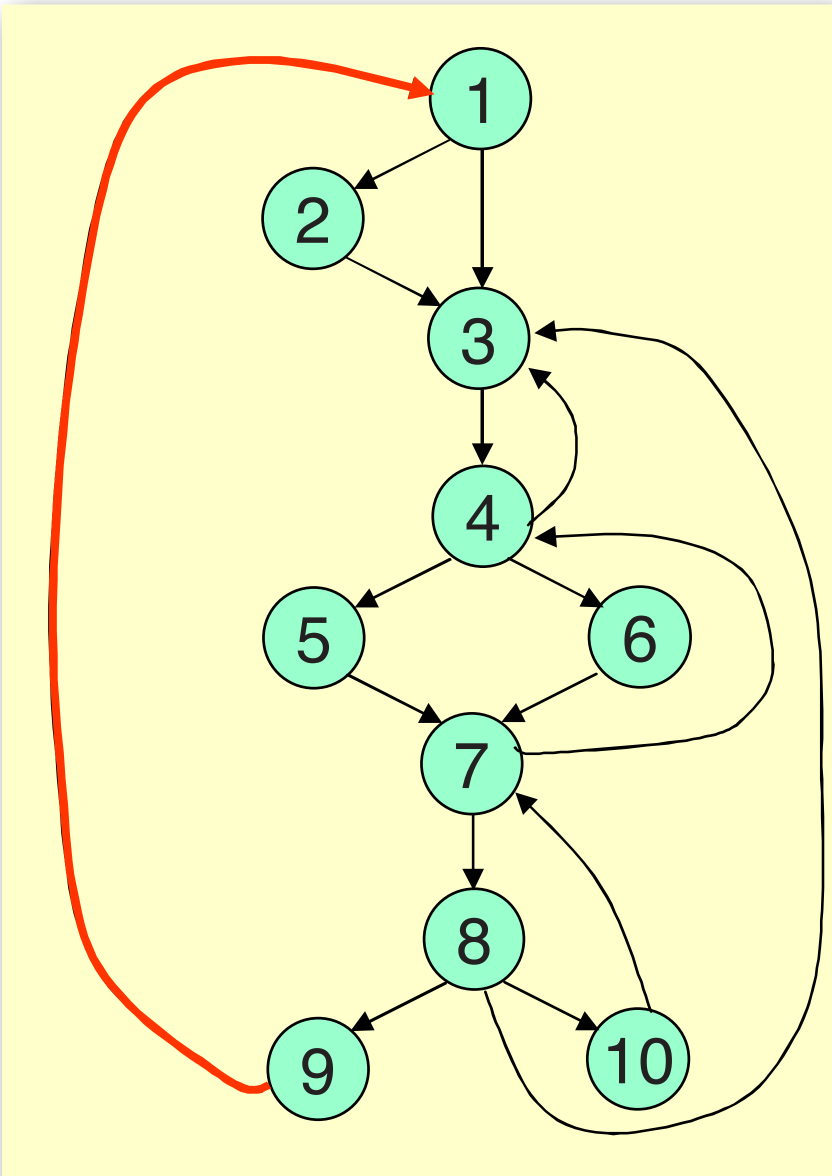


Find all back edges in this graph and the natural loop associated with each back edge

(9, 1)

(1) find a back edge (**b**, **a**)
(2) find the nodes that are dominated by **a**
(3) look for nodes that can reach **b** among the nodes dominated by **a**

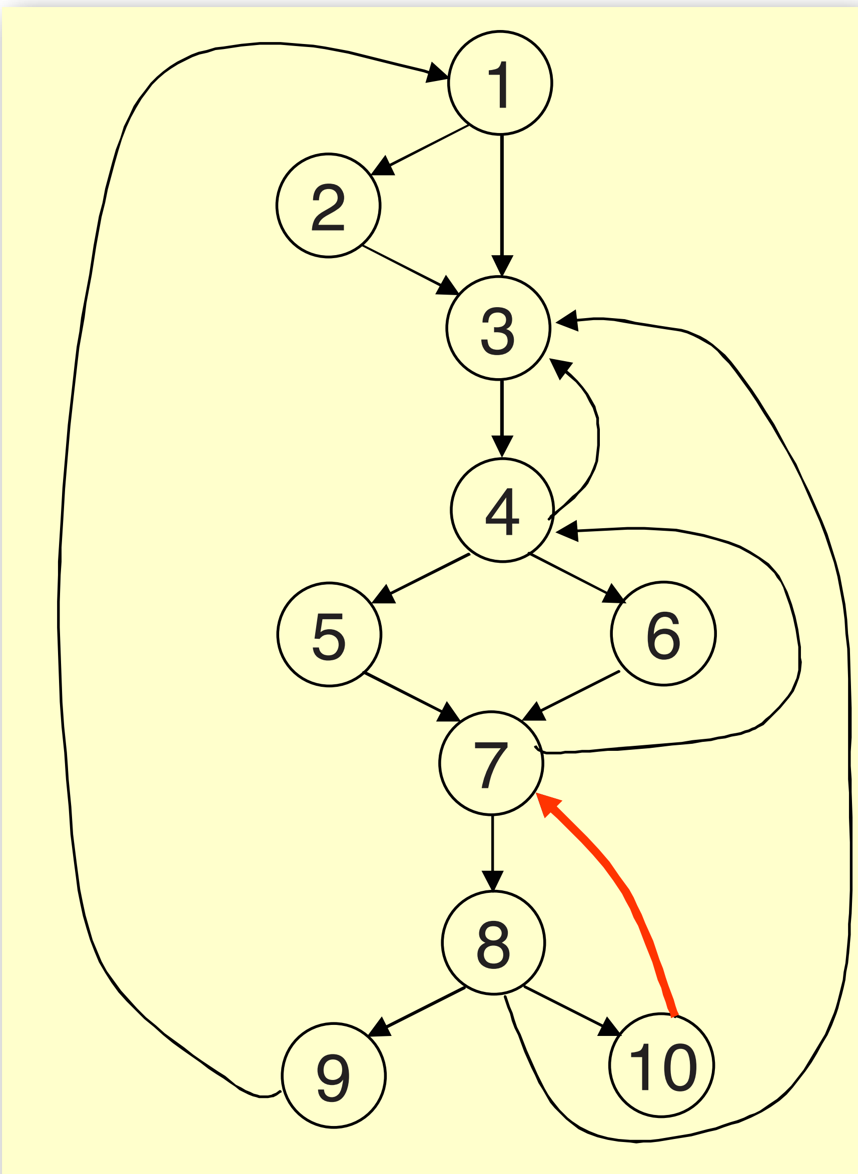
Example



Find all back edges in this graph and the natural loop associated with each back edge

