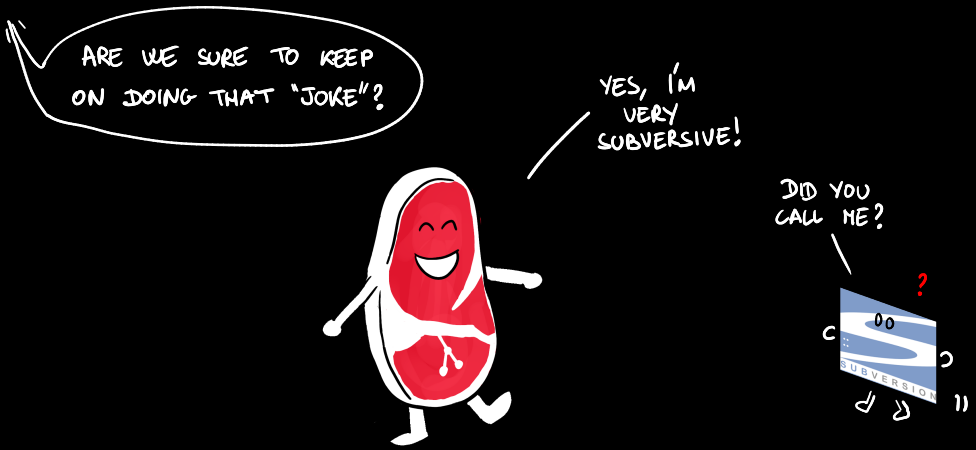
Theoretical Analysis of Git Bisect

Julien COURTIEL (Université de Caen Normandie)
with Paul DORBEC and Romain LECOQ (Université de Caen Normandie)

Journées APR (30 Juin 2024)

Part 1 Looking for the Faulty Commit



QUESTION: What do you use to share files with your coauthors?

THE TIER LIST

QUESTION: What do you use to share files with your coauthors?

god-like

good

trash

spawn of the devil

THE TIER LIST

QUESTION: What do you use to share files with your coauthors?

god-like

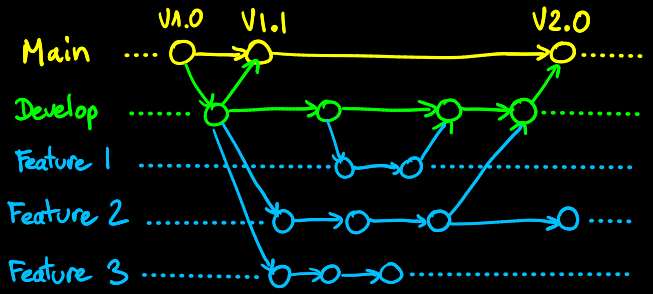
good

trash

spawn of the devil

GIT AND ITS COMMIT GRAPHS

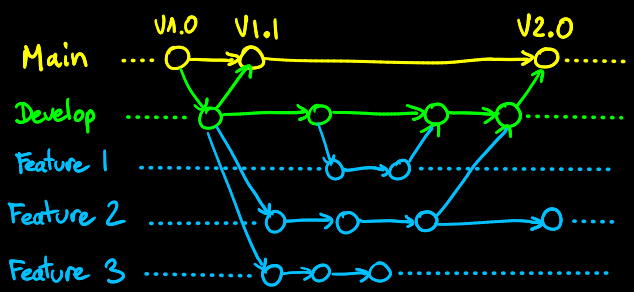
git is a distributed version control system where the revisions (or "commits") are arranged as a Directed Acyclic Graph (DAG)



Commit graph

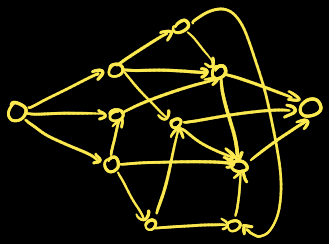
GIT AND ITS COMMIT GRAPHS

git is a distributed version control system where the revisions (or "commits") are arranged as a Directed Acyclic Graph (DAG)



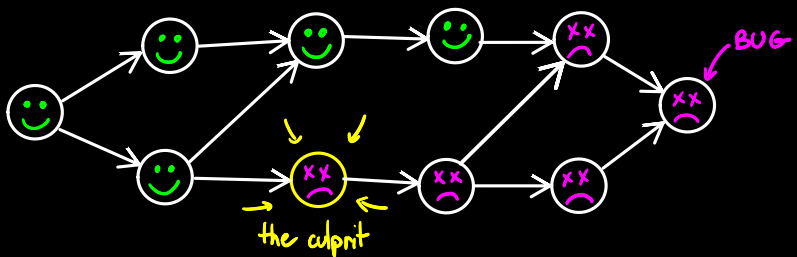
Commit graph

or



Also a legit commit graph!

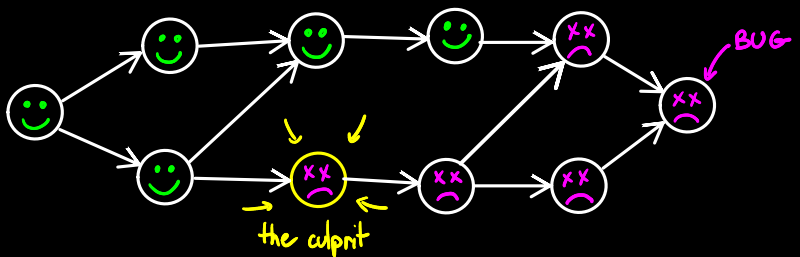
PROBLEM: FINDING A REGRESSION



Input A commit graph in which a commit is known to be bugged, the other commits are bugged or clean (= bug-free)

Question Which commit has originally introduced the bug?

PROBLEM: FINDING A REGRESSION

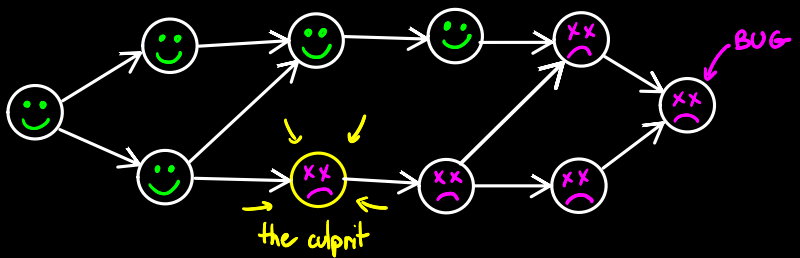


Input A commit graph in which a commit is known to be bugged, the other commits are bugged or clean (= bug-free)

Question Which commit has originally introduced the bug?

