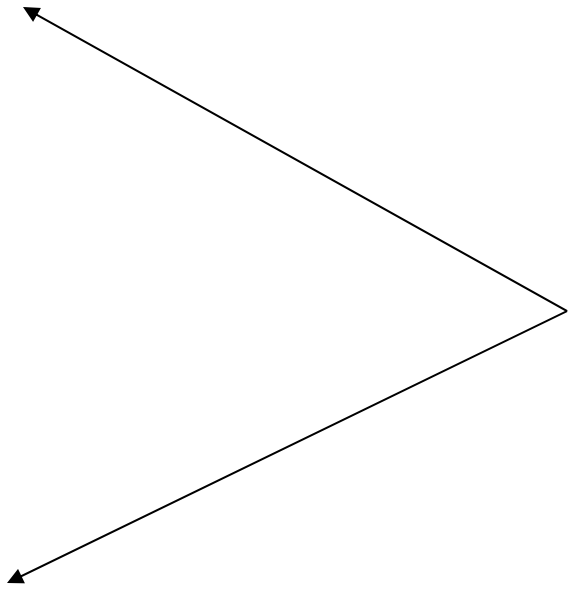


Profiling: Understanding Program Behavior - Where to Optimize?

Why profiling is useful?

- Things that static analysis can not do
 - Opportunities observed during program execution
 - Profiling can be used to identify new optimization opportunities
 - Opportunities involves performance trade-offs
 - Profiles can be used to carry out sophisticated cost-benefit analysis

Example

- If () then
 - $Z = x * y$
 - Else
 - ...
 - Endif
 - If () then
 - $W = x * y$
 - endif
- 
- Redundancy
- The diagram consists of two arrows pointing from a central point on the right towards the left. The top arrow points to the assignment $Z = x * y$ in the first if-then block. The bottom arrow points to the assignment $W = x * y$ in the second if-then block. This visualizes that both branches of the code perform the same calculation, which is a form of redundancy.

Example

- If () then
 - $Z = x * y$
- Else
 - ...
- Endif
- If () then
 - $W = x * y$
- Endif
- If () then
 - $T = Z = x * y$
- Else
 - $T = x * y$
 - ...
- Endif
- If () then
 - $W = T$
- Endif

Redundancy
removal

Example

- If () then
 - $Z = x * y$
- Else
 - ...
- Endif
- If () then
 - $W = x * y$
- endif

- If () then
 - $T = Z = x * y$
- Else
 - $T = x * y$
 - ...
- Endif
- If () then
 - $W = T$
- Endif

Redundancy
removal

- If () then
 - $T = Z = x * y$
- Else
 - $T =$
 $(y = 1) ? x : x * y$
 - ...
- Endif
- If () then
 - $W = T$
- Endif

Strength reduction

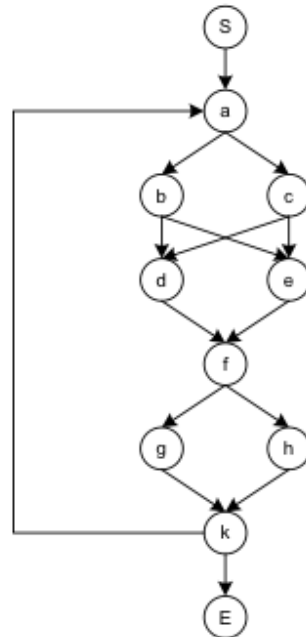
Profiling types

- Control flow profile
 - Captures a trace of the execution path taken by the program
- Value profile
 - Records values of operands of an instruction
- Address profile
 - Records stream of memory addresses that are referenced by the program

Control flow profile

- Node profile
 - Execution frequencies of basic blocks
- Edge profile
 - Execution frequencies of each edge in CFG
- Two-edge profile
 - Execution frequencies of each pair of consecutive edges
- Path profile
 - Execution frequencies of acyclic subpaths in CFG

Example



Control Flow Trace:
 $S(abdfgk)^{10}(acefhk)^{10}$
 $(abdfhk)^{10}(acefgk)^{10}$
 $(abefgk)^{50}(abefhk)^{10}E$

Node	Frequency
S	1
a	100
b	80
c	20
d	20
e	80
f	100
g	70
h	30
k	100
E	1

Edge	Frequency
Sa	1
ab	80
ac, ce	20
bd, df	20
be	60
ef	80
fg, gk	70
fh, hk	30
ka	99
kE	1

Two-Edge	Frequency
Sab	1
abd, bdf	20
abe, bef	60
ace, cef	20
dfg	10
dfh	10
efg	60
efh	20
fgk	70
fhk	30
gka	80
hka	20
kac	20
kab	79
hkE	1

Path	Frequency
Sabdfgk	1
abdfgk	9
abdfhk	10
abefgk	50
acefhk	10
acefgk	10
abefhk	9
abefhkE	1

Calculate path frequency

- abefgk
- Estimate based upon node profiles
 - 30-70
- Estimate based upon edge profiles
 - 30-60
- Estimate based upon two-edge profiles
 - 40-60
- Estimate based upon path profiles
 - 50

Overhead of profiling

- Node profiles
 - Only one counter for each control dependence region
- Edge profiles
 - Two edges may have identical frequencies
 - The frequency of an edge might be deducted from frequencies of other edges

Overhead of profiling

- Two-edge and path profiles
 - Expensive
 - Assign number to each edge so that sums of edge number are unique for every path
 - Reducing number of instrumenting points using optimizations in collecting edge profiles

Value profiles

- Collect only the most frequently appearing values
- Collect for only interesting instructions
 - Using control flow profile

Code:

I1: load R3, 0(R4)
I2: R2 ← R3 & 0xff

Value profile:

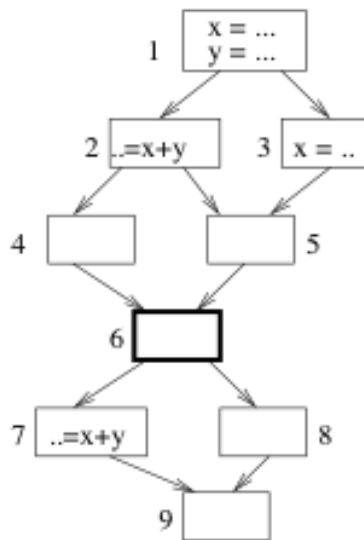
(instruction, register)	profiles (value,freq)
(I1,R3)	(0xb8d003400,10) ...
...	...
(I1,R2)	(0,1000)
(I2,R3)	(0,100),(0x8900,200) ..., (0x2900,100)