(9, 1) Entire graph

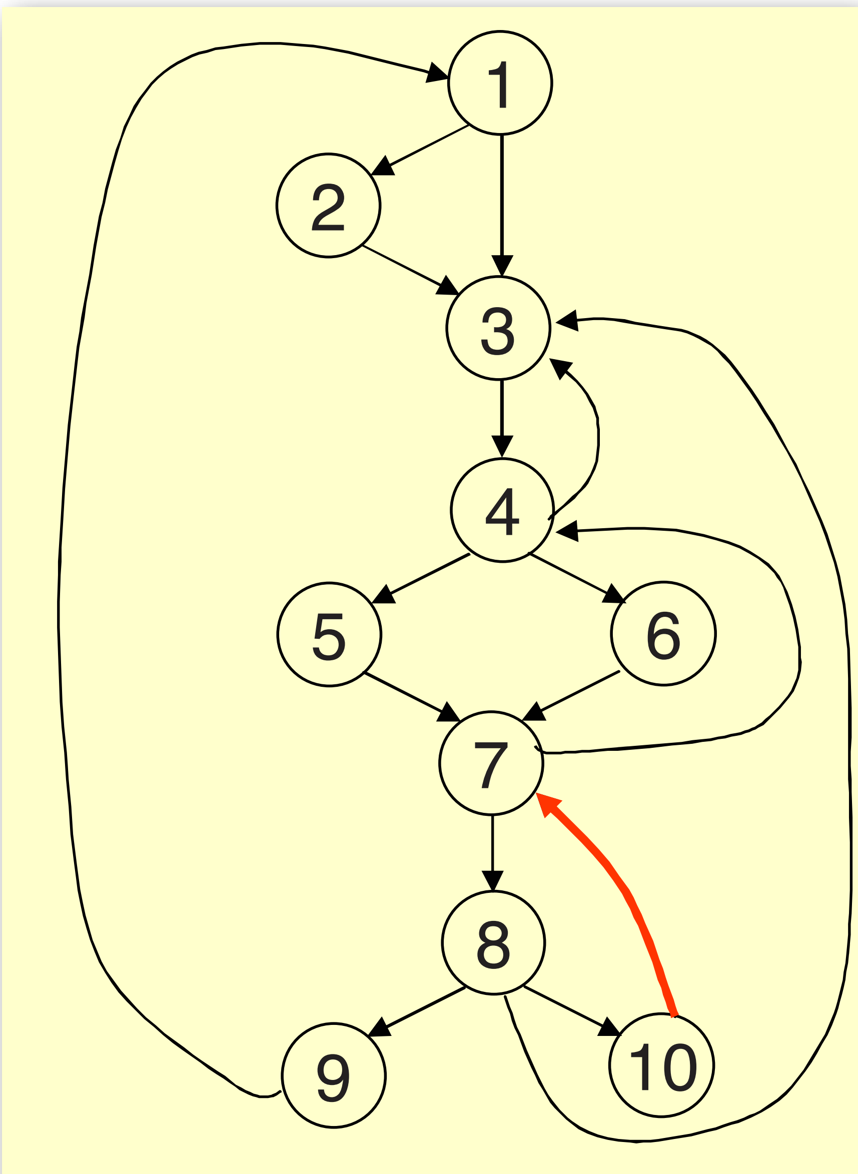
Example



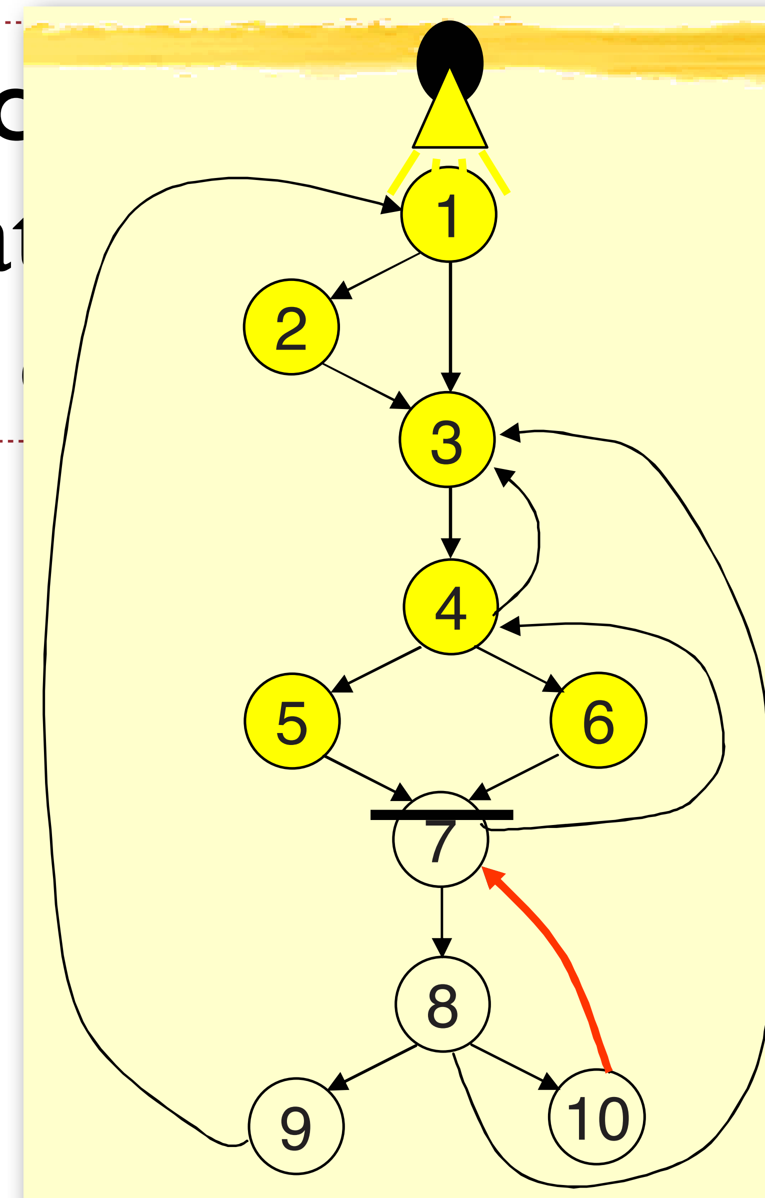
Find all back edges in this graph and the natural loop associated with each back edge

(9, 1)	Entire graph
(10, 7)	

Example

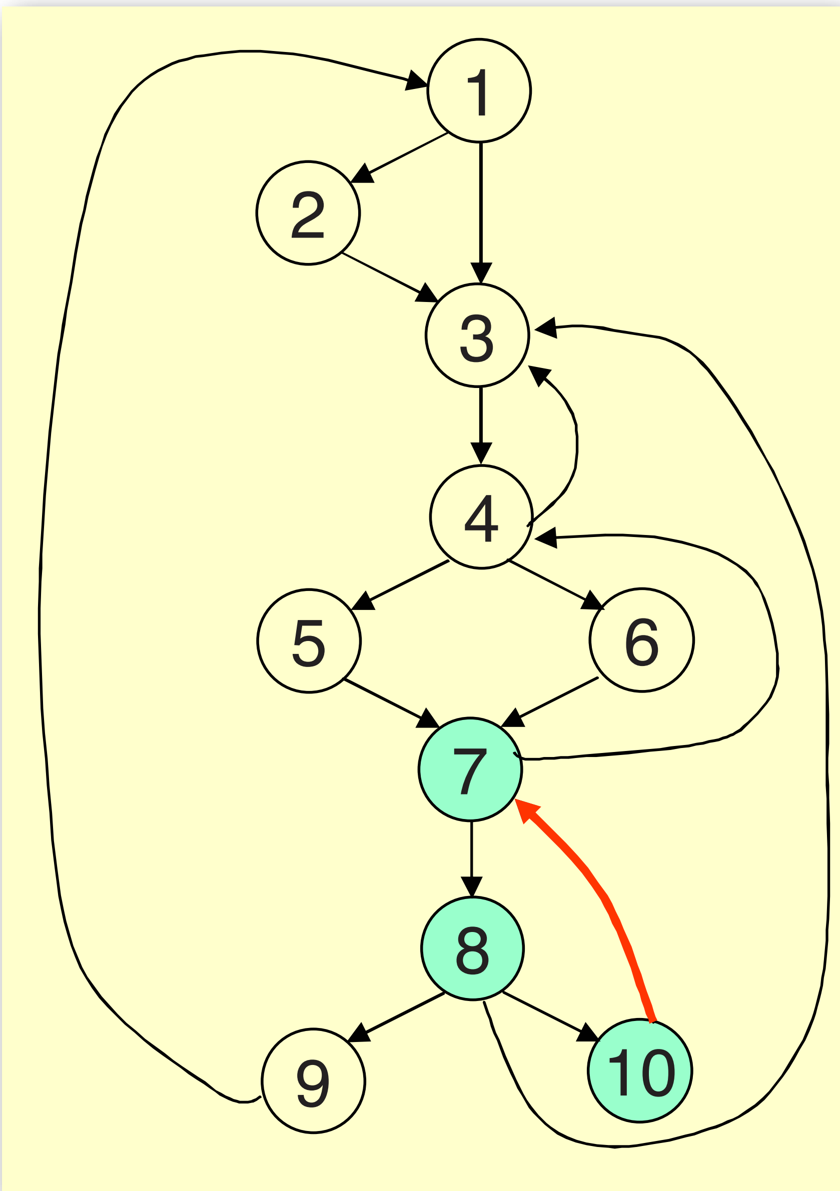


Find all bac… raph and the nat… ated with …



(9, 1)
(10, 7)

