Dynamic Analysis

Rupesh Nasre.

CS6843 Program Analysis
IIT Madras
July 2025

Learning Outcomes

- Recall applications of dynamic analysis
- Compare static versus dynamic analysis
- Given a CFG, perform efficient path profiling
- Optimize instrumentation using spanning tree method

Outline

- Applications of dynamic analysis
  - Limitations of static analysis
  - Trade-offs
- Profiling techniques
- Finding invariants
  - Equality
  - Affine
- Dynamic type inferencing

Applications

- Bug finding (testing)
- Data race detection
- Identifying security vulnerabilities
- Improved precision of static analysis
- Input-dependent analysis

Limitations of Static Analysis

- Reduced precision: Over-approximations
- Cannot perform input-dependent analysis

Static versus Dynamic

- Sound
- Imprecise
- Input-oblivious

- Incomplete
- Precise
- Input-dependent

- Choosing between static and dynamic analysis often requires a trade-off between soundness and precision.
- Current trend is to combine the two techniques to get better precision at improved scalability.

Profiling

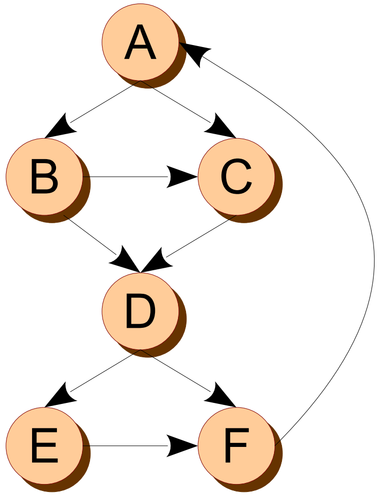
- Profiling is a method of collecting information of interest during program execution.
- The information is often useful to find hot-spots in the program.
- Examples
  - Number of times an instruction is executed
  - Number of page faults
  - Number of cache hits
  - Total memory used
  - ...

Profiling

- Intrusive: inserts instructions in the program (source, IR, assembly) statically, which get executed at runtime
  - File log
  - Memory locations pointed to by a pointer
  - Execution time of a function
- Non-intrusive: the program is unaltered; uses external means to profile
  - Hardware counters
  - Program execution time

Path Profiling

- Consider a program with an entry node and an exit node. There are several execution paths (traces) that the program takes from entry to exit.
- The task is to find the frequency of execution of each path.



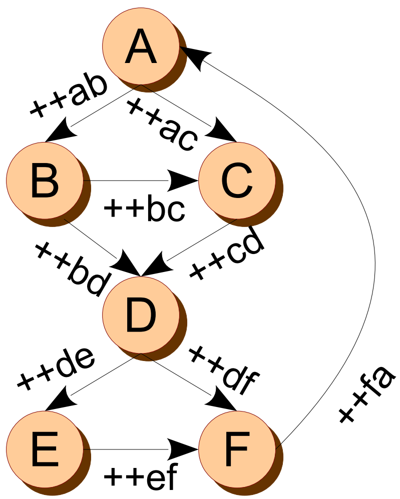
Path	Frequency
ACDF	90
ACDEF	60
ABCDF	0
ABCDEF	100
ABDF	20
ABDEF	0

Path Profiling

- Naïve path profiling is expensive: instrumenting each path may lead to exponential blow up in computation and storage.
- This can lead to unacceptable program slowdown.

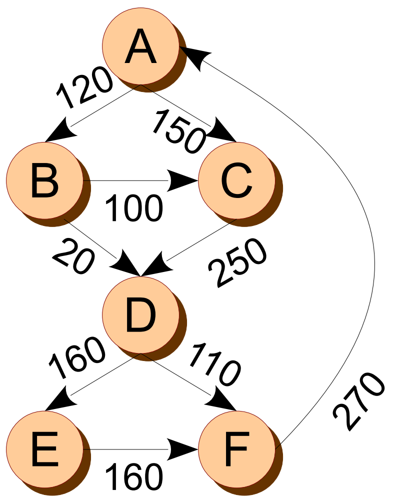
Edge Profiling

- Path profile is approximated as an edge profile
- The frequency of each edge is calculated – which is used to find the path frequency

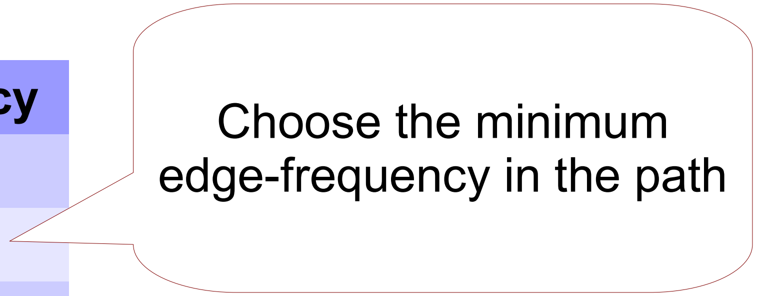


Edge Profiling

- Path profile is approximated as an edge profile
- The frequency of each edge is calculated – which is used to find the path frequency