Assumption If a parent of a commit is bugged, then the commit is bugged.

PROBLEM: FINDING A REGRESSION



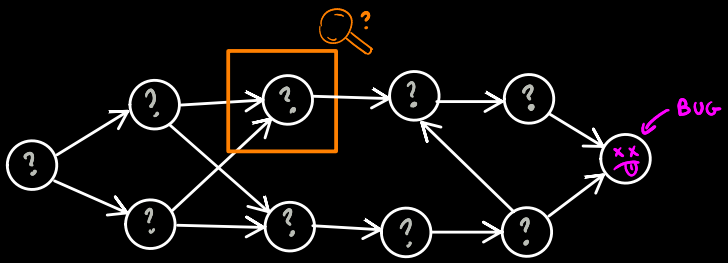
Input A commit graph in which a commit is known to be bugged, the other commits are bugged or clean (= bug-free)

Question Which commit has originally introduced the bug?

Assumptions
- If a parent of a commit is bugged, then the commit is bugged.
- Only one commit has introduced the bug, namely the faulty commit (or regression)

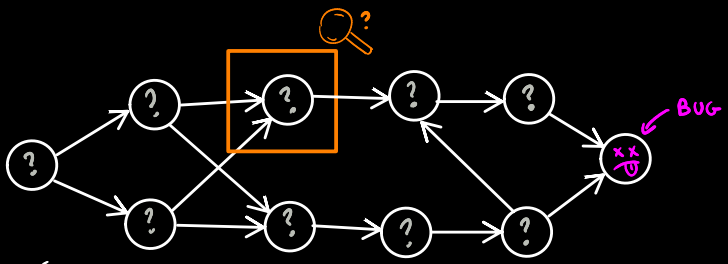
HOW TO INVESTIGATE

Unique operation: QUERY of a commit with unknown status



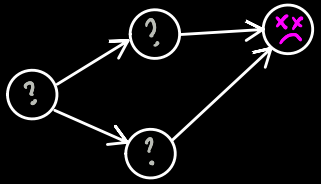
How to investigate

Unique operation: **Query** of a commit with unknown status



If bugged,

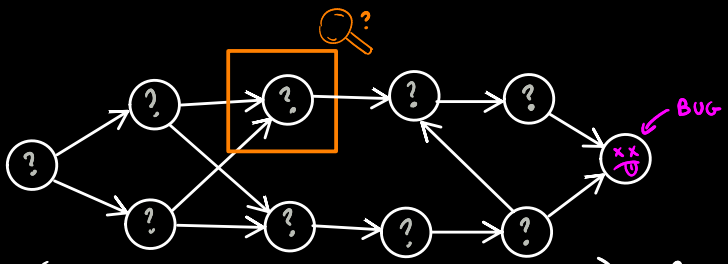
then the faulty commit is an ancestor of this commit



ancestor of a vertex v = v or an ancestor of a parent of v

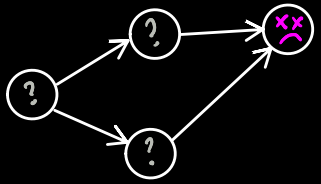
How to investigate

Unique operation: **Query** of a commit with unknown status



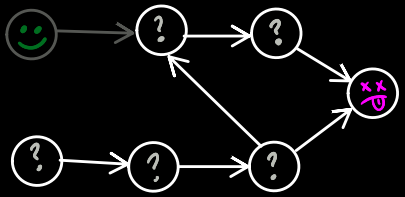
If bugged,

then the faulty commit is an ancestor of this commit



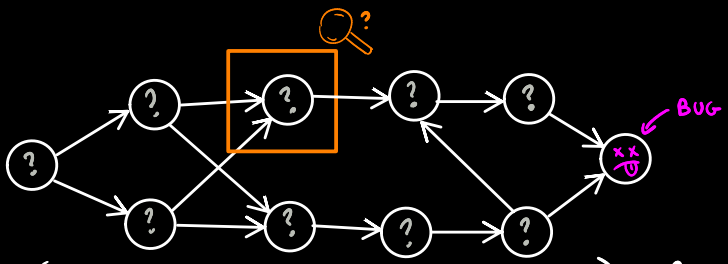
If clean,

then the faulty commit is not an ancestor of this commit



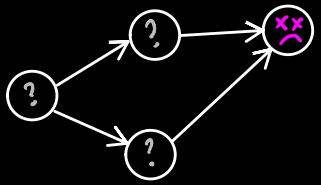
How to investigate

Unique operation: **Query** of a commit with unknown status



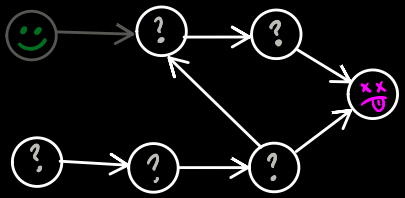
If **bugged**,

then the faulty commit is an ancestor of this commit



If **clean**,

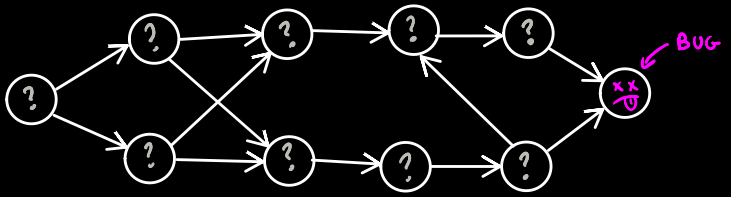
then the faulty commit is not an ancestor of this commit



The faulty commit is found whenever there remains only 1 suspect.

REGRESSION SEARCH PROBLEM

Input: a DAG where each vertex has an unknown status, except one, which is bugged.

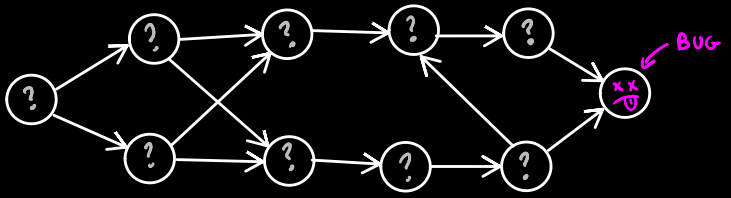


Output: A strategy that finds the faulty commit with a minimal number of queries in the worst-case scenario
= optimal strategy

(the faulty commit can be any ancestor of the bugged vertex)

REGRESSION SEARCH PROBLEM

Input: a DAG where each vertex has an unknown status, except one, which is bugged.