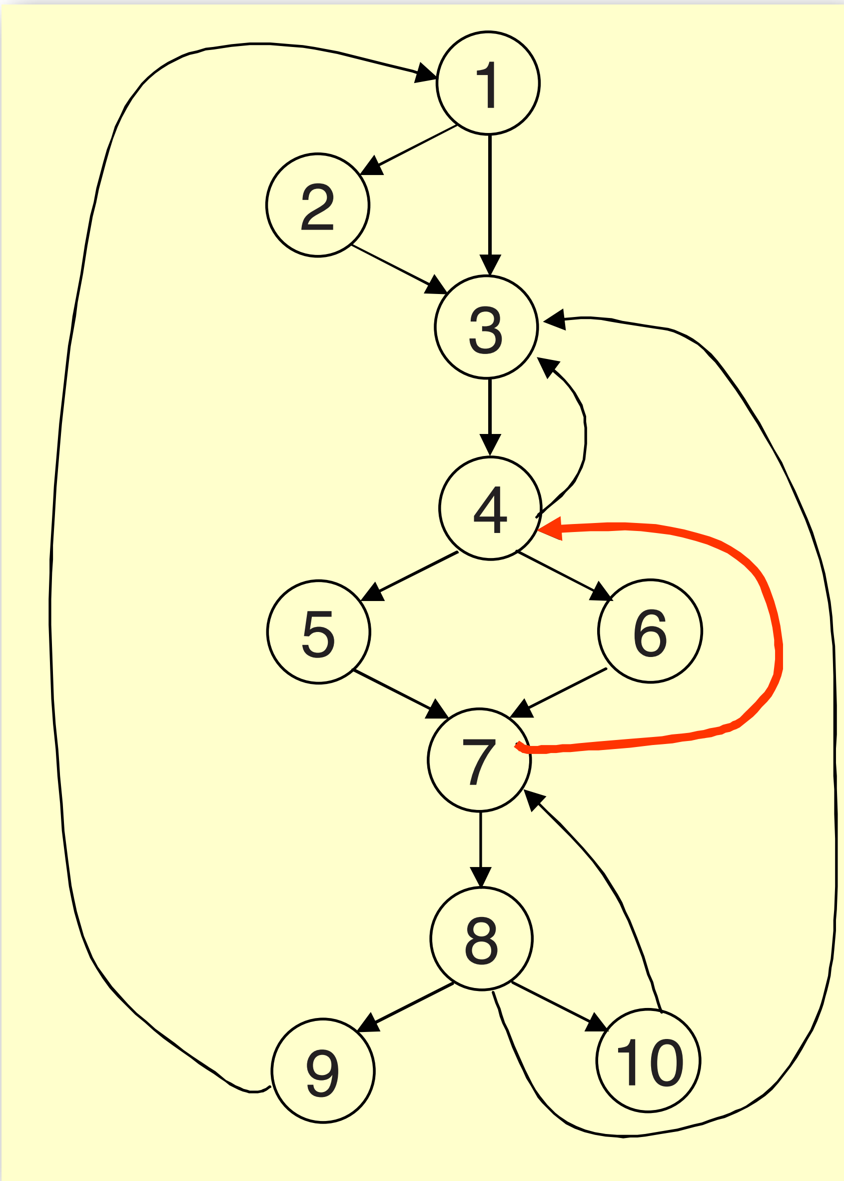
Example



Find all back edges in this graph and the natural loop associated with each back edge

(9, 1)	Entire graph
(10, 7)	{7, 8, 10}

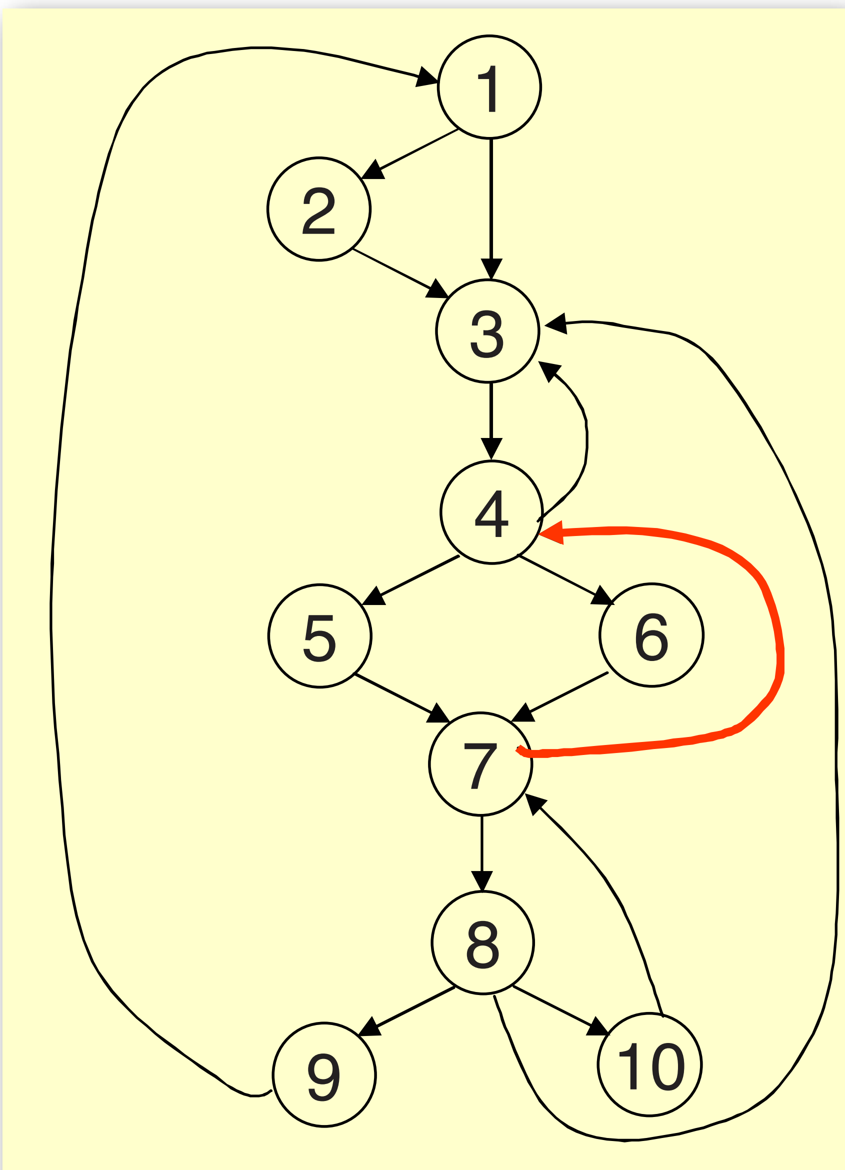
Example



Find all back edges in this graph and the natural loop associated with each back edge

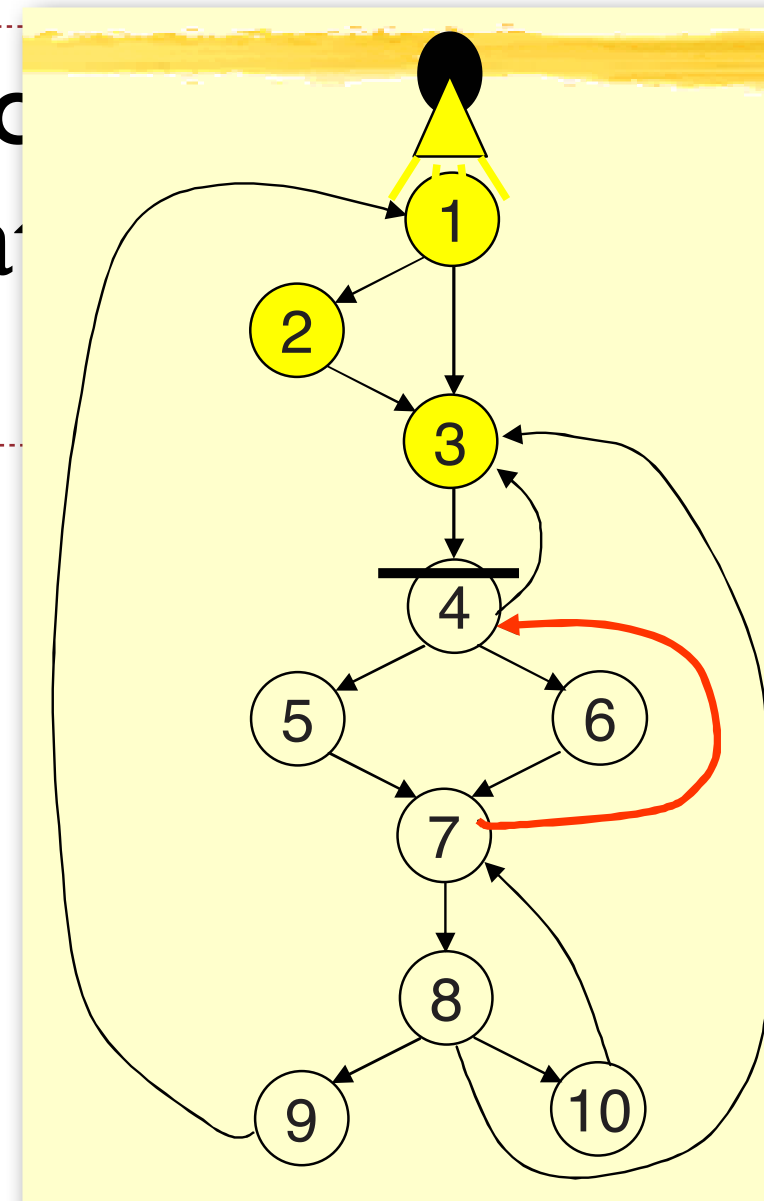
(9, 1)	Entire graph
(10, 7)	{7, 8, 10}
(7, 4)	