Address profile

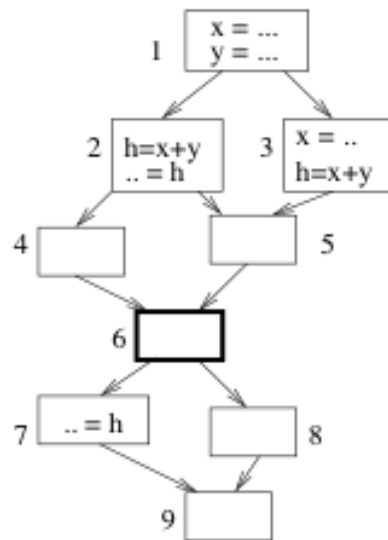
- Addresses referenced by a program
- Usually used for data layout and placement optimizations
- Complete address traces are large and expensive to get
 - Compress address traces
 - Collect only address streams that are accessed together

Profiling enables optimizations

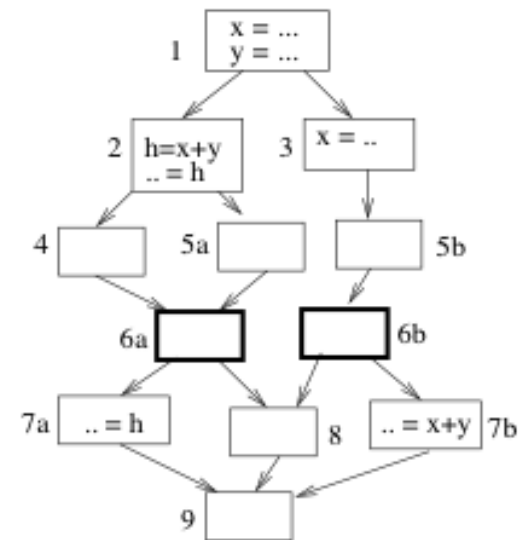
- Partial Redundancy Elimination (PRE)



(a) Code with Partial Redundancy.

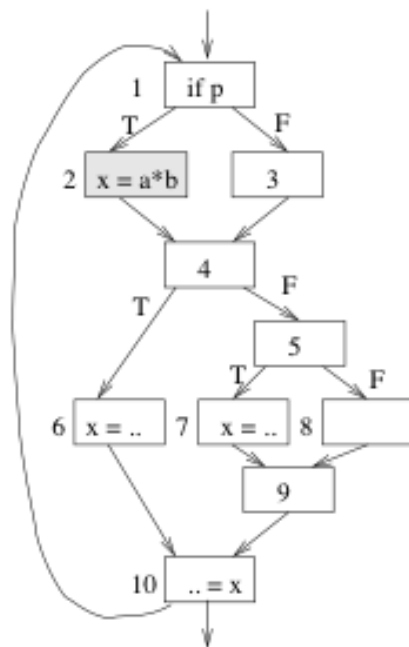


(b) PRE using Speculative Code Motion.

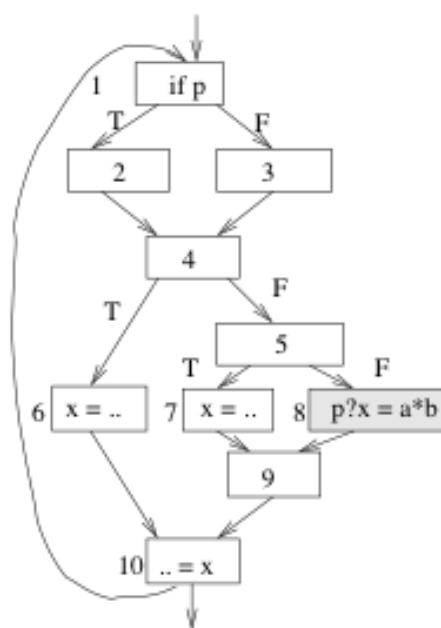


(c) PRE using Control Flow Restructuring.

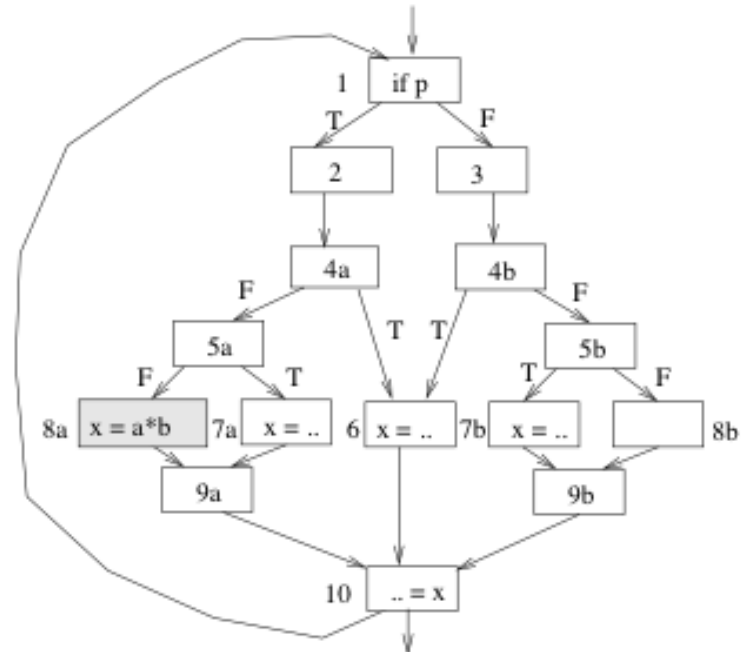
Partial dead code elimination



(d) Partially Dead Code.

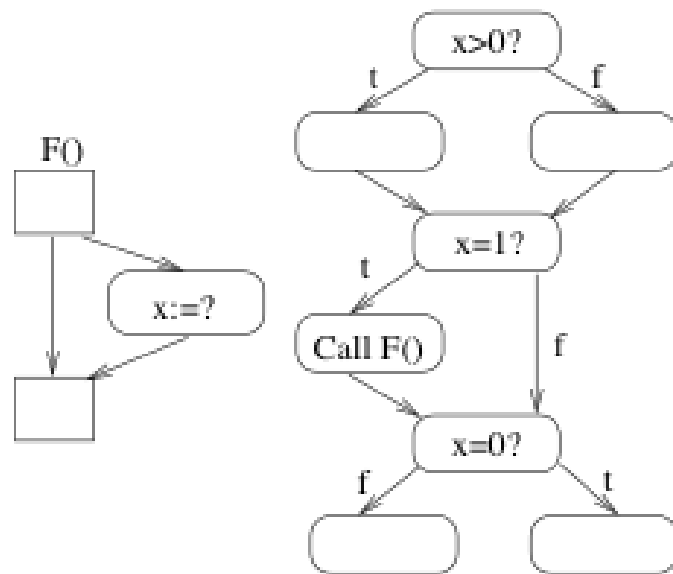


(e) PDE using Predicated Code Motion.