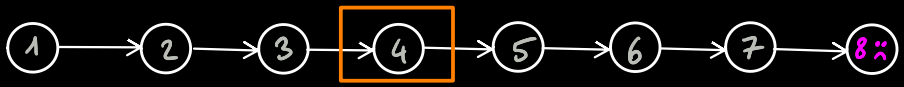
Output: A strategy that finds the faulty commit with a minimal number of queries in the worst-case scenario
= optimal strategy

(the faulty commit can be any ancestor of the bugged vertex)

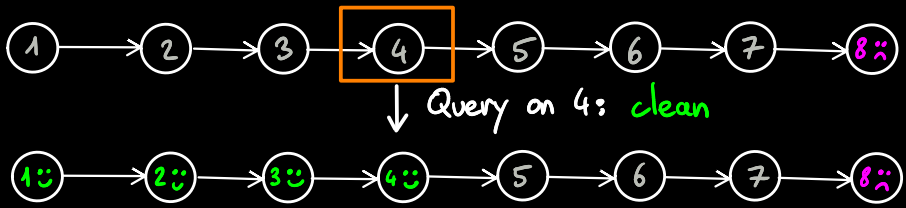
In real life, queries are costly.

FIRST EXAMPLE : CHAINS

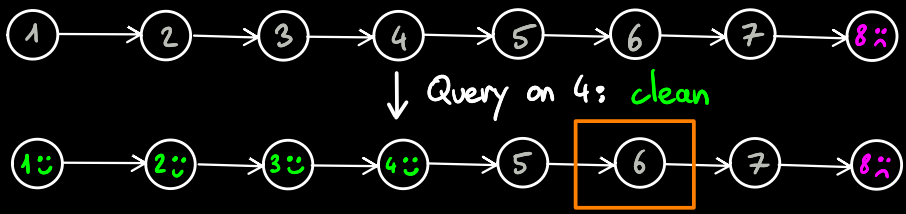
FIRST EXAMPLE : CHAINS



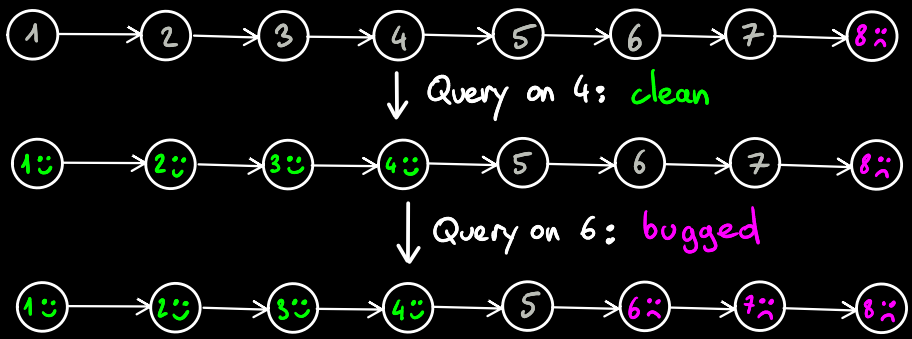
FIRST EXAMPLE : CHAINS



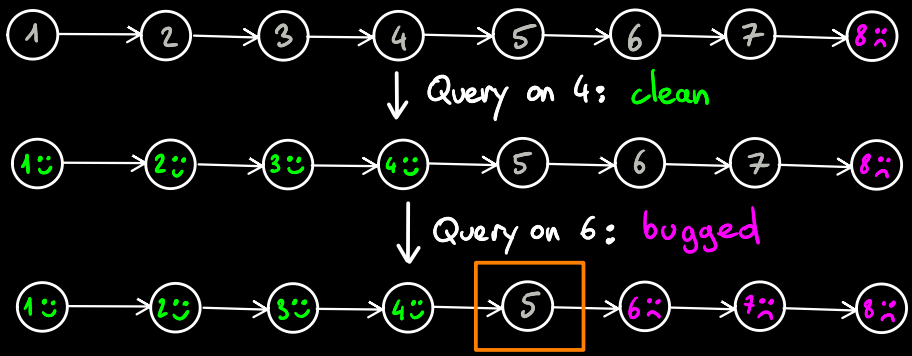
FIRST EXAMPLE : CHAINS



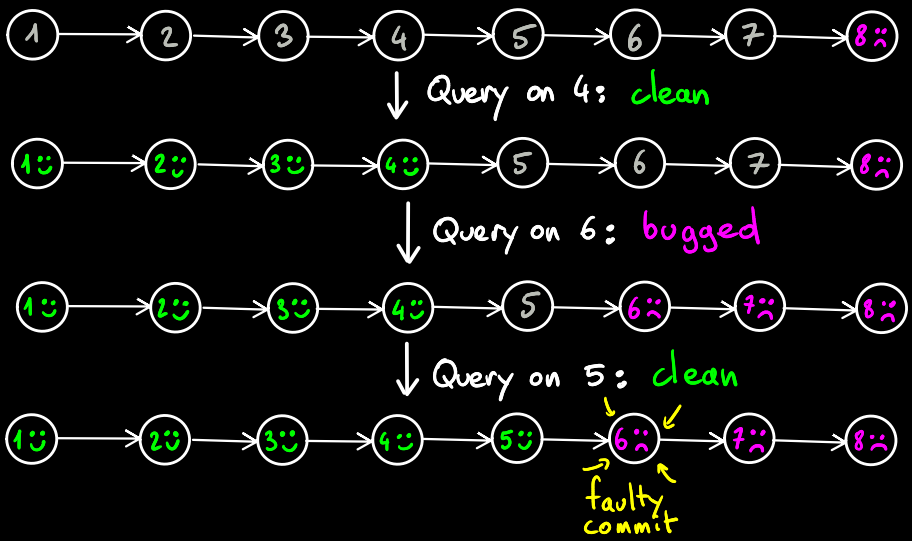
FIRST EXAMPLE : CHAINS



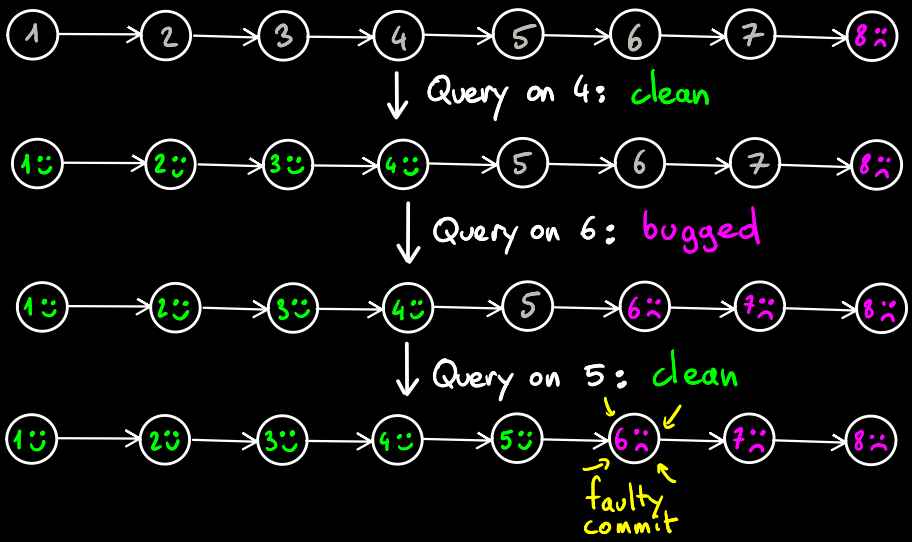
FIRST EXAMPLE : CHAINS



FIRST EXAMPLE : CHAINS

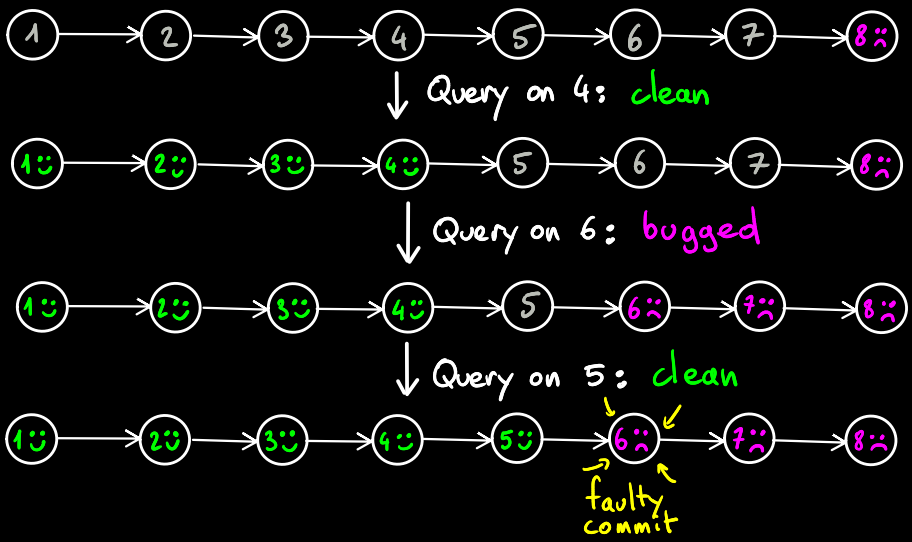


FIRST EXAMPLE : CHAINS



optimal strategy = binary search

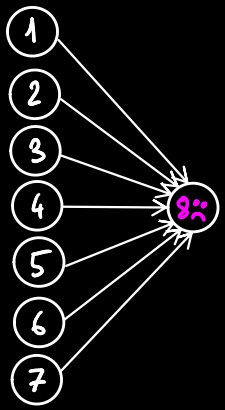
FIRST EXAMPLE : CHAINS



optimal strategy = binary search

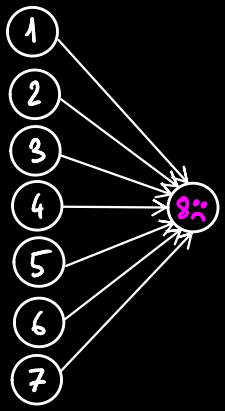
