Program Testing and Analysis: Path Profiling

Prof. Dr. Michael Pradel

Software Lab, TU Darmstadt

Warm-up Quiz

What does the following code print?

```
function d(a, b) {
  var sum = 0;
  for (var i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  console.log(sum == (a + b + 67));
}

d(23, 45, 67);
```

true **false** **Something else**

Warm-up Quiz

What does the following code print?

```
function d(a, b) {
  var sum = 0;
  for (var i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  console.log(sum == (a + b + 67));
}

d(23, 45, 67);
```

Includes all passed arguments, independent of declared parameters

true | **false** | **Something else**

Outline

1. **Motivation and Challenges**
2. **Ball-Larus algorithm for DAGs**
3. **Generalization and Applications**

Mostly based on this paper:

- *Efficient path profiling*, Ball and Larus, MICRO 1996

Other reading material:

- *Whole program paths* , Larus, PLDI 1999
- *HOLMES: Effective statistical debugging via efficient path profiling*, Chilimbi et al., ICSE 2009

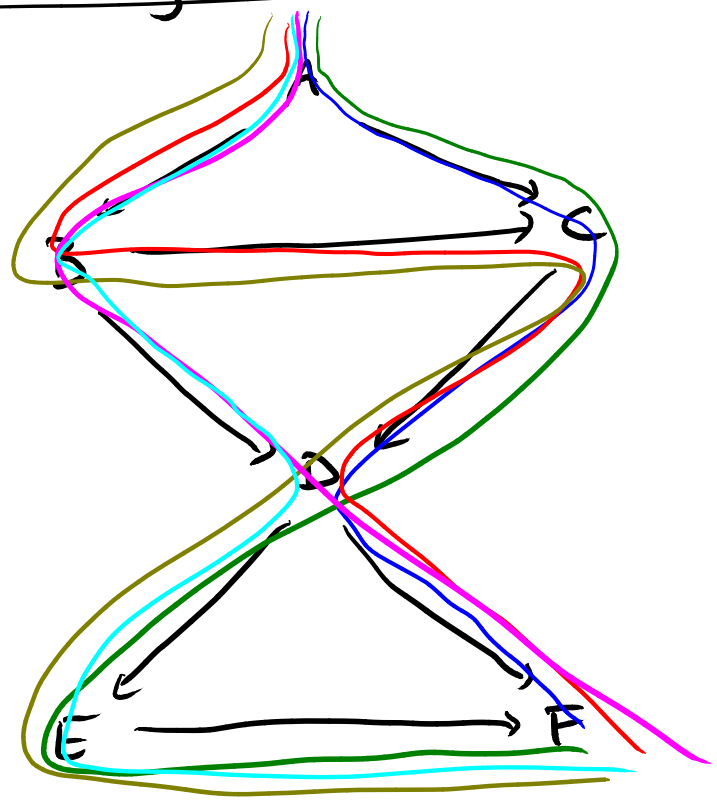
Path Profiling

- **Goal: Count how often a path through a function is executed**
- **Interesting for various applications**
  - Profile-directed compiler optimizations
  - Performance tuning: Which paths are worth optimizing?
  - Test coverage: Which paths are not yet tested?

Challenges

- **Runtime overhead**
  - Limit slowdown of program
- **Accuracy**
  - Ideally: Precise profiles (no heuristics, no approximations)
- **Infinitely many paths**
  - Cycles in control flow graph

Running example



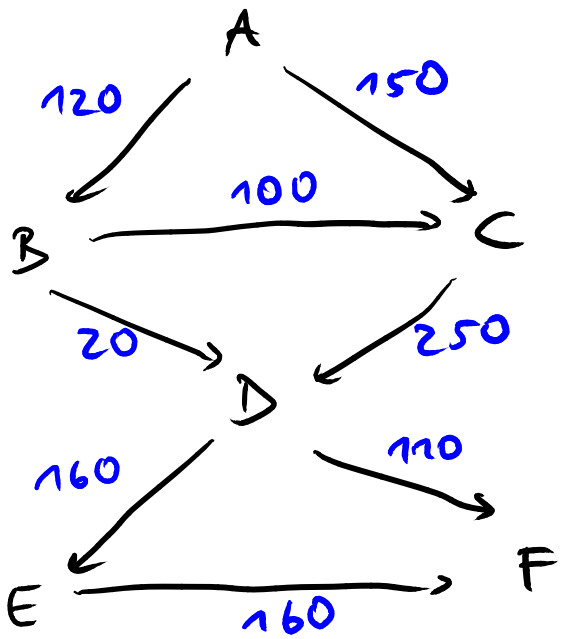
#	Path	Frequency
0	ACDF	2
1	ACDEF	
2	ABCDF	
3	ABCDEF	
4	ABDF	
5	ABDEF	0

