Range-Based Set Reconciliation

Aljoscha Meyer

Set Reconciliation

- set union over a network
- between (exactly) two machines
- unstructured data
- no shared state or history

Trivial Reconciliation

Sam Gwilym

Model and Analysis

- Alfie and Betty talk over a network
- reliable communication, rounds of unit length, unlimited bandwidth
- probabilistic solutions

Model and Analysis

- Alfie and Betty talk over a network
- reliable communication, rounds of unit length, unlimited bandwidth
- probabilistic solutions
- n: size of the union
- $n_\triangle$: size of the symmetric difference

Model and Analysis

- roundtrips
- communicated bytes
- computation time
- computation space

Traditional Approaches

- obtain approximation of $n_{\triangle}$
- compute message of size $\mathcal{O}(n_{\triangle})$ by iterating over all n items
- exchange messenges
- recover symmetric difference from those messages using at least $\mathcal{O}(n_{\triangle})$ time and memory

P2P Reconciliation

Peer-to-peer systems:

- iterating over local set every time infeasible
- loading local set into memory infeasible
- some peers are out to get us

P2P Reconciliation

Peer-to-peer systems:

- iterating over local set every time infeasible
- loading local set into memory infeasible
- some peers are out to get us

$\implies$ traditional approaches don't work

Reducing Computational Complexities

- allow for logarithmic number of rounds

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
    - if equal, we are done

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
    - if equal, we are done
- divide-and-conquer

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
    - if equal, we are done
- divide-and-conquer
    - chose predicate for partitioning

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
  - if equal, we are done
- divide-and-conquer
  - chose predicate for partitioning
  - restrict to simple predicates (split into ranges)

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
  - if equal, we are done
- divide-and-conquer
  - chose predicate for partitioning
  - restrict to simple predicates (split into ranges)
  - peers alternatingly choose splits that are optimal for themselves

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
    - if equal, we are done
- divide-and-conquer
    - chose predicate for partitioning
    - restrict to simple predicates (split into ranges)
    - peers alternatingly choose splits that are optimal for themselves
- binary-search for differences

Reducing Computational Complexities

- allow for logarithmic number of rounds
- compare fingerprints of local sets
    - if equal, we are done
- divide-and-conquer
    - chose predicate for partitioning
    - restrict to simple predicates (split into ranges)
    - peers alternatingly choose splits that are optimal for themselves
- binary-search for differences
    - collaboratively
    - over the wire
    - in parallel
    - ok, the analogy is not perfect

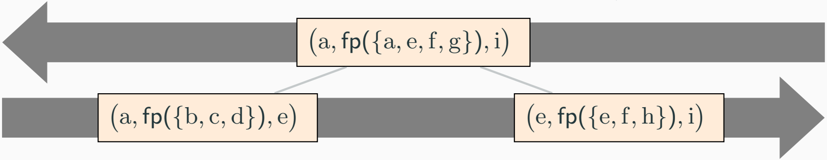
Range-Based Set Reconciliation

$X_\alpha := \{\mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}, \mathrm{h}\}$

$X_\beta := \{\mathrm{a}, \mathrm{e}, \mathrm{f}, \mathrm{g}\}$

$\leftarrow$ $\big(\mathrm{a}, \mathsf{fp}(\{\mathrm{a}, \mathrm{e}, \mathrm{f}, \mathrm{g}\}), \mathrm{i}\big)$

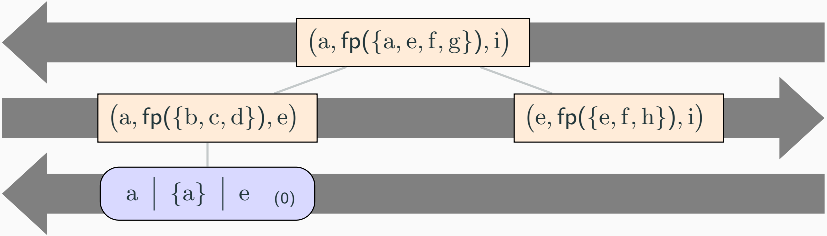
$X_\alpha := \{b, c, d, e, f, h\}$ $\quad$ $X_\beta := \{a, e, f, g\}$