More generally, number of queries in an optimal strategy in a chain of length $n = \lceil \log_2(n) \rceil$

SECOND EXAMPLE : OCTOPUSES



optimal strategy =

SECOND EXAMPLE : OCTOPUSES



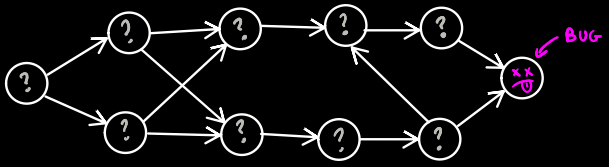
optimal strategy = whatever

More generally, number of queries in an optimal strategy in an octopus of size n = $n-1$

COMPLEXITY OF THE REGRESSION SEARCH PROBLEM

REGRESSION SEARCH PROBLEM (DECISION VERSION)

Input: a DAG where each vertex has an unknown status, except one, which is bugged + integer $k \geq 0$



Decide: Is there a strategy that finds the faulty commit with at most k queries in every scenario?

NP-complete for general DAGs

[Carmo Donadelli Kohayakawa Laber 2004]

[We've also proved it!]

but polynomial for trees ...

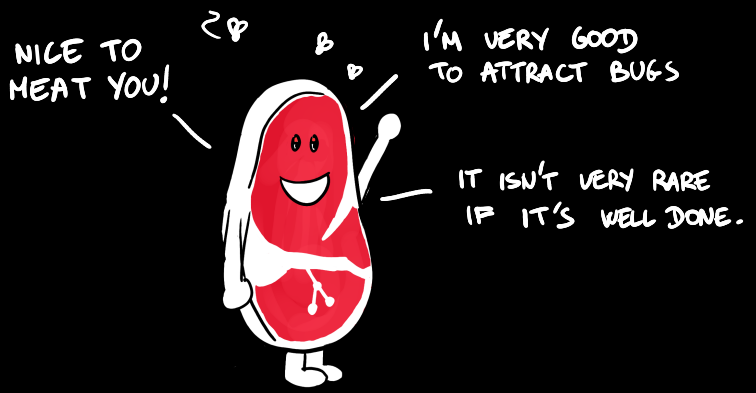
[Ben-Asher Farchi Newman 2000]

...more precisely, linear.

[Mozes Onak Weimann 2008]

The more well-known Decision Tree Problem generalizes the Regression Search Problem but it deals with the average case.