Edge Profiling

Naive approach: Edge profiling

- **Instrument each branching point**
- **Count how often each CFG edge is executed**
- **Estimate most frequent path: Always follow most frequent edge**

Example: Edge profiling



frequency of execution

Quiz: What is the most frequent path?

ACDEF

Really? Two possible path profiles

Path	Profile 1	Profile 2
ACDF	90	$\boxed{110}$
ACDEF	60	40
ABCDF	0	0
ABCDEF	$\boxed{100}$	100
ABDF	20	0
ABDEF	0	20

Edge Profiling

Naive approach: Edge profiling

- **Instrument each branching point**
- **Count how often each CFG edge is executed**
- **Estimate most frequent path: Always follow most frequent edge**

Edge Profiling

Naive approach: Edge profiling

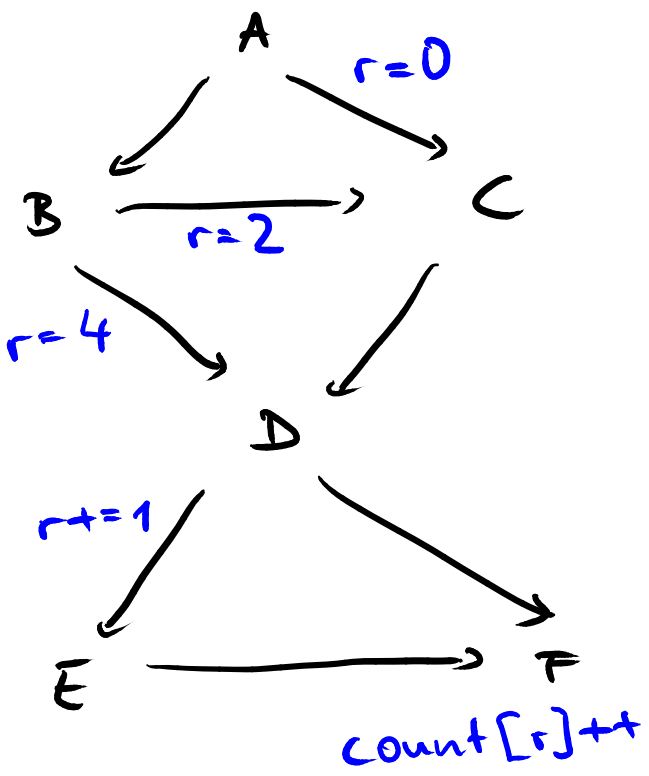
- **Instrument each branching point**
- **Count how often each CFG edge is executed**
- **Estimate most frequent path: Always follow most frequent edge**

Fails to uniquely identify most frequent path

Ball-Larus Algorithm

- **Assign a number to each path**
- **Compute path number by incrementing a counter at branching points**
- **Properties of path encoding**
  - Precise: A single unique encoding for each path
  - Minimal: Instruments subset of edges with minimal cost

Example: Path Encoding



instrumentation:

- state "r"
- array of counters "count"

