PROGRAM ANALYSIS

Baishakhi Ray

What is Program Analysis

- A process of automatically analyzing a program behavior
- Different program properties can be analyzed for
  - Program Optimization
  - Program Correctness

Example:

```
int foo(void)
{
  int a = 24;
  if (a > 25)
  {
    return(25);
    a = 25
  }
  return(a);
  b = 24;
  return(b);
}
```

- How/Whether a statement will be executed?
  - Control Flow Analysis
- How the values propagate?
  - Data Flow Analysis

Underlined code is dead code

Control Flow Analysis

- Representation of Control Flow: Control Flow Graph (CFG)
  - Nodes represent statements (low-level linear IR)
  - Edges represent explicit flow of control

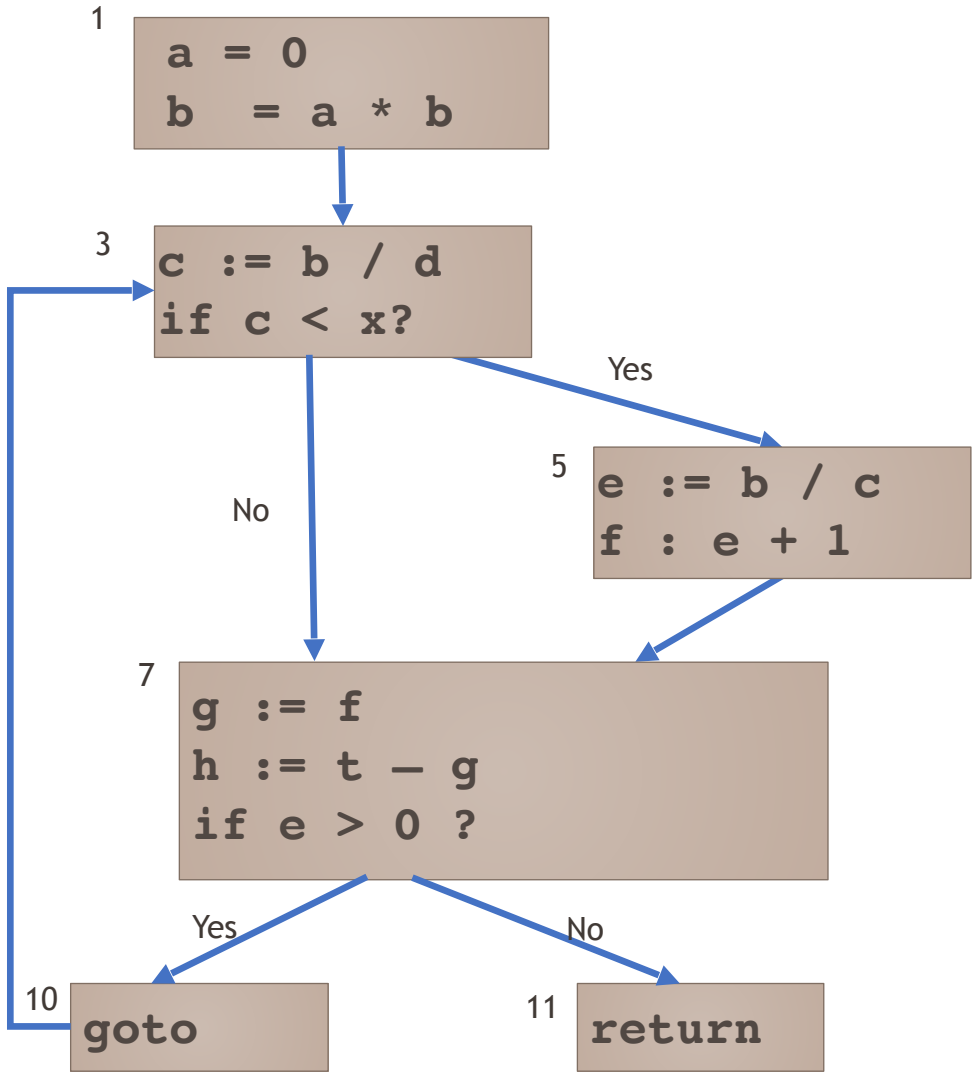
Control Flow Graph

```
1.      a = 0
2.      b = a * b
3. L1: c = b/d
4.      if c < x goto L2
5.         e = b/c
6.         f = e + 1
7. L2: g = f
8.      h = t -g
9.      if e > 0 goto L3
10. goto L1
11. L3: return
```

