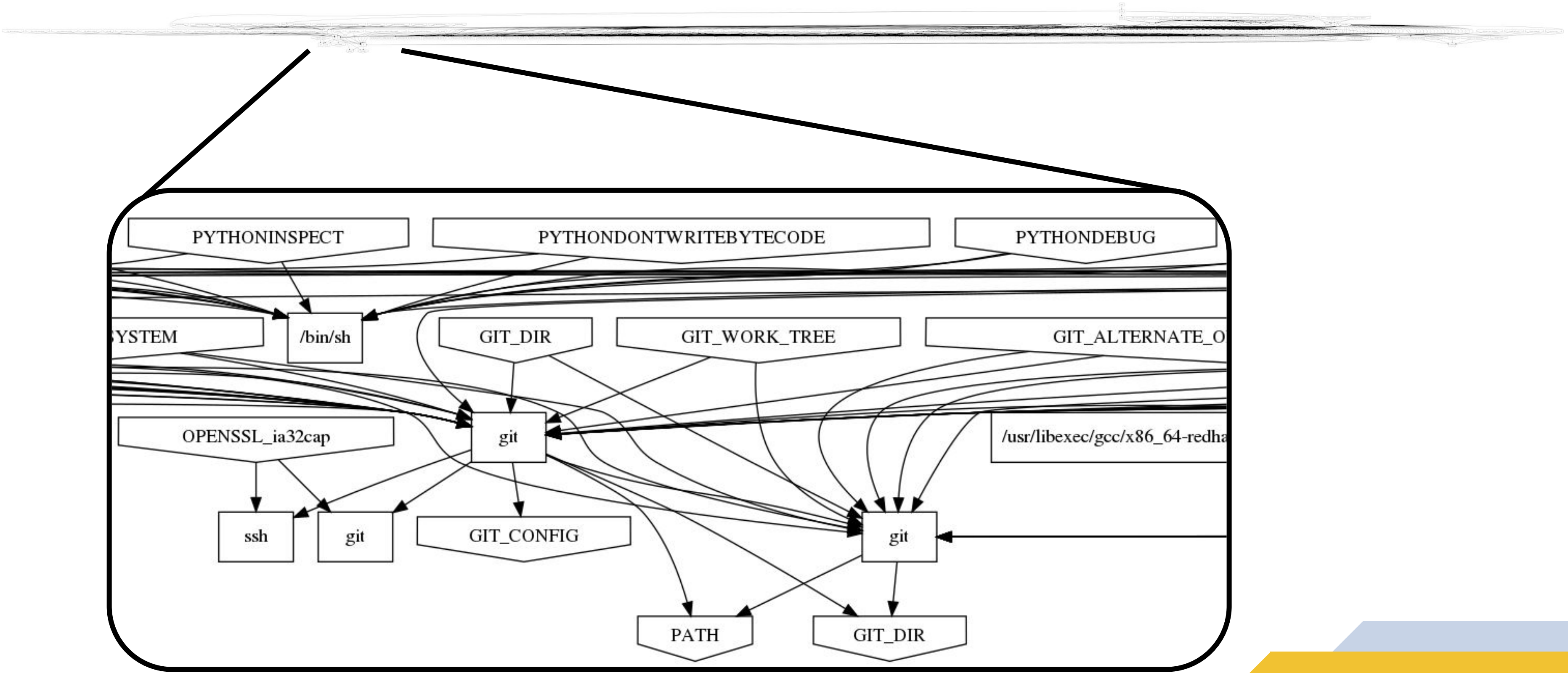
UNIVERSITY OF NOTRE DAME

Depth-First Search and Its Use Case in Distributed Systems Debugging

Nate Kremer-Herman


PYTHONINSPECT
PYTHONDONTWRITEBYTECODE
PYTHONDEBUG
YSTEM
/bin/sh
GIT_DIR
GIT_WORK_TREE
GIT_ALTERNATE_O
OPENSSL_ia32cap
git
/usr/libexec/gcc/x86_64-redha
ssh
git
GIT_CONFIG
git
PATH
GIT_DIR

Depth-first search kernel

Given a large graph.