Example

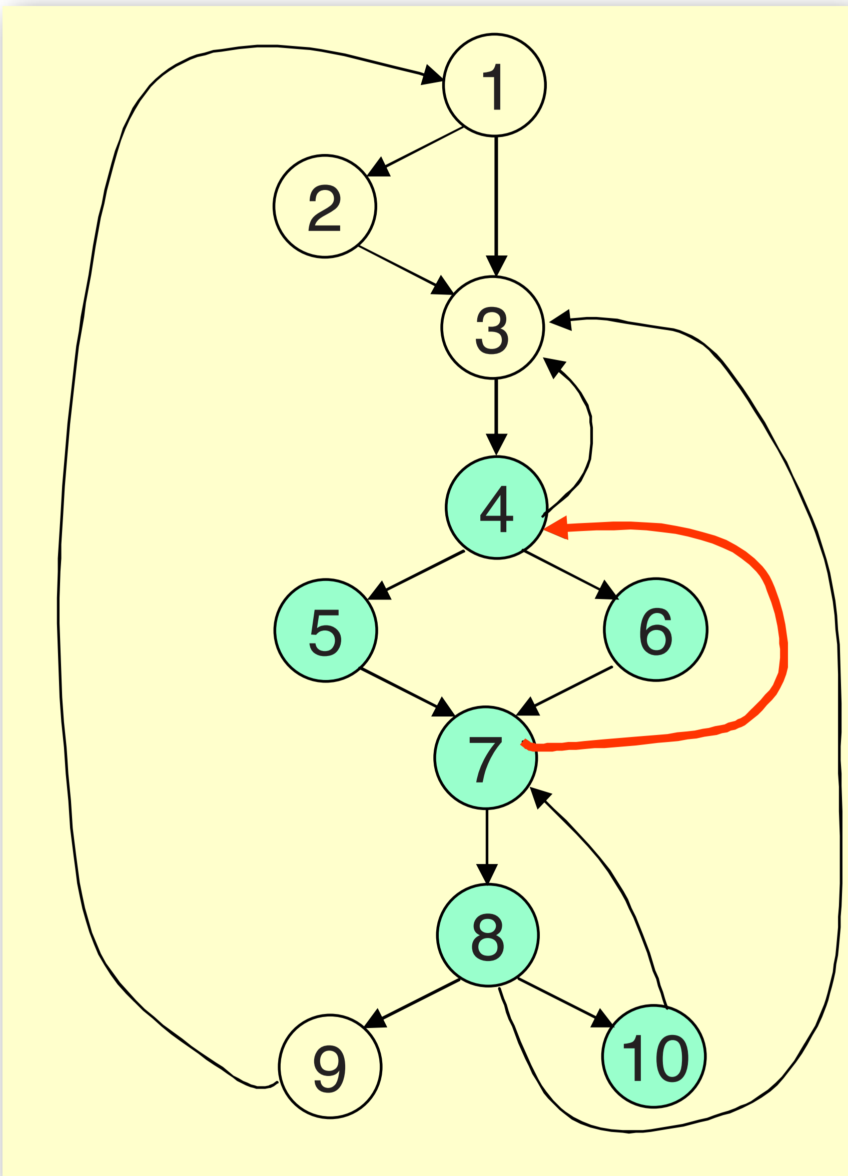


Find all bac… raph
and the na… ated
with

(9, 1)
(10, 7)
(7, 4)



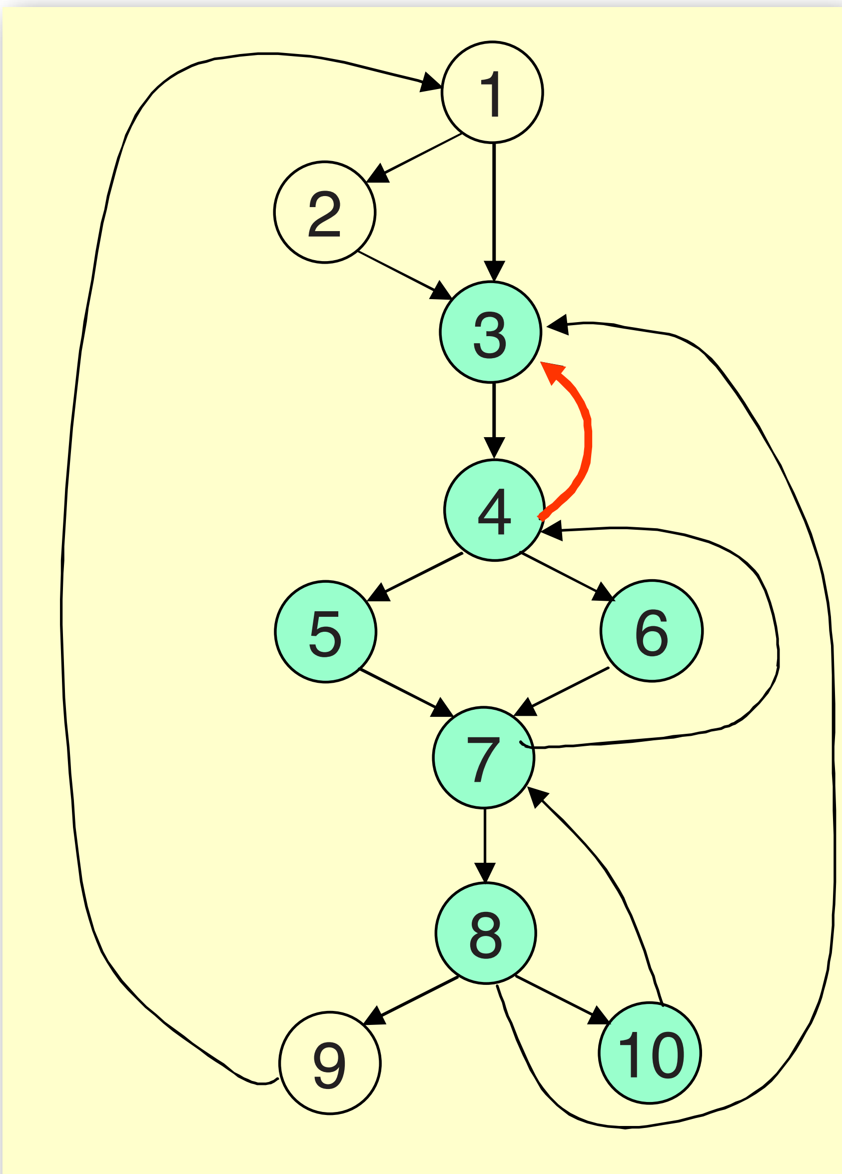
Example



Find all back edges in this graph and the natural loop associated with each back edge

(9, 1)	Entire graph
(10, 7)	{7, 8, 10}
(7, 4)	{4, 5, 6, 7, 8, 10}

Example



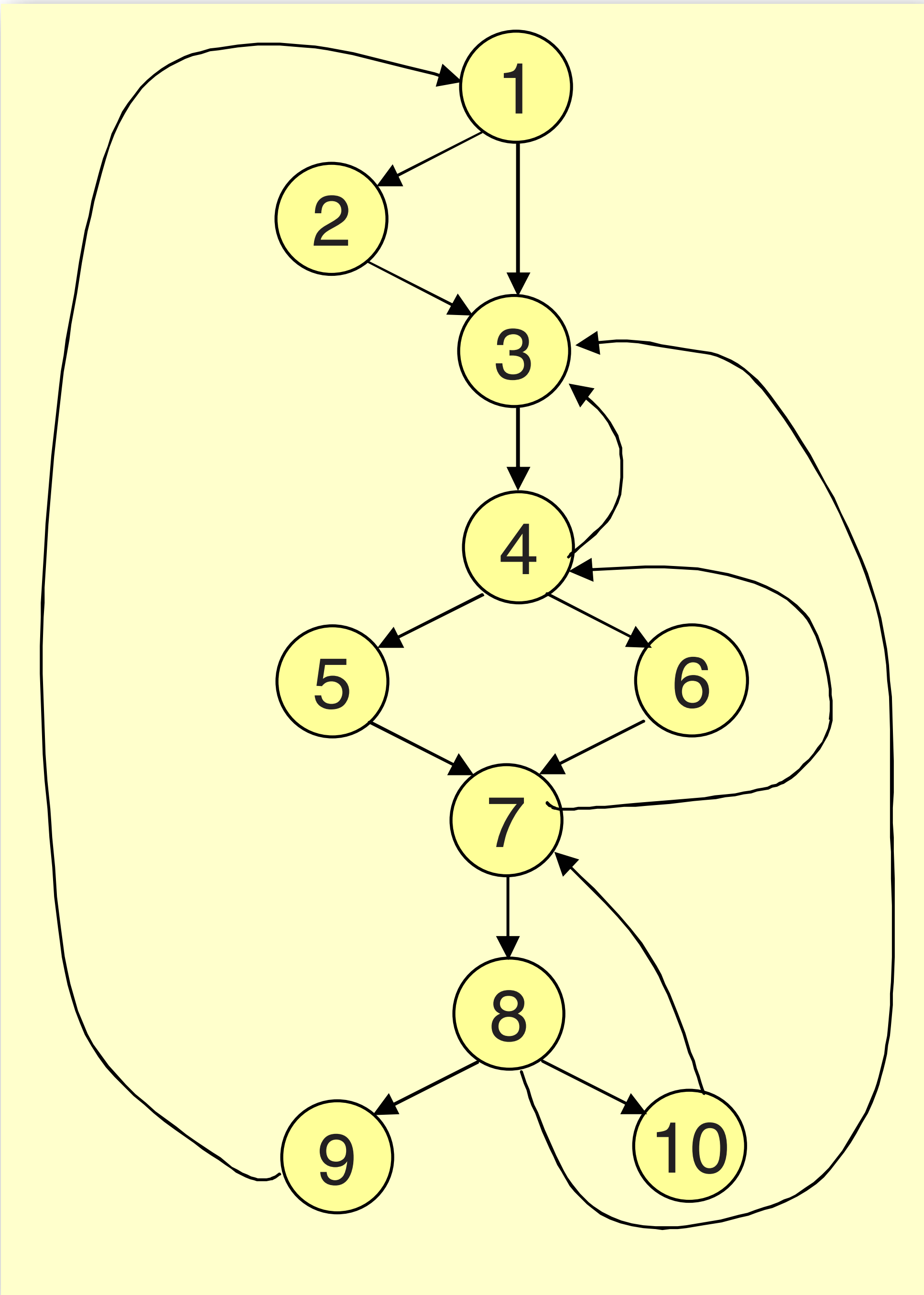
Find all back edges in this graph and the natural loop associated with each back edge