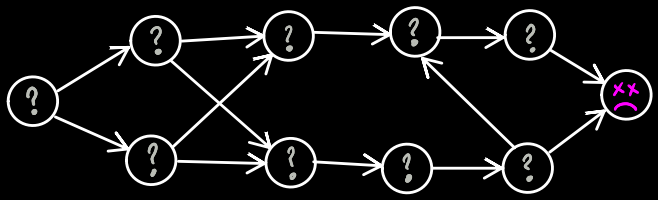
Part 11
Git Bisect



THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1:



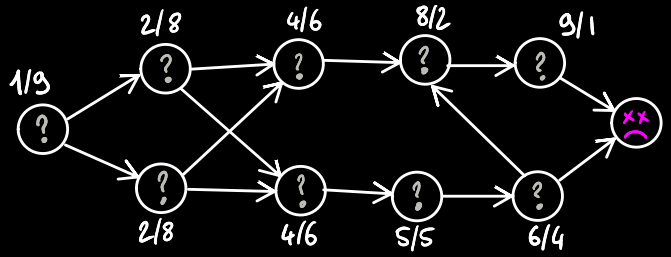
STEP 2 :

STEP 3:

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



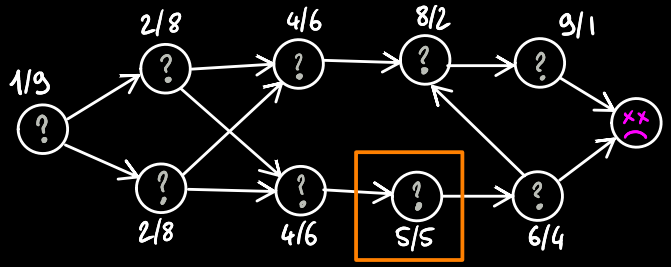
STEP 2 :

STEP 3 :

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



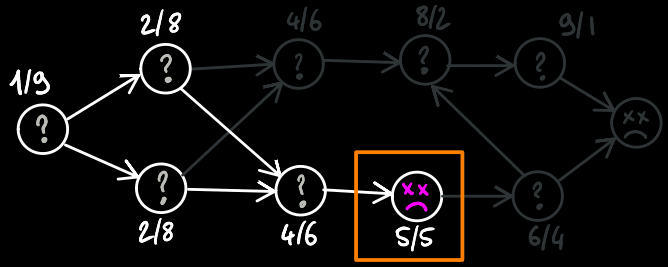
STEP 2: Query the vertex with the most balanced ratio

STEP 3:

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



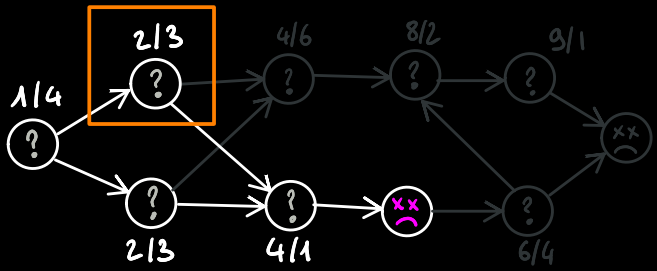
STEP 2: Query the vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



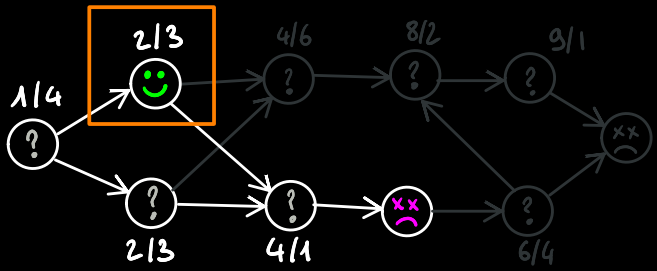
STEP 2: Query a vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



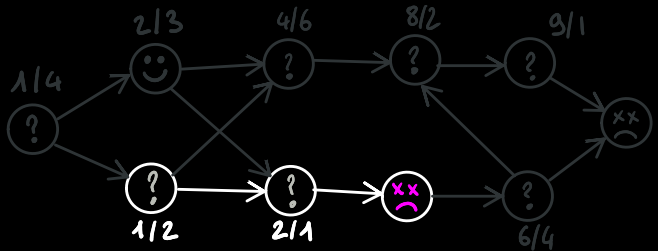
STEP 2: Query a vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



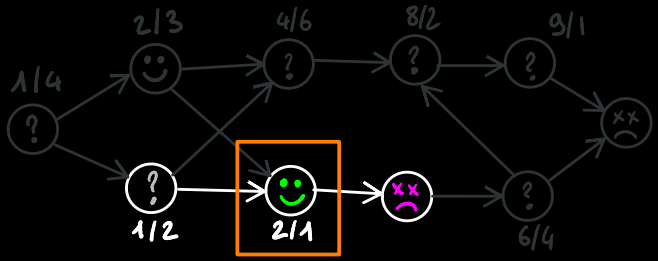
STEP 2: Query the vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



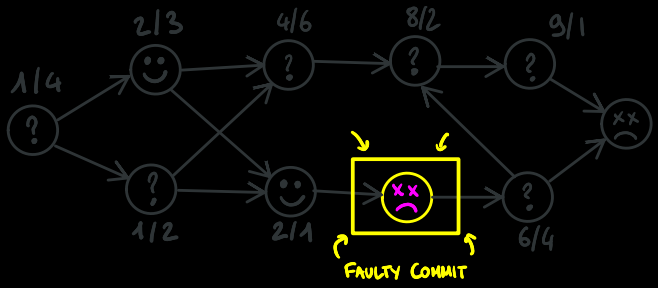
STEP 2: Query the vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



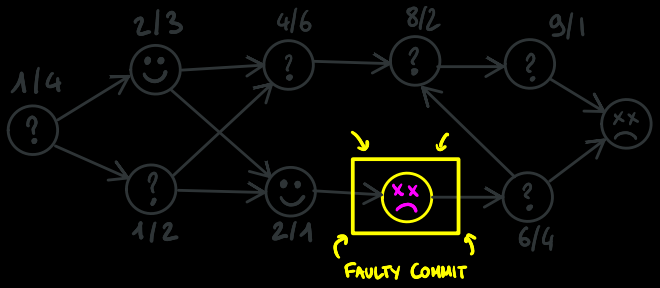
STEP 2: Query the vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

THE GIT BISECT ALGORITHM

git uses a heuristic to find the faulty commit: git bisect

STEP 1: Compute the number of ancestors/non-ancestors for each commit



STEP 2: Query the vertex with the most balanced ratio

STEP 3: Delete the innocent commits and recurse.

QUESTION: Does git bisect always give an optimal strategy?

HOW GOOD IS GIT BISECT?

QUESTION: Does git bisect always give an optimal strategy?

How good is git bisect?

Question: Does git bisect always give an optimal strategy?

NO The Regression Search Problem is NP-complete.

How good is git bisect?

Question: Does git bisect always give an optimal strategy?

NO The Regression Search Problem is NP-complete.

But is it really that bad?

HOW GOOD IS GIT BISECT?