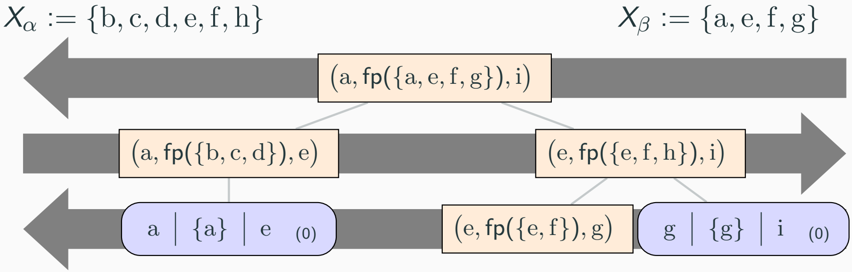
Range-Based Set Reconciliation

$X_\alpha := \{b, c, d, e, f, h\}$

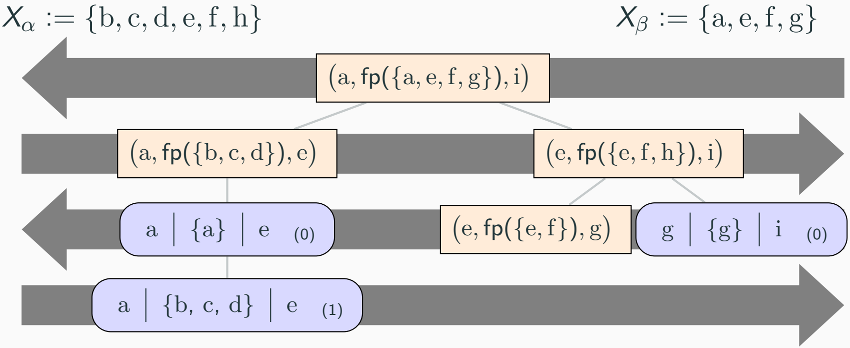
$X_\beta := \{a, e, f, g\}$



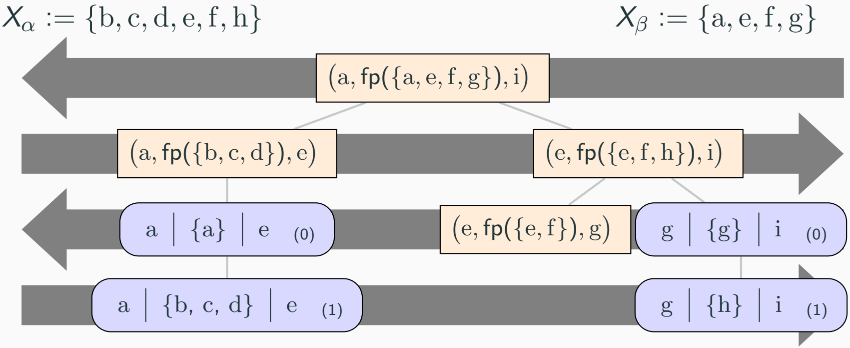
Range-Based Set Reconciliation



Range-Based Set Reconciliation



Range-Based Set Reconciliation



Some Nice Properties

- reasonably efficient: $\mathcal{O}(\min(n_\triangle \cdot \log(n), n))$ bytes communication, $\mathcal{O}(1)$ working memory
- can tune bandwidth vs roundtrip minimization
- arbitrary recursion anchor protocols
- arbitrary partition techniques

Some Nice Properties

- reasonably efficient: $\mathcal{O}(\min(n_\triangle \cdot \log(n), n))$ bytes communication, $\mathcal{O}(1)$ working memory
- can tune bandwidth vs roundtrip minimization
- arbitrary recursion anchor protocols
- arbitrary partition techniques
- but: linear computation times

Reducing Computation Times

- $\binom{n}{2}$ possible subranges $\implies$ cannot precompute all fingerprints

Reducing Computation Times

- $\binom{n}{2}$ possible subranges $\implies$ cannot precompute all fingerprints
- free space budget of $\mathcal{O}(n)$
- labeled trees!

Intermission — Merkle Search Trees

Auvolat, Alex, and François Taïani. "Merkle search trees: Efficient state-based CRDTs in open networks." 2019 38th Symposium on Reliable Distributed Systems (SRDS). IEEE, 2019.

- define a unique tree shape for every set
- use that shape for a Merkle tree of your set
- only exchange fingerprints for ranges that correspond to subtrees

Intermission — Merkle Search Trees

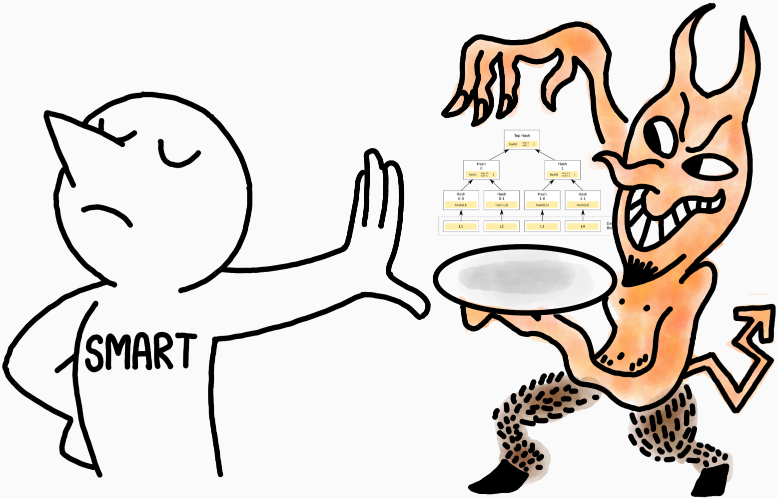
Auvolat, Alex, and François Taïani. "Merkle search trees: Efficient state-based CRDTs in open networks." 2019 38th Symposium on Reliable Distributed Systems (SRDS). IEEE, 2019.

- define a unique tree shape for every set
- use that shape for a Merkle tree of your set
- only exchange fingerprints for ranges that correspond to subtrees

Problems:

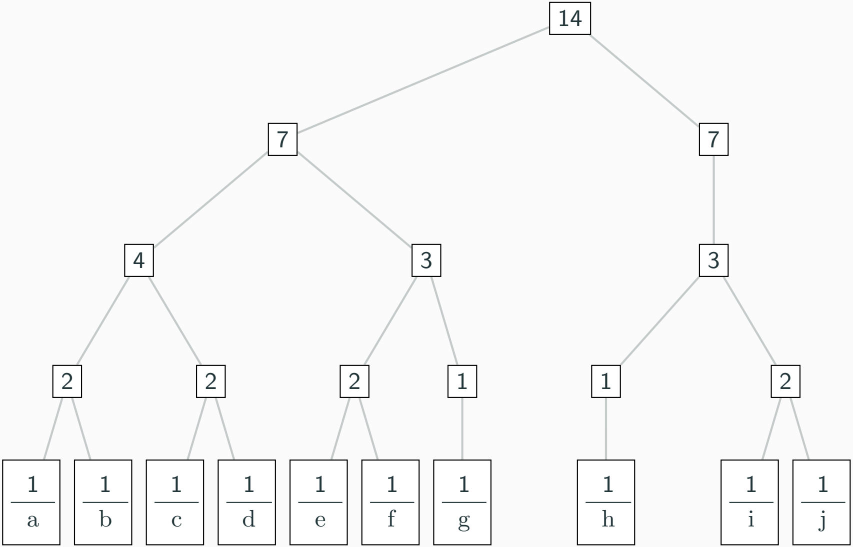
- easily attacked with degenerate tree shapes
- peers cannot optimize data representation for their use-case

