Hypersuccinct Trees

New universal tree source codes for optimal compressed tree data structures and range minima

Sebastian Wild

joint work with Ian Munro, Pat Nicholson, and Louisa Seelbach Benkner

arxiv.org/abs/2104.13457



European Symposium on Algorithms 2021

Outline



1 Three Roots



2 Hypersuccinct Trees



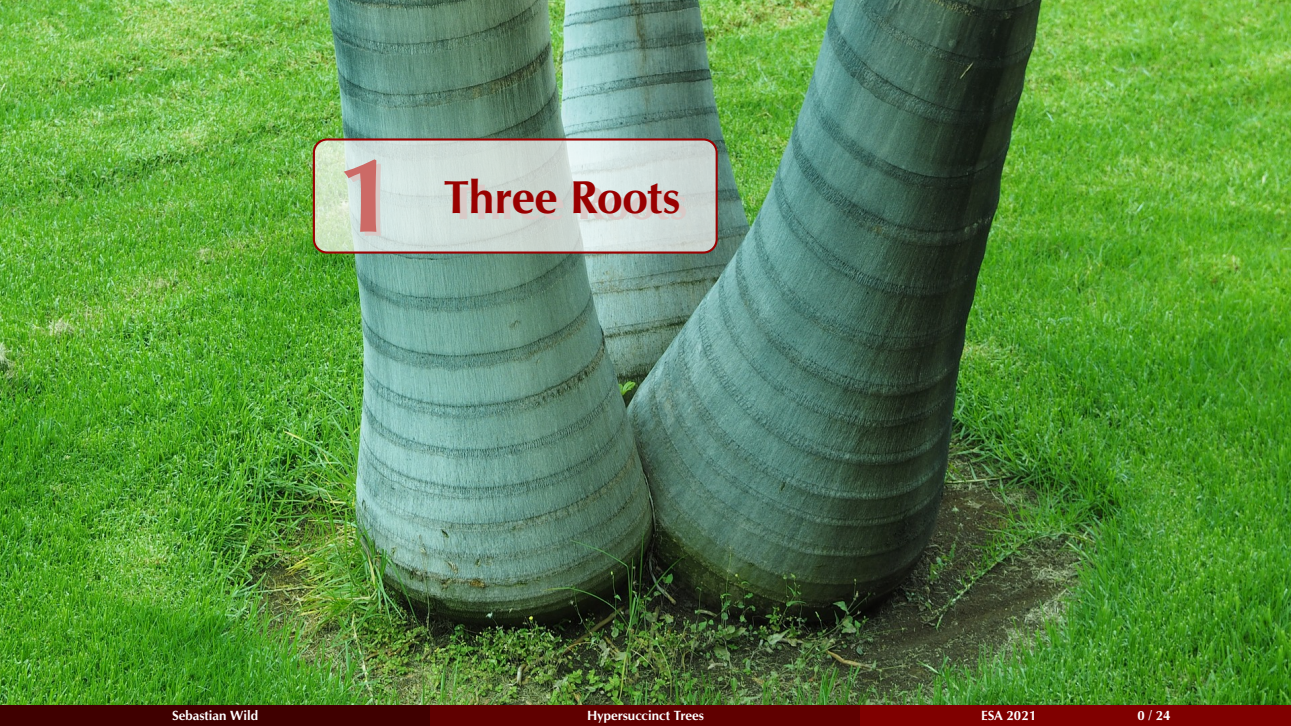
3 Tree Sources



4 Two Favorite Trees



5 Range-Minimum Queries



1 Three Roots

Three roots

Data structures

- *succinct* data structures
- optimal space usage in the **worst case** up to l. o.t.:

  $\lg |\mathcal{U}_n|(1 + o(1))$ bits
- support many **operations** efficiently
- (potentially: update object)

  not today

Information theory

- *universal source code*
- encode random object x generated by ***source*** with few bits:
  - source **entropy** + l. o.t. on average
  - or better: instance-optimal $\lg(1/\mathbb{P}[x])(1 + o(1))$

Analysis of Algorithms

- average-case analysis (+ more)
- **precise** asymptotic approx. for number of objects in a class
- asymptotics for distribution of **parameters**

Three roots

Data structures

- *succinct* data structures
- optimal space usage in the **worst case** up to l. o.t.:

  $\lg |\mathcal{U}_n|(1 + o(1))$ bits
- support many **operations** efficiently
- (potentially: update object)

  not today

Information theory

- *universal source code*
- encode random object x generated by ***source*** with few bits:
  - source **entropy** + l. o.t. on average
  - or better: instance-optimal

    $\lg(1/\mathbb{P}[x])(1 + o(1))$

Analysis of Algorithms

- average-case analysis (+ more)
- **precise** asymptotic approx. for number of objects in a class
- asymptotics for distribution of **parameters**

Three roots

Data structures

- *succinct* data structures
- optimal space usage in the **worst case** up to l. o.t.:

  $\lg |\mathcal{U}_n|(1 + o(1))$ bits
- support many **operations** efficiently
- (potentially: update object)

  not today

Information theory

- *universal source code*
- encode random object x generated by ***source*** with few bits:
  - source **entropy** + l. o.t. on average
  - or better: instance-optimal

    $\lg(1/\mathbb{P}[x])(1 + o(1))$

Analysis of Algorithms

- average-case analysis (+ more)
- **precise** asymptotic approx. for number of objects in a class
- asymptotics for distribution of **parameters**

Three roots

Data structures

- *succinct* data structures
- optimal space usage in the **worst case** up to l. o.t.:

  $\lg |\mathcal{U}_n|(1 + o(1))$ bits
- support many **operations** efficiently
- (potentially: update object)

  not today

Information theory

- *universal source code*
- encode random object x generated by ***source*** with few bits:
  - source **entropy** + l. o.t. on average
  - or better: instance-optimal

    $\lg(1/\mathbb{P}[x])(1 + o(1))$

Analysis of Algorithms

- average-case analysis (+ more)
- **precise** asymptotic approx. for number of objects in a class
- asymptotics for distribution of **parameters**

Hypersuccinct trees

A *single*, simple code for binary trees (hypersuccinct code)

that can be augmented to support all queries of the best succinct trees in O(1) time,

simultaneously achieves optimal compression up to l. o.t. for all tree sources for which *any* universal source code is known,

building on precise analysis of trees and their properties.

Outline



1 Three Roots



2 Hypersuccinct Trees



3 Tree Sources



4 Two Favorite Trees



5 Range-Minimum Queries



2 Hypersuccinct Trees

Succinct Binary Trees

What is known?

a.k.a. plane trees — e.g. binary

- data structure for ordinal or cardinal trees
- $2n + o(n)$ bits of space
  - optimal in worst case
  - some isolated works on better space for restricted scenarios
  - but tailored approaches for each tree distribution
- supports huge list of operations in $O(1)$ time
  on a standard word-RAM
- several competing approaches (BP, DFUDS, TC)
  (largely incompatible with each other)

Operations in Tree Covering

`parent(`v`)`	the parent of v, same as `anc(`v`, 1)`
`degree(`v`)`	the number of children of v
`left_child(`v`)`	the left child of node v
`right_child(`v`)`	the right child of node v
`depth(`v`)`	the depth of v, i.e., the number of edges between the root and v
`anc(`v`, `i`)`	the ancestor of node v at depth `depth(`v`)` $-$ i
`subtree_size(`v`)`	the number of descendants of v
`height(`v`)`	the height of the subtree rooted at node v
`LCA(`v`, `u`)`	the lowest common ancestor of nodes u and v
`leftmost_leaf(`v`)`	the leftmost leaf descendant of v
`rightmost_leaf(`v`)`	the rightmost leaf descendant of v
`level_leftmost(`ℓ`)`	the leftmost node on level ℓ
`level_rightmost(`ℓ`)`	the rightmost node on level ℓ
`level_predecessor(`v`)`	the node immediately to the left of v on the same level
`level_successor(`v`)`	the node immediately to the right of v on the same level
`node_rank`$_X$`(`v`)`	the position of v in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$, i.e., in a preorder, postorder, or inorder traversal
`node_select`$_X$`(`i`)`	the ith node in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$
`leaf_rank(`v`)`	the number of leaves before and including v in preorder
`leaf_select(`i`)`	the ith leaf in preorder

Succinct Binary Trees

What is known?

a.k.a. plane trees — e.g. binary

- data structure for ordinal or cardinal trees
- $2n + o(n)$ bits of space
  - optimal in worst case
  - some isolated works on better space for restricted scenarios
  - but tailored approaches for each tree distribution
- supports huge list of operations in $O(1)$ time
  on a standard word-RAM
- several competing approaches (BP, DFUDS, TC)
  (largely incompatible with each other)