(9, 1)	Entire graph
(10, 7)	{7, 8, 10}
(7, 4)	{4, 5, 6, 7, 8, 10}
(8, 3)	{3, 4, 5, 6, 7, 8, 10}
(4, 3)	{3, 4, 5, 6, 7, 8, 10}

Dominator Tree

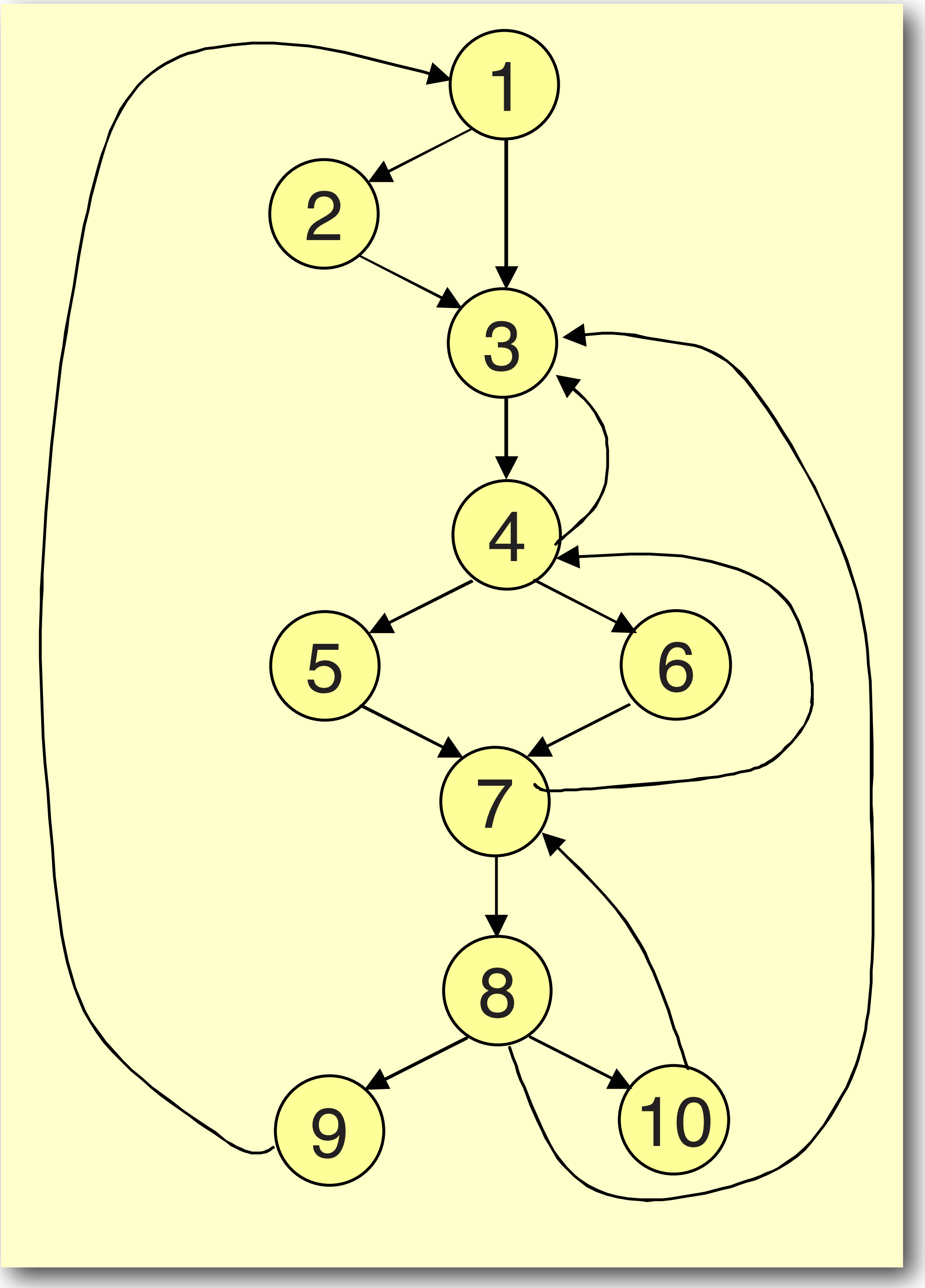
- A *dominator tree* is a useful way to represent the dominance relation
- In a dominator tree the start node **s** is the root, and each node **d** dominates only its descendents in the tree
- Efficient construction with the help of the *Lengauer Tarjan algorithm* (see Moodle for link to paper)

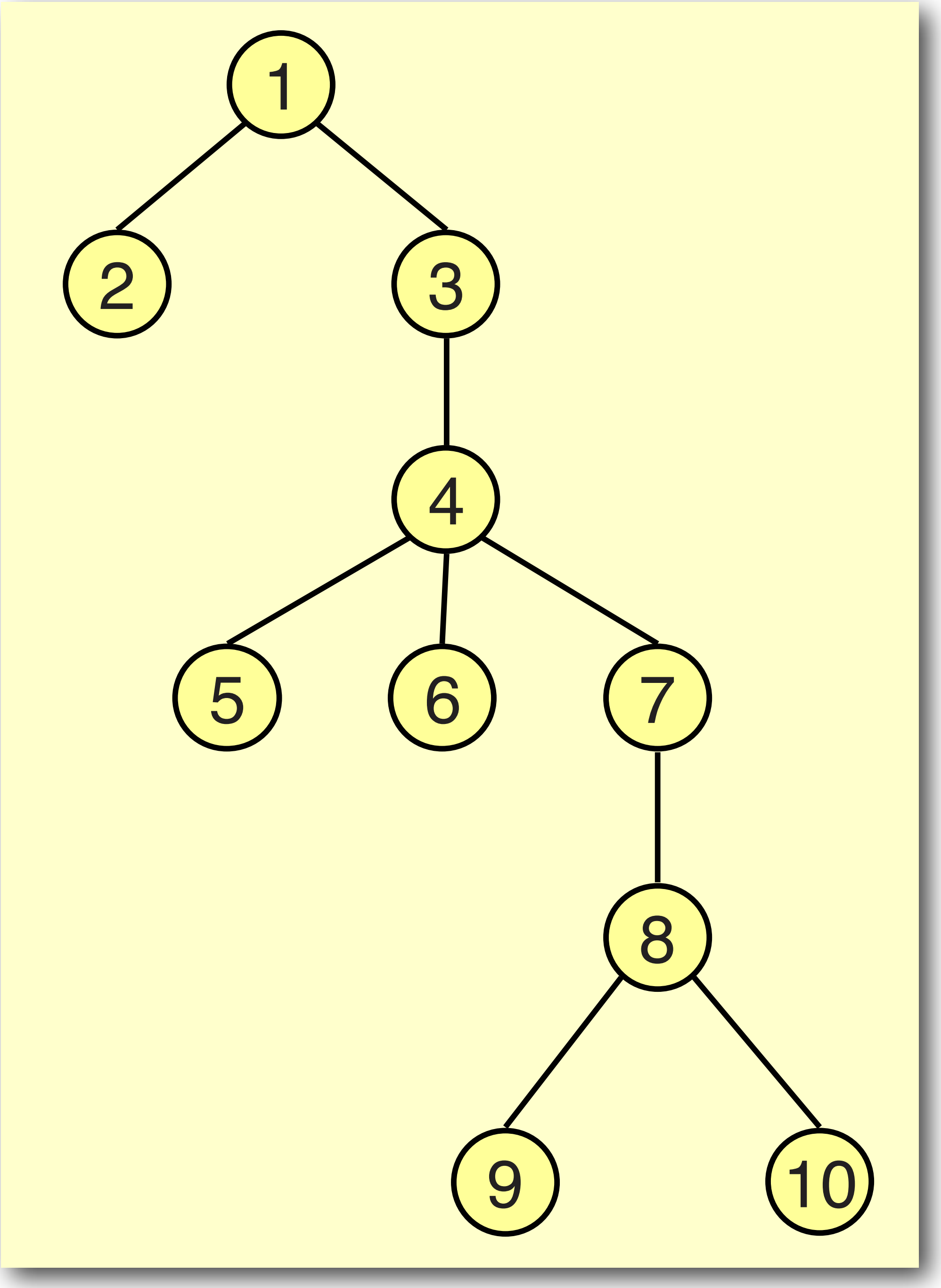
Example



In a dominator tree the start node **s** is the root, and each node **d** dominates only its descendents in the tree