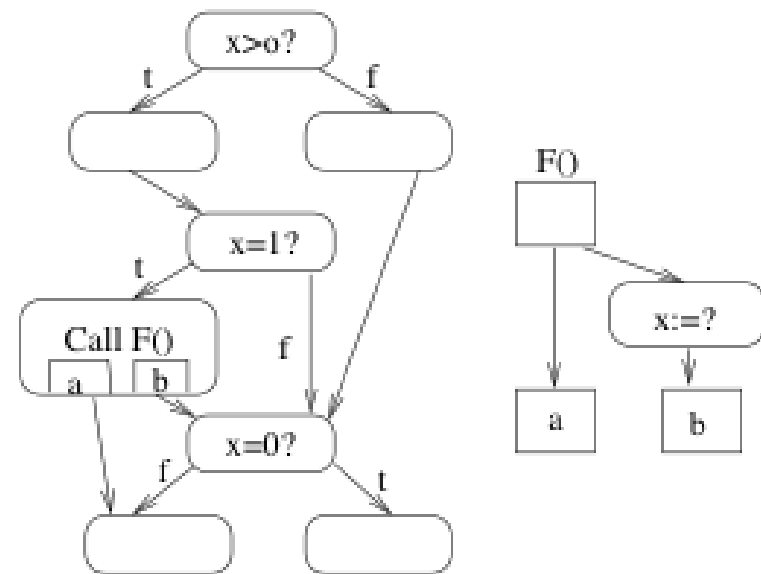


(f) PDE using Control Flow Restructuring.

Conditional branch elimination

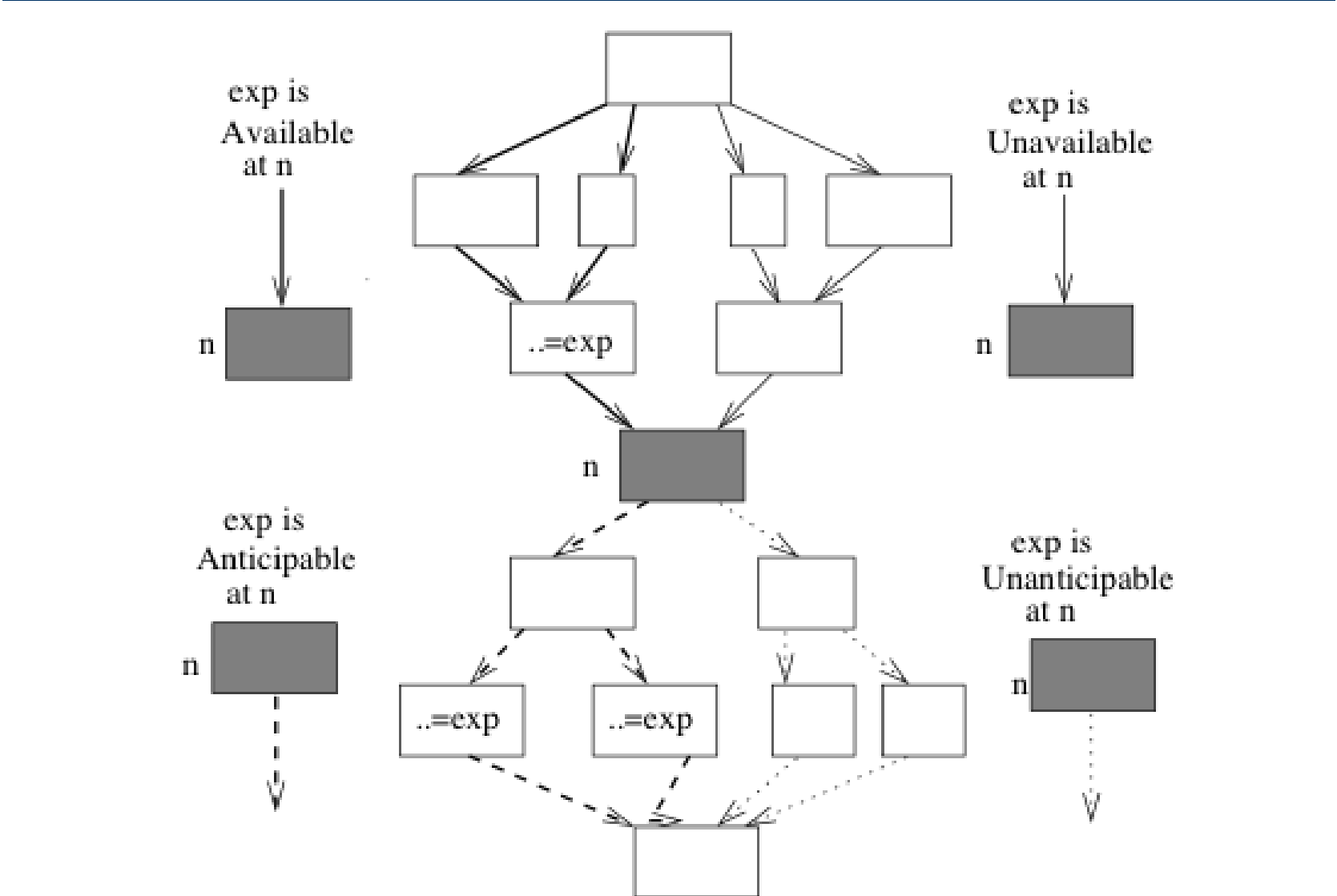


(g) Partially Redundant Conditional Branch.



(h) Redundancy Removal using Control Flow Restructuring.