Operations in Tree Covering

`parent(`v`)`	the parent of v, same as `anc(`v`, 1)`
`degree(`v`)`	the number of children of v
`left_child(`v`)`	the left child of node v
`right_child(`v`)`	the right child of node v
`depth(`v`)`	the depth of v, i.e., the number of edges between the root and v
`anc(`v`,` i`)`	the ancestor of node v at depth `depth(`v`)` $-$ i
`subtree_size(`v`)`	the number of descendants of v
`height(`v`)`	the height of the subtree rooted at node v
`LCA(`v`,` u`)`	the lowest common ancestor of nodes u and v
`leftmost_leaf(`v`)`	the leftmost leaf descendant of v
`rightmost_leaf(`v`)`	the rightmost leaf descendant of v
`level_leftmost(`ℓ`)`	the leftmost node on level ℓ
`level_rightmost(`ℓ`)`	the rightmost node on level ℓ
`level_predecessor(`v`)`	the node immediately to the left of v on the same level
`level_successor(`v`)`	the node immediately to the right of v on the same level
`node_rank`$_X$`(`v`)`	the position of v in the X-order, $X \in$ {`PRE`, `POST`, `IN`}, i.e., in a preorder, postorder, or inorder traversal
`node_select`$_X$`(`i`)`	the ith node in the X-order, $X \in$ {`PRE`, `POST`, `IN`}
`leaf_rank(`v`)`	the number of leaves before and including v in preorder
`leaf_select(`i`)`	the ith leaf in preorder

Succinct Binary Trees

What is known?

a.k.a. plane trees — e.g. binary

- data structure for ordinal or cardinal trees
- $2n + o(n)$ bits of space
  - optimal in worst case
  - some isolated works on better space for restricted scenarios
  - but tailored approaches for each tree distribution
- supports huge list of operations in $O(1)$ time
  on a standard word-RAM
- several competing approaches (BP, DFUDS, TC)
  (largely incompatible with each other)

Operations in Tree Covering

`parent`(v)	the parent of v, same as `anc`$(v, 1)$
`degree`(v)	the number of children of v
`left_child`(v)	the left child of node v
`right_child`(v)	the right child of node v
`depth`(v)	the depth of v, i.e., the number of edges between the root and v
`anc`(v, i)	the ancestor of node v at depth `depth`$(v) - i$
`subtree_size`(v)	the number of descendants of v
`height`(v)	the height of the subtree rooted at node v
`LCA`(v, u)	the lowest common ancestor of nodes u and v
`leftmost_leaf`(v)	the leftmost leaf descendant of v
`rightmost_leaf`(v)	the rightmost leaf descendant of v
`level_leftmost`(ℓ)	the leftmost node on level ℓ
`level_rightmost`(ℓ)	the rightmost node on level ℓ
`level_predecessor`(v)	the node immediately to the left of v on the same level
`level_successor`(v)	the node immediately to the right of v on the same level
`node_rank`$_X(v)$	the position of v in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$, i.e., in a preorder, postorder, or inorder traversal
`node_select`$_X(i)$	the ith node in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$
`leaf_rank`(v)	the number of leaves before and including v in pre-order
`leaf_select`(i)	the ith leaf in preorder

Succinct Binary Trees

What is known?

a.k.a. plane trees — e.g. binary

- data structure for ordinal or cardinal trees
- $2n + o(n)$ bits of space
  - optimal in worst case
  - some isolated works on better space for restricted scenarios
  - but tailored approaches for each tree distribution
- supports huge list of operations in $O(1)$ time
  on a standard word-RAM
- several competing approaches (BP, DFUDS, TC)
  (largely incompatible with each other)

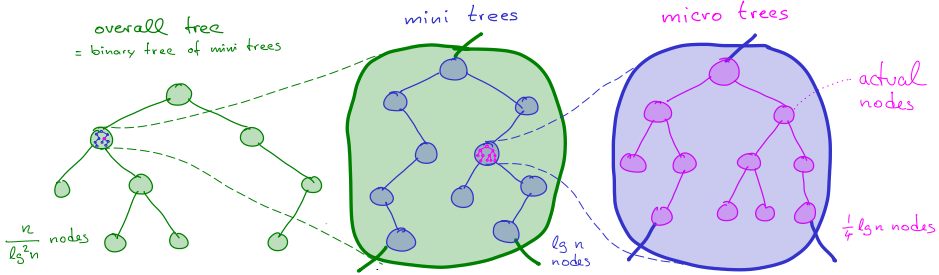
Operations in Tree Covering

`parent`(v)	the parent of v, same as `anc`$(v, 1)$
`degree`(v)	the number of children of v
`left_child`(v)	the left child of node v
`right_child`(v)	the right child of node v
`depth`(v)	the depth of v, i.e., the number of edges between the root and v
`anc`(v, i)	the ancestor of node v at depth `depth`$(v) - i$
`subtree_size`(v)	the number of descendants of v
`height`(v)	the height of the subtree rooted at node v
`LCA`(v, u)	the lowest common ancestor of nodes u and v
`leftmost_leaf`(v)	the leftmost leaf descendant of v
`rightmost_leaf`(v)	the rightmost leaf descendant of v
`level_leftmost`(ℓ)	the leftmost node on level ℓ
`level_rightmost`(ℓ)	the rightmost node on level ℓ
`level_predecessor`(v)	the node immediately to the left of v on the same level
`level_successor`(v)	the node immediately to the right of v on the same level
`node_rank`$_X(v)$	the position of v in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$, i.e., in a preorder, postorder, or inorder traversal
`node_select`$_X(i)$	the ith node in the X-order, $X \in \{$`PRE`, `POST`, `IN`$\}$
`leaf_rank`(v)	the number of leaves before and including v in pre-order
`leaf_select`(i)	the ith leaf in preorder

Tree-Covering Data Structures

Key idea:

- **decompose** tree into ***mini trees*** and mini trees into ***micro trees***

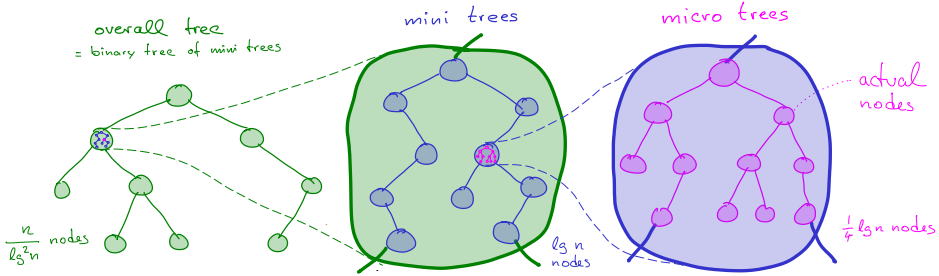


- within $o(n)$ space:
  can store $\tilde{O}(\log n)$ bits per mini tree
  and $\tilde{O}(\log \log n)$ bits per micro tree
  $\rightsquigarrow$ enough to support many operations
  (most comprehensive of all succinct trees)
- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table ("4 Russians technique")

Tree-Covering Data Structures

Key idea:

- **decompose** tree into ***mini trees*** and mini trees into ***micro trees***

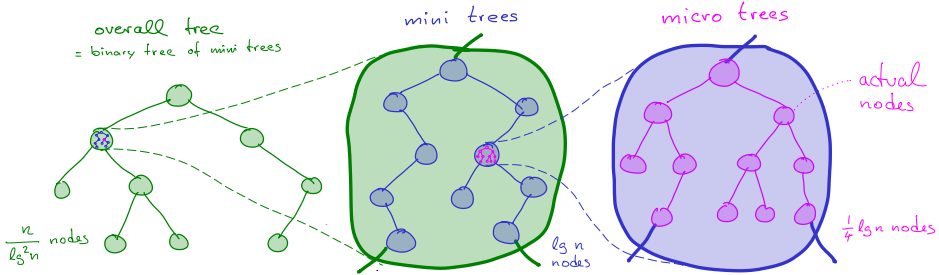


- within $o(n)$ space:
  can store $\tilde{O}(\log n)$ bits per mini tree
  and $\tilde{O}(\log \log n)$ bits per micro tree
  $\rightsquigarrow$ enough to support many operations
  most comprehensive of all succinct trees
- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table ("4 Russians technique")

Tree-Covering Data Structures

Key idea:

- **decompose** tree into ***mini trees*** and mini trees into ***micro trees***



- within $o(n)$ space:
  can store $\tilde{O}(\log n)$ bits per mini tree
  and $\tilde{O}(\log\log n)$ bits per micro tree
  $\rightsquigarrow$ enough to support many operations

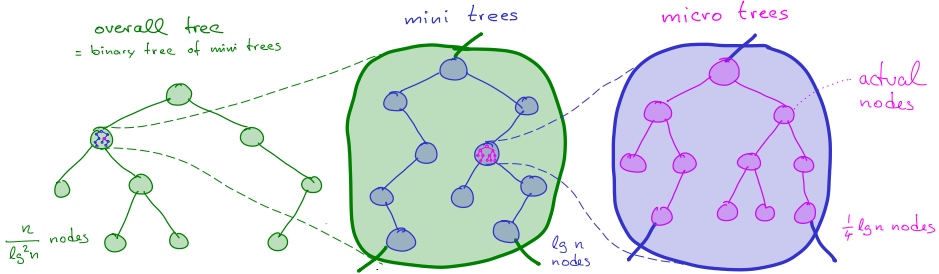
  most comprehensive of all succinct trees

- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table ("4 Russians technique")

Tree-Covering Data Structures

Key idea:

- **decompose** tree into ***mini trees*** and mini trees into ***micro trees***

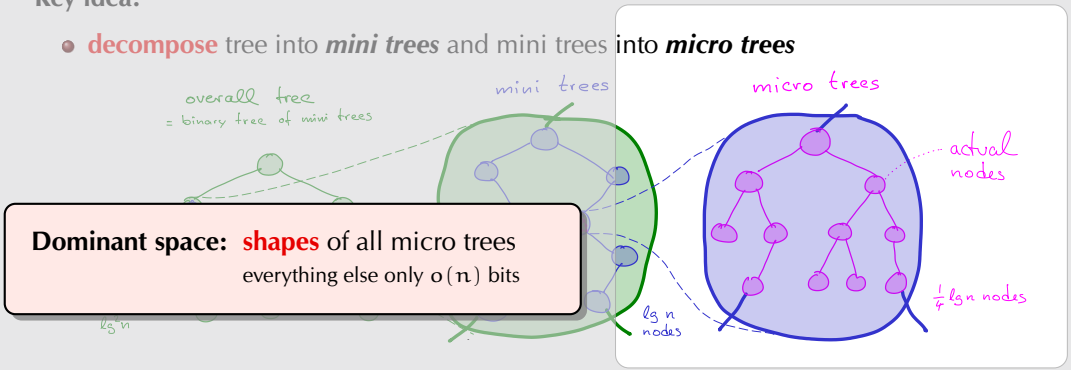


- within $o(n)$ space:
  can store $\tilde{O}(\log n)$ bits per mini tree
  and $\tilde{O}(\log\log n)$ bits per micro tree
  $\rightsquigarrow$ enough to support many operations
  most comprehensive of all succinct trees
- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table ("4 Russians technique")

Tree-Covering Data Structures

Key idea:

- **decompose** tree into ***mini trees*** and mini trees into ***micro trees***



Dominant space: **shapes** of all micro trees
everything else only $o(n)$ bits

- within $o(n)$ space:
  can store $\tilde{O}(\log n)$ bits per mini tree
  and $\tilde{O}(\log\log n)$ bits per micro tree
  $\rightsquigarrow$ enough to support many operations

most comprehensive of all succinct trees

- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table ("4 Russians technique")

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \dots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect $\quad (o(n)$ bits$)$
2. Huffman codes $C(\mu_i)$ of all micro trees

$$\rightsquigarrow \quad |H(t)| \;=\; \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \ldots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect $\quad (o(n)$ bits$)$
2. Huffman codes $C(\mu_i)$ of all micro trees

$$\rightsquigarrow \quad |H(t)| \;=\; \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \ldots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect ($o(n)$ bits)
2. Huffman codes $C(\mu_i)$ of all micro trees

$$\rightsquigarrow \quad |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \dots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect ($o(n)$ bits)
2. Huffman codes $C(\mu_i)$ of all micro trees

$$\rightsquigarrow \quad |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

- **A** n and m (Elias gamma code)
- **B** balanced-parenthesis (BP) bitstring for Υ ($2m$ bits).
- **C** Huffman code for $\mu_1, \dots, \mu_m$:
  list of codewords and corresponding trees (size + BP)
- **D** position of portals in micro trees ($2\ O(\log\log n)$-bit integers per μ_i)

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \ldots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect ($o(n)$ bits)
2. Huffman codes $C(\mu_i)$ of all micro trees

- **A** n and m (Elias gamma code)
- **B** balanced-parenthesis (BP) bitstring for Υ ($2m$ bits).
- **C** Huffman code for $\mu_1, \ldots, \mu_m$: list of codewords and corresponding trees (size + BP)
- **D** position of portals in micro trees (2 $O(\log\log n)$-bit integers per μ_i)

$$\rightsquigarrow \quad |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

Hypersuccinct code

Essence of tree covering data structure yields simple code for binary trees!

Given a binary tree t with micro trees $\mu_1, \ldots, \mu_m$.

Hypersuccinct code $H(t)$ stores

1. How micro trees connect ($o(n)$ bits)
2. Huffman codes $C(\mu_i)$ of all micro trees

- **A** n and m (Elias gamma code)
- **B** balanced-parenthesis (BP) bitstring for Υ ($2m$ bits).
- **C** Huffman code for $\mu_1, \ldots, \mu_m$: list of codewords and corresponding trees (size + BP)
- **D** position of portals in micro trees ($2\ O(\log\log n)$-bit integers per μ_i)

$$\rightsquigarrow \quad |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$

Farzan-Munro Algorithm



Farzan-Munro Algorithm

How to (best) decompose a binary tree?

Farzan-Munro Algorithm

- Recursively: components C_1, C_2 for left and right child
- u_1 and u_2 light? $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- u_1 and u_2 heavy? $\rightsquigarrow$ $C = \{v\}$, C_1, C_2, all marked permanent
- u_1 heavy and u_2 light?
  - C_1 permanent? $\rightsquigarrow$ $C = \{v\} \cup C_2$
  - otherwise $\rightsquigarrow$ $|C_1| < B$ $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- If $|C| \geqslant B$, mark it as permanent.
  Return C.

Definition: v is heavy $\iff$ `subtree_size`$(v) \geqslant B$

Farzan-Munro Algorithm

How to (best) decompose a binary tree?

Farzan-Munro Algorithm

- Recursively: components C_1, C_2 for left and right child
- u_1 and u_2 light? $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- u_1 and u_2 heavy? $\rightsquigarrow$ $C = \{v\}$, C_1, C_2, all marked permanent
- u_1 heavy and u_2 light?
  - C_1 permanent? $\rightsquigarrow$ $C = \{v\} \cup C_2$
  - otherwise $\rightsquigarrow$ $|C_1| < B$ $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- If $|C| \geqslant B$, mark it as permanent.
  Return C.

Definition: v is heavy $\iff$ `subtree_size`$(v) \geqslant B$

Farzan-Munro Algorithm

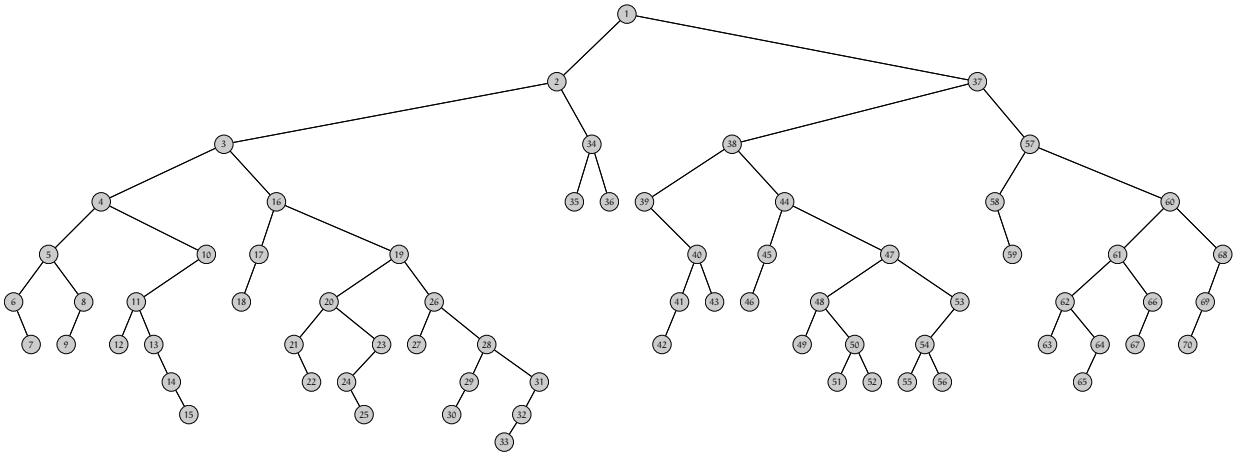
How to (best) decompose a binary tree?

Farzan-Munro Algorithm

- Recursively: components C_1, C_2 for left and right child
- u_1 and u_2 light? $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- u_1 and u_2 heavy? $\rightsquigarrow$ $C = \{v\}$, C_1, C_2, all marked permanent
- u_1 heavy and u_2 light?
  - C_1 permanent? $\rightsquigarrow$ $C = \{v\} \cup C_2$
  - otherwise $\rightsquigarrow$ $|C_1| < B$ $\rightsquigarrow$ $C = \{v\} \cup C_1 \cup C_2$
- If $|C| \geqslant B$, mark it as permanent.
  Return C.