Say No to Merkle Trees

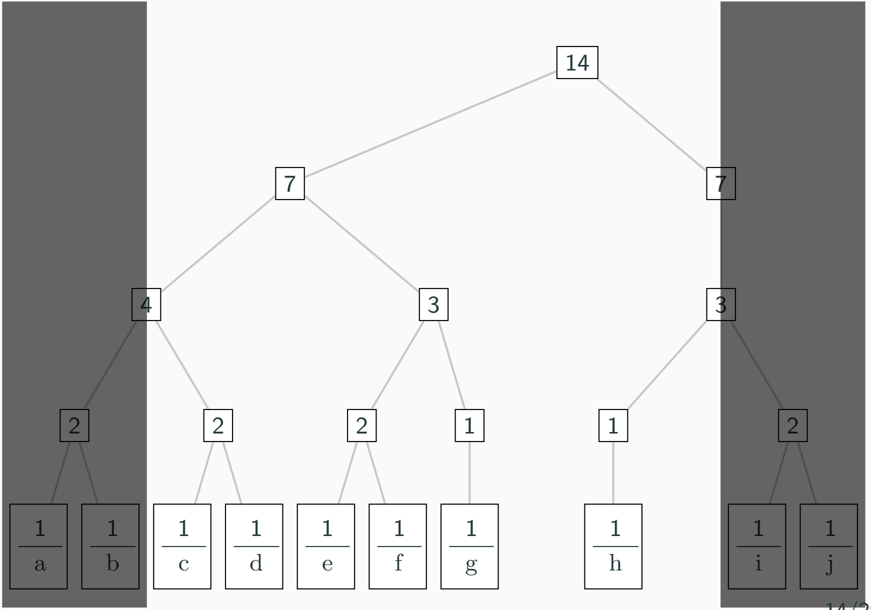


Drawing: Sam Gwilym

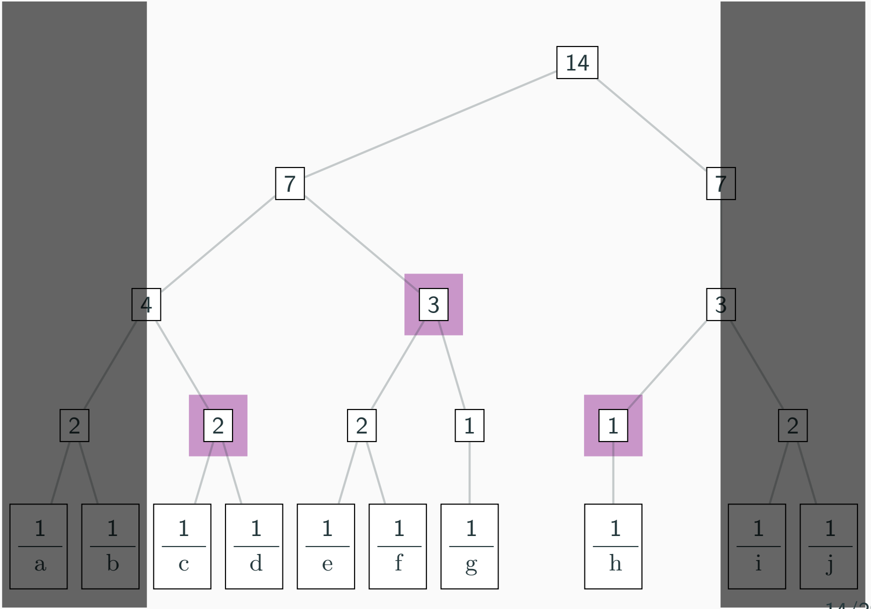
Order-Statistic Trees



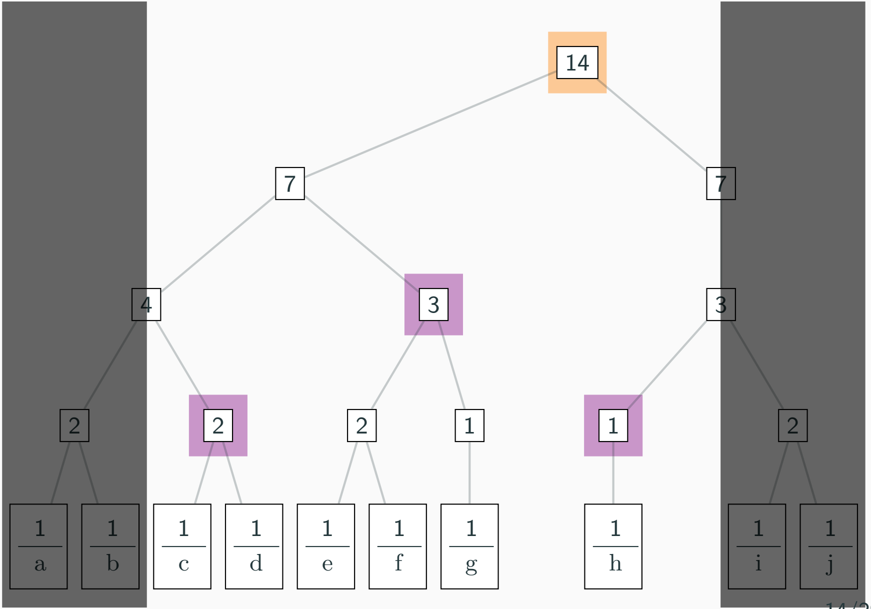
Order-Statistic Trees $[c, i)$



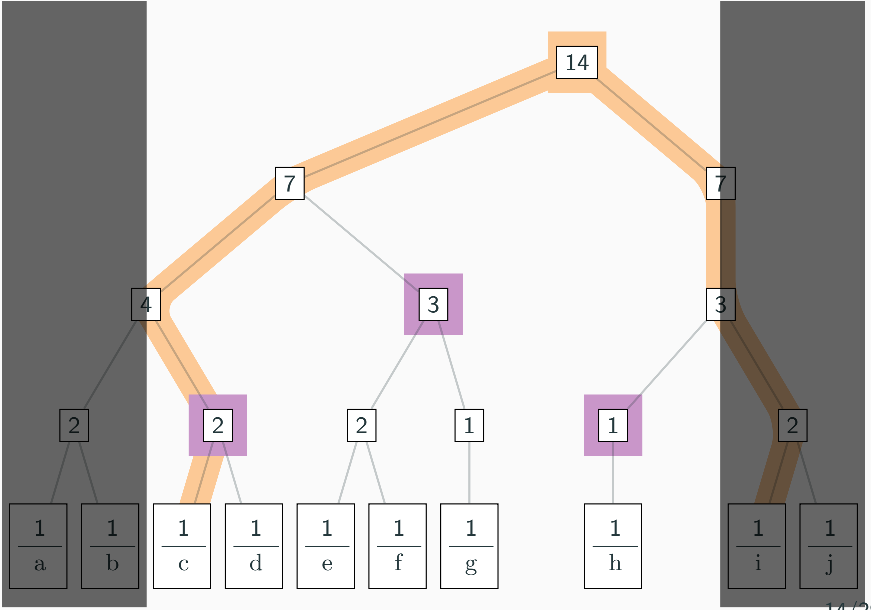
Order-Statistic Trees $[c, i)$



Order-Statistic Trees $[c, i)$



Order-Statistic Trees $[c, i)$



Monoid Trees

- set of labels: $\mathbb{N}$
- binary associative function: $+$
- neutral element: 0

Monoid Trees

Monoid:

- set of labels: $\mathbb{N}$
- binary associative function: $+$
- neutral element: 0

Monoid Trees

Monoid:

- set of labels: $\mathbb{N}$
- binary associative function: $+$
- neutral element: 0
- transforming into the monoid: $\lambda x.1$

Monoid Trees

Monoid:

- set of labels: $\mathbb{N}$
- binary associative function: $+$
- neutral element: 0
- transforming into the monoid: $\lambda x.1$

- set of labels: $\{n : 0 <= n < 2^{256} - 1\}$

- transforming into the monoid: sha256

Monoid Trees

Monoid:

- set of labels: $\mathbb{N}$