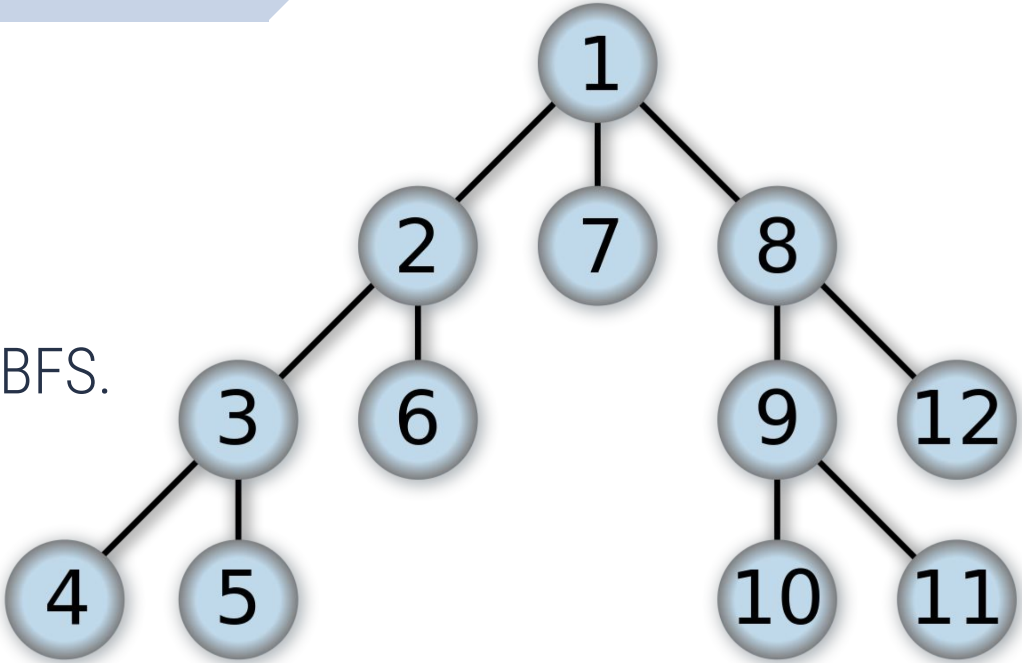
Start at a root node.

Find all reachable vertices.

Measured in TEPS, just like BFS.

Worst case performance:

$O(|V| + |E|)$



Iterative pseudocode

```
procedure DFS-iterative(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.pop()
        if v is not labeled as discovered:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Implementation techniques

- Implemented in Perl
  - Regex matching
  - Data structures
- Used the iterative algorithm
  - Can only use recursion to a certain depth
  - Faster albeit less elegant
- Is essentially a bottom-up DFS (kinda)
  - I cheat and make the leaves the roots

For Perl wizards:

```perl
sub iterative {
    my ($v) = @_;
    my $return = "";
    my @stack;
    push(@stack, $v);
    while(scalar(@stack)) {
        my $n = pop(@stack);
        if(($nodes{$n}{'v'} != $i)) {
            $nodes{$n}{'v'} = $i;
            $return = $return . "$n:";
            $traversed++;
            my @children = split(":", $nodes{$n}{'c'});
            my @attrs = split(":", $nodes{$n}{'a'});
            foreach my $c (@children) {
                my @cattrs = split(":", $nodes{$c}{'a'});
                my $cf = 0;
                foreach my $ca (@cattrs) {
                    foreach my $na (@attrs) {
                        if(substr($na, 1) eq substr($ca, 1)) {
                            if($c >= 0) { push(@stack, $c); }
                            $cf = 1;
                            last;
                        }
                    }
                    if($cf) { last; }
                }
            }
        }
    }
    return $return;
}
```

Notional summary (for everyone else)