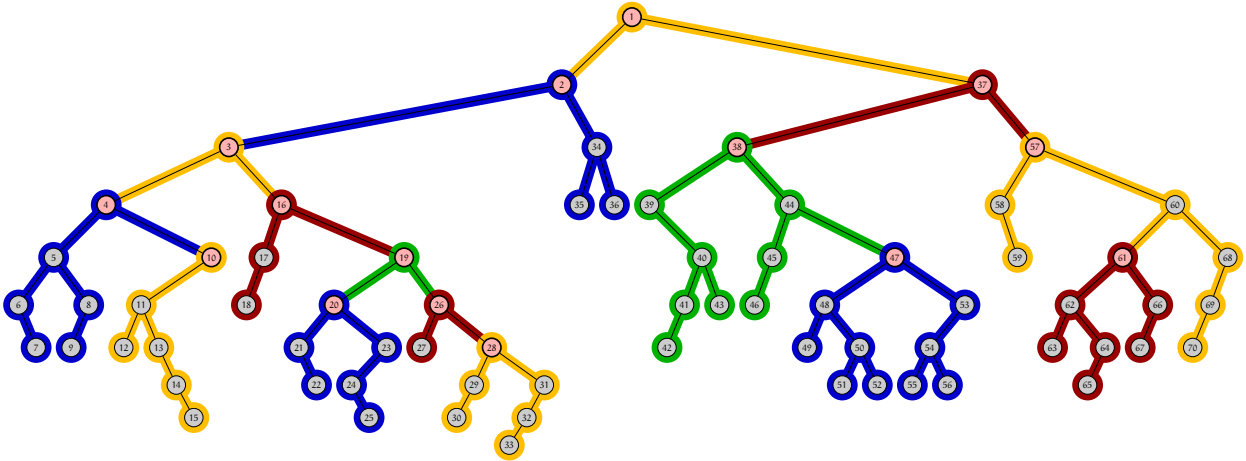
Definition: v is heavy $\iff$ `subtree_size`$(v) \geqslant B$

Example Partitioning



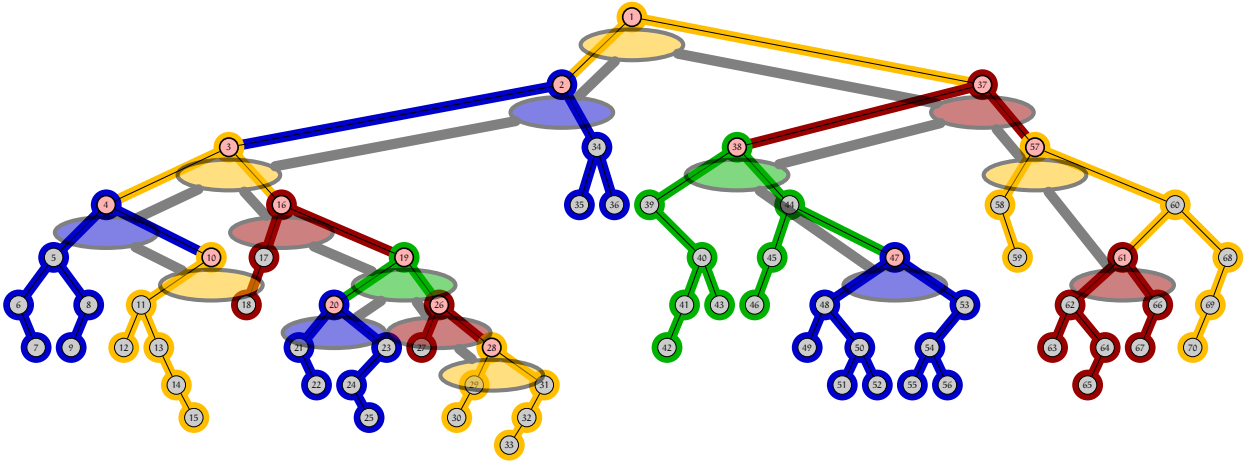
$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Example Partitioning



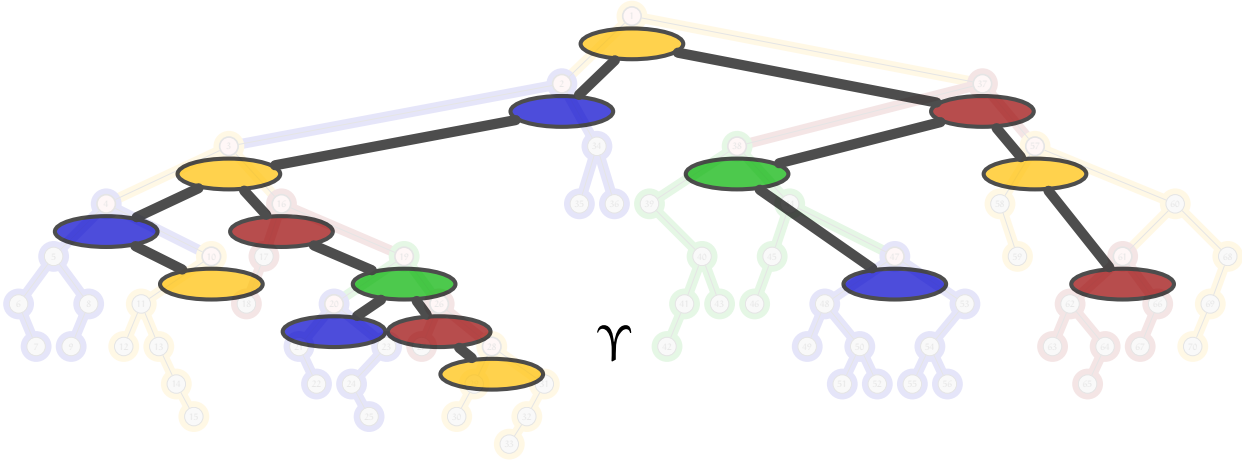
$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Example Partitioning



$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Example Partitioning

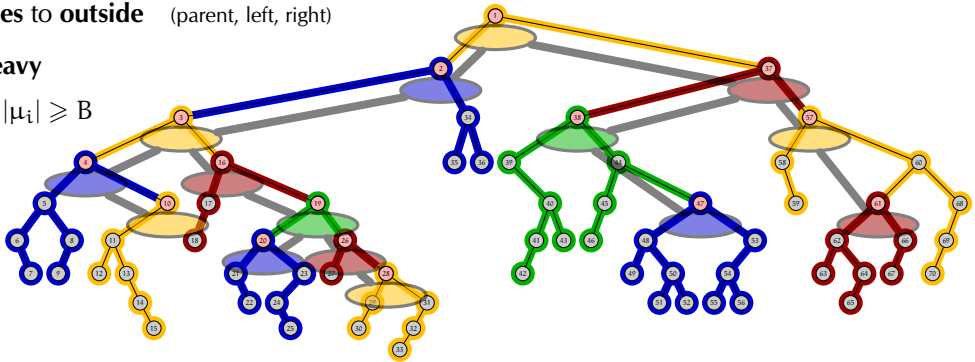


$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Properties of Micro Trees

Binary tree t on n nodes is decomposed into $\mu_1, \ldots, \mu_m$

- (a) $m = O(n/B)$ (few micro trees)
- (b) $|\mu_i| = O(B)$ (all small)
- (c) $|\mu_1| + \cdots + |\mu_m| = n$ (partition vertices)
- (d) μ_i has $\leqslant 3$ **edges** to **outside** (parent, left, right)
- (e) **root** of μ_i is **heavy**
- (f) μ_i **fringe** $\rightsquigarrow$ $|\mu_i| \geqslant B$



Outline



1 Three Roots



2 Hypersuccinct Trees



3 Tree Sources



4 Two Favorite Trees



5 Range-Minimum Queries



3 Tree Sources

Universal Codes

Information theory

- Study **family** of sources (e.g., memoryless sources for text, Markov sources)
- within that family: try to find ***universal codes*** (e.g., Lempel-Ziv compression)
  - matches entropy of source up to l.o.t.
  - without knowing source
  - ⇝ widely applicable compression method

⇝ *(Often) (relatively) simple algorithms whose analysis isn't.*

Universal Codes

Information theory

- Study **family** of sources (e.g., memoryless sources for text, Markov sources)
- within that family: try to find ***universal codes*** (e.g., Lempel-Ziv compression)
  - matches entropy of source up to l.o.t.
  - without knowing source
  - ⇝ widely applicable compression method

⇝ *(Often) (relatively) simple algorithms whose analysis isn't.*

Universal Codes

Information theory

- Study **family** of sources (e.g., memoryless sources for text, Markov sources)
- within that family: try to find ***universal codes*** (e.g., Lempel-Ziv compression)
  - matches entropy of source up to l.o.t.
  - without knowing source
  - $\rightsquigarrow$ widely applicable compression method

$\rightsquigarrow$ *(Often) (relatively) simple algorithms whose analysis isn't.*

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ $=$ prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ $=$ probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad \text{type}(v) \in \{$ $\}$
- *k-th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size(v.left)}, \texttt{subtree\_size(v.right)})$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ = prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ = probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad \text{type}(v) \in \{$ $\}$
- *kth-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size(v.left)}, \texttt{subtree\_size(v.right)})$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ = prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow \mathbb{P}_{\mathcal{S}}[t]$ = probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad \text{type}(v) \in \{$ $\}$
- *k-th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow \mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size(v.left), subtree\_size(v.right)})$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ $=$ prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ $=$ probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad$ $\text{type}(v) \in \{$  $\}$
- k*th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size}(v.\texttt{left}), \texttt{subtree\_size}(v.\texttt{right}))$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ $=$ prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ $=$ probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad$ $\text{type}(v) \in \{$  $\}$
- k*th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size}(v.\texttt{left}), \texttt{subtree\_size}(v.\texttt{right}))$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ $=$ prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ $=$ probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad$ $\text{type}(v) \in \{$  $\}$
- k*th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size}(v.\texttt{left}), \texttt{subtree\_size}(v.\texttt{right}))$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees

Binary Tree Sources

(Binary) tree source $\mathcal{S}$ $=$ prob. distribution over tree shapes with a filter, e.g., tree of size n

$\rightsquigarrow$ $\mathbb{P}_{\mathcal{S}}[t]$ $=$ probability that $\mathcal{S}$ emits t

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\quad$ $\text{type}(v) \in \{$  $\}$
- k*th-order type process:* type prob. depends on types of k ancestors
- ***fixed-size** source:* for target size n, draw subtree sizes of root from given distribution

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size}(v.\texttt{left}), \texttt{subtree\_size}(v.\texttt{right}))$
- ***fixed-height** source:* same with height of tree
- ***uniform subclass** source:* uniform distribution over subclass of trees



Universal codes can't exist in full generality!

can be different for every n!

