Segmented Changelog

Jun Wu

Facebook

April 15, 2019

Goals

Improve $O(changelog^{\dagger})$ DAG operations.

- $O(1)$-ish clone with edenfs
- $O(\log changelog)$-ish ancestor related DAG calculations

Improve $O(changelog)$ set operations.

- `a + b`
- `a - b`
- `a & b`

[†] Number of commits in the repo

Non-goals

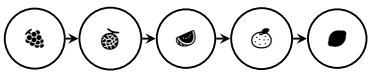
Filter operations. Expect they run on a small subset.

- `author(alice)`
- `date(2017)`
- `branchpoint()`
- `merge()`

File DAG operations. Expect server-side support (ex. fastlog).

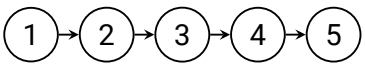
- `follow(path, commit)`

Commit Identities



[, , , ,]

$O(N)$ space



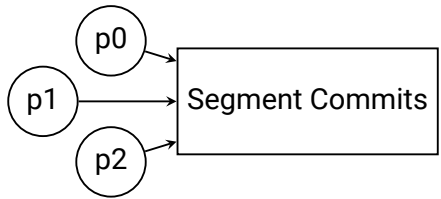
`1:5`

$O(1)$ space

Segments

Segments and parents

A Segment contains a list of numbers.
Numbers are sorted topologically in DAG.

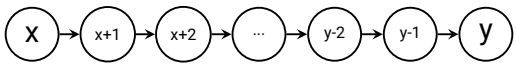


Parents of a segment: `parents(segment) - segment`. Stored with the segment.

Flat Segments

Constraints:

- `x:y` (using revision numbers)
- `heads(x:y)` is `y`
- `roots(x:y)` is `x`
- `(((x:y) - x) & merge())` is empty

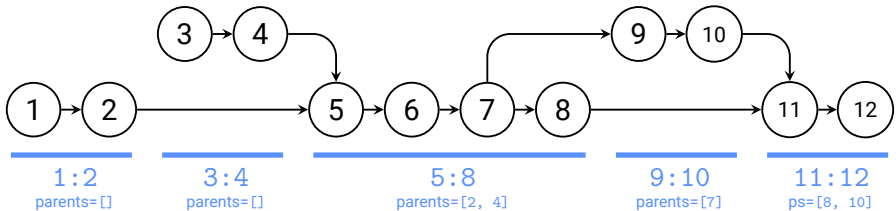


Properties:

- Reduce DAG complexity to $O(merges)$
- Parents of `x` are segment parents.
- Parent information is loseless.

Flat Segments

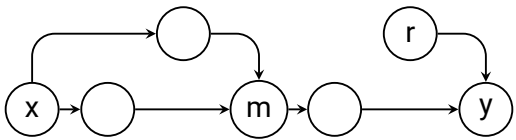
Example



High-Level Segments

Constraints:

- `x:y` (using revision numbers)
- `heads(x:y)` is `y`



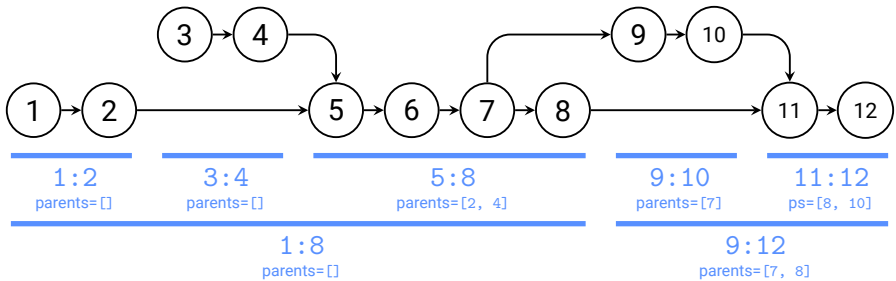
Properties:

- Compress commits across merges
- Parent information is lossy.[†]

[†]Cannot get parents of arbitrary commit by high-level segments only.

High-Level Segments

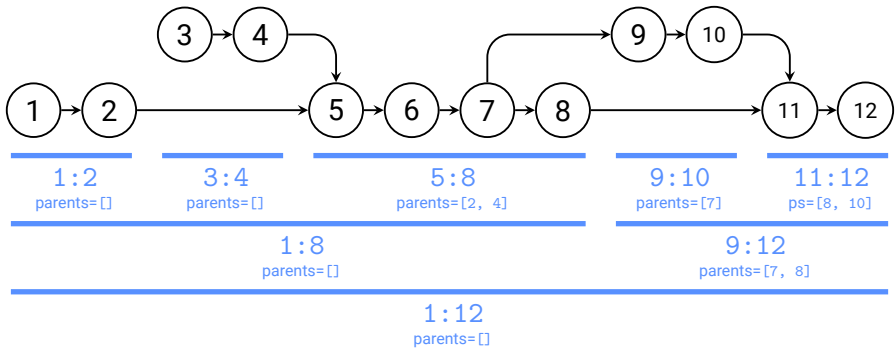
Example



Segment Size = 3. A high-level segment contains 3 lower-level segments at most.