- binary associative function: $+$
- neutral element: 0
- transforming into the monoid: $\lambda x.1$

- set of labels: $\{n : 0 <= n < 2^{256} - 1\}$
- binary associative function: xor
- neutral element: 0
- transforming into the monoid: sha256

Resulting Functions

Let U be a set, $\preceq$ a linear order on U, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathsf{h} : U \to M$.

We *lift* h *to finite sets via* $\mathcal{M}$ to obtain $\mathsf{lift}_{\mathsf{h}}^{\mathcal{M}} : \mathcal{P}(U) \rightharpoonup M$ with:

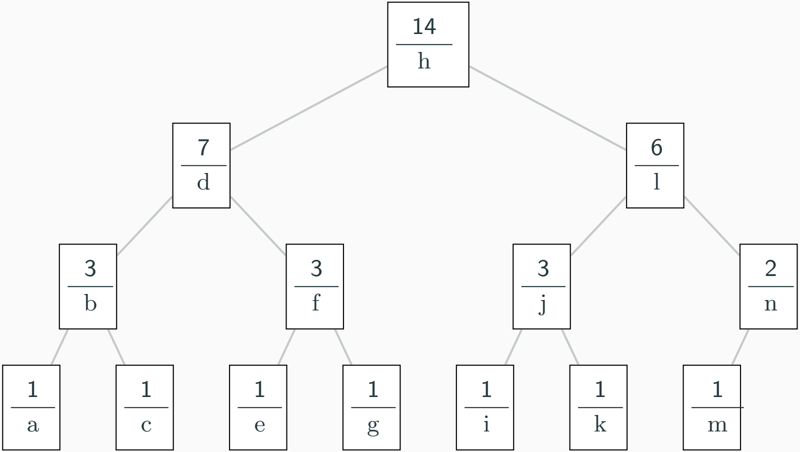
$$\begin{aligned} \mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}(\emptyset) &:= \mathbb{0}, \\ \mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}(S) &:= \mathsf{h}\big(\min(S)\big) \oplus \mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}\big(S \setminus \{\min(S)\}\big). \end{aligned}$$

That is: $\mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}(S) = \mathsf{h}(s_1) \oplus \mathsf{h}(s_2) \oplus \cdots \oplus \mathsf{h}(s_{|S|})$.