Tame Binary Tree Sources

Family of sources	Restriction	Redundancy
Memoryless node-type	—	$O(n \log\log n / \log n)$
kth-order node-type	—	$O((nk + n\log\log n)/\log n)$
Monotonic fixed-size	$p(\ell, r) \geqslant p(\ell+1, r)$ and $p(\ell, r) \geqslant p(\ell, r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n / \log n)$
Worst-case fringe-dominated fixed-size	$n_{\geqslant B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$; $n_{\geqslant B}(t) =$ #nodes with subtree size in $\Omega(\log n)$	$O(n_{\geqslant B}(t)\log\log n + n\log\log n/\log n)$
Weight-balanced fixed-size	$\sum_{\frac{n}{c} \leqslant \ell \leqslant n-\frac{n}{c}} p(\ell-1, n-\ell-1) = 1$ for constant $c \geqslant 3$	$O(n \log\log n / \log n)$
Average-case fringe-dominated fixed-size	$\mathbb{E}[n_{\geqslant B}(T)] = o(n/\log\log n)$ for random T generated by source $\mathcal{S}$	$O(n_{\geqslant B}(t)\log\log n + n\log\log n/\log n)$
Monotonic fixed-height	$p(\ell, r) \geqslant p(\ell+1, r)$ and $p(\ell, r) \geqslant p(\ell, r+1)$ for all $\ell, r \in \mathbb{N}_0$	$O(n \log\log n / \log n)$
Worst-case fringe-dominated fixed-height	$n_{\geqslant B}(t) = o(n/\log\log n)$ for all t with $\mathbb{P}[t] > 0$	$O(n_{\geqslant B}(t)\log\log n + n\log\log n/\log n)$
Tame uniform-subclass	class of trees $\mathcal{T}_n(\mathcal{P})$ is hereditary (i.e., closed under taking subtrees), $n_{\geqslant B}(t) = o(n/\log\log n)$ for $t \in \mathcal{T}_n(\mathcal{P})$, $\lg\lvert\mathcal{T}_n(\mathcal{P})\rvert = cn + o(n)$ for constant $c > 0$, heavy-twigged: if v has subtree size $\Omega(\log n)$, v's subtrees have size $\omega(1)$	$o(n)$

Optimally compressed binary tree distributions

Tree-Shape Distribution	Entropy	Corresponding Source
(Uniformly random) binary trees of size n	$2n$	Memoryless binary, monotonic fixed-size binary
(Uniformly random) full binary trees of size n	n	Memoryless binary
(Uniformly random) unary paths of length n	n	Memoryless binary
(Uniformly random) Motzkin trees of size n	$1.585n$	Memoryless binary
BSTs generated by insertions in random order	$1.736n$	Monotonic fixed-size binary
Binomial random trees	$P(\lg n)n$ [a)]	Average-case fringe-dominated fixed-size binary
Almost paths	— [b)]	Monotonic fixed-size binary
Random fringe-balanced binary search trees	— [b)]	Average-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of height h	— [b)]	Worst-case fringe-dominated fixed-height binary
(Uniformly random) weight-balanced binary trees of size n	— [b)]	Worst-case fringe-dominated fixed-size binary
(Uniformly random) AVL trees of size n	$0.938n$	Uniform-subclass
(Uniformly random) left-leaning red-black trees of size n	$0.879n$	Uniform-subclass

a) Here P is a nonconstant, continuous, periodic function with period 1.

b) No (concise) asymptotic approximation known.

Outline



1 Three Roots



2 Hypersuccinct Trees



3 Tree Sources



4 Two Favorite Trees



5 Range-Minimum Queries



4 Two Favorite Trees

Two Examples

Here: two representative examples

1. **Random BSTs**
   - Start with a random permutation π of $\{1, \dots, n\}$
   - Successively insert $\pi_1, \dots, \pi_n$ into initially empty (unbalanced) BST.
   - Challenge: highly non-uniform distribution

2. (uniform) **Weight-Balanced BSTs** (BB[α])
   - parameter $\alpha \in (0, \frac{1}{2})$
   - α-balanced $=$ at every node v holds: $\texttt{subtree\_size(v.left)}+1 \geqslant \alpha(\texttt{subtree\_size(v)}+1)$
     $\texttt{subtree\_size(v.right)}+1 \geqslant \alpha(\texttt{subtree\_size(v)}+1)$
   - Challenge: support is small subclass; non-fringe μ_i might not be α-balanced

$\rightsquigarrow$ ideas can be generalized to families of sources

Two Examples

Here: two representative examples

1. **Random BSTs**
   - Start with a random permutation π of $\{1, \dots, n\}$
   - Successively insert $\pi_1, \dots, \pi_n$ into initially empty (unbalanced) BST.
   - Challenge: highly non-uniform distribution

2. (uniform) **Weight-Balanced BSTs** (BB[α])
   - parameter $\alpha \in (0, \frac{1}{2})$
   - α-balanced $=$ at every node v holds: $\texttt{subtree\_size(v.left)}+1 \geqslant \alpha(\texttt{subtree\_size}(v)+1)$
     $\texttt{subtree\_size(v.right)}+1 \geqslant \alpha(\texttt{subtree\_size}(v)+1)$
   - Challenge: support is small subclass; non-fringe μ_i might not be α-balanced

$\rightsquigarrow$ ideas can be generalized to families of sources

Two Examples

Here: two representative examples

1. **Random BSTs**
   - Start with a random permutation π of $\{1, \dots, n\}$
   - Successively insert $\pi_1, \dots, \pi_n$ into initially empty (unbalanced) BST.
   - Challenge: highly non-uniform distribution

2. (uniform) **Weight-Balanced BSTs** (BB[α])
   - parameter $\alpha \in (0, \frac{1}{2})$
   - α-balanced $=$ at every node v holds: $\texttt{subtree\_size}(v.\texttt{left}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
     $\texttt{subtree\_size}(v.\texttt{right}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
   - Challenge: support is small subclass; non-fringe μ_i might not be α-balanced

$\rightsquigarrow$ ideas can be generalized to families of sources

Two Examples

Here: two representative examples

1. **Random BSTs**
   - Start with a random permutation π of $\{1, \dots, n\}$
   - Successively insert $\pi_1, \dots, \pi_n$ into initially empty (unbalanced) BST.
   - Challenge: highly non-uniform distribution

2. (uniform) **Weight-Balanced BSTs** (BB[α])
   - parameter $\alpha \in (0, \frac{1}{2})$
   - α-balanced $=$ at every node v holds: $\texttt{subtree\_size}(v.\texttt{left}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
     $\texttt{subtree\_size}(v.\texttt{right}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
   - Challenge: support is small subclass; non-fringe μ_i might not be α-balanced

$\rightsquigarrow$ ideas can be generalized to families of sources

Two Examples

Here: two representative examples

1. **Random BSTs**
   - Start with a random permutation π of $\{1, \dots, n\}$
   - Successively insert $\pi_1, \dots, \pi_n$ into initially empty (unbalanced) BST.
   - Challenge: highly non-uniform distribution

2. (uniform) **Weight-Balanced BSTs** (BB[α])
   - parameter $\alpha \in (0, \frac{1}{2})$
   - α-balanced $=$ at every node v holds: $\texttt{subtree\_size}(v.\texttt{left}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
     $\texttt{subtree\_size}(v.\texttt{right}) + 1 \geqslant \alpha(\texttt{subtree\_size}(v) + 1)$
   - Challenge: support is small subclass; non-fringe μ_i might not be α-balanced

$\rightsquigarrow$ ideas can be generalized to families of sources

Random BSTs – Outline

Random BSTs:

- rank of root uniform $\rightsquigarrow$ every possible split equally likely

$\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{\texttt{subtree\_size}_t(v)}$

$\rightsquigarrow$ random BSTs $=$ fixed-size source with $p(\ell, n-1-\ell) = \frac{1}{n}$ $\quad (n \in \mathbb{N}_{\geqslant 1}$ and $\ell \in \{0, \dots, n-1\})$

Step 1
Construct a source-specific micro-tree encoding
$D_S : \{\mu_1, \dots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_S(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

Step 2
By optimality of Huffman codes:
$\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_S(\mu_i)|$

Step 3
Use properties of S to show that
$\prod_{i=1}^{m} \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$

Step 4
Conclude
$\sum_{i=1}^{m} |C(\mu_i)| \approx \lg(1/\mathbb{P}[t])$

Random BSTs – Outline

Random BSTs:

- rank of root uniform $\rightsquigarrow$ every possible split equally likely

$\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{\texttt{subtree\_size}_t(v)}$

$\rightsquigarrow$ random BSTs $=$ fixed-size source with $p(\ell, n-1-\ell) = \frac{1}{n}$ $\quad (n \in \mathbb{N}_{\geqslant 1}$ and $\ell \in \{0, \dots, n-1\})$

Step 1
Construct a source-specific micro-tree encoding
$D_S \colon \{\mu_1, \dots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_S(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

Step 2
By optimality of Huffman codes:
$\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_S(\mu_i)|$

Step 3
Use properties of S to show that
$\prod_{i=1}^{m} \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$

Step 4
Conclude
$\sum_{i=1}^{m} |C(\mu_i)| \approx \lg(1/\mathbb{P}[t])$

Random BSTs – Outline

Random BSTs:

- rank of root uniform $\rightsquigarrow$ every possible split equally likely

$\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{\texttt{subtree\_size}_t(v)}$

$\rightsquigarrow$ random BSTs $=$ fixed-size source with $p(\ell, n-1-\ell) = \frac{1}{n}$ $\quad (n \in \mathbb{N}_{\geqslant 1}$ and $\ell \in \{0, \dots, n-1\})$

Step 1
Construct a source-specific micro-tree encoding
$D_S \colon \{\mu_1, \dots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_S(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

Step 2
By optimality of Huffman codes:
$\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_S(\mu_i)|$

Step 3
Use properties of S to show that
$\prod_{i=1}^{m} \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$

Step 4
Conclude
$\sum_{i=1}^{m} |C(\mu_i)| \approx \lg(1/\mathbb{P}[t])$

Random BSTs – Outline

Random BSTs:

- rank of root uniform $\rightsquigarrow$ every possible split equally likely

$\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} \frac{1}{\texttt{subtree\_size}_t(v)}$

$\rightsquigarrow$ random BSTs $=$ fixed-size source with $p(\ell, n-1-\ell) = \frac{1}{n}$ $\quad (n \in \mathbb{N}_{\geqslant 1}$ and $\ell \in \{0, \dots, n-1\})$

Step 1
Construct a source-specific micro-tree encoding
$D_S \colon \{\mu_1, \dots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_S(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

Step 2
By optimality of Huffman codes:
$\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_S(\mu_i)|$

Step 3
Use properties of S to show that
$\prod_{i=1}^{m} \mathbb{P}[\mu_i] \gtrsim \mathbb{P}[t]$

Step 4
Conclude
$\sum_{i=1}^{m} |C(\mu_i)| \approx \lg(1/\mathbb{P}[t])$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*

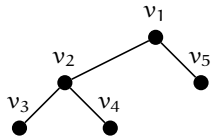
Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*

2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)
- $\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
  identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$
- $\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$
- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$
- $\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
  use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$
- $\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$
- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same
- $\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$



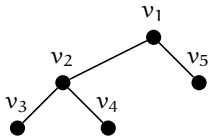
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)
- ⇝ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
  identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$
- ⇝ here $\ell_1 = 3$ ⇝ $I_1 = [\frac{3}{5}, \frac{4}{5})$
- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$
- ⇝ left subtree size $\ell_2 \in \{0,1,2\}$
  use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$
- ⇝ here $\ell_2 = 1$, ⇝ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$
- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. ⇝ nothing to store!
- v_4 and v_5 same
- ⇝ $I = [\frac{2}{3}, \frac{11}{15})$

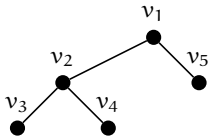
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

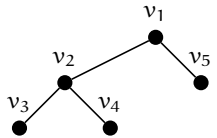
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

⇝ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

⇝ here $\ell_1 = 3$ ⇝ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

⇝ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

⇝ here $\ell_2 = 1$, ⇝ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. ⇝ nothing to store!
- v_4 and v_5 same

⇝ $I = [\frac{2}{3}, \frac{11}{15})$

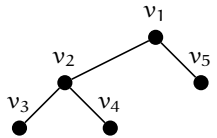
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

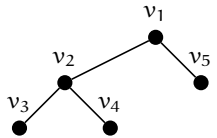
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know `subtree_size`$(v_1) = |\mu_i| = 5$ (from 1)

⇝ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

⇝ here $\ell_1 = 3$ ⇝ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know `subtree_size`$(v_2) = \ell_1 = 3$

⇝ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

⇝ here $\ell_2 = 1$, ⇝ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- `subtree_size`$(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. ⇝ nothing to store!
- v_4 and v_5 same

⇝ $I = [\frac{2}{3}, \frac{11}{15})$

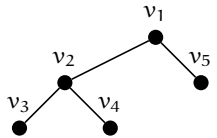
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: `10110`
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

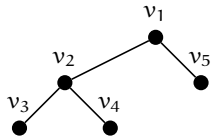
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

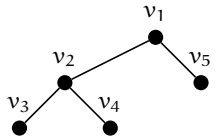
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

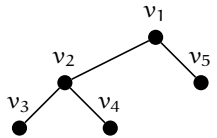
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

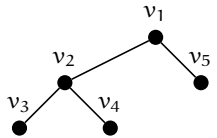
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

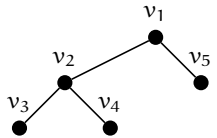
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

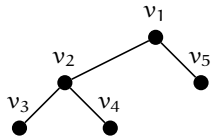
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

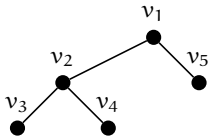
2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \dots, \mu_m$ with $|D_S(\mu_i)| \sim \lg(1/\mathbb{P}_S[\mu_i])$

1. store $|\mu_i|$ Elias code
2. store **left subtree sizes** in depth-first traversal using *arithmetic coding*



2a **Encode** sequence of outcomes **as subinterval** I of $[0,1) = I_0$

- Know $\texttt{subtree\_size}(v_1) = |\mu_i| = 5$ (from 1)

$\rightsquigarrow$ for v_1, **left** subtree size $\ell_1 \in \{0,1,2,3,4\}$
identify with subintervals of I_0 of lengths $p(\ell_1, 4-\ell_1)|I_0| = \frac{1}{5}$

$\rightsquigarrow$ here $\ell_1 = 3$ $\rightsquigarrow$ $I_1 = [\frac{3}{5}, \frac{4}{5})$

- know $\texttt{subtree\_size}(v_2) = \ell_1 = 3$

$\rightsquigarrow$ left subtree size $\ell_2 \in \{0,1,2\}$
use subintervals of I_1 of lengths $p(\ell_1, 2-\ell_1)|I_1| = \frac{1}{3} \cdot \frac{1}{5}$

$\rightsquigarrow$ here $\ell_2 = 1$, $\rightsquigarrow$ $I_2 = [\frac{3}{5} + \frac{1}{15}, \frac{3}{5} + \frac{2}{15}) = [\frac{2}{3}, \frac{11}{15})$

- $\texttt{subtree\_size}(v_3) = \ell_2 = 1$, so $\ell_3 = 0$. $\rightsquigarrow$ nothing to store!
- v_4 and v_5 same

$\rightsquigarrow$ $I = [\frac{2}{3}, \frac{11}{15})$

2b **Arithmetic coding:**

- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: $\texttt{10110}$
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $\mathrm{D}_{\mathcal{S}}$

- For node v with $\texttt{subtree\_size}(v) = n_v$,
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $\mathrm{D}_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v$,
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $\mathrm{D}_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v$,
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $\mathrm{D}_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v,$
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $\mathrm{D}_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v,$
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Step 2 Huffman optimality

- Hypersuccinct code uses Huffman code C for micro trees of t, not $\mathrm{D}_{\mathcal{S}}$
- but Huffman codes are optimal!

$$\rightsquigarrow \sum_{i=1}^{m} |\mathrm{C}(\mu_i)| \leqslant \sum_{i=1}^{m} |\mathrm{D}_{\mathcal{S}}(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2\Big)$$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $D_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v,$
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |D_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Step 2 Huffman optimality

- Hypersuccinct code uses Huffman code C for micro trees of t, not $D_{\mathcal{S}}$
- but Huffman codes are optimal!

$$\rightsquigarrow \sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $D_{\mathcal{S}}$

- For node v with $\texttt{subtree\_size}(v) = n_v$, $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$, $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

$\rightsquigarrow |D_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Step 2 Huffman optimality

- Hypersuccinct code uses Huffman code C for micro trees of t, not $D_{\mathcal{S}}$
- but Huffman codes are optimal!

$$\rightsquigarrow \sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2\Big)$$

Random BSTs – Source-specific code length

"Depth-First Arithmetic Code" $D_{\mathcal{S}}$

- For node v with $\quad \texttt{subtree\_size}(v) = n_v$,
  $\texttt{subtree\_size}(v.\texttt{left}) = \ell_v$
  $\texttt{subtree\_size}(v.\texttt{right}) = r_v$
  - shrink interval by factor $p(\ell_v, r_v)$

$\rightsquigarrow |I| = \mathbb{P}_{\mathcal{S}}[\mu_i]$

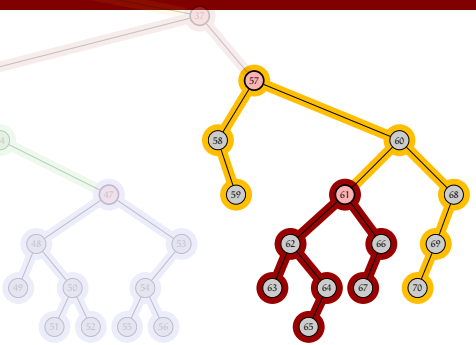
$\rightsquigarrow |D_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

Step 2 Huffman optimality

- Hypersuccinct code uses Huffman code C for micro trees of t, not $D_{\mathcal{S}}$
- but Huffman codes are optimal!

$$\rightsquigarrow \sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} |D_{\mathcal{S}}(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  $\rightsquigarrow$ uses wrong subtree sizes!