Example





Regions

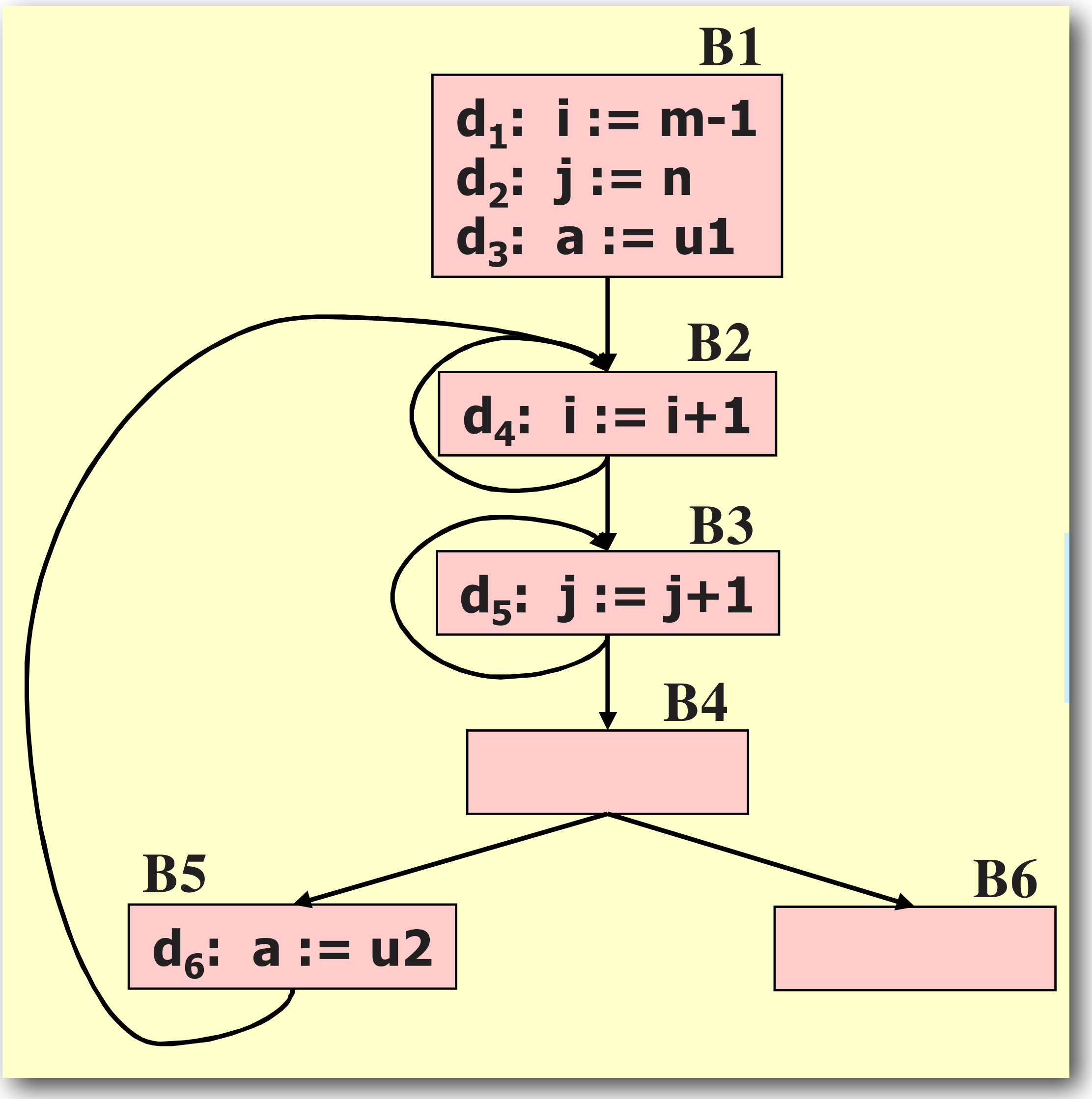
- A *region* is a set of nodes N that include a *header* with the following properties:
  1. header must dominate all the nodes in the region
  2. all the edges between nodes in N are in the region (except for some edges that enter the header)
- A loop is a special region with additional properties:
  1. it is strongly connected
  2. all back edges to the header are included in loop

Regions

- A *region* is a set of nodes N that include a *header* with the following properties:
   1. header must dominate all the nodes in the region
   2. all the edges between nodes in N are in the region (except for some edges that enter the header)
- A loop is a special region with additional properties:

Typically we are interested on studying the data flow into and out of regions. For instance, which definitions reach a region

Points and Paths

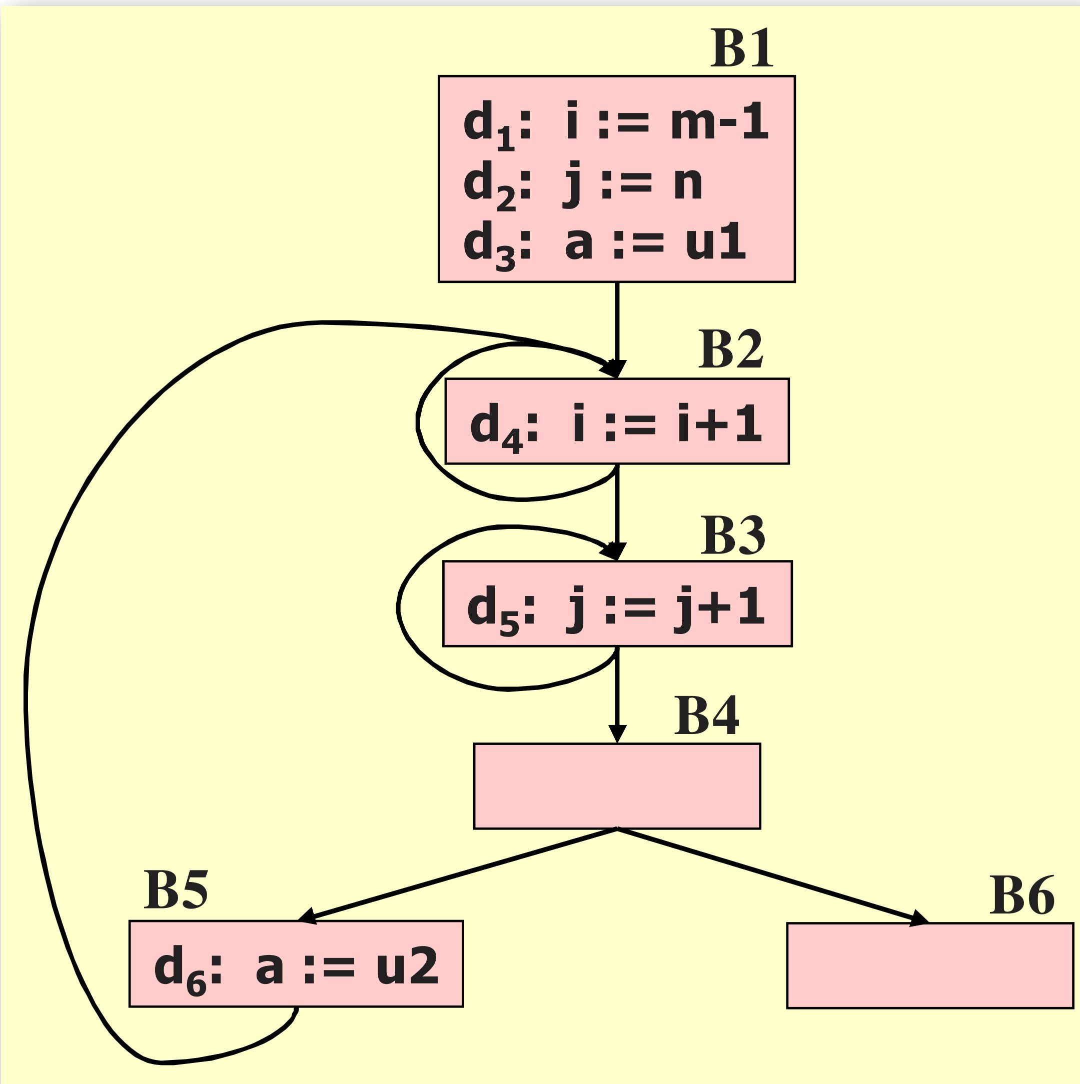


Points in a basic block:

- between statements
- before the first statement
- after the last statement

In the example, how many points do basic blocks B1, B2, B3, and B5 have?