Yes

Control Flow Graph

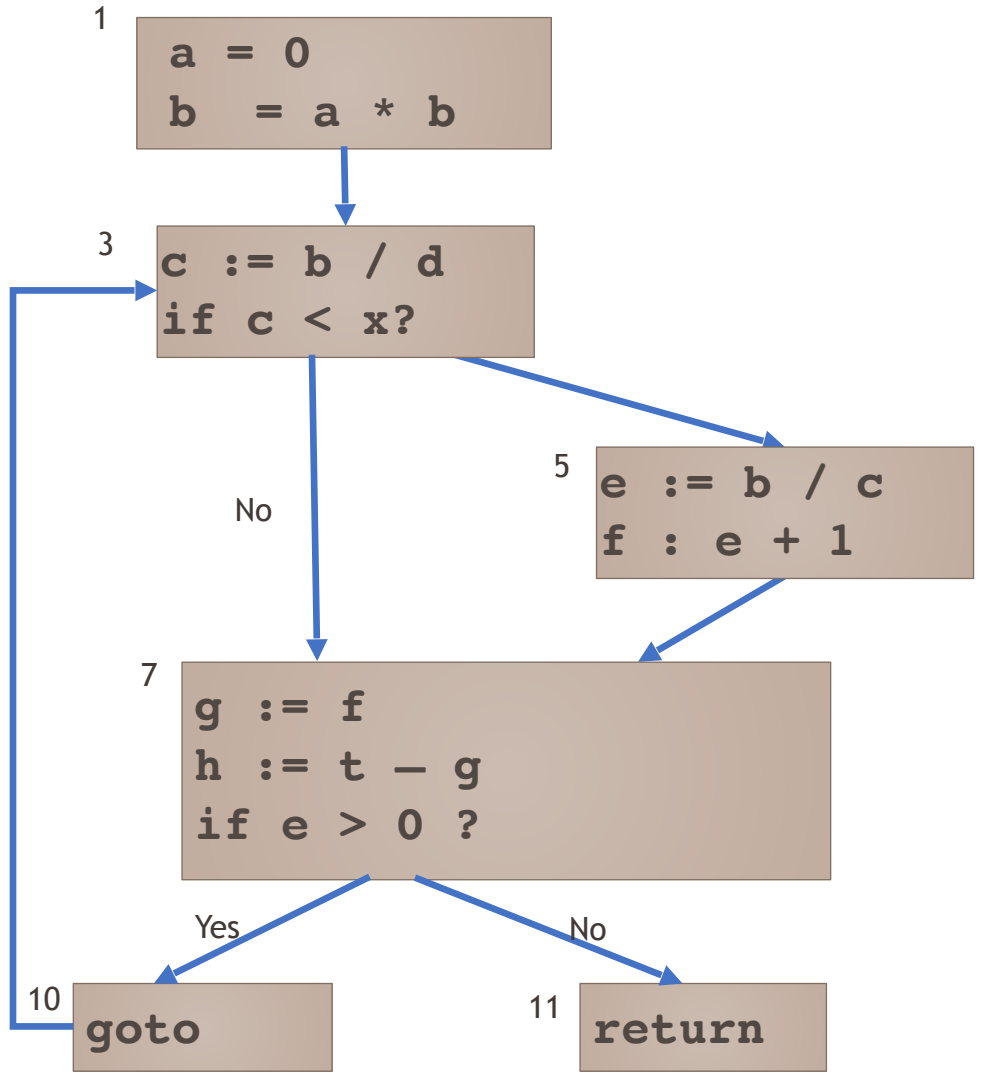
```
1.      a = 0
2.      b = a * b
3. L1: c = b/d
4.      if c < x goto L2
5.         e = b/c
6.         f = e + 1
7. L2: g = f
8.      h = t -g
9.      if e > 0 goto L3
10. goto L1
11. L3: return
```



Basic Blocks (BB)