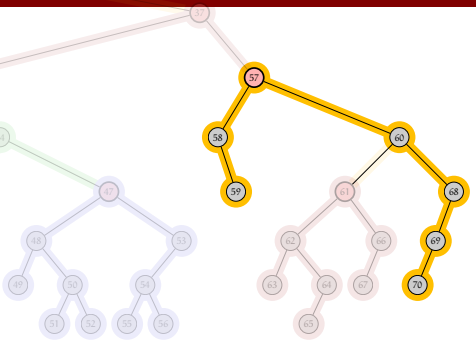
  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $p(\ell+1, r) \leqslant p(\ell, r)$ and $p(\ell, r+1) \leqslant p(\ell, r)$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\leqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  $\rightsquigarrow$ uses wrong subtree sizes!

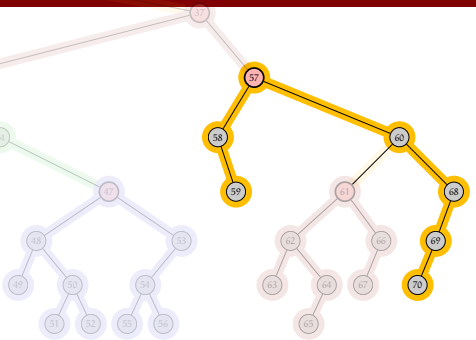
  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $p(\ell+1, r) \leqslant p(\ell, r)$ and $p(\ell, r+1) \leqslant p(\ell, r)$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  $\rightsquigarrow$ uses wrong subtree sizes!

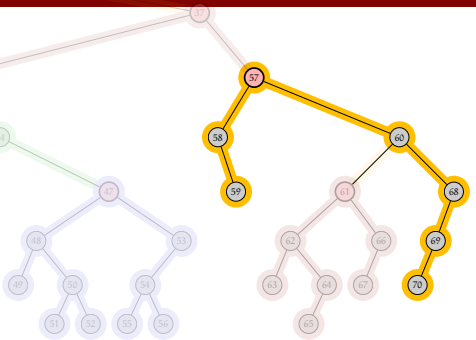
  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $p(\ell+1, r) \leqslant p(\ell, r)$ and $p(\ell, r+1) \leqslant p(\ell, r)$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  ⇝ uses wrong subtree sizes!

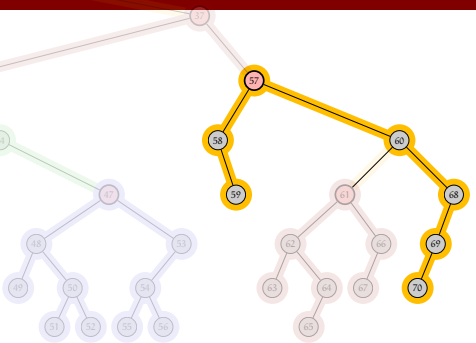
  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $p(\ell+1, r) \leqslant p(\ell, r)$ and $p(\ell, r+1) \leqslant p(\ell, r)$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  $\rightsquigarrow$ uses wrong subtree sizes!

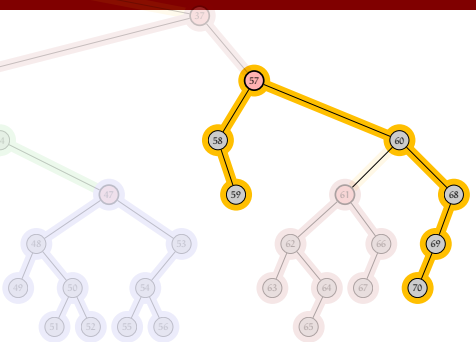
  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $\boxed{p(\ell+1,r) \leqslant p(\ell,r) \text{ and } p(\ell,r+1) \leqslant p(\ell,r)}$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Monotonicity



Step 3 From μ_i to t

- So far: optimal code for micro trees … but want code for t!

  Problem: non-fringe micro trees

  - Store yellow subtree as if red subtree was not there

  ⇝ uses wrong subtree sizes!

  But:
  - ℓ_v, r_v in μ_i only smaller, and
  - $\boxed{p(\ell+1,r) \leqslant p(\ell,r) \text{ and } p(\ell,r+1) \leqslant p(\ell,r)}$ (**monotonic** source)

$$\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] = \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{\mu_i}(v.\texttt{left}), \texttt{subtree\_size}_{\mu_i}(v.\texttt{right})\big)$$

$$\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big)$$

$$= \prod_{v \in t} p\big(\texttt{subtree\_size}_{t}(v.\texttt{left}), \texttt{subtree\_size}_{t}(v.\texttt{right})\big) = \mathbb{P}[t]$$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

- Step 2: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$
- Step 3: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

$$\rightsquigarrow |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$
$$\leqslant \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) + o(n)$$
$$\leqslant \lg(1/\mathbb{P}[t]) + o(n)$$

Random BSTs

- $\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim 1.736n$

$$\sum_{k=1}^{n} \frac{2\lg(k)}{(k+1)(k+2)}$$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

- Step 2: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$
- Step 3: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

$$\rightsquigarrow |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$
$$\leqslant \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) + o(n)$$
$$\leqslant \lg(1/\mathbb{P}[t]) + o(n)$$

Random BSTs

- $\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim 1.736n$

$$\sum_{k=1}^{n} \frac{2\lg(k)}{(k+1)(k+2)}$$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

- Step 2: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$
- Step 3: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

$$\rightsquigarrow |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$
$$\leqslant \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) + o(n)$$
$$\leqslant \lg(1/\mathbb{P}[t]) + o(n)$$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

- Step 2: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$
- Step 3: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

$$\rightsquigarrow |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$
$$\leqslant \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) + o(n)$$
$$\leqslant \lg(1/\mathbb{P}[t]) + o(n)$$

Random BSTs

- $\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim 1.736n$

$$\sum_{k=1}^{\infty} \frac{2\lg(k)}{(k+1)(k+2)}$$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

- Step 2: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \Big(\lg(1/\mathbb{P}[\mu_i]) + 2 \Big)$
- Step 3: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

$$\rightsquigarrow |H(t)| = \sum_{i=1}^{m} |C(\mu_i)| + o(n)$$
$$\leqslant \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) + o(n)$$
$$\leqslant \lg(1/\mathbb{P}[t]) + o(n)$$

Random BSTs

- $\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim 1.736n$

$$\sum_{k=1}^{\infty} \frac{2\lg(k)}{(k+1)(k+2)}$$

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \frac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

general recipe for uniform distributions

but **not monotonic**

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \frac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \dfrac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \dfrac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \dfrac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)
  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \dfrac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs

Uniform Weight-Balanced BSTs:

- W_n = set of all α-weight-balanced binary trees.
- Not so well-understood
  - *No counting results (!)* (to my knowledge)
- **Some properties:**
  - logarithmic height (obvious)
  - every fringe subtree is again weight balanced (obvious)
  - only $O(n/B)$ nodes have subtree size $\geqslant B$ (not obvious, but not hard to prove)

Keep this in mind!

  - Can be generated with a fixed-size source using

$$p(\ell, n-1-\ell) = \begin{cases} \dfrac{|W_\ell| \cdot |W_{n-1-\ell}|}{|W_n|} & \text{if } \min\{\ell+1, n-\ell\} \geqslant \alpha(n+1) \\ 0 & \text{otherwise} \end{cases}$$

but **not monotonic**

general recipe for uniform distributions

Weight-Balanced BSTs – Problems

Complication 1: **non-fringe subtree** in general **not** α-balanced!

Weight-Balanced BSTs – Problems

Complication 1: **non-fringe subtree** in general **not** α-balanced!

$\rightsquigarrow$ Cannot possibly hope to show $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

potentially 0

Weight-Balanced BSTs – Problems

Complication 1: **non-fringe subtree** in general **not** α-balanced!

$\rightsquigarrow$ Cannot possibly hope to show $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

potentially 0

trees are *"nicely balanced"* ... maybe we can ignore non-fringe subtrees?

i.e., encode trivially with 2 bits per node

Weight-Balanced BSTs – Problems

Complication 1: **non-fringe subtree** in general **not** α-balanced!

$\rightsquigarrow$ Cannot possibly hope to show $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

potentially 0

trees are *"nicely balanced"* ... maybe we can ignore non-fringe subtrees?

i.e., encode trivially with 2 bits per node

Complication 2: Can still have $\Theta(n)$ nodes in non-fringe μ_i.

Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$

Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs

$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bitvectors for boughs' end
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

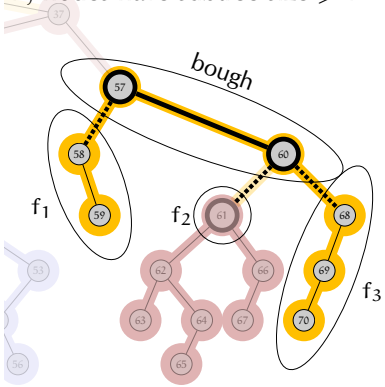
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bitvectors for bough and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

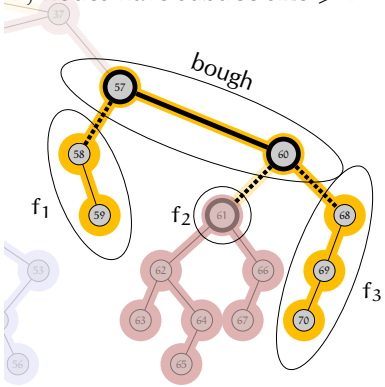
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

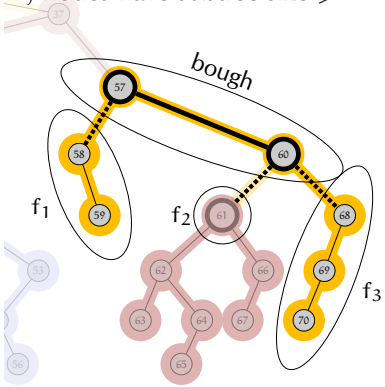
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

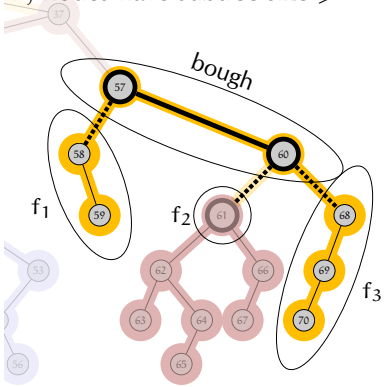
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

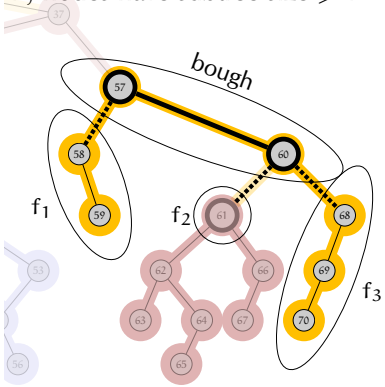
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

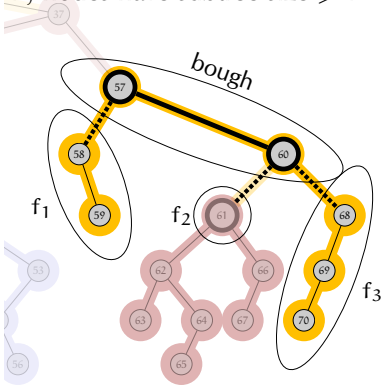
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

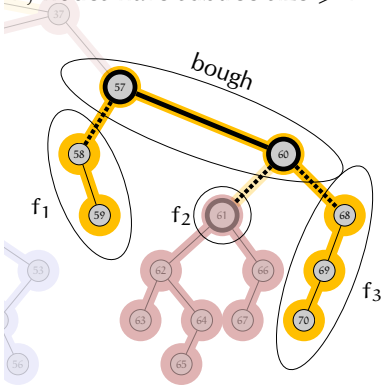
Weight-Balanced BSTs – Great-Branching Code

- Weight-balanced trees are **"fringe dominated"**: $O(n/B)$ nodes have subtree size $\geqslant B$



Inside D_S, break up micro trees into

1. "boughs" of heavy nodes
2. fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



$\rightsquigarrow$ D_S stores

- fringe μ_i using depth-first arithmetic code
- non-fringe μ_i using
  1. 2 bits/node for boughs and
  2. depth-first arithmetic code for twigs

$\rightsquigarrow$ Only boughs stored suboptimally and these are a vanishing fraction of t.

Outline



1 Three Roots



2 Hypersuccinct Trees



3 Tree Sources



4 Two Favorite Trees



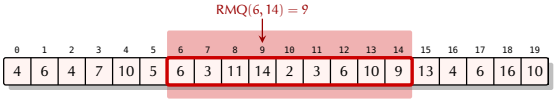
5 Range-Minimum Queries



5 Range-Minimum Queries

Range-maximum queries (RMQ)

- **Given:** Static array $A[0..n)$ of numbers (array/numbers don't change)
- **Goal:** Find maximum in a range; A known in advance and can be preprocessed



- **Nitpicks:**
  - Report **index** of maximum, not its value
  - Report **leftmost** position in case of ties

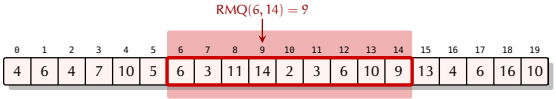
lowest common ancestor

RMQ is **equivalent** to LCA in binary trees:

Range-maximum queries (RMQ)

array/numbers don't change

- **Given:** Static array $A[0..n)$ of numbers
- **Goal:** Find maximum in a range; A known in advance and can be preprocessed



- **Nitpicks:**
  - Report **index** of maximum, not its value
  - Report **leftmost** position in case of ties

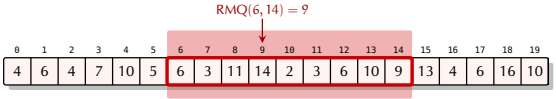
lowest common ancestor

RMQ is **equivalent** to LCA in binary trees:

Range-maximum queries (RMQ)

array/numbers don't change

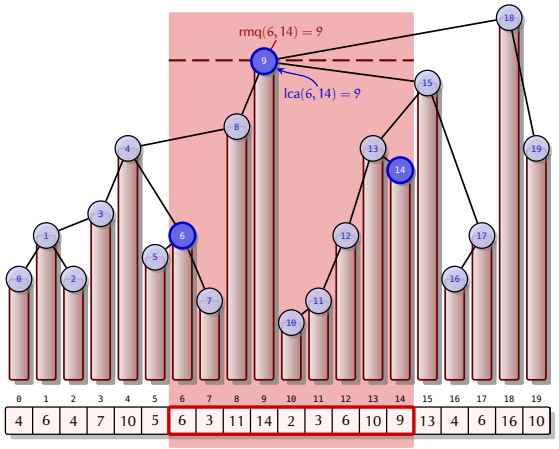
- **Given:** Static array $A[0..n)$ of numbers
- **Goal:** Find maximum in a range; A known in advance and can be preprocessed



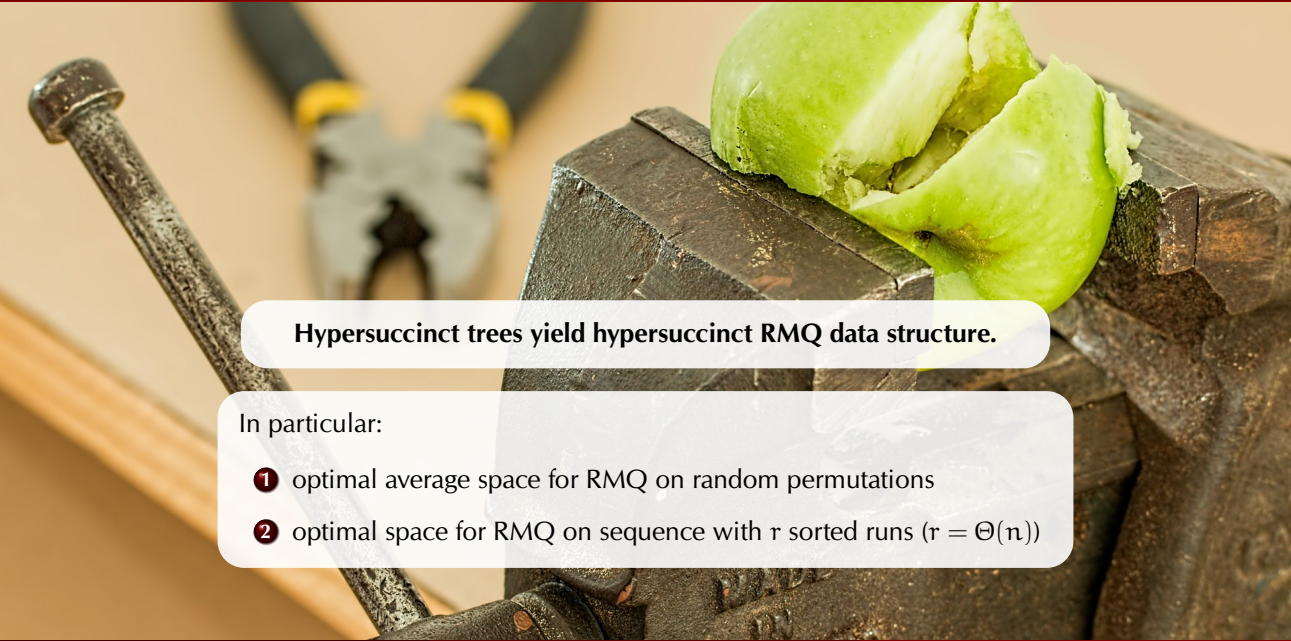
- **Nitpicks:**
  - Report **index** of maximum, not its value
  - Report **leftmost** position in case of ties

lowest common ancestor

RMQ is **equivalent** to LCA in binary trees:



Hypersuccinct RMQ



Hypersuccinct trees yield hypersuccinct RMQ data structure.

In particular:

1. optimal average space for RMQ on random permutations
2. optimal space for RMQ on sequence with r sorted runs ($r = \Theta(n)$)

Conclusion

Hypersuccinct trees

- simple universal tree source code
- as versatile as any known universal code for trees
- but also supports efficient queries

What's next?

- tree with labels
- other combinatorial structures

Conclusion

Hypersuccinct trees

- simple universal tree source code
- as versatile as any known universal code for trees
- but also supports efficient queries

What's next?

- tree with labels
- other combinatorial structures



Icons made by *Freepik*, *Gregor Cresnar*, *Those Icons*, *Smashicons*, *Good Ware*, *Pause08*, and *Madebyoliver* from `www.flaticon.com`.
Other photos from `www.pixabay.com`.