QUESTION: Does git bisect always give an optimal strategy?

NO The Regression Search Problem is NP-complete.

But is it really that bad? YEAH

HOW GOOD IS GIT BISECT?

QUESTION: Does git bisect always give an optimal strategy?

NO The Regression Search Problem is NP-complete.

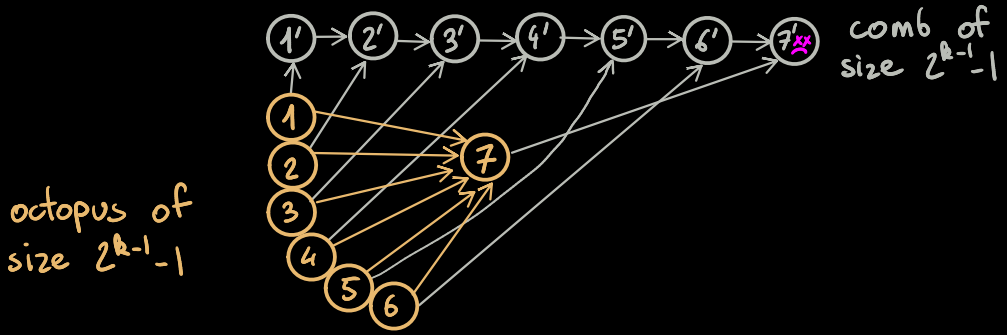
But is it really that bad? YEAH

Proposition For any k, there exists a DAG such that an optimal strategy uses k queries and git bisect always uses $2^{k-1}-1$ queries.

How good is git bisect?

Proposition For any k, there exists a DAG such that an optimal strategy uses k queries and git bisect always uses $2^{k-1}-1$ queries.

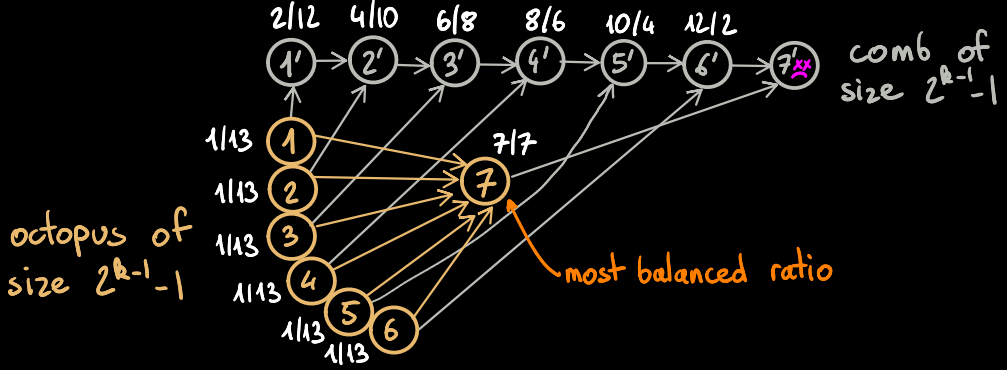
Proof for $k=4$:



How good is git bisect?

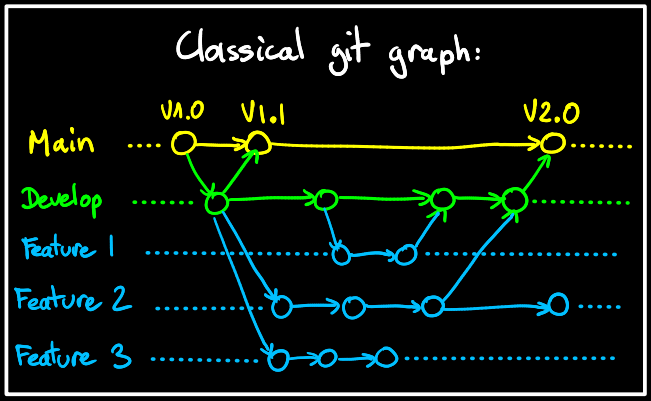
Proposition For any k, there exists a DAG such that an optimal strategy uses k queries and git bisect always uses $2^{k-1}-1$ queries.

Proof for $k=4$:



BACK TO REALITY?

Octopus substructures are unrealistic

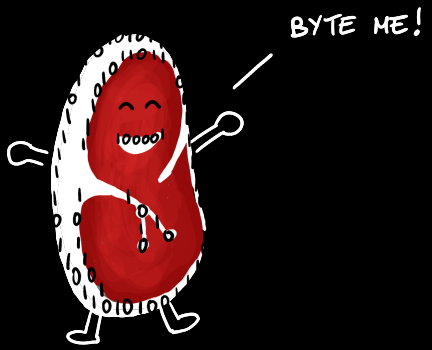


Usually, we never merge more than 2 branches.

(Otherwise it is called an octopus merge

)

Part III - git bisect on binary DAGs

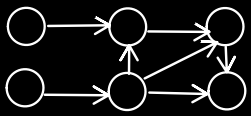


BINARY DAG

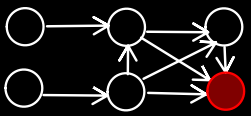
Definition

binary DAG = DAG where the vertices have indegree ≤ 2

Ex:



Good



Bad

Theorem

git bisect is a $\frac{1}{\log_2(\frac{3}{2})}$-approximation algorithm when it is used on binary DAGs.

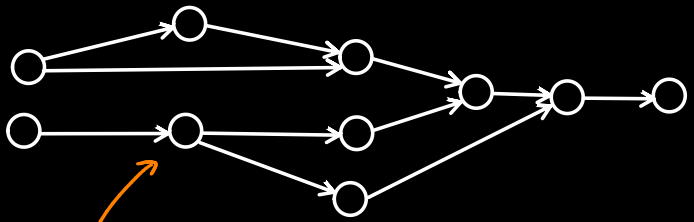
$\frac{1}{\log_2(\frac{3}{2})} \simeq 1{,}71$ is the optimal constant.

EXISTENCE OF A BALANCED VERTEX

Lemma

Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$



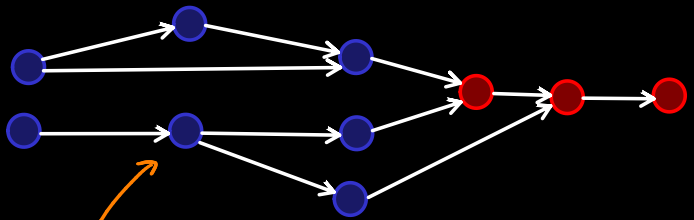
Where is it?