Profitability of PRE

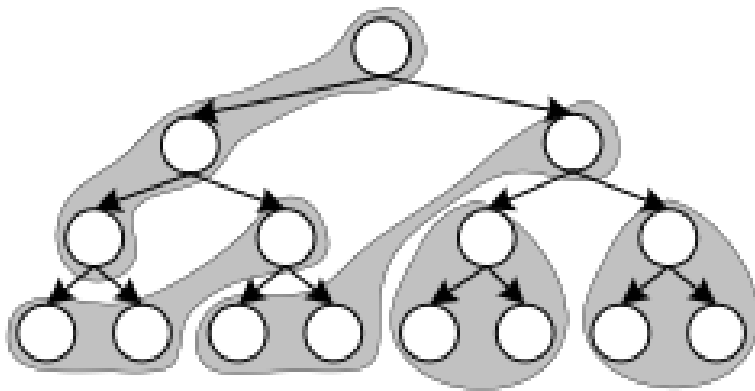


Profile guided memory optimization

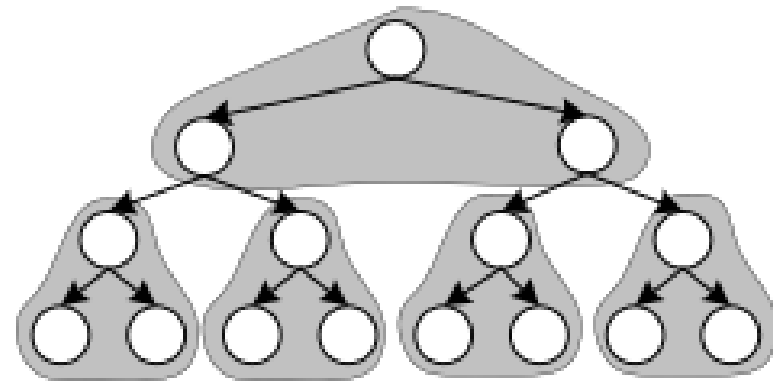
- Improve tolerance of cache miss penalty
- Reduce the number of cache misses
 - Object placement
 - Object layout
 - Object layout and placement
 - Object compression

