On the combinatorics of space-efficient data structures

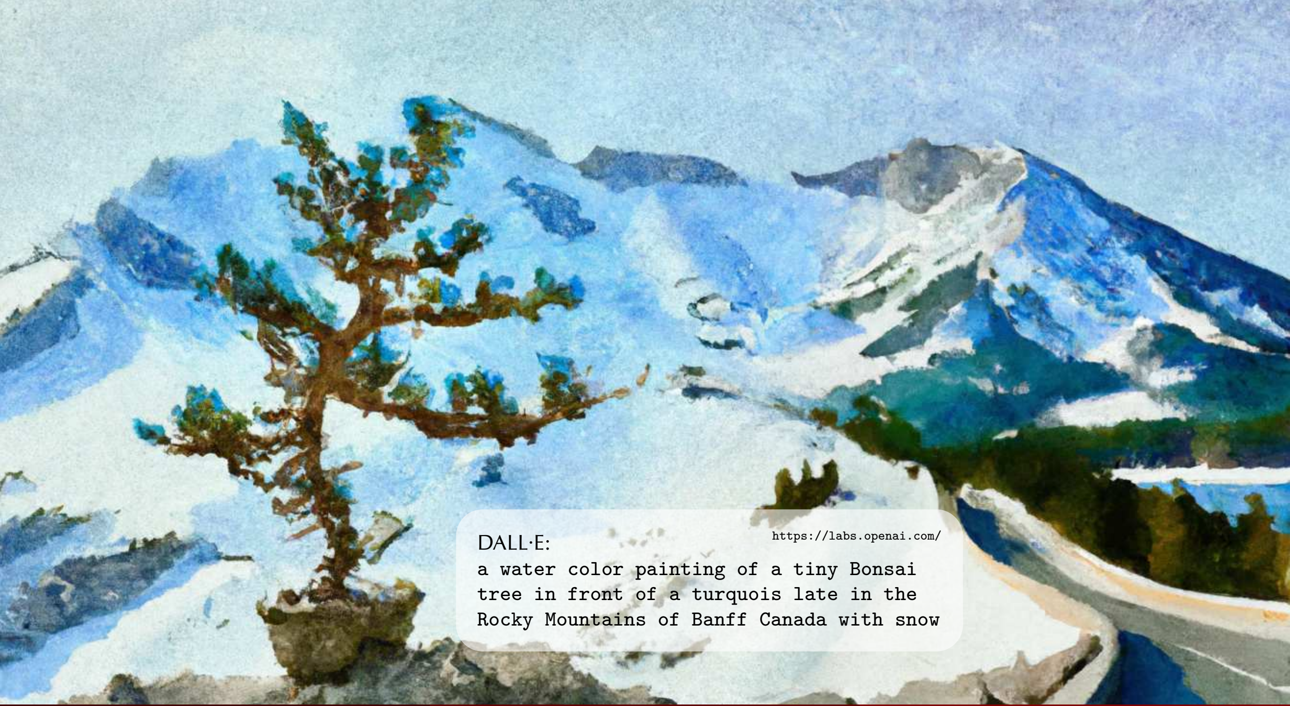
Sebastian Wild

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

`arXiv:2104.13457`

UNIVERSITY OF LIVERPOOL

Banff Workshop 22w5004 – Analytic and Probabilistic Combinatorics



DALL·E:

https://labs.openai.com/

`a water color painting of a tiny Bonsai tree in front of a turquois late in the Rocky Mountains of Banff Canada with snow`

Outline

1 Hypersuccinct Trees

2 Two Favorite Trees

3 Beyond Trees

4 Bonus: Range-Minimum Queries

5 Bonus: Succinct Bitvectors

1 Hypersuccinct Trees

Three roots

Data structures

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

  $\lg|U_n|(1+o(1))$ 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.:

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

instance-optimal

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.:

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

    instance-optimal

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.:

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

instance-optimal

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.