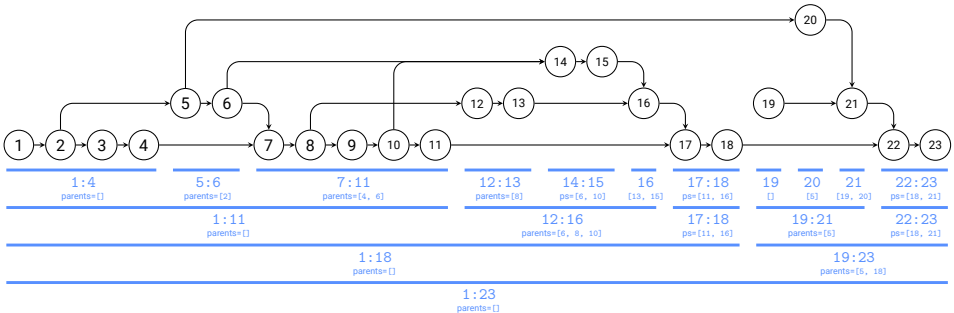
High-Level Segments

Example



High-Level Segments

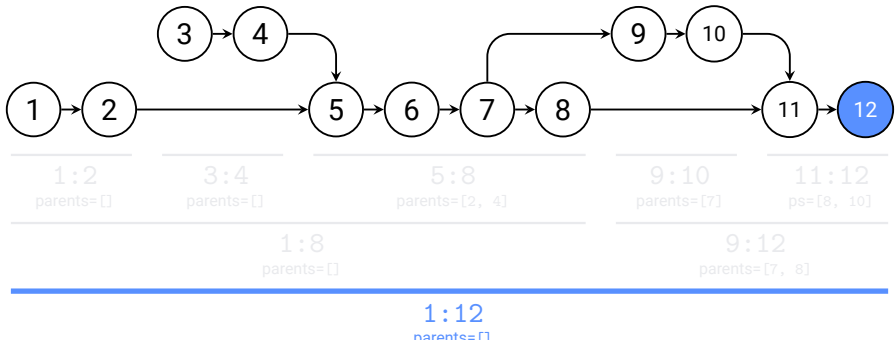
Example



High-Level Segments

Example: Ancestors Selection

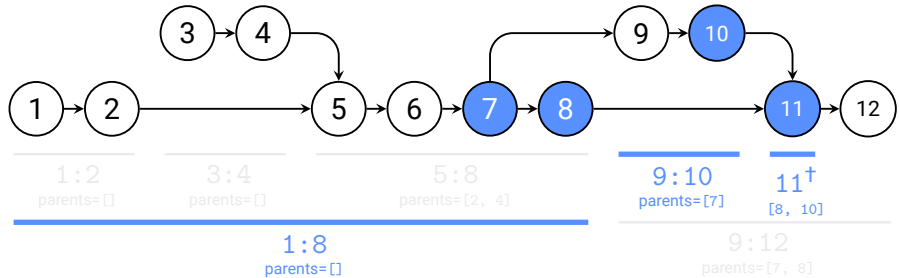
Selecting `::12`



High-Level Segments

Example: Ancestors Selection

Selecting `::11`

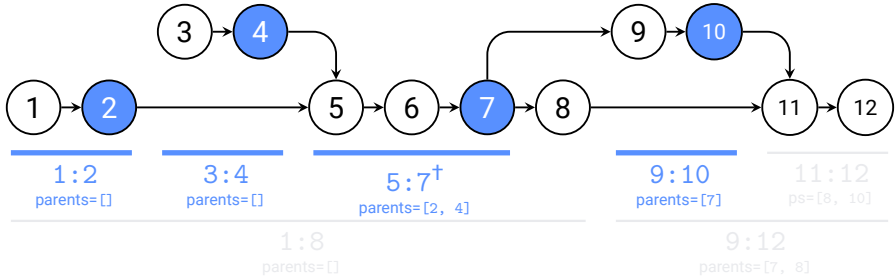


† `11:11` is generated from `11:12`.

High-Level Segments

Example: Ancestors Selection

Selecting `::10`

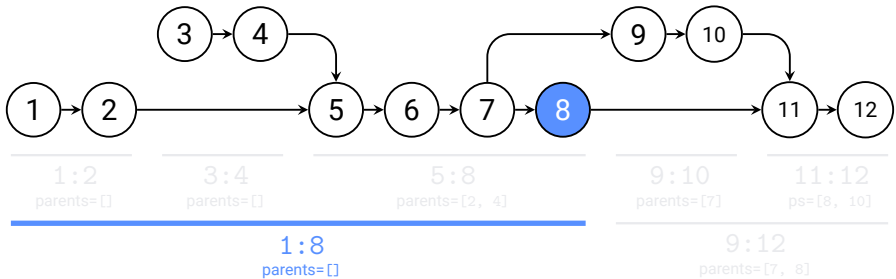


†`5:7` is generated from `5:8`.