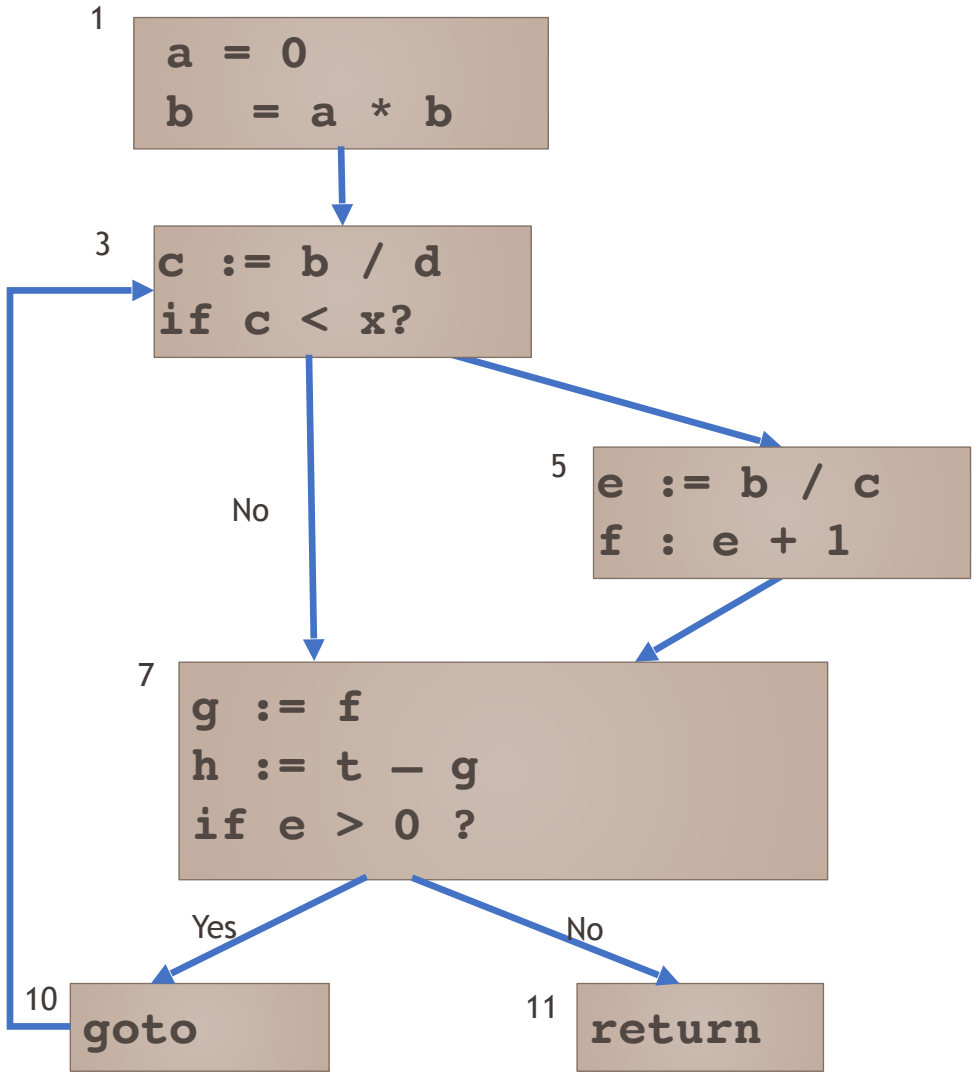
- A sequence of straight line code that can be entered only *at the beginning* and exited only

- Building BB
  - Identify Leaders
    - The first instruction in a procedure, or
    - The target of any branch, or
    - An instruction immediately following a branch (implicit target)
  - Gobble all subsequent instructions until the next leader



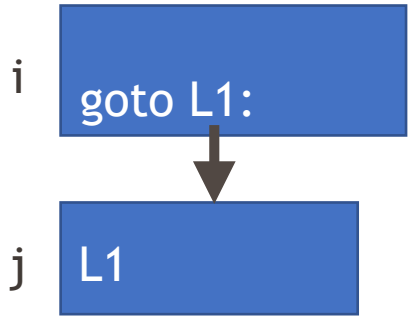
Basic Blocks (BB)

- Leaders:
  - {1, 3, 5, 7, 10, 11}
- Blocks
  - {1, 2}
  - {3, 4}
  - {5, 6}
  - {7, 8, 9}
  - {10}
  - {11}