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 $\sim \log_2(\text{Catalan}_n)$
  - 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, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal
`node_select`$_X(i)$	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
`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 $\sim \log_2(\text{Catalan}_n)$
  - 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

$\texttt{parent}(v)$	the parent of v, same as $\texttt{anc}(v, 1)$
$\texttt{degree}(v)$	the number of children of v
$\texttt{left\_child}(v)$	the left child of node v
$\texttt{right\_child}(v)$	the right child of node v
$\texttt{depth}(v)$	the depth of v, i.e., the number of edges between the root and v
$\texttt{anc}(v, i)$	the ancestor of node v at depth $\texttt{depth}(v) - i$
$\texttt{subtree\_size}(v)$	the number of descendants of v
$\texttt{height}(v)$	the height of the subtree rooted at node v
$\texttt{LCA}(v, u)$	the lowest common ancestor of nodes u and v
$\texttt{leftmost\_leaf}(v)$	the leftmost leaf descendant of v
$\texttt{rightmost\_leaf}(v)$	the rightmost leaf descendant of v
$\texttt{level\_leftmost}(\ell)$	the leftmost node on level ℓ
$\texttt{level\_rightmost}(\ell)$	the rightmost node on level ℓ
$\texttt{level\_predecessor}(v)$	the node immediately to the left of v on the same level
$\texttt{level\_successor}(v)$	the node immediately to the right of v on the same level
$\texttt{node\_rank}_X(v)$	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal
$\texttt{node\_select}_X(i)$	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
$\texttt{leaf\_rank}(v)$	the number of leaves before and including v in preorder
$\texttt{leaf\_select}(i)$	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 $\sim \log_2(\text{Catalan}_n)$
  - 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

$\texttt{parent}(v)$	the parent of v, same as $\texttt{anc}(v, 1)$
$\texttt{degree}(v)$	the number of children of v
$\texttt{left\_child}(v)$	the left child of node v
$\texttt{right\_child}(v)$	the right child of node v
$\texttt{depth}(v)$	the depth of v, i.e., the number of edges between the root and v
$\texttt{anc}(v, i)$	the ancestor of node v at depth $\texttt{depth}(v) - i$
$\texttt{subtree\_size}(v)$	the number of descendants of v
$\texttt{height}(v)$	the height of the subtree rooted at node v
$\texttt{LCA}(v, u)$	the lowest common ancestor of nodes u and v
$\texttt{leftmost\_leaf}(v)$	the leftmost leaf descendant of v
$\texttt{rightmost\_leaf}(v)$	the rightmost leaf descendant of v
$\texttt{level\_leftmost}(\ell)$	the leftmost node on level ℓ
$\texttt{level\_rightmost}(\ell)$	the rightmost node on level ℓ
$\texttt{level\_predecessor}(v)$	the node immediately to the left of v on the same level
$\texttt{level\_successor}(v)$	the node immediately to the right of v on the same level
$\texttt{node\_rank}_X(v)$	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal
$\texttt{node\_select}_X(i)$	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
$\texttt{leaf\_rank}(v)$	the number of leaves before and including v in preorder
$\texttt{leaf\_select}(i)$	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 $\sim \log_2(\mathrm{Catalan}_n)$
  - 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)