Points and Paths

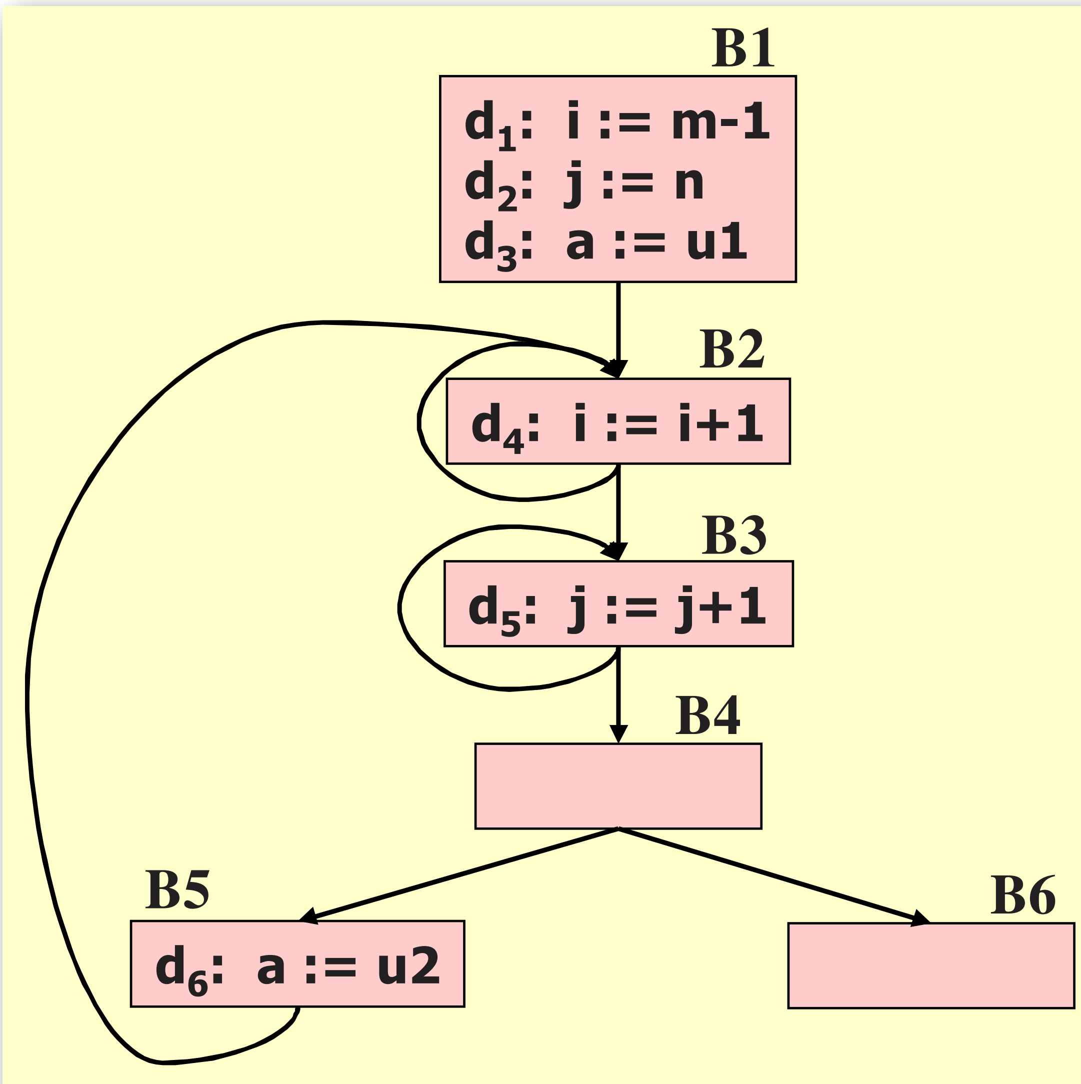


Points in a basic block:

- between statements
- before the first statement
- after the last statement

B1 has four, B2, B3, and B5 have two points each

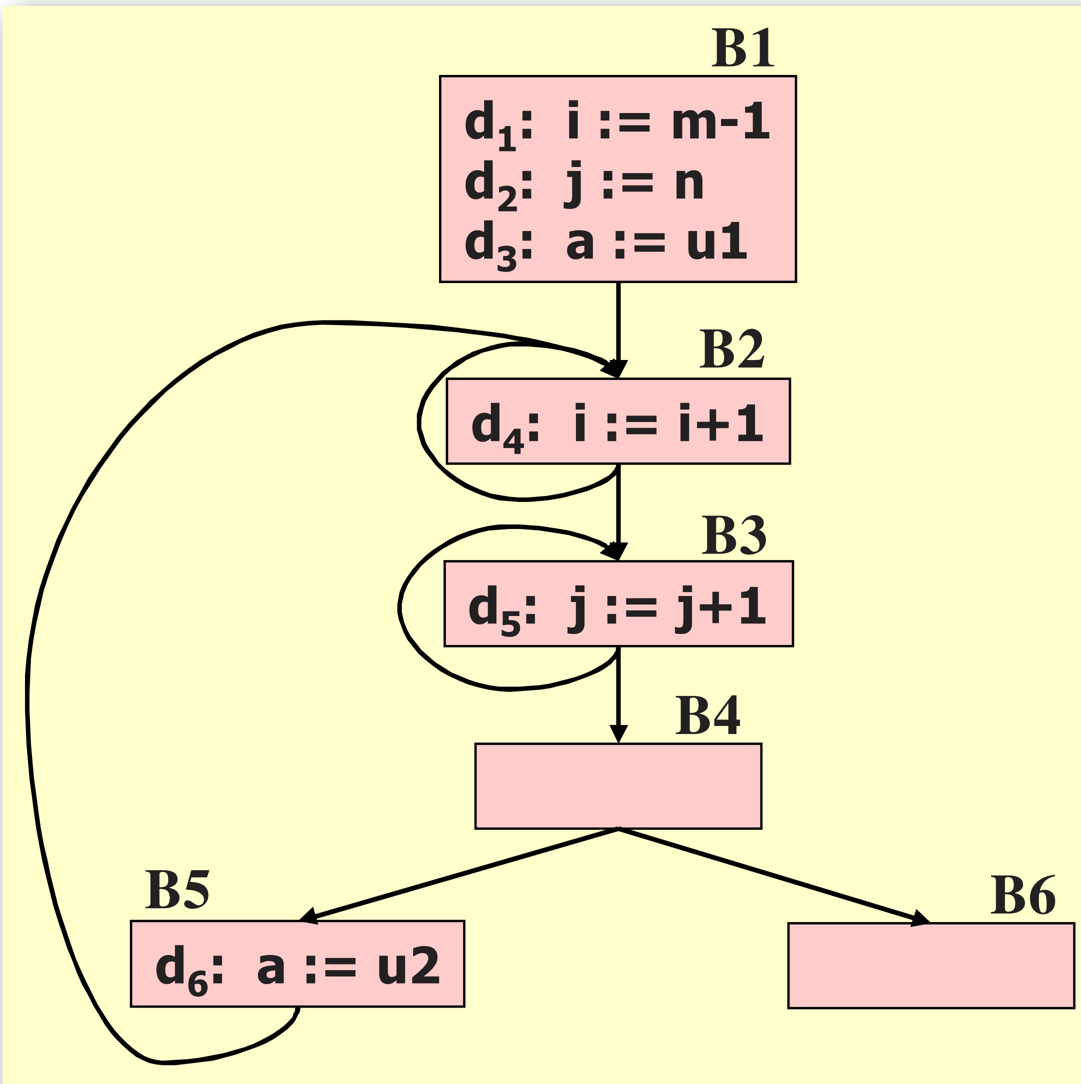
Points and Paths



A *path* is a sequence of points p_1, p_2, ..., p_n such that either:

1. if p_i immediately precedes *S*, then p_{i+1} immediately follows *S*
2. or p_i is the end of a basic block and p_{i+1} is the beginning of a successor block