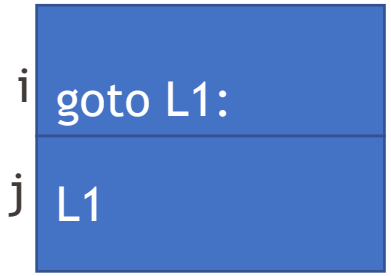


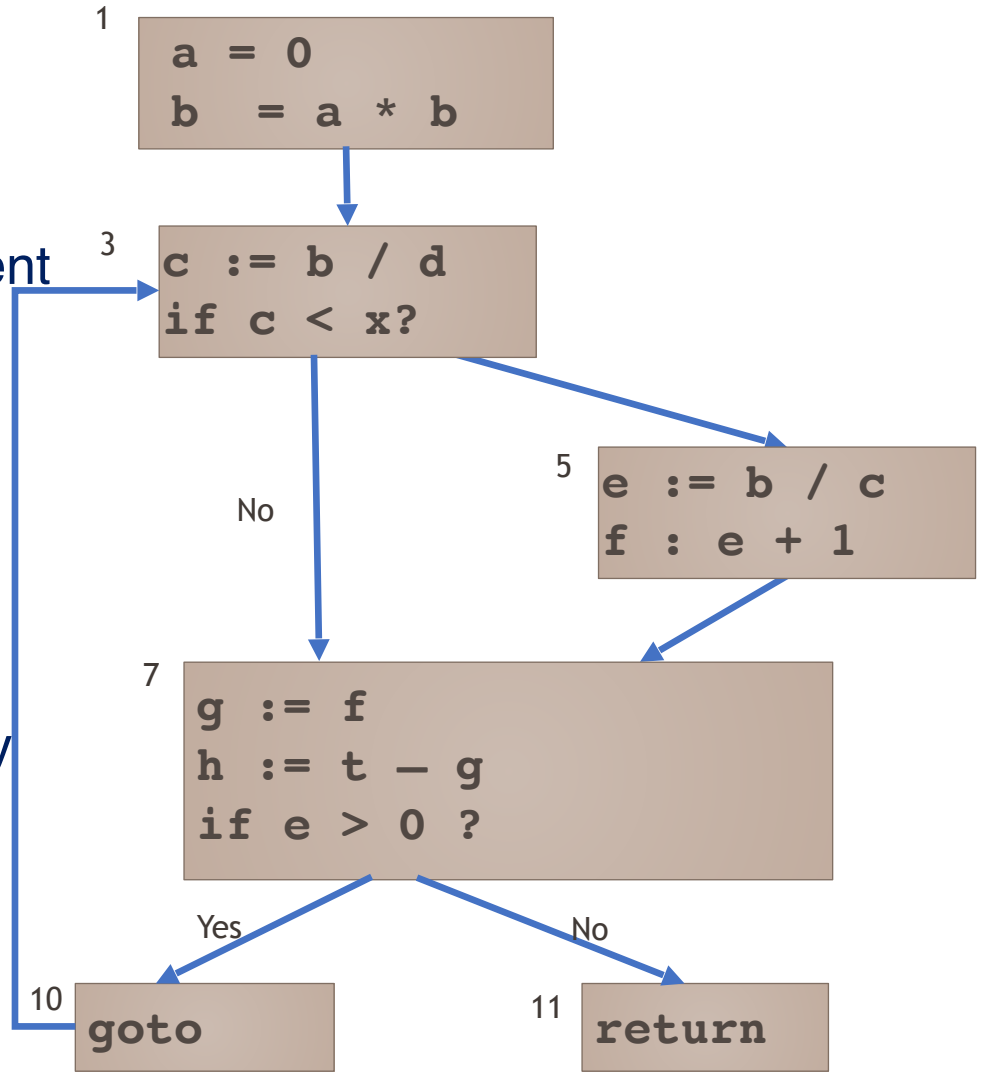
Constructing CFG

- Each CFG node represents a basic block
- There is an edge from node i to j if
  - Last statement of block i branches to the first statement of j, or



  - Block i does not end with a branch and is immediately followed in program order by block j (fall through)