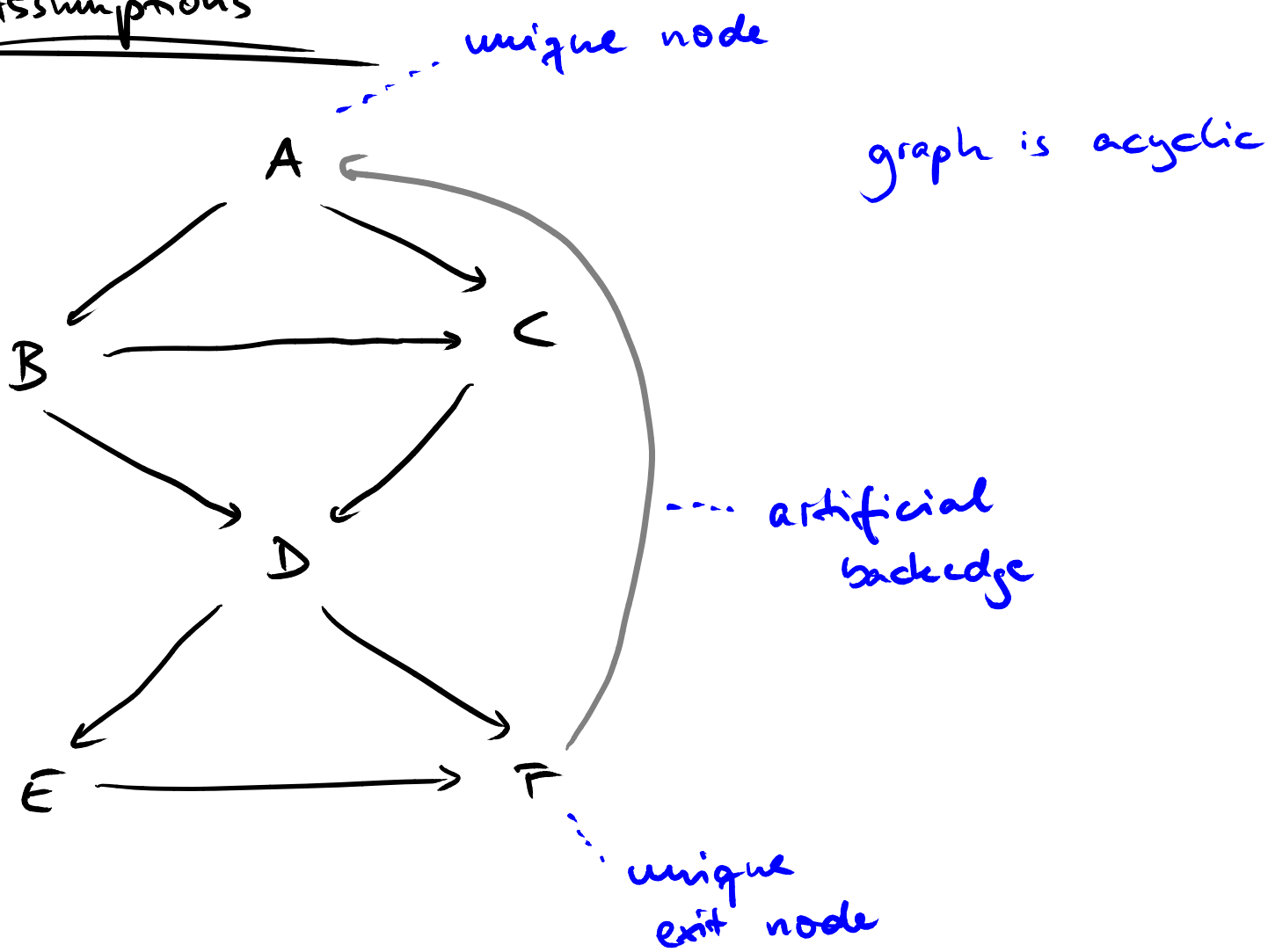
Path	Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Algorithm for DAGs

Assumptions

- **Control flow graph is a directed acyclic graph (DAG)**
- n **paths (numbered** 0 **to** $n-1$**)**
- **Graph has unique entry and exit nodes**
- **Artificial back edge from exit to entry**