Points and Paths

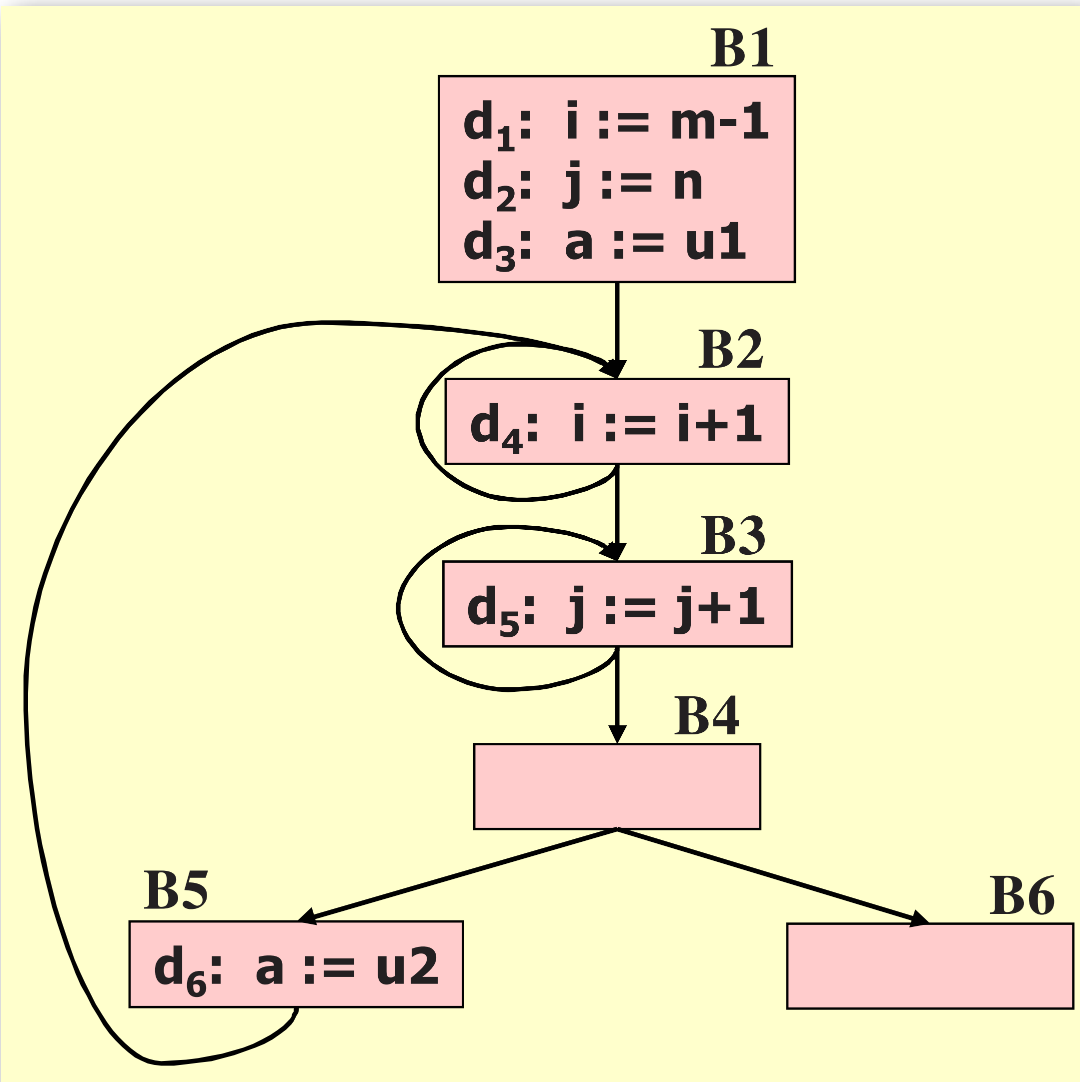


A *path* is a sequence of points p_1, p_2, ..., p_n such that either:

1. if p_i immediately precedes *S*, then p_{i+1} immediately follows *S*
2. or p_i is the end of a basic block and p_{i+1} is the beginning of a successor block

In the example, is there a path from the beginning of block B5 to the beginning of block B6?

Points and Paths



A *path* is a sequence of points p_1, p_2, ..., p_n such that either:

1. if p_i immediately precedes *S*, then p_{i+1} immediately follows *S*
2. or p_i is the end of a basic block and p_{i+1} is the beginning of a successor block

There is a path through the end point of B5 and then through all the points in B2, B3, and B4

Global Dataflow Analysis

- We need to know variable **def** and **use** information between basic blocks for:
  - constant folding
  - dead-code elimination
  - redundant computation elimination
  - code motion
  - induction variable elimination
  - build data dependence graph (DDG)

Definition and Use

S_k: $V_1 = V_2 + V_3$

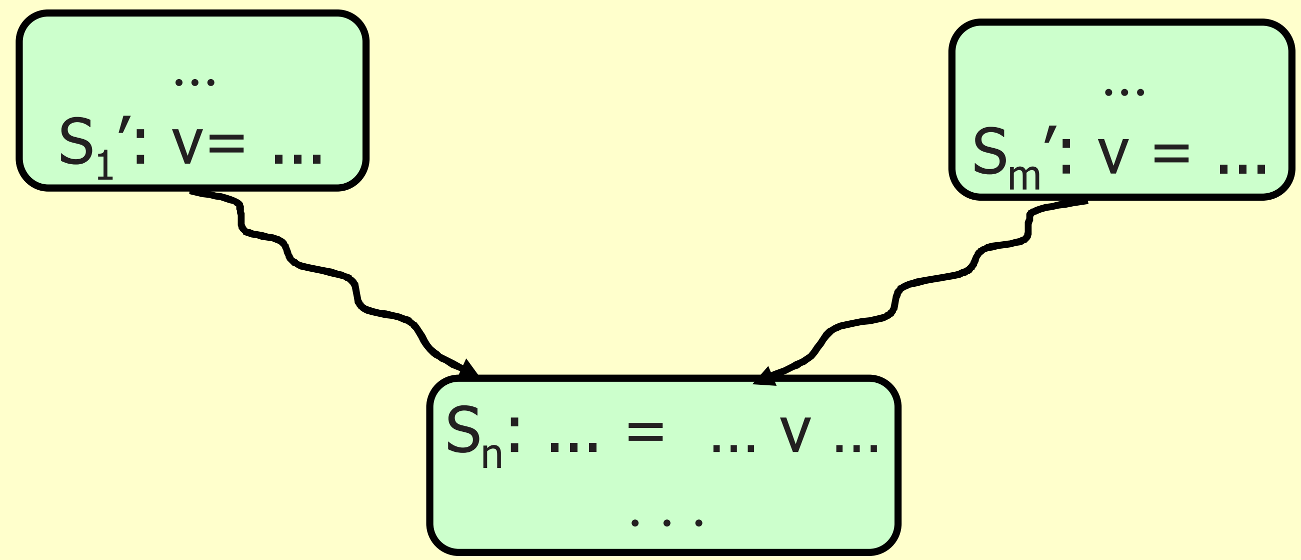
S_k is a *definition* of V_1
S_k is an *use* of V_2 and V_3

- We often need to study *def-use chains* to track how data propagates within a program
- Useful for other purposes as well

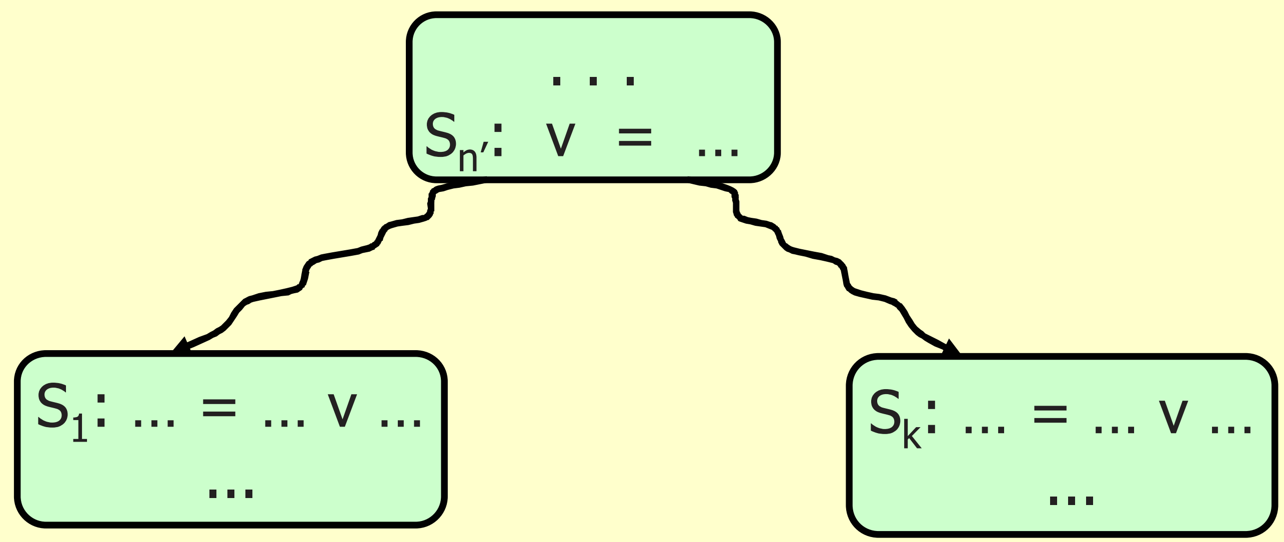
DU and UD Chains

- Many dataflow analyses need to find the use-sites of each defined variable or the definition-sites of each variable used in an expression
- *Def-Use* (D-U) and *Use-Def* (U-D) chains are efficient data structures that keep this information
  - An UD chain is a list of all definitions that can reach a given use of a variable
  - A DU chain is a list of all uses that can be reached by a given definition of a variable

Example



A **UD chain**: $UD(S_n, v) = (S_1', \ldots, S_m')$.



A **DU chain**: $DU(S_{n'}, v) = (S_1, \ldots, S_k)$.

Questions?

Contact:
Prof. Thorsten Holz
thorsten.holz@rub.de
@thorstenholz on Twitter

More information:
http://syssec.rub.de
http://moodle.rub.de

Sources

- Lecture *Software Reverse Engineering* at University of Mannheim, spring term 2010 (Ralf Hund, Carsten Willems and Felix Freiling)
- Lecture *Compiler Design and Optimization* at University of Alberta by José Nelson Amaral
  - http://webdocs.cs.ualberta.ca/~amaral/courses/680/index.html