```
procedure DFS-iterative(G,v):
    push v on a stack, S
    while S is not empty
        v = S.pop()
        if v has not been visited in this round:
            label v as visited
            for all child edges of v do
                if child has a matching attribute with v:
                    S.push(child)
```

Updated complexity analysis

- Algorithm is still $O(|V| + |E|)$
  - Worst case, we look at all vertices
  - Best case, we look at no vertices
- Added overhead for attribute analysis
  - We only keep vertices which share attributes (for debugging)
  - Attribute checking slows down traversal by an order of magnitude or two (fun)

Metrics

- Measured performance in TEPS
- Captured error nodes in separate graphs
  - Examples to come

Datasets

- All datasets are currently synthetic
  - Each is a binary graph
  - Generated via Perl script

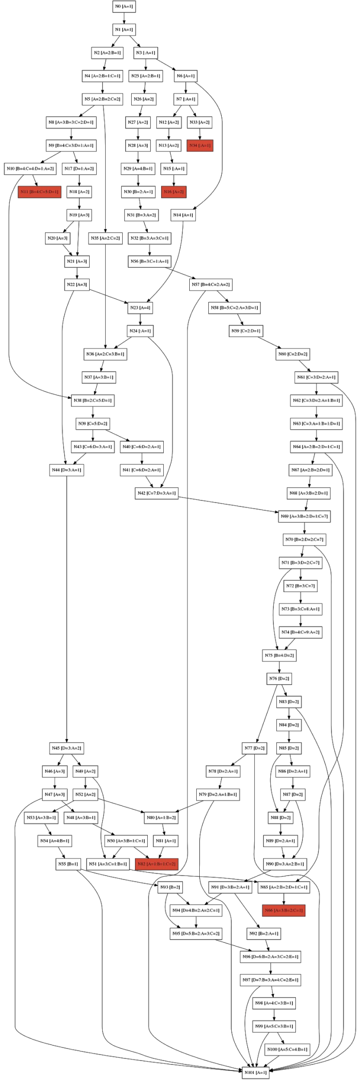
- Number of nodes ranges from 10 - 1,000,000
  - Realistic dataset size O(100) - O(10,000)
  - Tiny: 10 nodes
  - Small: 100 nodes
  - …
  - Colossal: 1,000,000 nodes
  - Any bigger runs into memory limitations

Small DAG example:

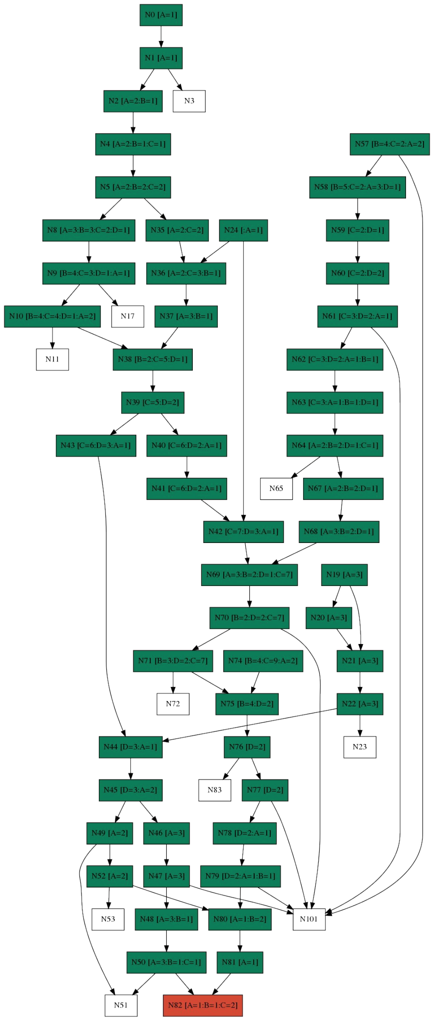