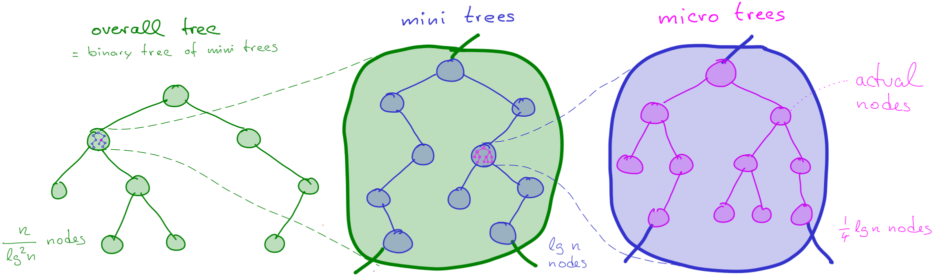
Operations in Tree Covering

$\texttt{parent}(v)$	the parent of v, same as $\texttt{anc}(v, 1)$
$\texttt{degree}(v)$	the number of children of v
$\texttt{left\_child}(v)$	the left child of node v
$\texttt{right\_child}(v)$	the right child of node v
$\texttt{depth}(v)$	the depth of v, i.e., the number of edges between the root and v
$\texttt{anc}(v, i)$	the ancestor of node v at depth $\texttt{depth}(v) - i$
$\texttt{subtree\_size}(v)$	the number of descendants of v
$\texttt{height}(v)$	the height of the subtree rooted at node v
$\texttt{LCA}(v, u)$	the lowest common ancestor of nodes u and v
$\texttt{leftmost\_leaf}(v)$	the leftmost leaf descendant of v
$\texttt{rightmost\_leaf}(v)$	the rightmost leaf descendant of v
$\texttt{level\_leftmost}(\ell)$	the leftmost node on level ℓ
$\texttt{level\_rightmost}(\ell)$	the rightmost node on level ℓ
$\texttt{level\_predecessor}(v)$	the node immediately to the left of v on the same level
$\texttt{level\_successor}(v)$	the node immediately to the right of v on the same level
$\texttt{node\_rank}_X(v)$	the position of v in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$, i.e., in a preorder, postorder, or inorder traversal
$\texttt{node\_select}_X(i)$	the ith node in the X-order, $X \in \{\texttt{PRE}, \texttt{POST}, \texttt{IN}\}$
$\texttt{leaf\_rank}(v)$	the number of leaves before and including v in preorder
$\texttt{leaf\_select}(i)$	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 ⇝ enough to support many operations
  and $\tilde{O}(\log\log n)$ bits per micro tree

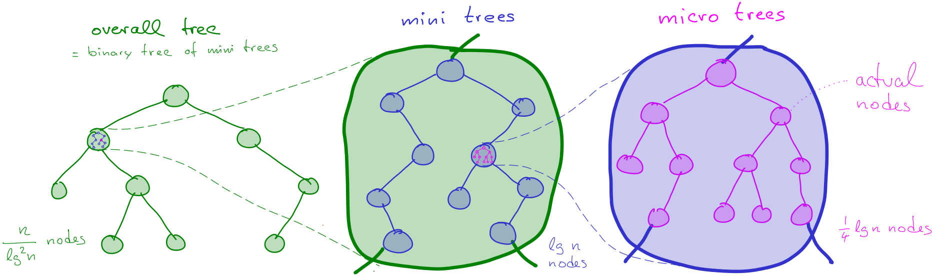
  ↑ most comprehensive of all succinct trees

- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table
  (“exhaustive lookup table”, “bootstrapping”, “4 Russians trick”)

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 ⇝ enough to support many operations
  and $\tilde{O}(\log \log n)$ bits per micro tree

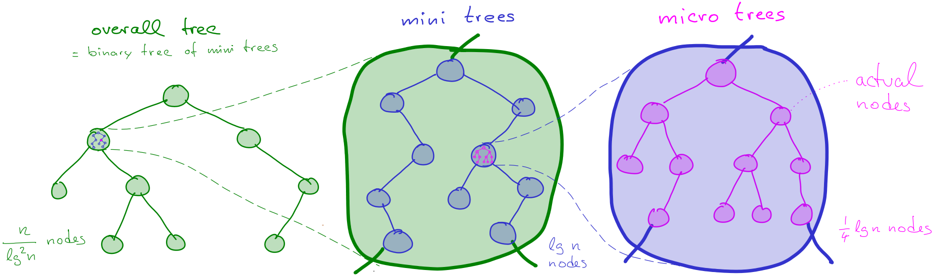
most comprehensive
of all succinct trees

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

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 $\rightsquigarrow$ enough to support many operations
  and $\tilde{O}(\log \log n)$ bits per micro tree

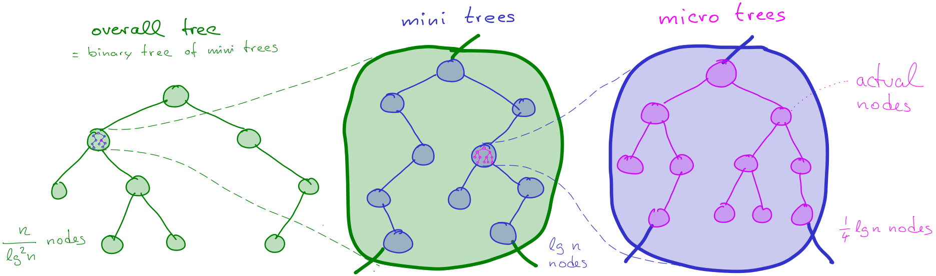
most comprehensive
of all succinct trees

- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table
  (“exhaustive lookup table”, “bootstrapping”, “4 Russians trick”)