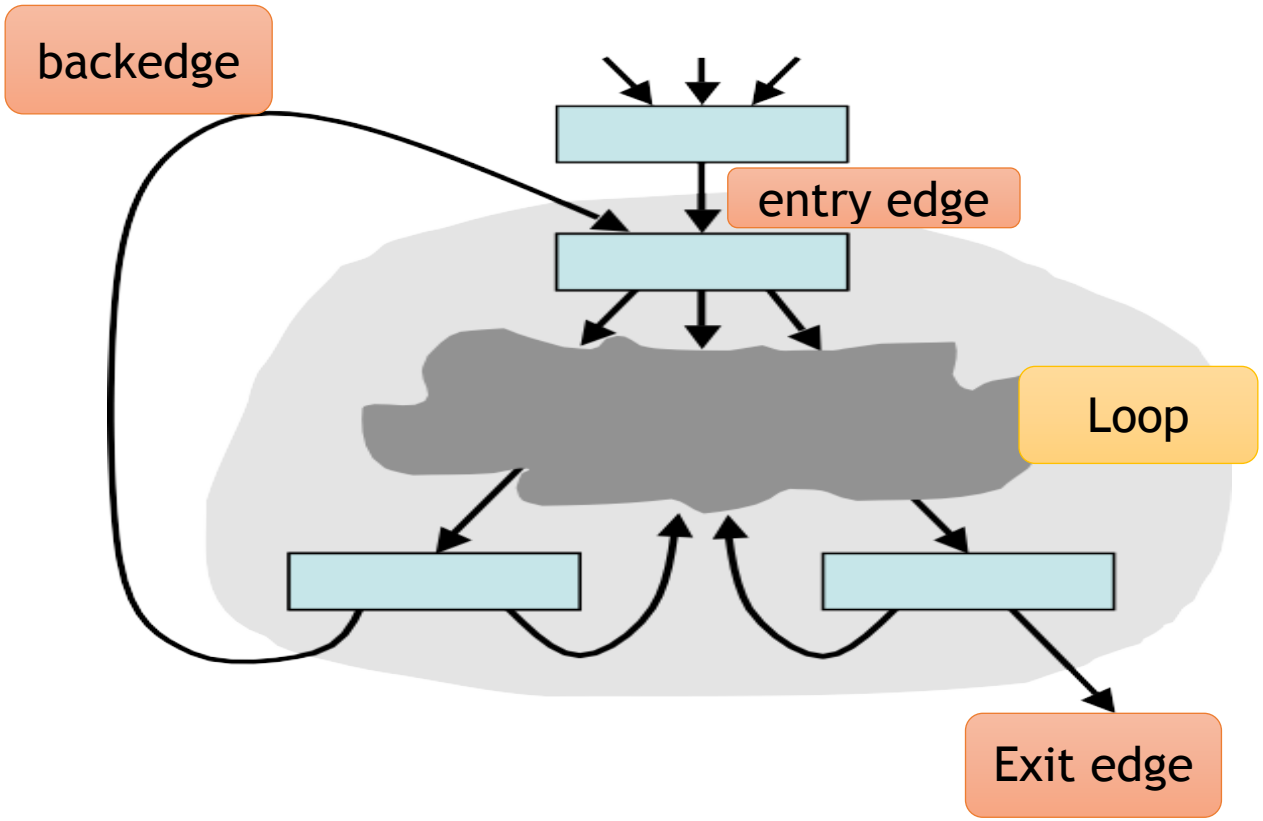
CFG Paths

- Consider a flow graph G= (N, E).
- A sequence of k edges, k>0, $(e_1, e_2, e_3, \ldots . e_k)$, denotes a path of length k through the flow graph if the following sequence condition holds.
- Given that n_p, n_q, n_r, and n_s are nodes belonging to N, and 0< i<k, if $e_i = (n_p, n_q)$, and $e_i + 1 = (n_r, n_s)$, then $n_q = n_r$.
- Complete Path: a path from start to exit
- Subpath: a subsequence of complete path

CFG Paths

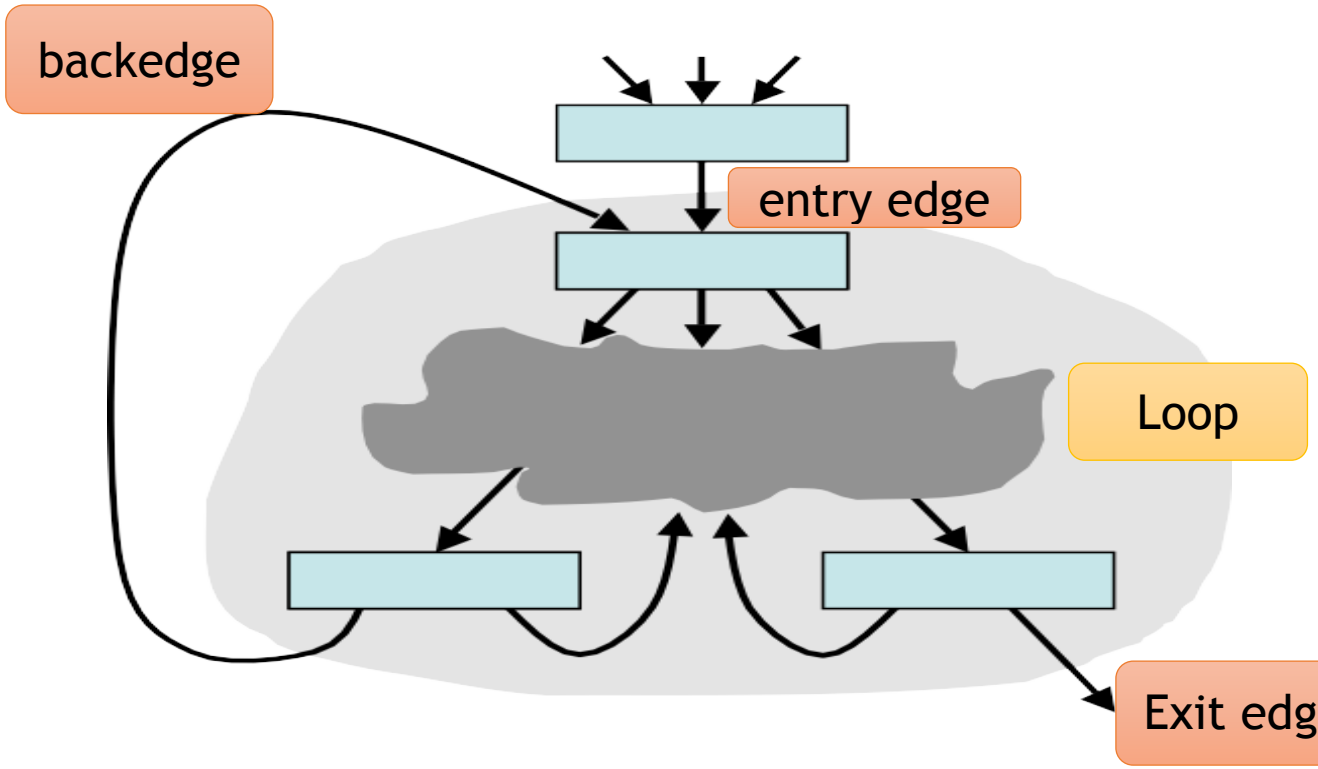
- There can be many distinct paths in a program
- A program with no condition will have only one path
- Each additional condition increases the number of path by at least one
- Depending on their location and nature, condition can have multiplicative effect on the number of path.