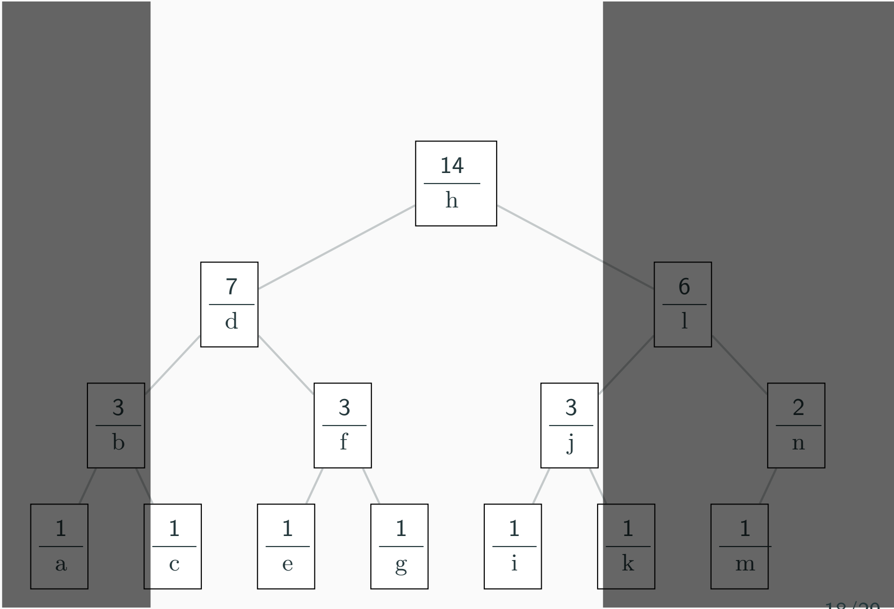
Advantages

- solid worst-case communication complexity
- implementation independence

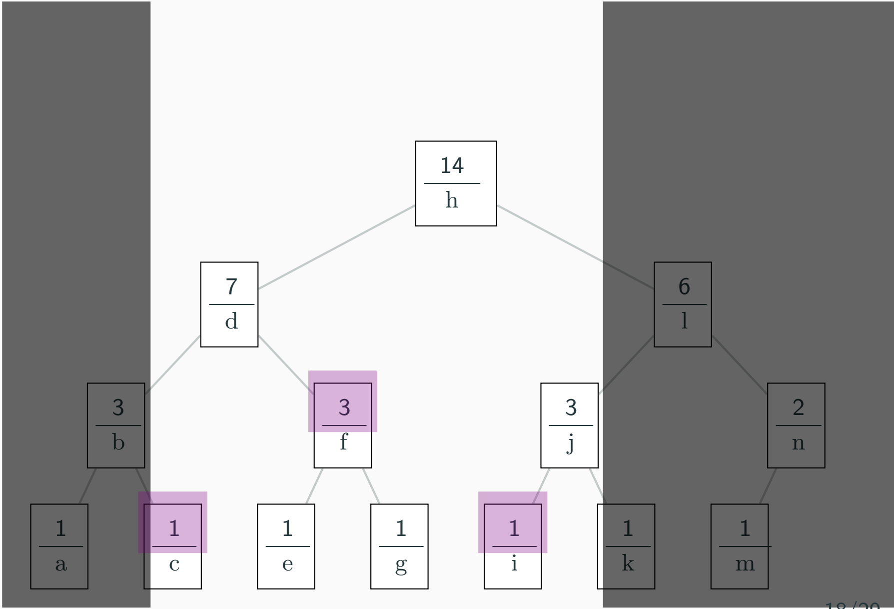
Alternative Datastructures



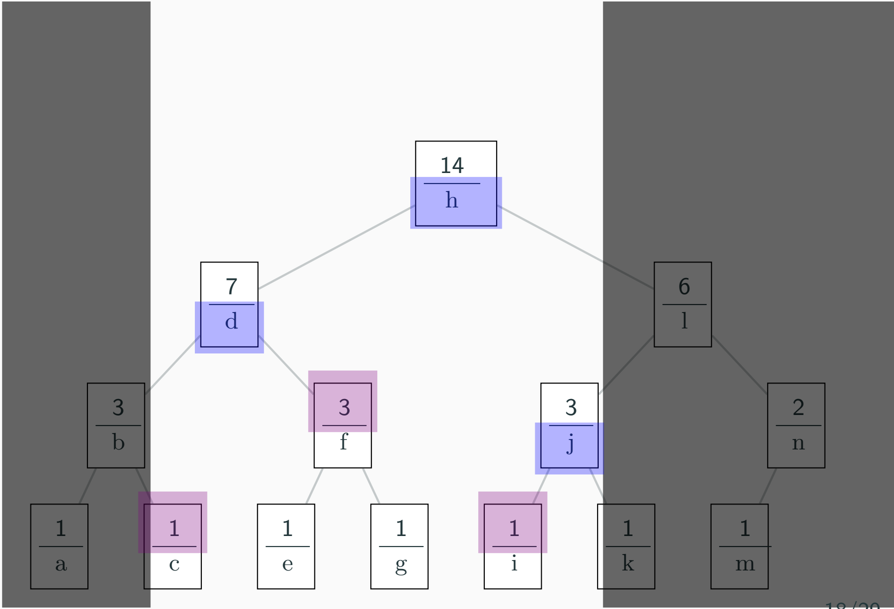
Alternative Datastructures



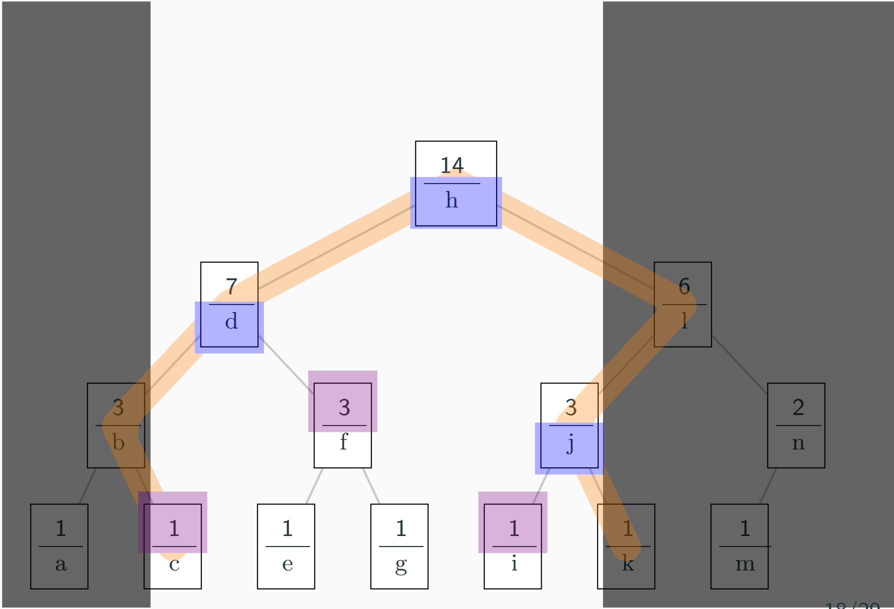