Assumptions



graph is acyclic

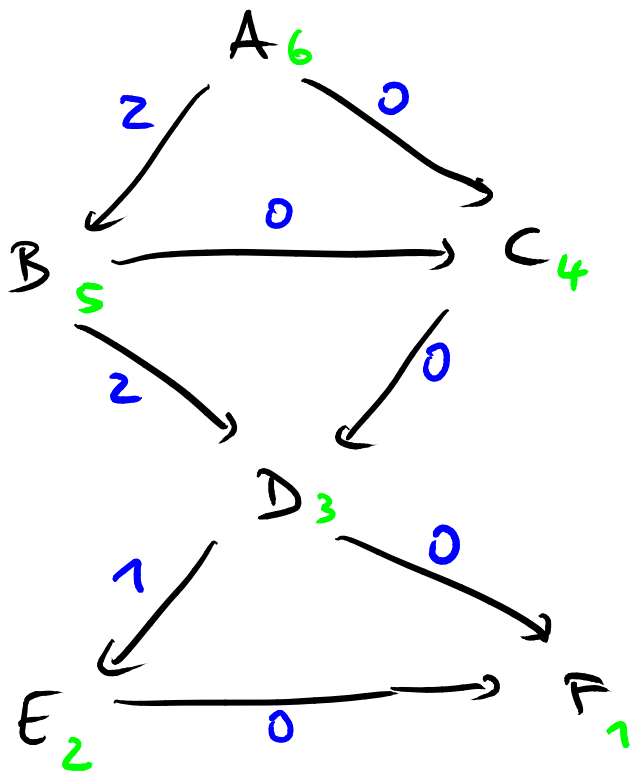
Algorithm: Overview

- **Step 1: Assign integers to edges**
  - Goal: Sum along a path yields unique number for path
  - Enough to achieve ”precise” goal
- **Step 2: Assign increment operations to edges**
  - Goal: Minimize additions along edges
  - Instrument subset of all edges
  - Assumes to know/estimate how frequent edges are executed

Representing Paths with Sums

- **Associate with each node a value:** $NumPaths(n)$ **= number of paths from** n **to exit**
- **Computing** $NumPaths$
  - Visit nodes in reverse topological order
  - If n is leaf node: $NumPaths(n) = 1$
  - Else: $NumPaths(n) =$ sum of $NumPaths$ of destination of outgoing edges

Example: NumPaths



Values for edges

Reverse topological order:
Successor of n visited before n

Node n	NumPaths(n)
F	1
E	1
D	2
C	2
B	4
A	6

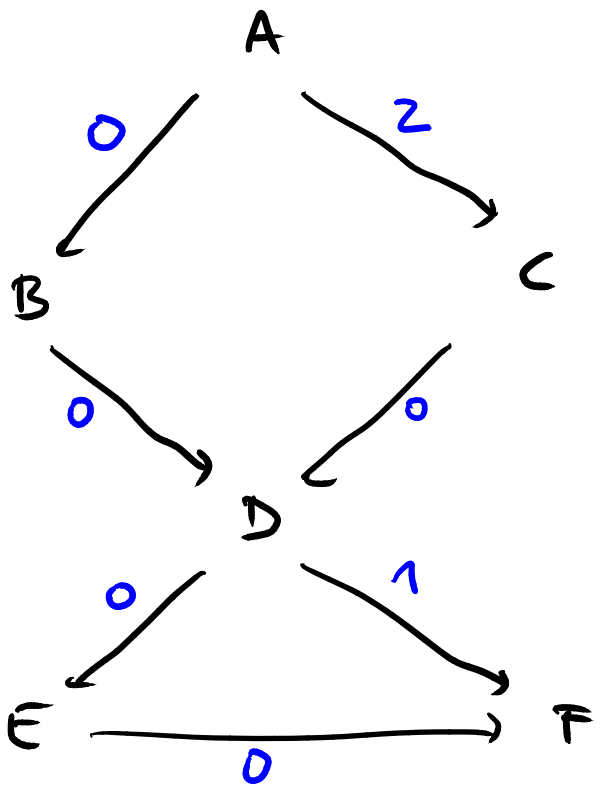
Representing Paths with Sums (2)

For each node in reverse topological order:

- **If n is leaf node:**

  $NumPaths(n) = 1$
- **Else:**
  - $NumPaths(n) = 0$
  - For each edge $n \to m$:
    - $Val(n \to m) = NumPaths(n)$
    - $NumPaths(n) \mathrel{+}= NumPaths(m)$