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

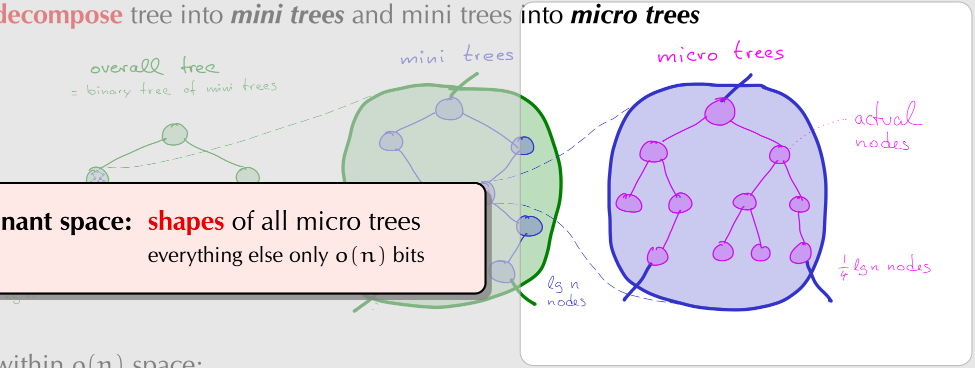
most comprehensive
of all succinct trees

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

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 ⇝ enough to support many operations

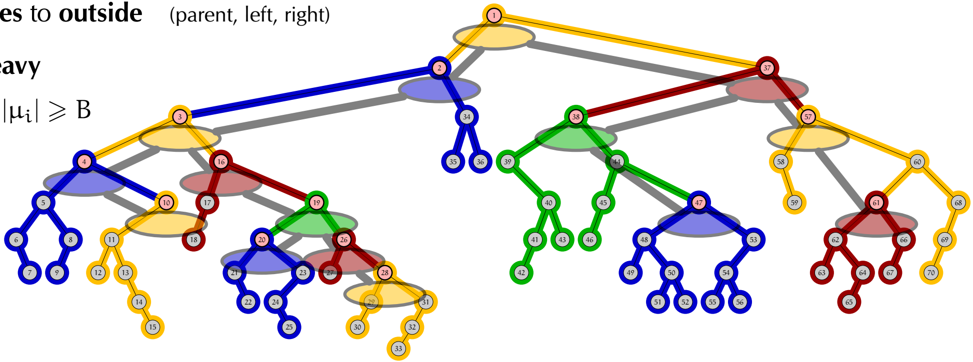
most comprehensive
of all succinct trees

- only $O(\sqrt{n})$ different micro tree shapes
  can store micro-tree-local operations in global lookup table
  (“exhaustive lookup table”, “bootstrapping”, “4 Russians trick”)

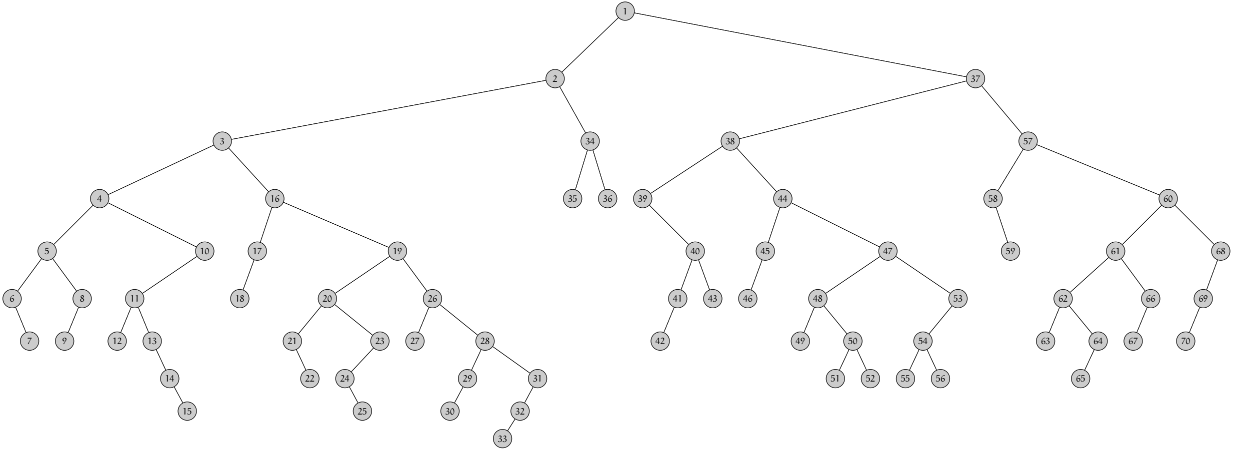
Properties of Micro Trees

Binary tree t on n nodes is decomposed into $\mu_1, \ldots, \mu_m$ with parameter $B = \frac{1}{8} \lg n$.