Path	Frequency
ACDF	110
ACDEF	150
ABCDF	100
ABCDEF	100
ABDF	20
ABDEF	20

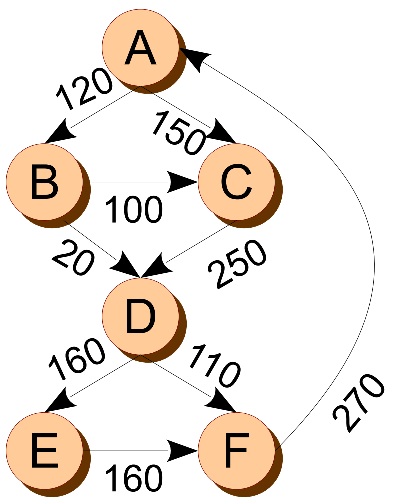


Path vs. Edge Profiling

- Path profile is approximated as an edge profile
- The frequency of each edge is calculated – which is used to find the path frequency

- **Can this instrumentation be optimized?**
- **Can we have better precision?**

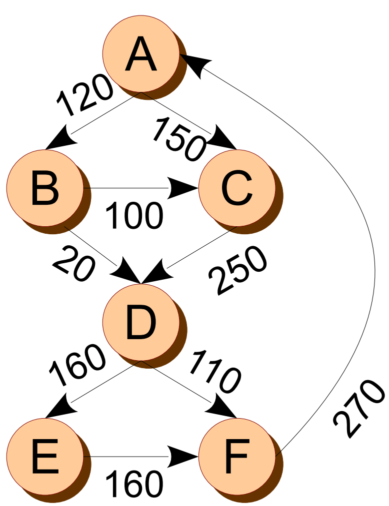
Is this a viable profile?



Path	Path Frequency (actual)	Path Frequency (estimated)
ACDF	90	110
ACDEF	60	150
ABCDF	0	100
ABCDEF	100	100
ABDF	20	20
ABDEF	0	20

Edge Profiling

- Edge profile may not always be a good indicator of a path profile.
- The same profile may refer to different actual path profiles.

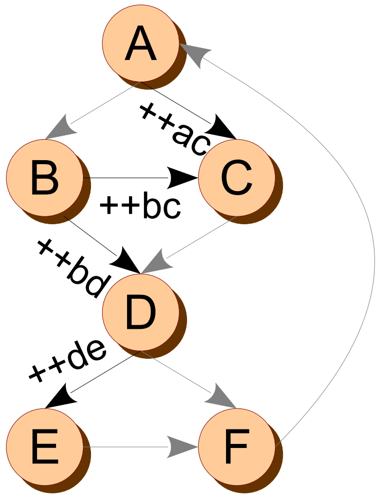


Path	Frequency	Actual Freq.	Actual Freq. 2
ACDF	110	90	110
ACDEF	**150**	**60**	**40**
ABCDF	100	0	0
ABCDEF	100	100	100
ABDF	20	20	0
ABDEF	20	0	20

Efficient Edge Profiling

- Observation: We do not need to instrument every edge.
- How to find a minimal, low-cost set of edges to instrument?
- Use a spanning tree (instrument non-st edges):
  - reduced instrumentation along paths,
  - not all edges carry instrumentation

Classwork: Find counts for uninstrumented edges.



Path	Frequency
C → D	ac + bc
D → F	ac + bc + bd - de
E → F	de
A → B	bc + bd
F → A	ac + bc + bd

Needs a unique variable along each instrumented edge.

Proof of Why Instrumenting non-ST Edges Suffices

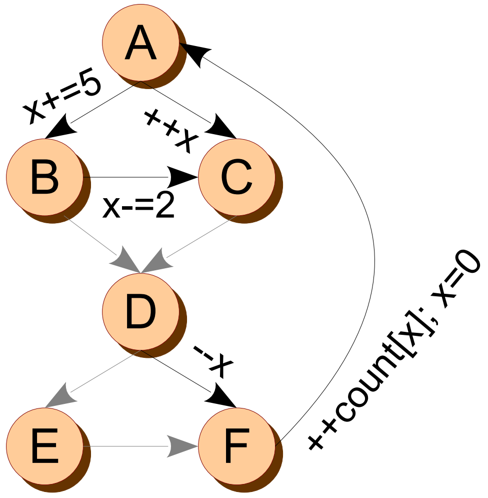
- **Vaishnavi's Proof:** For a leaf node in the ST, all but one incoming edge are instrumented. Thus, we can know the non-instrumented edge's count via flow-conservation. This can be repeated for the whole ST.
- **Sahil's Approach:** Writing these flow-conservation equations with ST and non-ST counters gives a set of linear equations. If we can show that the equations can be simplified with only the non-ST counters, that will be a proof. This relates to the rank of the matrix w.r.t. only the non-ST counters.