Object placement

- Object migration
 - Dynamically relocate objects
- Memory clustering



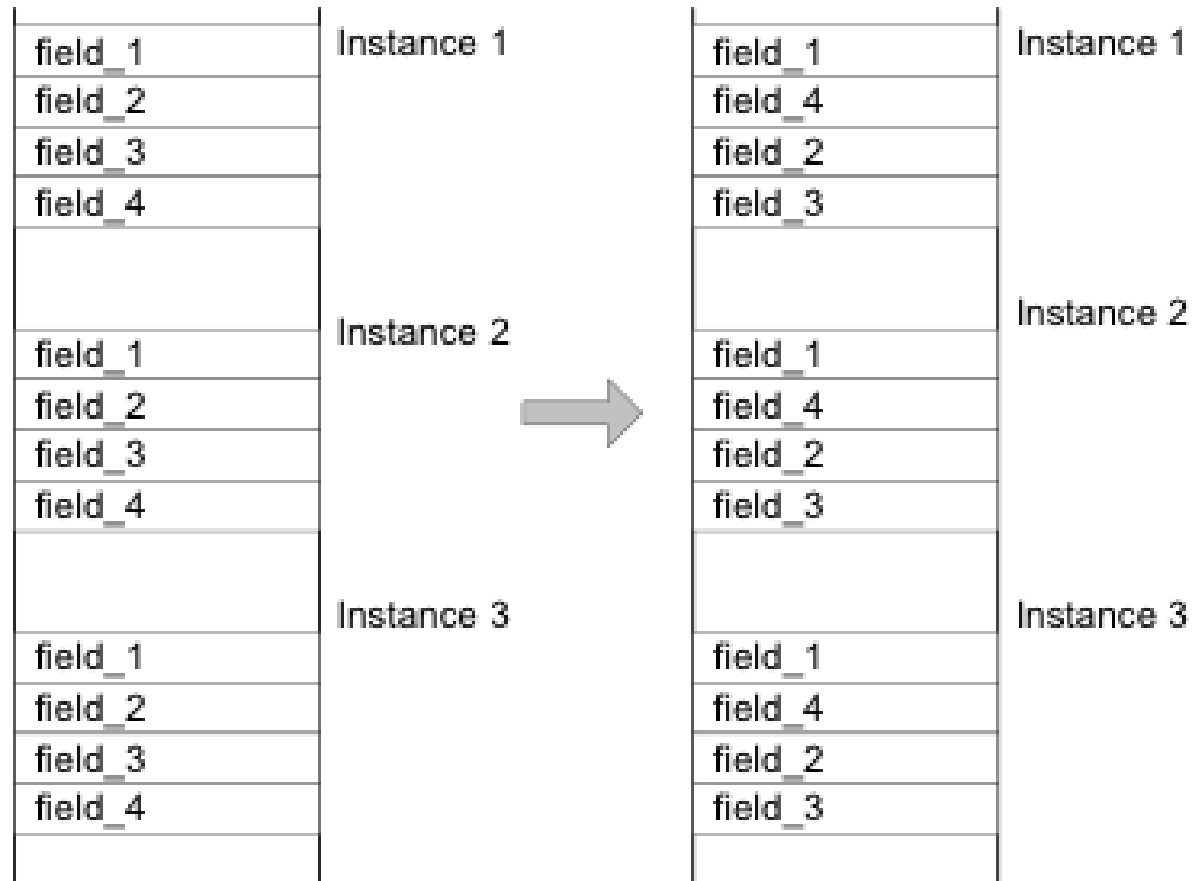
(a) Unoptimized



(b) Optimized

Object layout

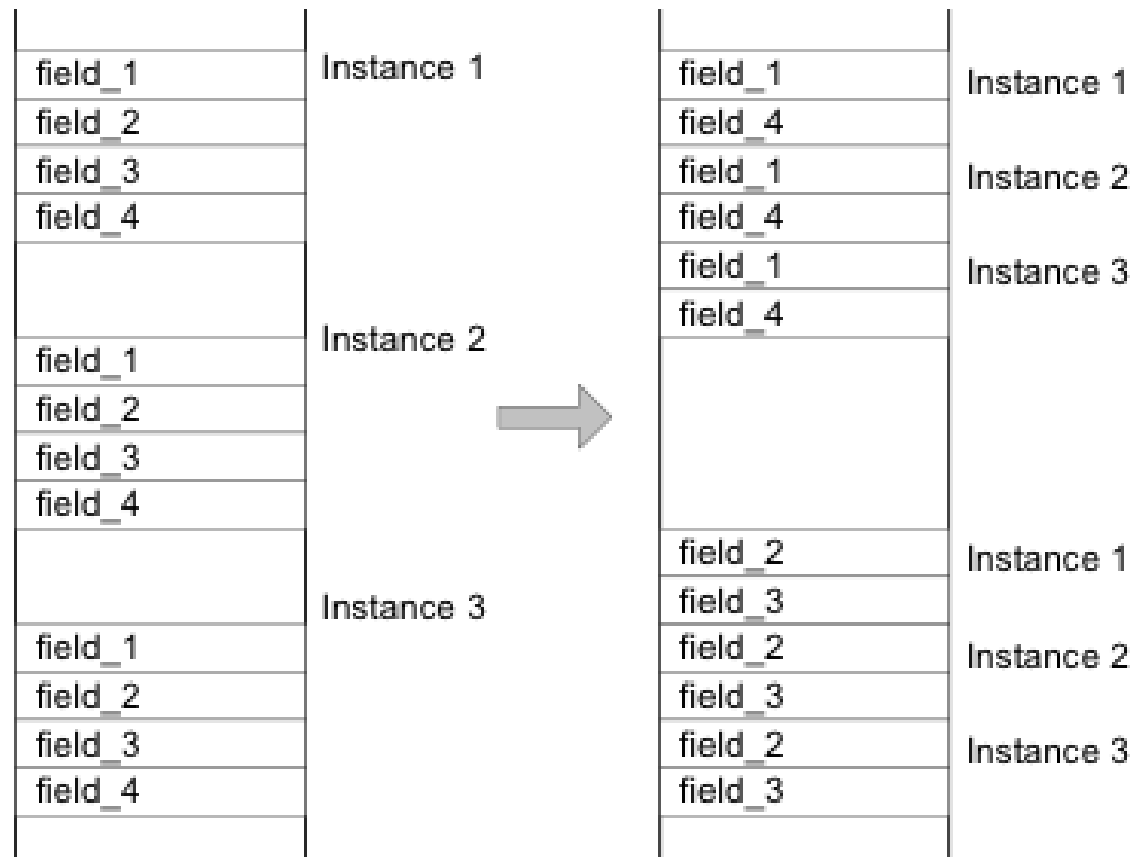
```
struct node {  
    int field_1;  
    int field_2;  
    int field_3;  
    int field_4;  
};
```



Object layout and placement

- Reorganize fields storage across objects

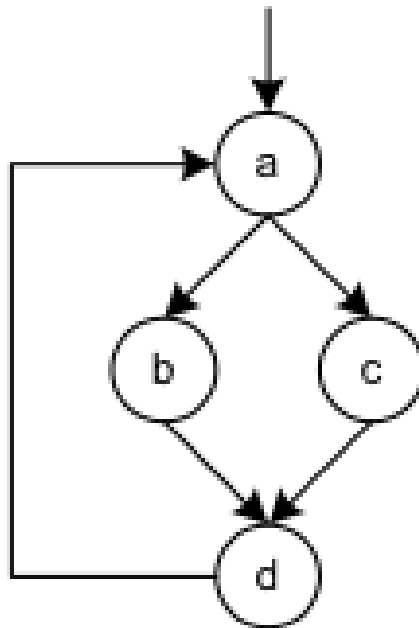
```
struct node {  
    int field_1;  
    int field_2;  
    int field_3;  
    int field_4;  
};
```



Object compression

- Identical values referenced by multiple objects
- Partial compressible objects
 - Only partial bits are identical

Profile guided code layout



control flow trace:
 $(abd)^{10}(acd)$

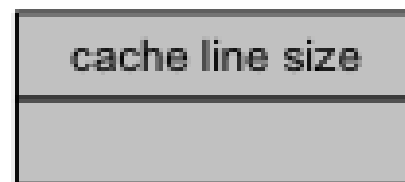
control flow profiles:
abd: 10 times
acd: 1 time

cache(2-entry full associative)