Alternative Datastructures



Alternative Datastructures



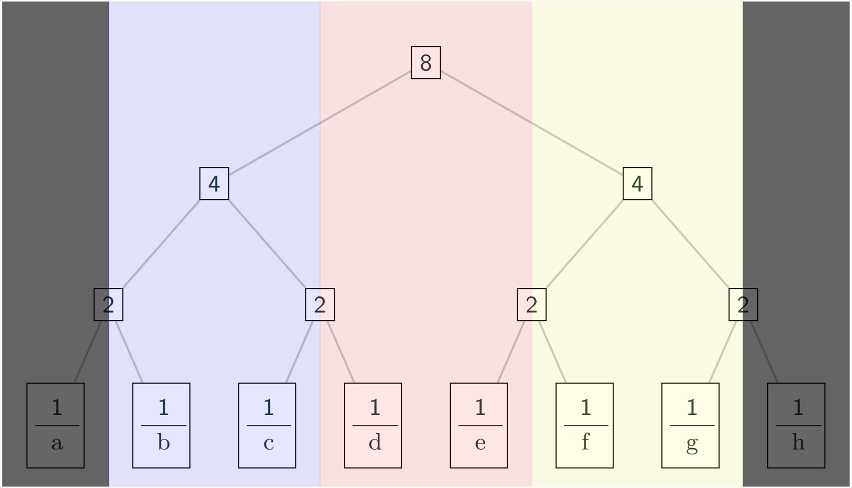
Alternative Datastructures



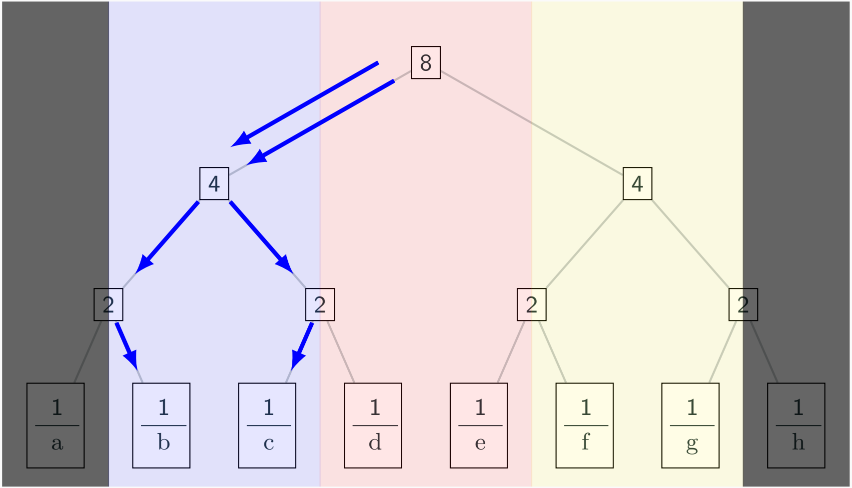
Alternative Datastructures

- monoid B-Tree
- monoid prefix tree
- monoid skip list
- monoid zip-tree
- no datastructure at all
- ...