Quiz: Values for Edges



n	NumPath (n)
A	4
B	2
C	2
D	2
E	1
F	1

- Sum of all NumPaths? 12
- Encoding for ACDF? 3

Algorithm: Overview

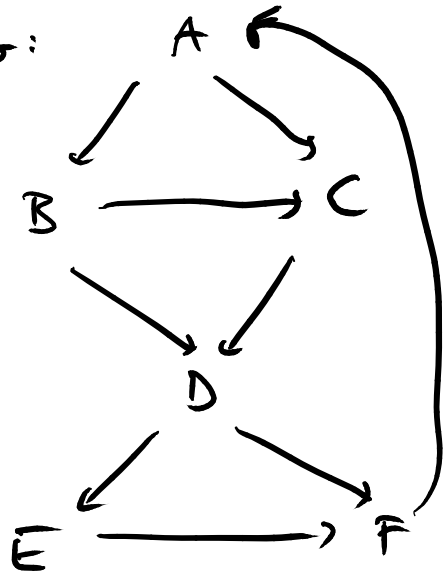
- **Step 1: Assign integers to edges**
  - Goal: Sum along a path yields unique number for path
  - Enough to achieve ”precise” goal
- **Step 2: Assign increment operations to edges**
  - Goal: Minimize additions along edges
  - Instrument subset of all edges
  - Assumes to know/estimate how frequent edges are executed

Spanning Tree

- **Given: Graph G**
- **Spanning tree T:**
  Undirected subgraph of G that is a tree and that contains all nodes of G
- **Chord edges: Edges in G but not in T**

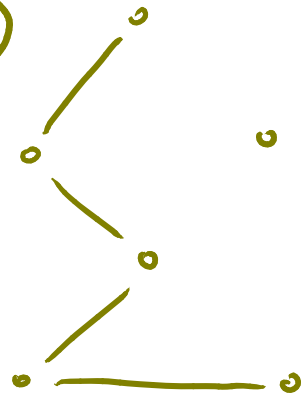
Example: Spanning Tree

G:

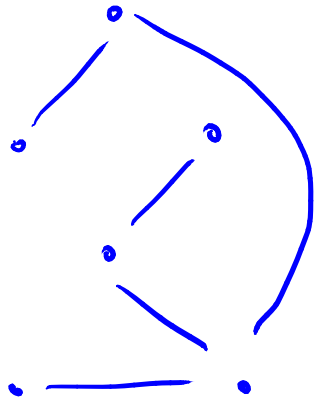


Which of these is a spanning tree of G?

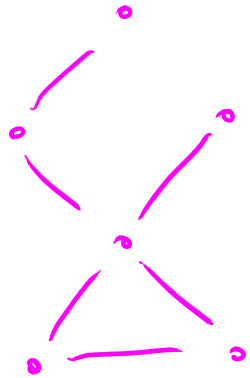
1.



2.

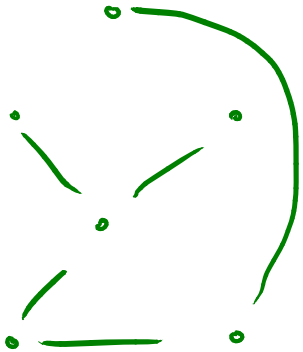


3.



4.

5.



Solution:

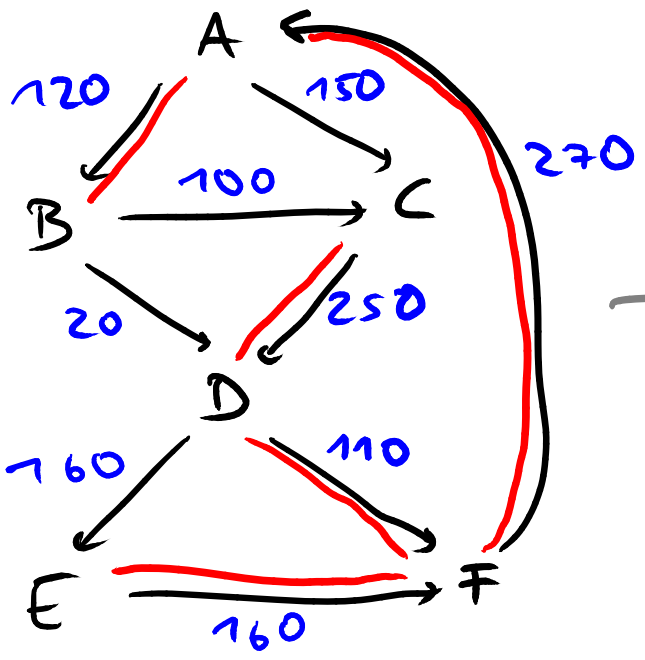
2, 4, 5

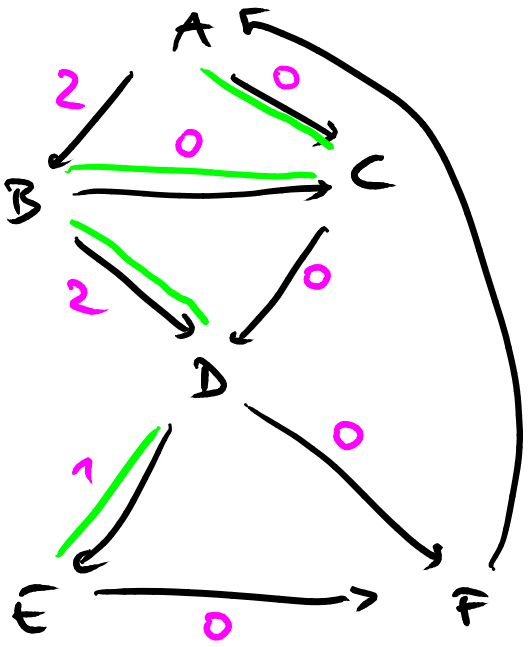
Increments for Edges

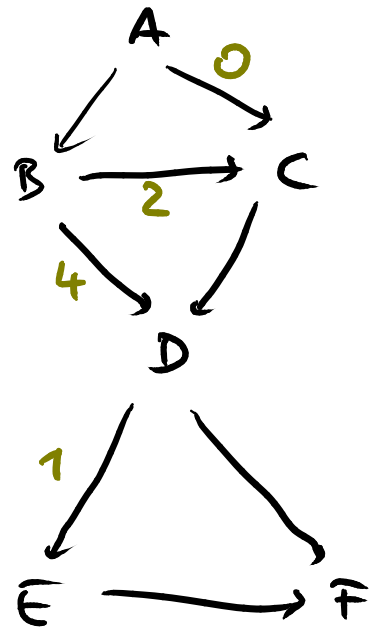
Goal: Increment sum at subset of edges

- **Choose spanning tree with maximum edge cost**
  - Cost of individual edges is assumed to be known
- **Compute increments at the chords of the spanning tree**
  - Based on existing event counting algorithm

Example: Increments for Edges







edge cost

most expensive spanning tree

Chord edges

non-minimal increments

minimal increments

path encoding

Instrumentation

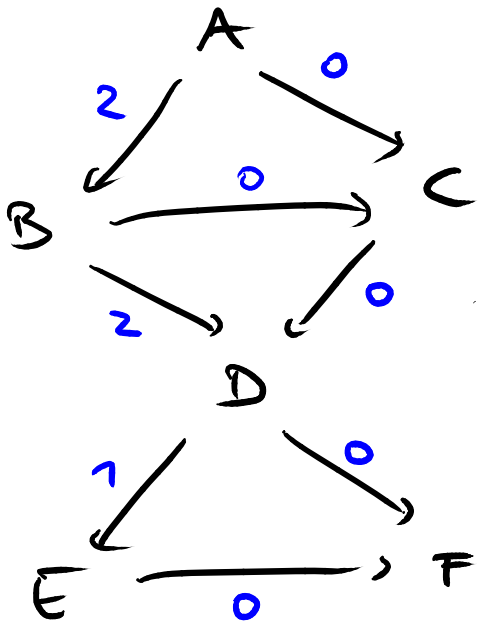
- **Basic idea**
  - Initialize sum at entry: `r=0`
  - Increment at edges: `r+=..`
  - At exit, increment counter for path: `count[r]++`
- **Optimization**
  - Initialize with incremented value, if first chord on path: `r=..`
  - Increment sum and counter for path, if last chord on path: `count[r+..]++`

Regenerating the Path

Knowing the sum r, how to determine the path?