original layout

a	
b	c
d	

33 cache misses

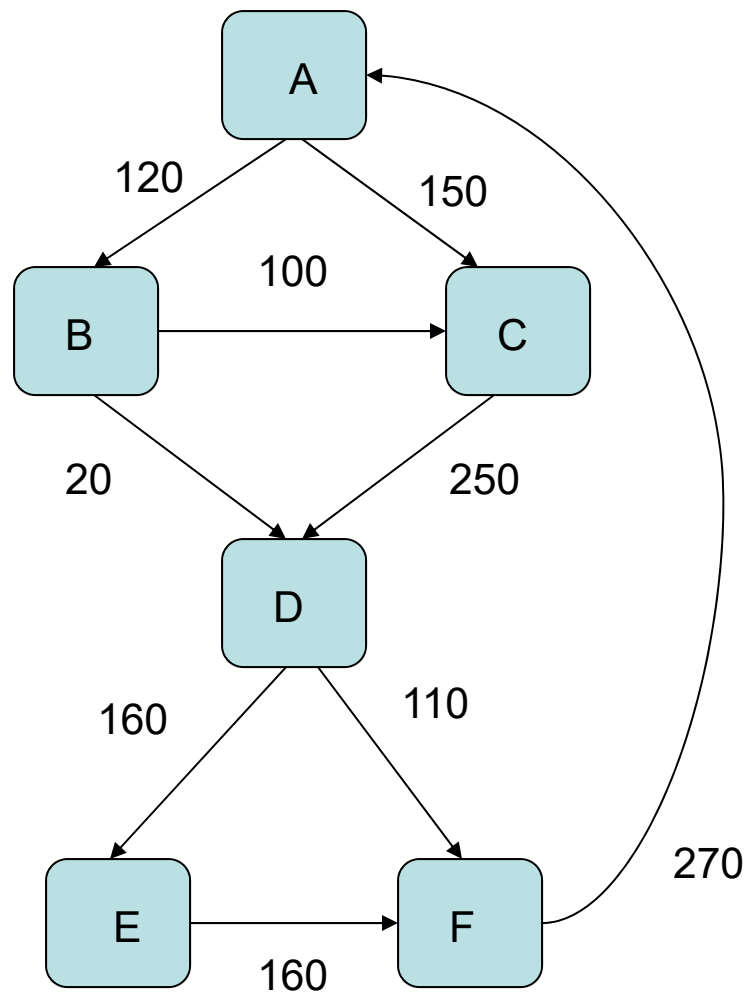


optimized layout

a	
b	d
c	

4 cache misses

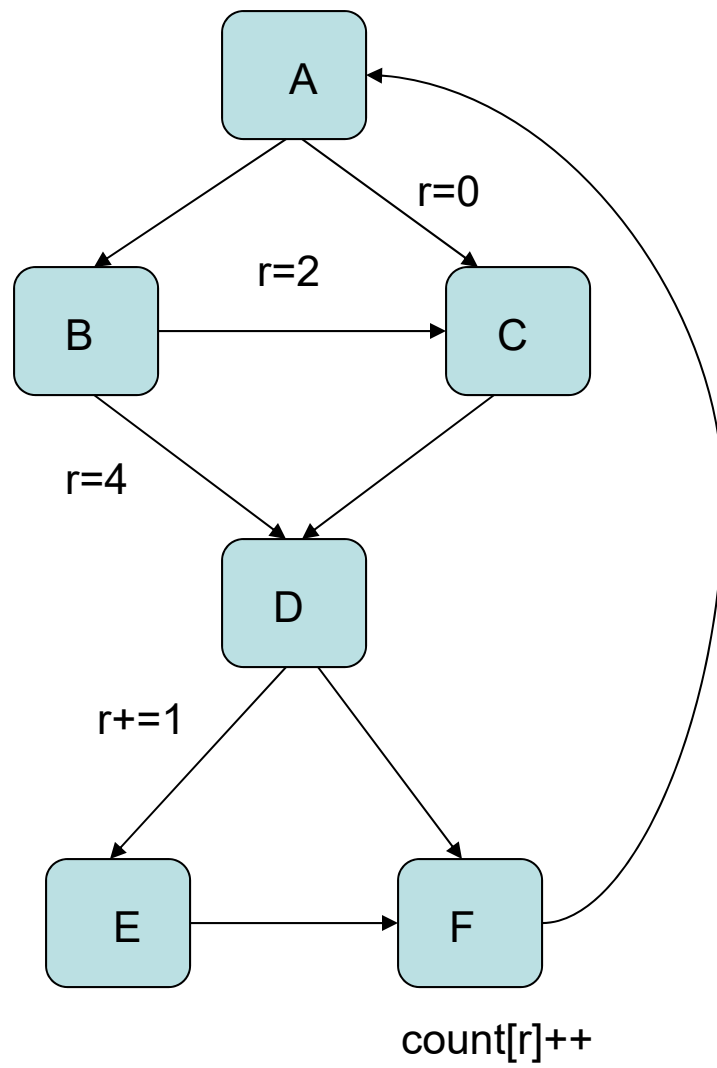
Efficient Path Profiling



Path	Prof1	Prof2
ACDF	90	110
ACDEF	60	40
ABCDF	0	0
ABCDEF	100	100
ABDF	20	0
ABDEF	0	20