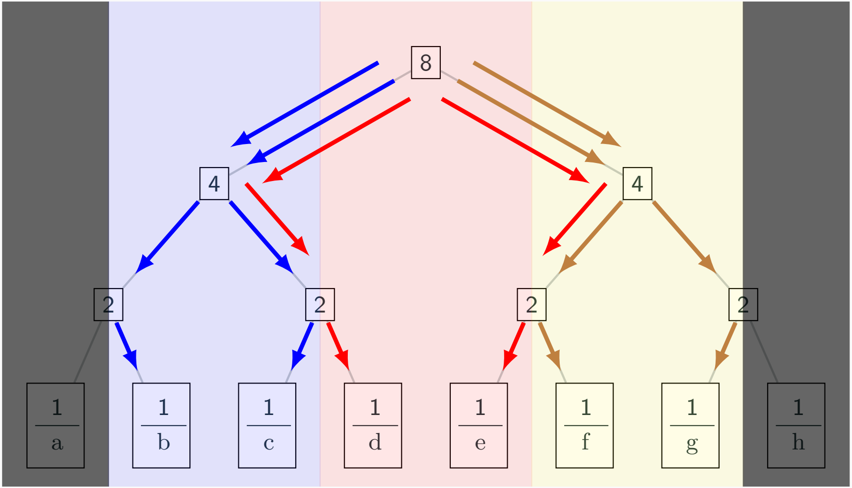
Successive Ranges



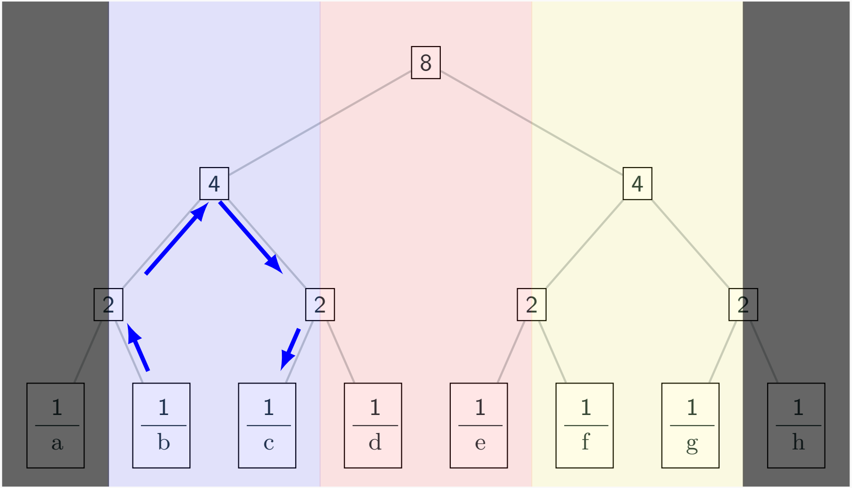
Successive Ranges



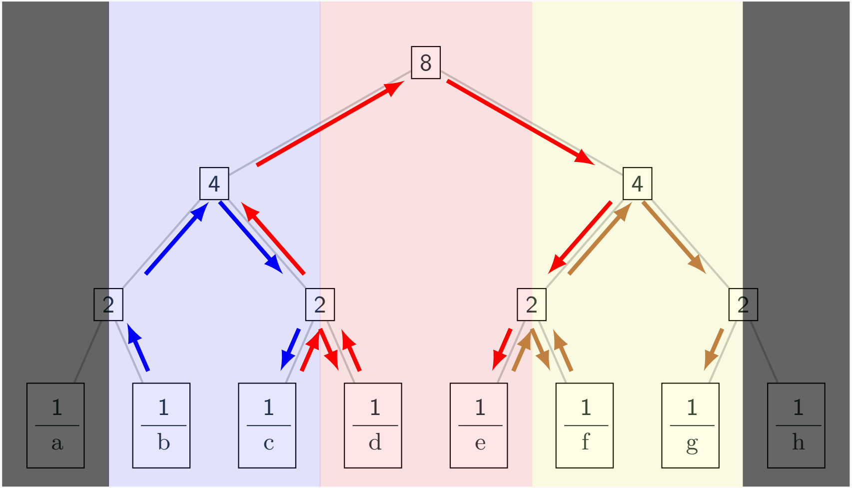
Successive Ranges



Successive Ranges



Successive Ranges



Some Peers Are Out To Get Us

- adversary must sabotage reconciliation

Some Peers Are Out To Get Us

- adversary must sabotage reconciliation
- active and passive adversaries

Some Peers Are Out To Get Us

- adversary must sabotage reconciliation
- active and passive adversaries
- randomized boundaries defeat individual collisions

Some Peers Are Out To Get Us

- adversary must sabotage reconciliation
- active and passive adversaries
- randomized boundaries defeat individual collisions
- better protection: secure hash functions

Secure Hash Functions

Let U be a set, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathsf{f} : \mathcal{P}(U) \to M$.

f is *set-homomorphic* if $\mathsf{f}(U_1 \cup U_2) = \mathsf{f}(U_1) \oplus \mathsf{f}(U_2)$.

Secure Hash Functions

Let U be a set, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathsf{f} : \mathcal{P}(U) \to M$.

f is *set-homomorphic* if $\mathsf{f}(U_1 \cup U_2) = \mathsf{f}(U_1) \oplus \mathsf{f}(U_2)$.

~~xor~~, ~~addition~~, multiplication, lattices, RSA, eliptic curves.

Secure Hash Functions

Let U be a set, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathrm{f} : \mathcal{P}(U) \to M$.

f is *set-homomorphic* if $\mathrm{f}(U_1 \cup U_2) = \mathrm{f}(U_1) \oplus \mathrm{f}(U_2)$.

~~xor~~, ~~addition~~, multiplication, lattices, RSA, eliptic curves.

Our functions: $\mathrm{lift}_{\mathrm{h}}^{\mathcal{M}}(S) = \mathrm{h}(s_1) \oplus \mathrm{h}(s_2) \oplus \cdots \oplus \mathrm{h}(s_{|S|})$

Secure Hash Functions

Let U be a set, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathsf{f} : \mathcal{P}(U) \to M$.

f is *set-homomorphic* if $\mathsf{f}(U_1 \cup U_2) = \mathsf{f}(U_1) \oplus \mathsf{f}(U_2)$.

~~xor~~, ~~addition~~, multiplication, lattices, RSA, eliptic curves.

Our functions: $\mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}(S) = \mathsf{h}(s_1) \oplus \mathsf{h}(s_2) \oplus \cdots \oplus \mathsf{h}(s_{|S|})$

Let further $\preceq$ a linear order on U.

h is a *tree-friendly function* if for $S_0, S_1 \in \mathcal{P}(U)$ with $\max(S_0) \prec \min(S_1)$, we have $\mathsf{h}(S_0 \cup S_1) = \mathsf{h}(S_0) \oplus \mathsf{h}(S_1)$.

Secure Hash Functions

Let U be a set, $\mathcal{M} = (M, \oplus, \mathbb{0})$ a monoid, and $\mathsf{f} : \mathcal{P}(U) \to M$.

f is *set-homomorphic* if $\mathsf{f}(U_1 \cup U_2) = \mathsf{f}(U_1) \oplus \mathsf{f}(U_2)$.