Looping



- Loop: Strongly connected component of CFG
- Entry Edge: Source not in loop but target in the loop
- Exit Edge: Source in the loop but target not in the loop
- Header node: Target of loop entry edge
- Back edge: Target is loop header, and source is in the loop
- Tail node: source of back edge
- Preheader: Single node that is source to the loop entry edge
- Nested Loop: Loop whose header is inside another loop

Looping

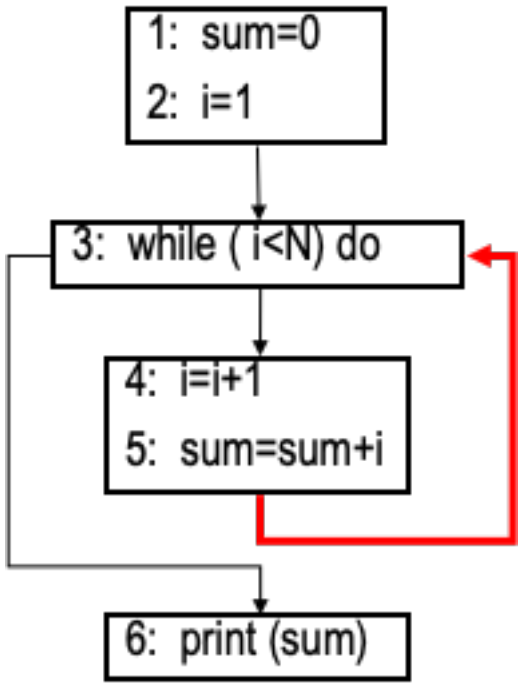


- backedges indicate that we might need to traverse the CFG more than once for data flow analysis

- Not all loops have preheaders
  - Sometimes it is useful to create them.

Without preheader node, there can be multiple entry edges. With single preheader node, there is only one entry edge.

Back Edge



Identifying loop

- Why is it important?
  - Most execution time spent in loops, so optimizing loops will often give most benefit
- Exploit hierarchical structure of programs
- Identify dominators to discover loops

Dominator

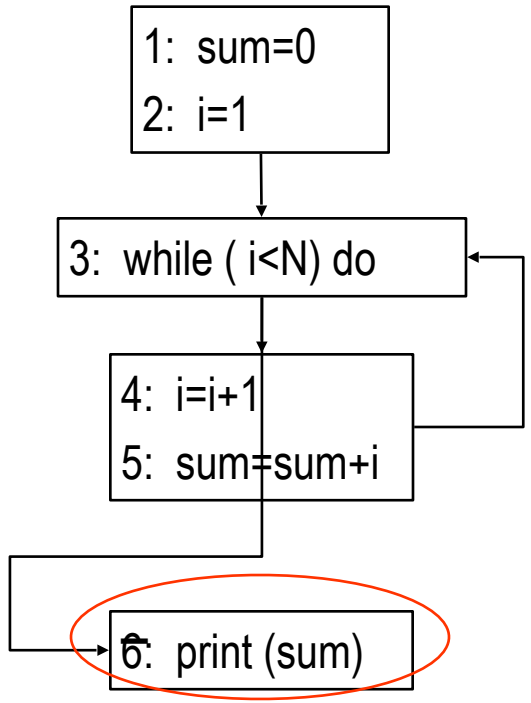
X dominates Y if all possible program paths from START to Y have to pass X.

Dominator

X strictly dominates Y if X dominates Y and X!=Y

```
1:    sum=0
2:    i=1
3:    while ( i<N) do
4:          i=i+1
5:          sum=sum+i
      endwhile
6:  print(sum)
```

SDOM(6)={1,3}

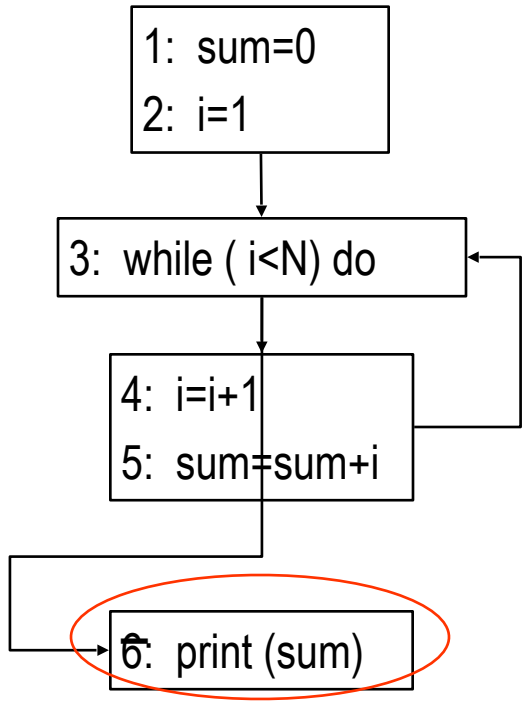


Dominator

X is the immediate dominator of Y if X is the last dominator of Y along a path from Start to Y.

```
1:    sum=0
2:    i=1
3:    while ( i<N) do
4:          i=i+1
5:          sum=sum+i
      endwhile
6:  print(sum)
```