EXISTENCE OF A BALANCED VERTEX

Lemma

Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$



Where is it ??

= more ancestors than non-ancestors

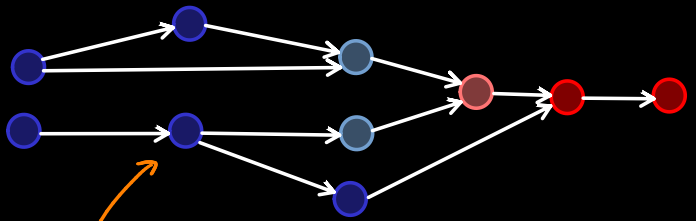
= more non-ancestors than ancestors

EXISTENCE OF A BALANCED VERTEX

Lemma

Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$



Where is it?

= more ancestors than non-ancestors

= more non-ancestors than ancestors

= where all parents are

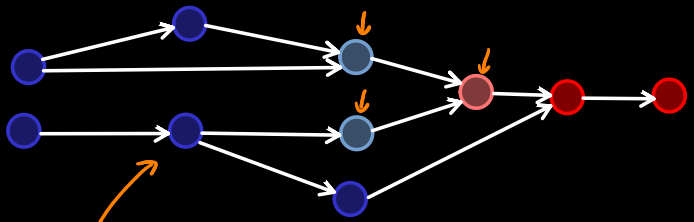
= parent of a

EXISTENCE OF A BALANCED VERTEX

Lemma

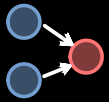
Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$



Where is it?

It's in any



= more ancestors than non-ancestors

= more non-ancestors than ancestors

= where all parents are

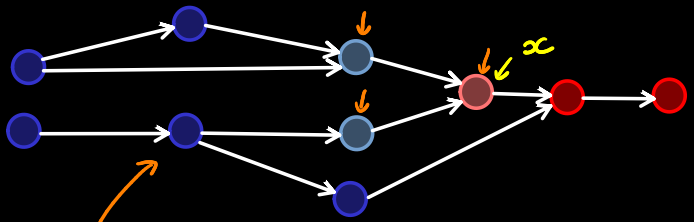
= parent of a

EXISTENCE OF A BALANCED VERTEX

Lemma

Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$



Where is it?

It's in any

● = more ancestors than non-ancestors

● = more non-ancestors than ancestors

● = ● where all parents are ●

● = ● parent of a ●

EXISTENCE OF A BALANCED VERTEX

Lemma

Any binary DAG with n vertices has a vertex x such that

$$\frac{n}{3} \leq \text{nb of ancestors of } x \leq \frac{2n+1}{3}$$

Proof of the $\frac{1}{\log_2(\frac{3}{2})}$-approximation:

At each query, git bisect eliminates $\geq 1/3$ commits by choosing x or a better vertex

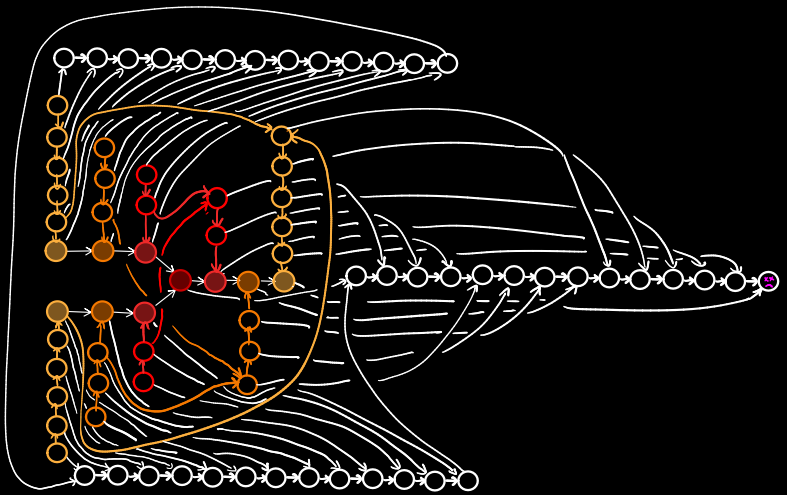
- number of git bisect queries $\simeq \log_{\frac{3}{2}}(n)$
- optimal number of queries $\geq \log_2(n)$

Problematic binary DAGs for git bisect

For every $k > 0$, there exists a binary DAG with:

- $\simeq 6k \left(\frac{3}{2}\right)^k$ vertices,
- $\geqslant k$ git bisect queries,
- $\simeq \log_2\left(\frac{3}{2}\right) \times k$ queries in the optimal strategy.

e.g. for $k = 3$:

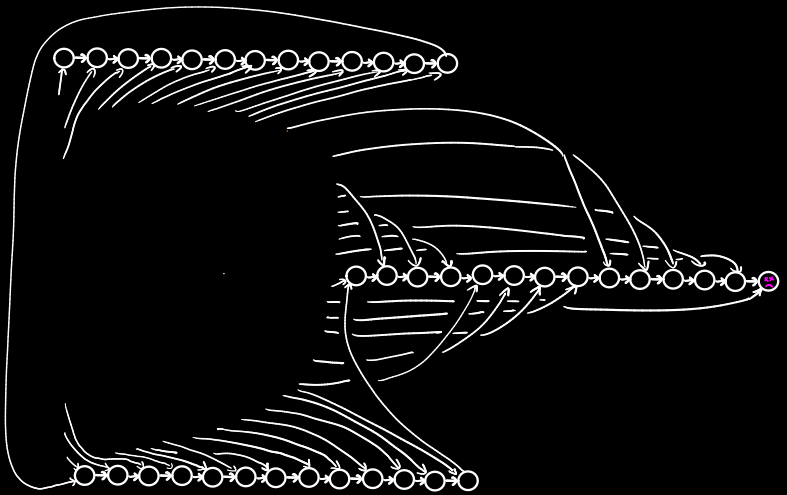


Problematic binary DAGs for git bisect

For every $k > 0$, there exists a binary DAG with:

- $\simeq 6k \left(\frac{3}{2}\right)^k$ vertices,
- $\geqslant k$ git bisect queries,
- $\simeq \log_2\left(\frac{3}{2}\right) \times k$ queries in the optimal strategy.

e.g. for $k = 3$:

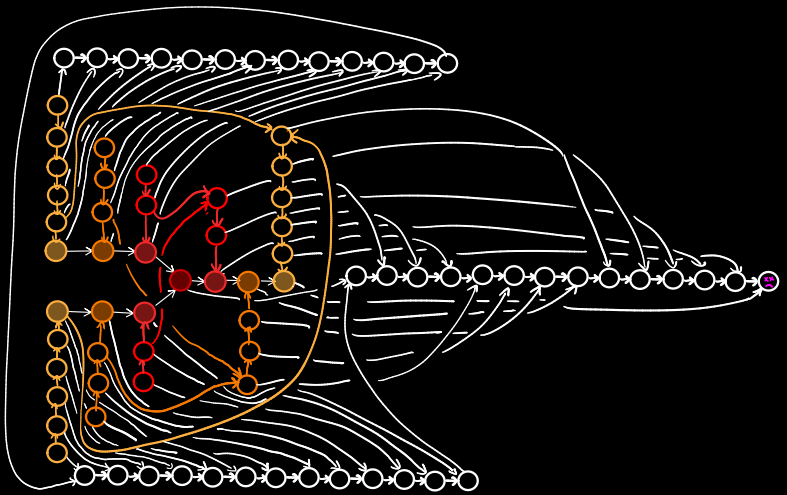


Problematic binary DAGs for git bisect

For every $k > 0$, there exists a binary DAG with:

- $\simeq 6k \left(\frac{3}{2}\right)^k$ vertices,
- $\geqslant k$ git bisect queries,
- $\simeq \log_2\left(\frac{3}{2}\right) \times k$ queries in the optimal strategy.

e.g. for $k = 3$:

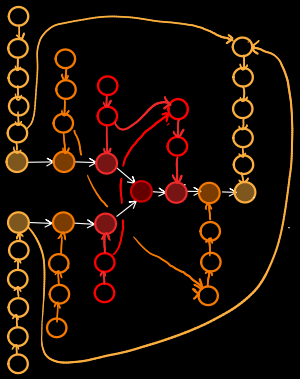


Problematic binary DAGs for git bisect

For every $k > 0$, there exists a binary DAG with:

- $\simeq 6k \left(\frac{3}{2}\right)^k$ vertices,
- $\geqslant k$ git bisect queries,
- $\simeq \log_2\left(\frac{3}{2}\right) \times k$ queries in the optimal strategy.

e.g. for $k = 3$:



A BETTER ALGORITHM? GOLDEN BISECT

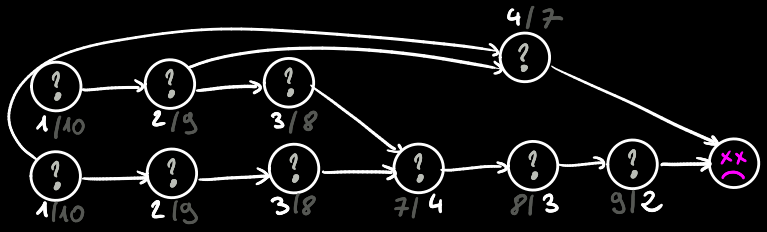
theoritical analysis of git bisect → new (better?) algorithm

A BETTER ALGORITHM? GOLDEN BISECT

theoritical analysis of git bisect → new (better?) algorithm

☆ GOLDEN BISECT ☆

Step 1: Compute the number of ancestors/number of non-ancestors for each vertex and keep the smaller of the 2 numbers.

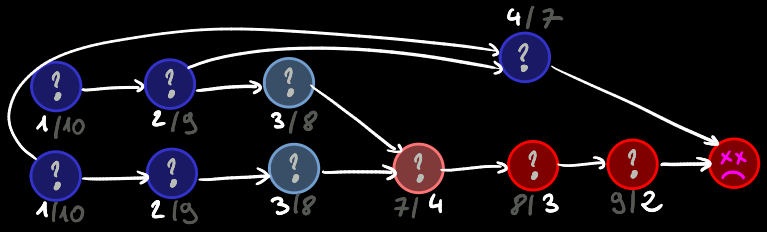


A BETTER ALGORITHM? GOLDEN BISECT

theoritical analysis of git bisect → new (better?) algorithm

☆ GOLDEN BISECT ☆

Step 1: Compute the number of ancestors/number of non-ancestors for each vertex and keep the smaller of the 2 numbers.



Step 2: If the largest number is $\geq \frac{\text{nb vertices}}{\phi^2} \simeq 0.38 \times \text{nb vertices}$, query a commit with this number.

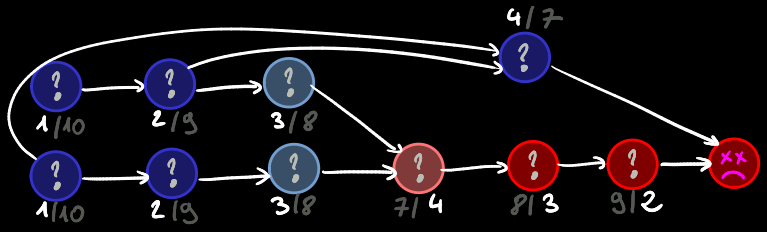
ϕ = golden ratio = $\frac{1+\sqrt{5}}{2}$

A BETTER ALGORITHM? GOLDEN BISECT

theoritical analysis of git bisect → new (better?) algorithm

☆ GOLDEN BISECT ☆

Step 1: Compute the number of ancestors/number of non-ancestors for each vertex and keep the smaller of the 2 numbers.



Step 2: If the largest number is $\geq \frac{\text{nb vertices}}{\phi^2} \simeq 0.38 \times \text{nb vertices}$, query a commit with this number.

ϕ = golden ratio = $\frac{1+\sqrt{5}}{2}$

Otherwise, query a 🔴 or a 🔵 with the largest number

- 🔴 = more ancestors than non-ancestors
- 🔵 = more non-ancestors than ancestors
- 🟣 = 🔴 where all parents are 🔵
- 🟣 = 🔵 parent of a 🔴

Step 3: Delete the innocent commits and recurse.

THEORETICAL ANALYSIS OF GOLDEN BISECT

Theorem

golden bisect is a $\frac{1}{\log_2(\phi)}$ - approximation algorithm for binary DAGs, where ϕ = golden ratio.

$\frac{1}{\log_2(\phi)} \simeq 1{,}44$ is the optimal constant.

But sometimes git bisect is better than golden bisect!

OPEN QUESTIONS

→ Is the Regression Search Problem NP-complete when restricted to binary DAGs?

(We proved that a variant, the Confined Regression Search Problem is NP-complete for binary DAGs)

→ study of git bisect in average case?

but what is a random "git graph"?

(⇒ definition of git graph in [Lecoq's PhD], random generation in [C., Pépin])

→ Is git-bisect a 2-approximation algorithm for trees?

THANK YOU!