Story So Far...

- We want path profile (most precise).
  - But it is expensive to compute and store.
- So we approximated it with edge profile.
  - But that is also inefficient.
- So we performed efficient edge profiling.
  - It is better, but can be improved.
  - Also, what we want is path profile.
- Can we perform efficient path profiling?
  - Apply the learning from edge profiling to develop a technique for efficient path profiling?

Efficient Path Profiling

- Single index variable across all paths
- Path linearization: Unique (and consecutive) path numbering, which enables indexing
- Most hardware support registers, fast increment and indexing



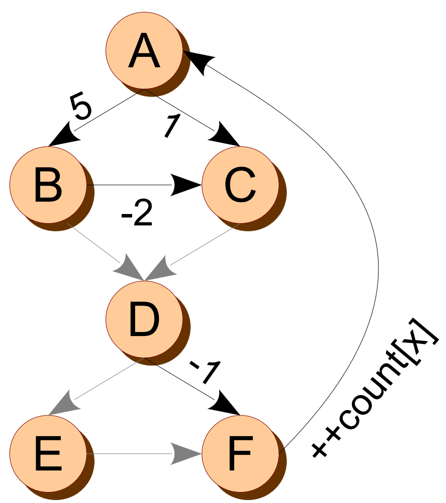
Path	x
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

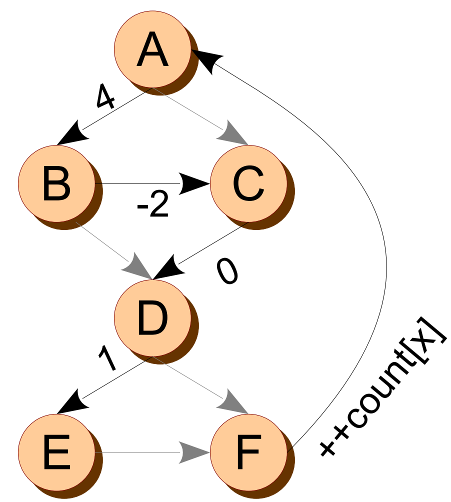
Classwork: Prove that such a path numbering is unique.

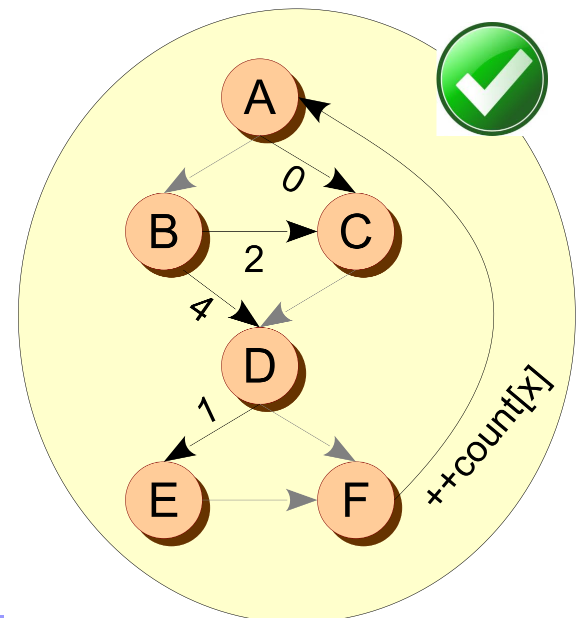
Check the value of x for each path.

Efficient Path Profiling

- Path numbering is not unique







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

In all the above cases, the path numbering is the same, number of instrumented edges (5) is the same

So, which instrumentation should we choose?