High-Level Segments

Example: Ancestors Selection

Selecting `::8`



High-Level Segments

Example: Common Ancestor

```
ancestor(10, 8)
= max(::10† & ::8)
= max((1:7+9:10) & 1:8)
= max(1:7)
= 7
```

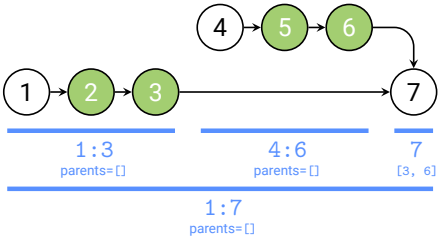
---

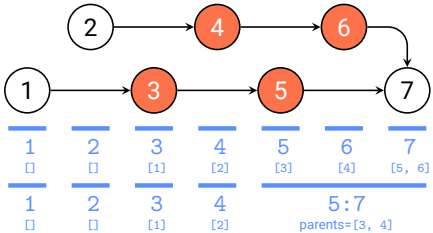
† `::10` can be lazy‡ to avoid `1:2` and `3:4` lookups.

‡ Laziness has a cost. Non-lazy version has $O(\log changelog)$-ish overhead.

Assigning Numbers

Flat Segments

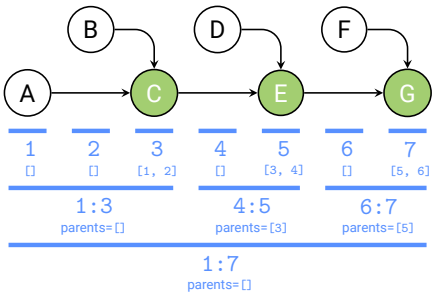


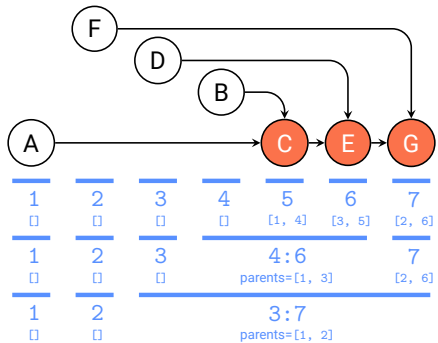


Use Depth-First Serach, not Breadth-First Search, from a merge.

Assigning Numbers

Merges

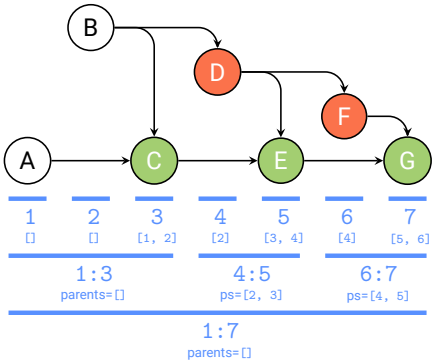


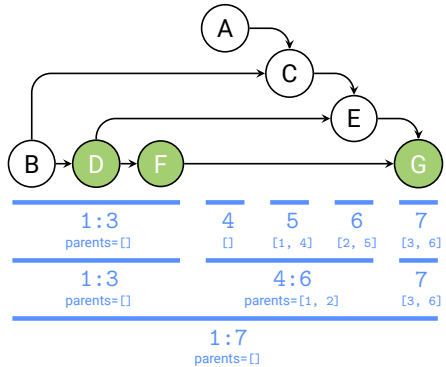


Use Depth-First Serach.

Assigning Numbers

Merges





Pick parent branch with less merges first.

Assigning Numbers

Algorithm

To assign a number for `x`, check its parents.

- If all parents have numbers, assign the next available number to `x`.
- Otherwise, pick the parent branch with less merges. Assign it recursively.

Real-world Repos

Repo	Commits	Flat	Level 2†	Level 3	Level 4
fbsource	millions	19961	1325	87	6
mozilla	469k	34179	2390	149	8
cpython	98k	21744	1367	81	4
pypy	74k	9008	609	39	2
git	55k	23884	1534	90	5
hg	42k	4654	297	17	0

†Segment Size = 16. Last segment per level is removed.

New Structures

- Number - Commit Hash Mapping
  - Large
  - Stored Server-side
- Commit Hash - Commit Data (user, date, message) Mapping
  - Larger
  - Stored Server-side
- Segments
  - Tiny (<1MB)
  - Calculated Server-side
  - Stored client-side