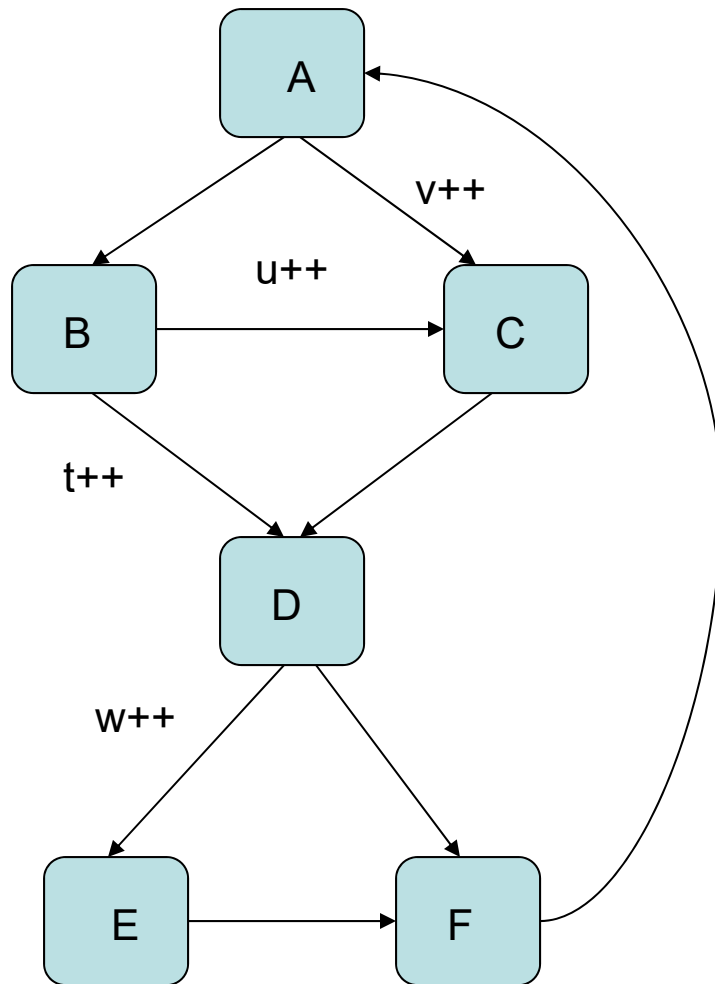
Goal of path profiling

- Each path from entry to exit produces a unique state at the exit



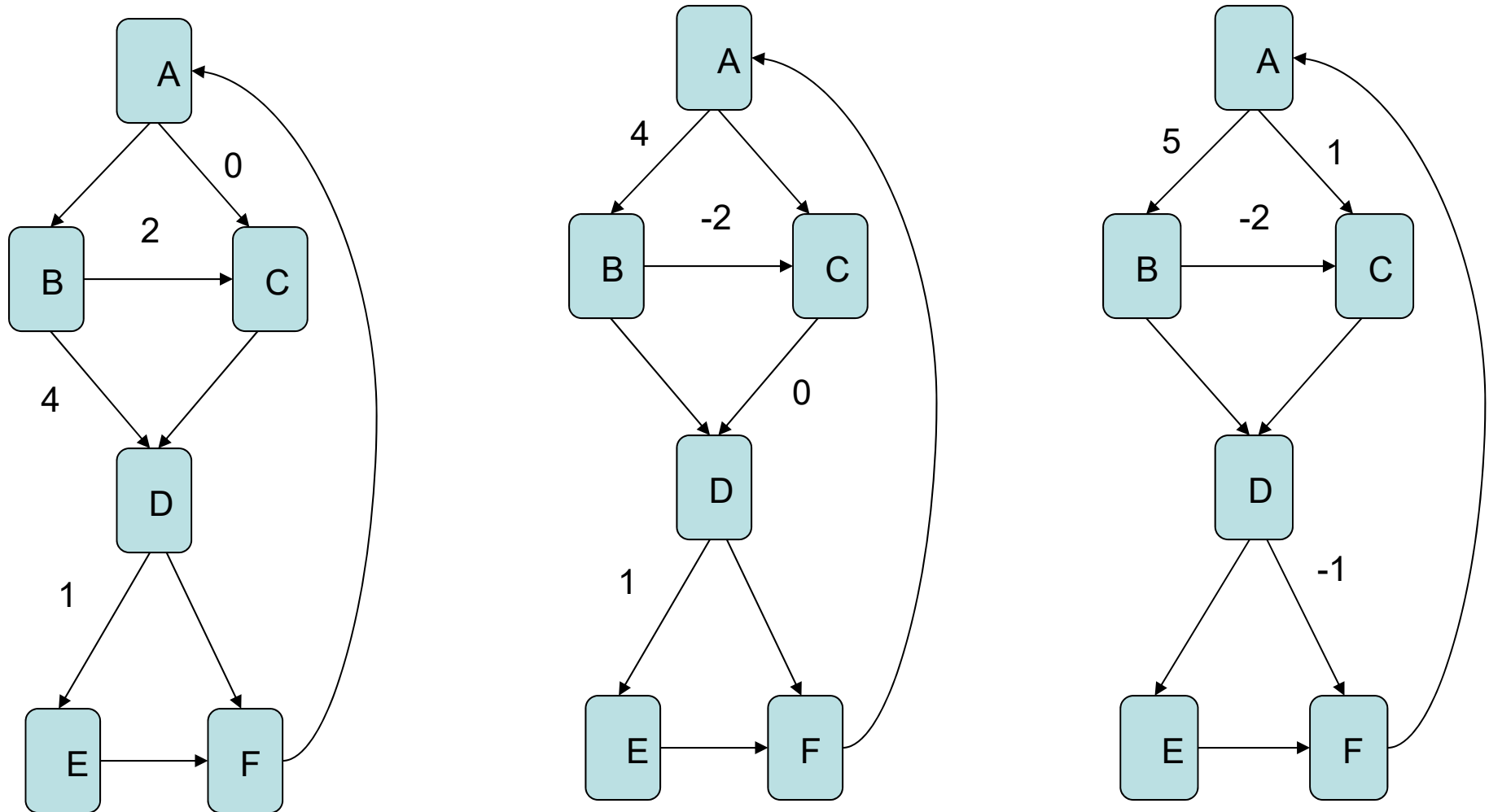
Path	Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Edge profiling



- $C \rightarrow D = u + v$
- $D \rightarrow F = t + u + v - w$
- $E \rightarrow F = w$
- $A \rightarrow B = t + u$
- $F \rightarrow A = t + u + v$

Path profiling is not unique

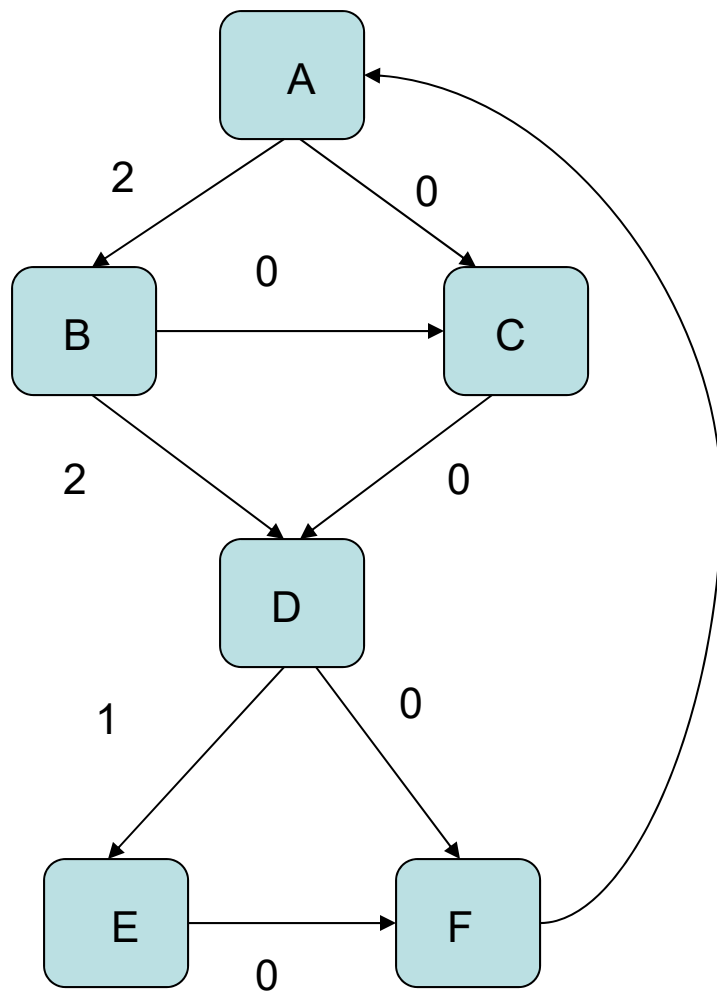


Basic steps to profile a DAG

- Assign integer values to edges such that no two paths compute the same path sum.
- Use a spanning tree to select edges to instrument and compute the appropriate increment for each instrumented edge.
- Select appropriate instrumentation.
- After collecting the run-time profile, derive the executed paths.

Step 1: assign increments

- Visit vertex v in reverse topological order
 - if v is leaf
 - $\text{NumPaths}(v) = 1$;
 - else
 - $\text{NumPaths}(v) = 0$;
 - for each edge $e = v \rightarrow w$
 - $\text{Val}(e) = \text{NumPaths}(v)$
 - $\text{NumPaths}(v) = \text{NumPaths}(v) + \text{NumPaths}(w)$



Vertex	NumPaths
A	6
B	4
C	2
D	2
E	1
F	1

Step 2: select edge subset to instrument

- Theorem: Let G be a control-flow graph, let T be a spanning tree of G , and Let D be a directed (not necessarily simple) cycle in G . For all edges e :

$$\#(D, e) = \sum_{\substack{\forall f \in E - T \\ s.t. e \in C(f)}} \#(D, f) Dir(C(f), e, f)$$

Goal of subset selection

- Find a subset of edges F and for each edge f in F a constant $\text{Increment}(f)$ such that for any directed cycle EX

$$\sum_{\forall f \in F} \#(EX, f) \text{Increment}(f) = \sum_{\forall e \in E} \#(EX, f) \text{Events}(e)$$

- For any incomplete execution path IX that ends with edge g

$$\left(\sum_{\forall f \in F} \#(IX, f) \text{Increment}(f) \right) + \text{QueryInc}(g) = \sum_{\forall e \in E} \#(IX, f) \text{Events}(e)$$