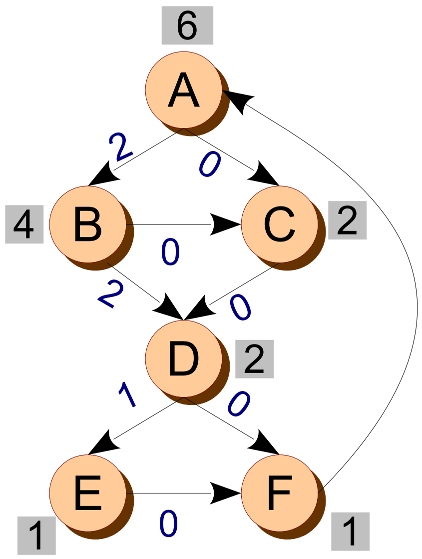
Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

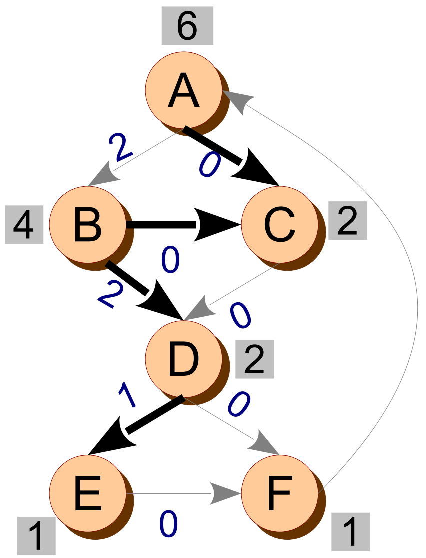
```
NumPaths(node) = 0
NumPaths(leaf) = 1
In reverse topological order
    For each edge v → w {
        Val(v → w) = NumPaths(v)
        NumPaths(v) += NumPaths(w)
    }
```



Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

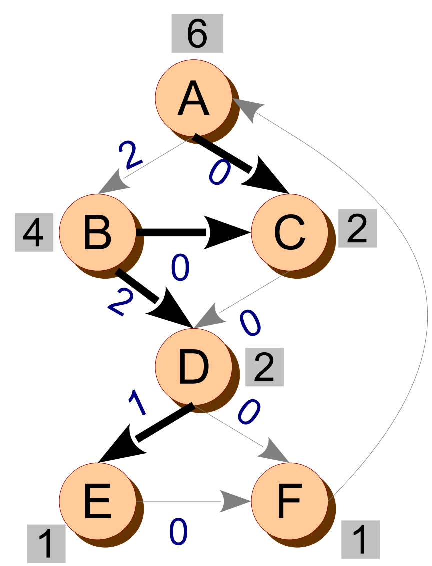
- Find a spanning tree.
- Find **chord** (non-ST) edges.
- For each chord, find fundamental cycle.

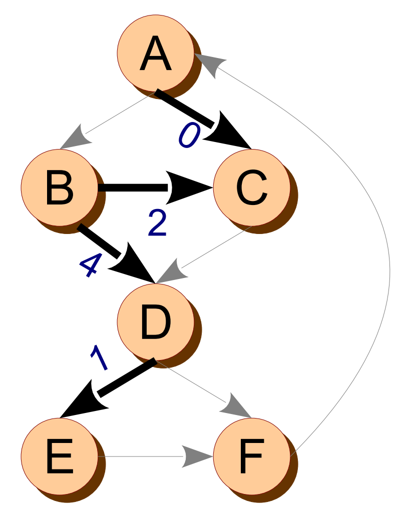


Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Chord AC: cycle ACDF : 0
Chord BC: cycle ABCDF : 2
Chord BD: cycle ABDF : 4
Chord DE: cycle DEF : 1





Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Prelude: Allocate and initialize the array of counters

Postlude: Write the array to permanent storage

Main:

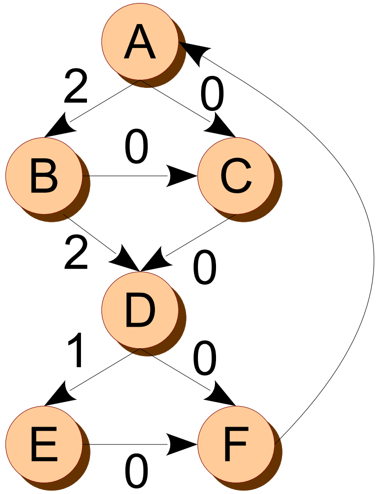
- Initialize path register r in the entry vertex
- Increment path memory counter in the exit vertex
- Optimizations

Efficient Path Profiling

1. Assign integer values to edges such that no two paths compute the same path-sum.
2. Use a spanning tree to select edges to instrument and compute the appropriate increment.
3. Select appropriate instrumentation.
4. After collecting the run-time profile, derive the execution paths.

Path Regeneration

- Path id → Path mapping?
- Traverse the CFG with the number (as the budget) and greedily exhaust the number (e.g., check 3).



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

Classwork

Find the instrumentation for the following CFG such that

- each path receives a unique number
- each path id belongs to [0..P-1] where P is the number of paths