IDOM(6)={3}



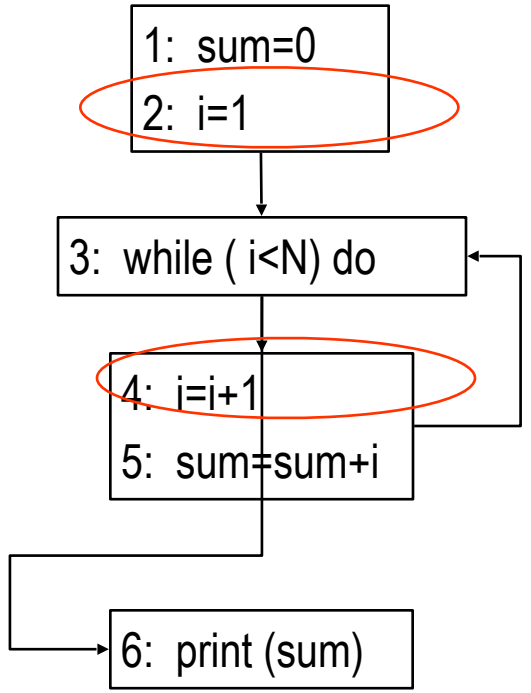
Postdominator

X post-dominates Y if every possible program path from Y to End has to pass X.

- Strict post-dominator, immediate post-dominance.

```
1:    sum=0
2:    i=1
3:    while ( i<N) do
4:          i=i+1
5:          sum=sum+i
      endwhile
6:  print(sum)
```

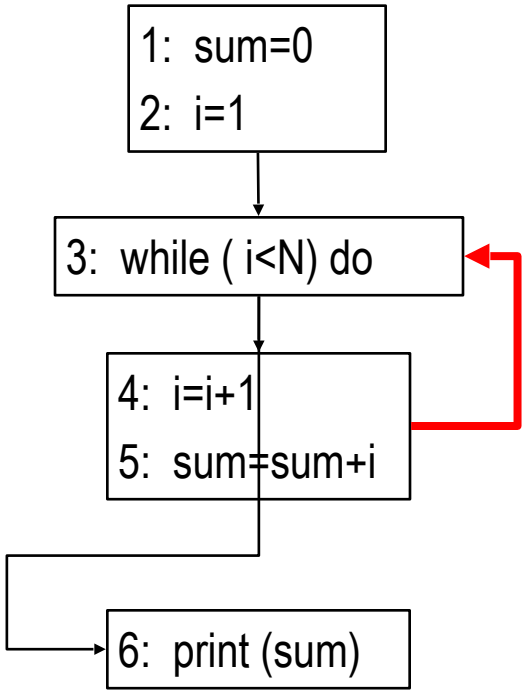
SPDOM(4)={3,6} IPDOM(4)=3



Back Edges

A back edge is an edge whose head dominates its tail

- Back edges often identify loops

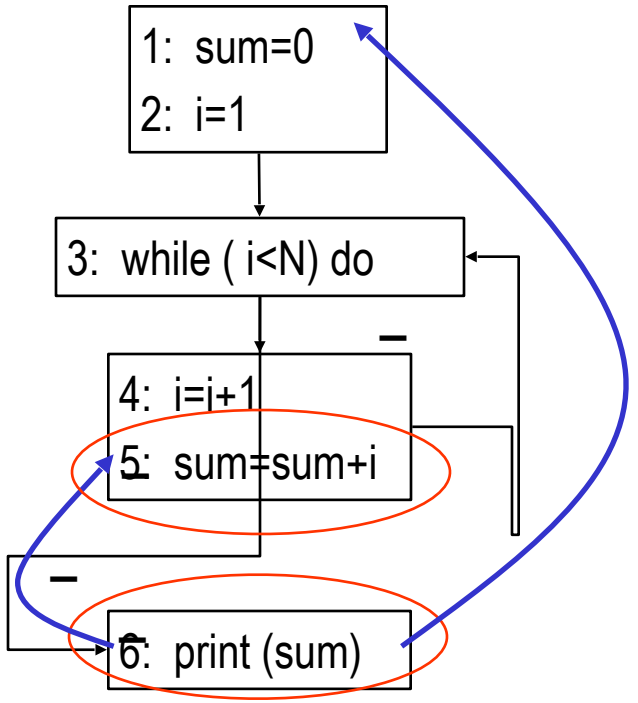


Program Dependence Graph

- The second widely used program representation.
- Nodes are constituted by statements instead of basic blocks.
- Two types of dependences between statements
  - Data dependence
  - Control dependence

Data Dependence

X is data dependent on Y if (1) there is a variable v that is defined at Y and used at X and (2) there exists a path of nonzero length from Y to X along which v is not re-defined.