Only 100 nodes

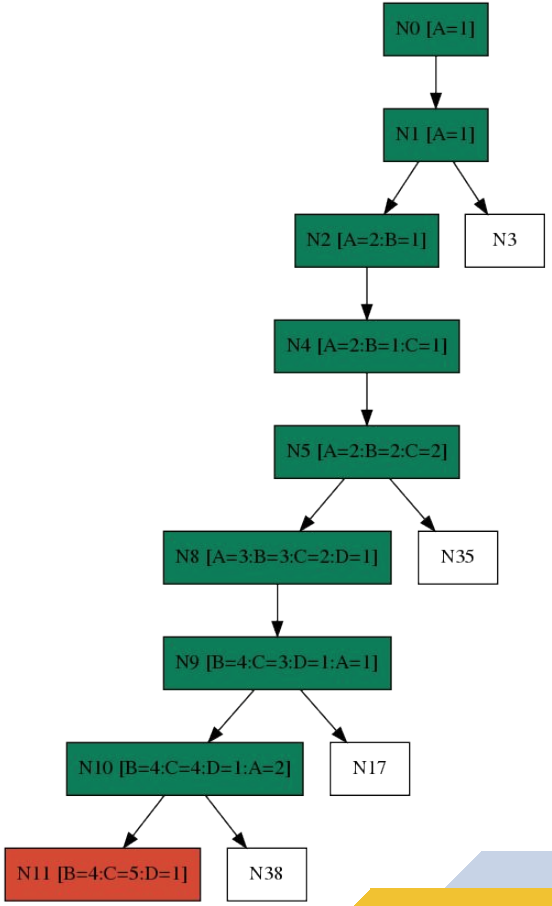
Still a headache to parse through by hand.

No highlighting of failed task lineage! I have to switch between a debug log and this graph.



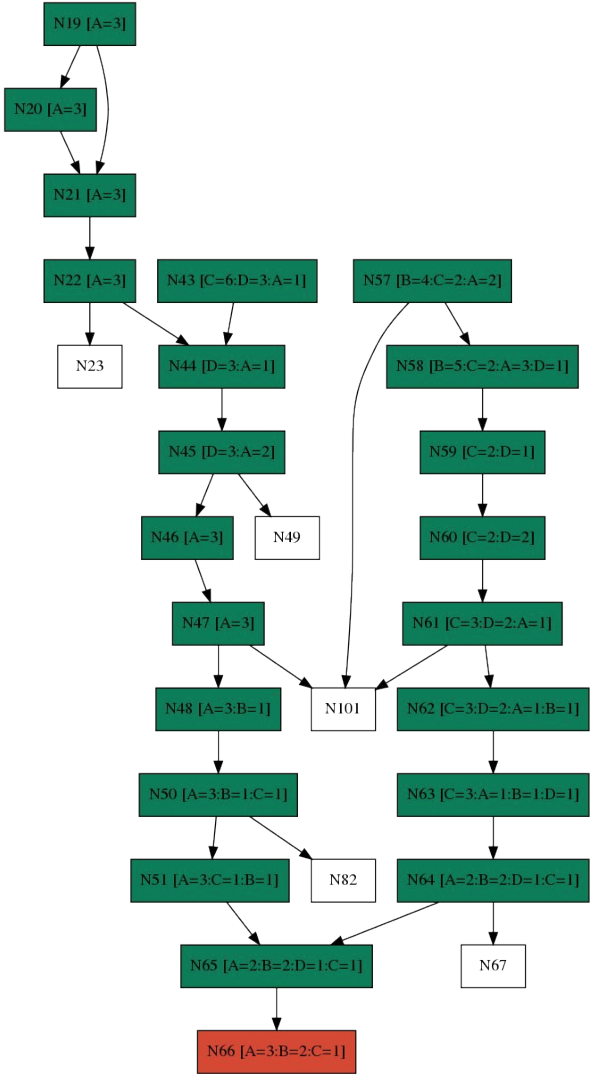
Much more manageable output per failed task.





N16 [A=2]

N34 [:A=1]



Sometimes a task fails on its own, not because of a previous task.

Implementation performance results

Only Traversal	
Nodes	**TEPS**
10	361,347.81
100	375,014.11
1,000	367,753.35
10,000	500,512.47
100,000	476,622.42
1,000,000	458,047.21

Traversal + Computation	
Nodes	**TEPS**
6 (10)	60,676.46
71 (100)	74,231.57
872 (1,000)	69,404.00
9,724 (10,000)	118,423.95
99,032 (100,000)	99,258.17
998,270 (1,000,000)	120,060.52

Plans for parallel implementation

- Use Work Queue master-worker framework to parallelize traversal
  - Cascading traversal pattern
  - May not be faster than serial implementation for a realistic dataset (resource acquisition)

- Use real data if there is time
  - Only roadblock is transforming debug logs into graphs, traverser is done
  - Each type of log has its own syntax, so each requires a handwritten parser

Questions?

UNIVERSITY OF NOTRE DAME