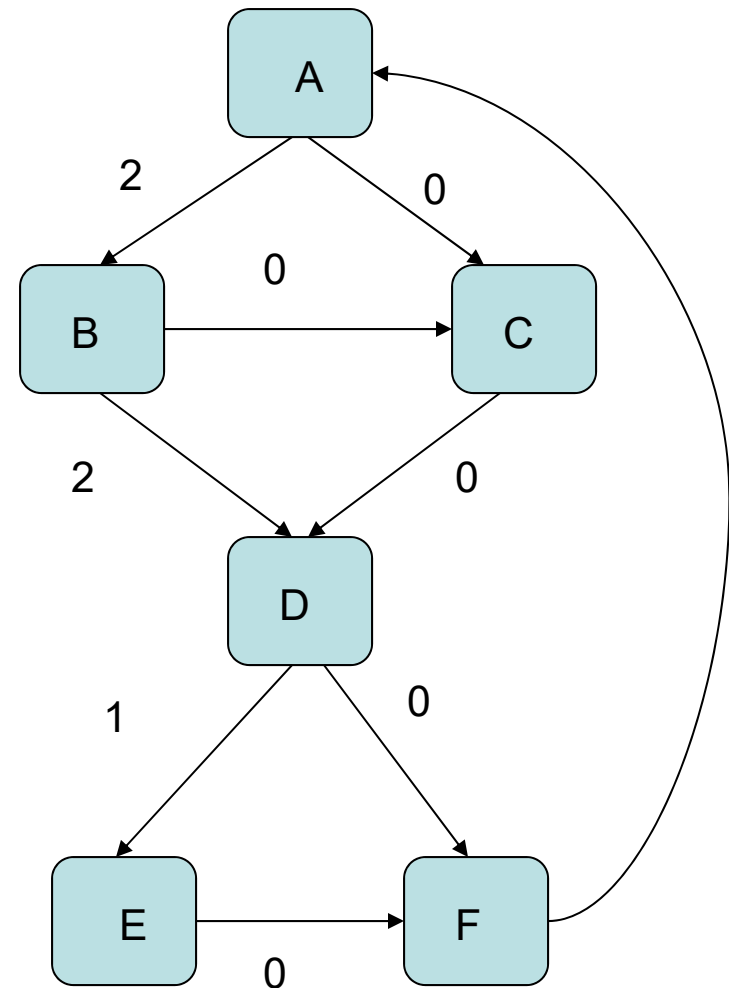
- 
- F is the set of *chord* in the graph

$$Increment(f) = \sum_{\forall e \in C(f)} Events(e) Dir(C(f), e, f)$$

- Compute *QueryInc*
 - $g' = \text{tgt}(g) \rightarrow \text{root}$ --- root is the root of spanning tree
 - $\text{QueryInc}(g) = \text{Increment}(g')$

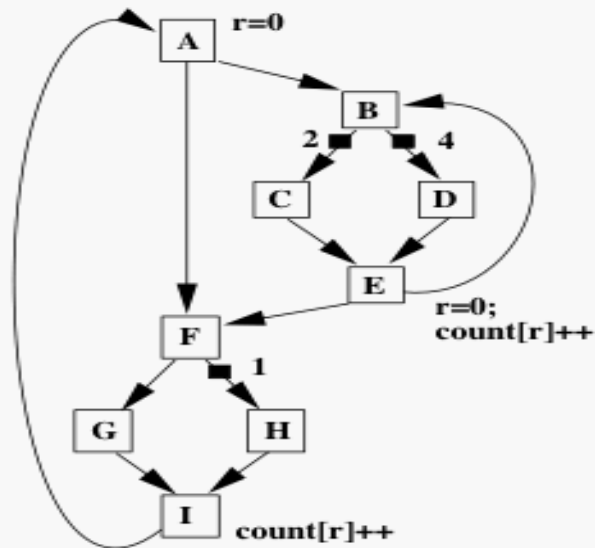
Regenerate a path

- Start from *Entry*, go along edges with largest valid *Val*

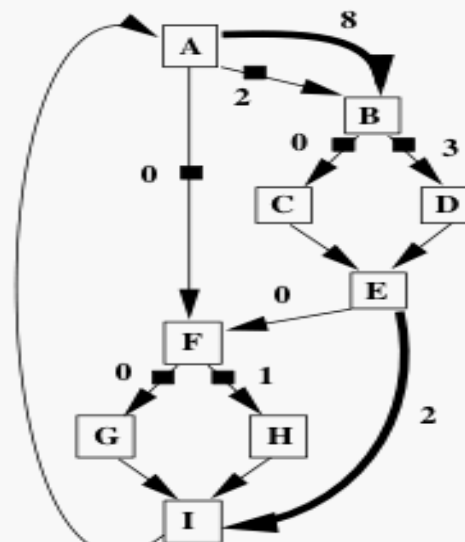


Profiling arbitrary CFG

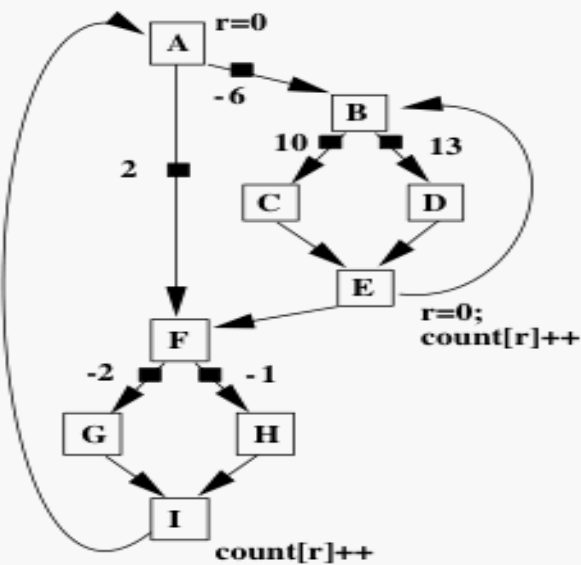
- Transform a cyclic CFG into a DAG.
 - Back edge $x \rightarrow y$
 - entry $\rightarrow y$, $x \rightarrow \text{exit}$



(a)



(b)



(c)

Path	Path Sum
<i>AFGI</i>	0
<i>AFHI</i>	1
<i>ABCEFGI</i>	2
<i>ABCEFHI</i>	3
<i>ABDEFGI</i>	5
<i>ABDEFHI</i>	6
<i>ABCE</i>	4
<i>ABDE</i>	7
<i>BCE</i>	10
<i>BDE</i>	13
<i>BCEFGI</i>	8
<i>BCEFHI</i>	9
<i>BDEFGI</i>	11
<i>BDEFHI</i>	12