- Use edge values from step 1 (”non-minimal increments”)
- Start at entry with $R = r$
- At branches, use edge with largest value v that is smaller than R and set $R -= v$

Example: Regenerating the Path



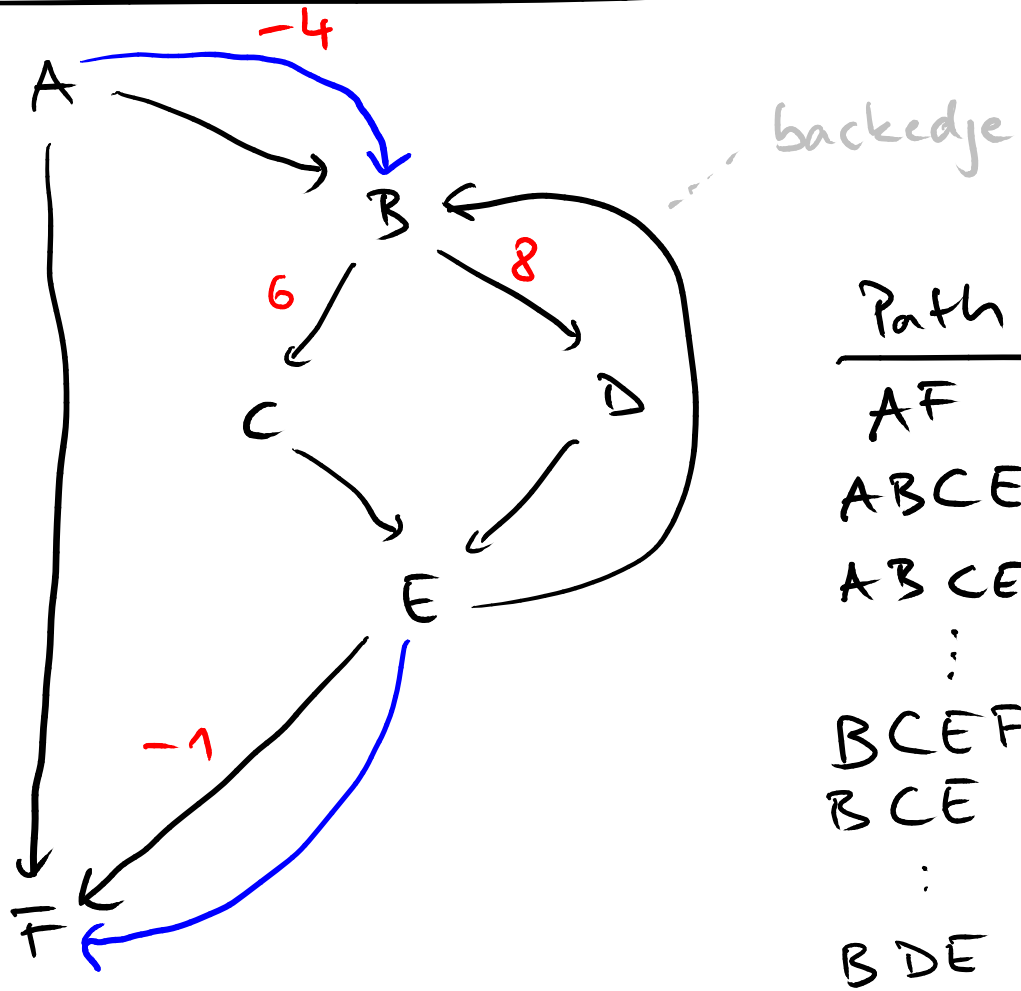
$r = 4$: A B D F

$r = 1$: A C D E F

Generalizing to Cyclic CGFs

- **For each backedge $n \rightarrow m$, add dummy edges**
  - $Entry \rightarrow m$
  - $n \rightarrow Exit$
- **Remove backedges and add DAG-based increments**
- **In addition, add instrumentation to each backedge**
  - `count[r]++; r=0`

Example: Generalizing



backedge

dummy edges

increments without backedge

Path	Encoding
AF	0
ABCEF	1
ABCE	2
⋮	
BCEF	5
BCE	6
⋮	
BDE	8

Conclusions

- **Ball-Larus profiling**:
  Beautiful algorithm with various applications
- **Path profiles also determine accurate edge and basic block profiles**
- **Can generalize to other graphs, e.g., to compute call graph profiles**