Control Dependence

Intuitively, Y is control-dependent on X iff X directly determines whether Y executes (statements inside one branch of a predicate are usually control dependent on the predicate)

- X is not strictly post-dominated by Y

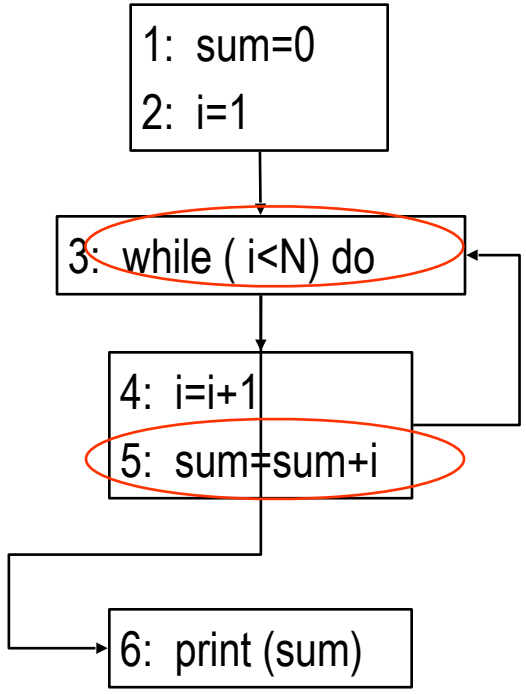
- There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y

Control Dependence - Example

Y is control-dependent on X iff X directly determines whether Y executes

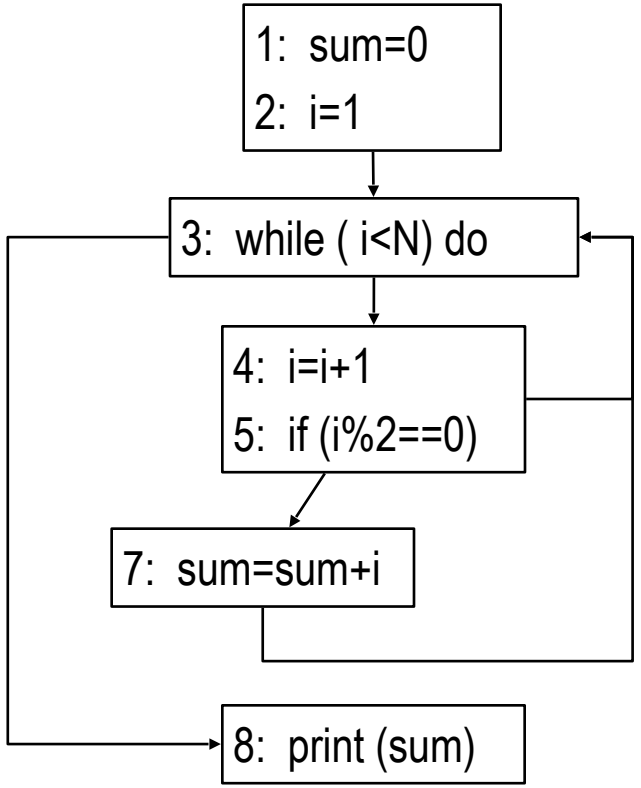
- X is not strictly post-dominated by Y
- There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y

```
1:    sum=0
2:    i=1
3:    while ( i<N) do
4:          i=i+1
5:          sum=sum+i
      endwhile
6:  print(sum)
```



Control Dependence - Example

```
1:    sum=0
2:    i=1
3:    while ( i<N) do
4:          i=i+1
5:          if (i%2==0)
6:               continue;
7:          sum=sum+i
      endwhile
8:    print(sum)
```



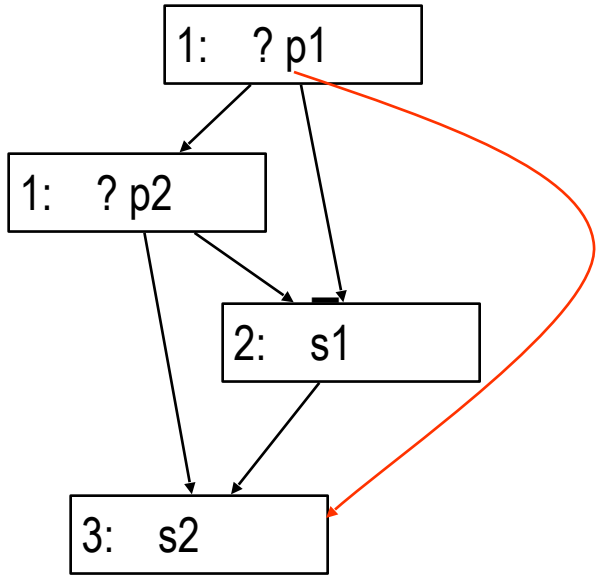
Control Dependence is Tricky!

Can one statement control depends on two predicates?

```
1:    if ( p1 || p2 )
2:          s1;
3:    s2;
```

What if ?

```
1:    if ( p1 && p2 )
2:          s1;
3:    s2;
```



The Use of PDG

A program dependence graph consists of control dependence graph and data dependence graph.

Call Graph (CG)

Each node represents a function; each edge represents a function invocation

```
void A( ) {
   B( );
   C( );
}

void B( ) {
L1:   D( );
L2:   D( );
}

void C ( ) {
   D( );
   A( );
}

void D ( ) {
}
```