~~xor~~, ~~addition~~, multiplication, lattices, RSA, eliptic curves.

Our functions: $\mathsf{lift}_{\mathsf{h}}^{\mathcal{M}}(S) = \mathsf{h}(s_1) \oplus \mathsf{h}(s_2) \oplus \cdots \oplus \mathsf{h}(s_{|S|})$

Let further $\preceq$ a linear order on U.

h is a *tree-friendly function* if for $S_0, S_1 \in \mathcal{P}(U)$ with $\max(S_0) \prec \min(S_1)$, we have $\mathsf{h}(S_0 \cup S_1) = \mathsf{h}(S_0) \oplus \mathsf{h}(S_1)$.

No commutativity: Cayley hash functions

Summary

- divide-and-conquer to find differences by checking fingerprint equality
- tree-friendly functions allow for efficient and flexible implementation
- cryptographically secure tree-friendly functions exist
- pretty simple compared to traditional solutions

Summary

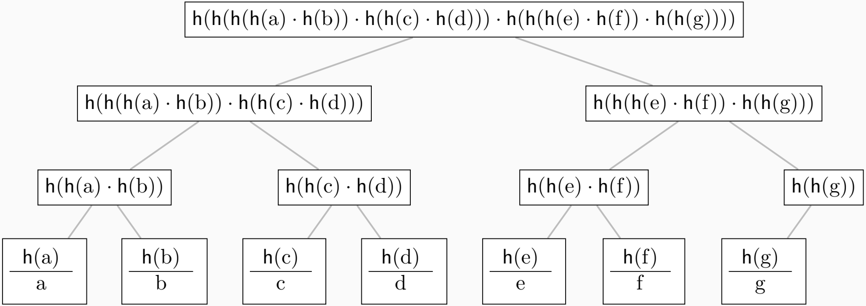
- divide-and-conquer to find differences by checking fingerprint equality
- tree-friendly functions allow for efficient and flexible implementation
- cryptographically secure tree-friendly functions exist
- pretty simple compared to traditional solutions
- remember to say no to Merkle trees

Bonus Slides!

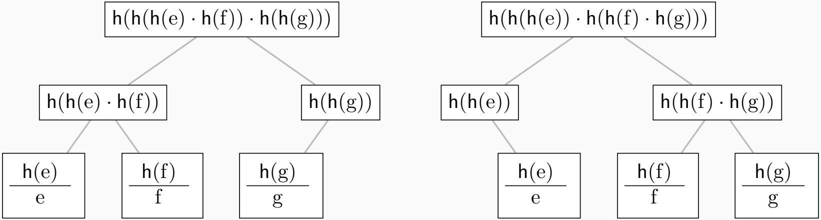
Reducing Computation Times

- Step 1: Put a Merkle tree on it
- Step 2: ???
- Step 3: Profit

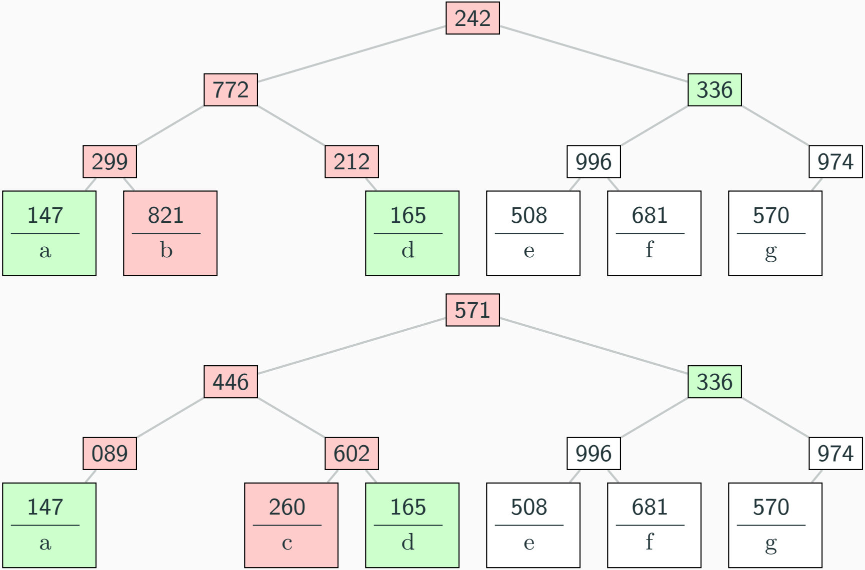
Auvolat, Alex, and François Taïani. "Merkle search trees: Efficient state-based CRDTs in open networks." 2019 38th Symposium on Reliable Distributed Systems (SRDS). IEEE, 2019.


$h(h(h(h(a) \cdot h(b)) \cdot h(h(c) \cdot h(d))) \cdot h(h(h(e) \cdot h(f)) \cdot h(h(g))))$
$h(h(h(a) \cdot h(b)) \cdot h(h(c) \cdot h(d)))$
$h(h(h(e) \cdot h(f)) \cdot h(h(g)))$
$h(h(a) \cdot h(b))$
$h(h(c) \cdot h(d))$
$h(h(e) \cdot h(f))$
$h(h(g))$
$\frac{h(a)}{a}$
$\frac{h(b)}{b}$
$\frac{h(c)}{c}$
$\frac{h(d)}{d}$
$\frac{h(e)}{e}$
$\frac{h(f)}{f}$
$\frac{h(g)}{g}$

Merkle Trees



Merkle Tree Reconciliation



Merkle Tree Reconciliation

- inflexible data representation
- inacceptable worst-case complexity
    - remember, some peers are out to get us