- **a** $m = O(n/B)$ (few micro trees)
- **b** $|\mu_i| \leqslant 2B$ (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** $\rightsquigarrow$ $|\mu_i| \geqslant B$

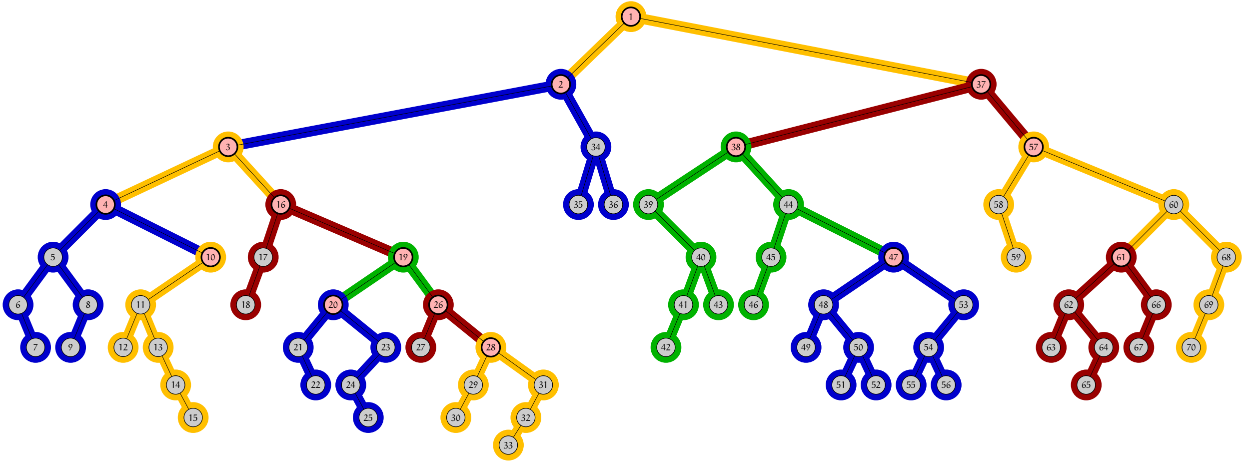


Example Partitioning



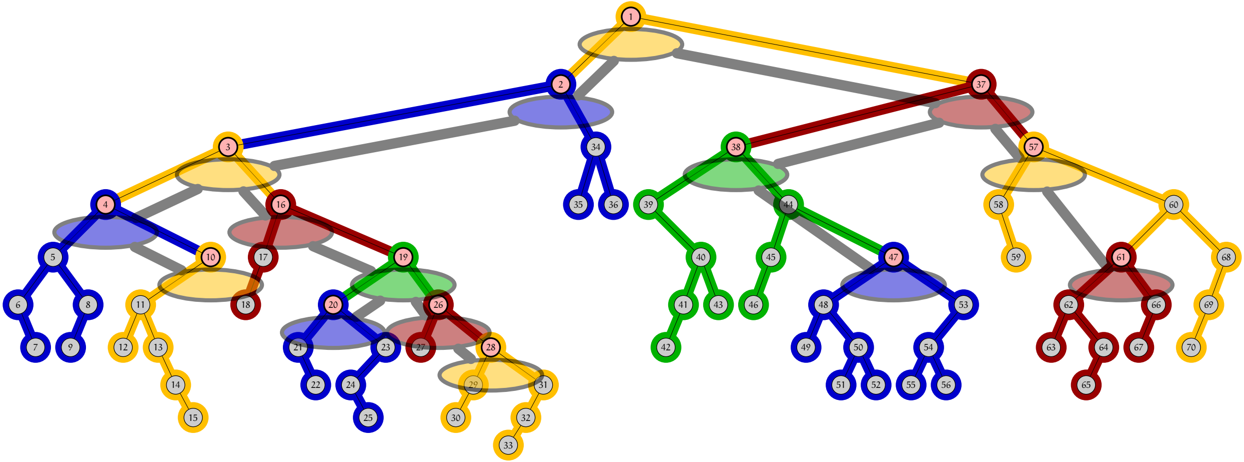
$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Example Partitioning



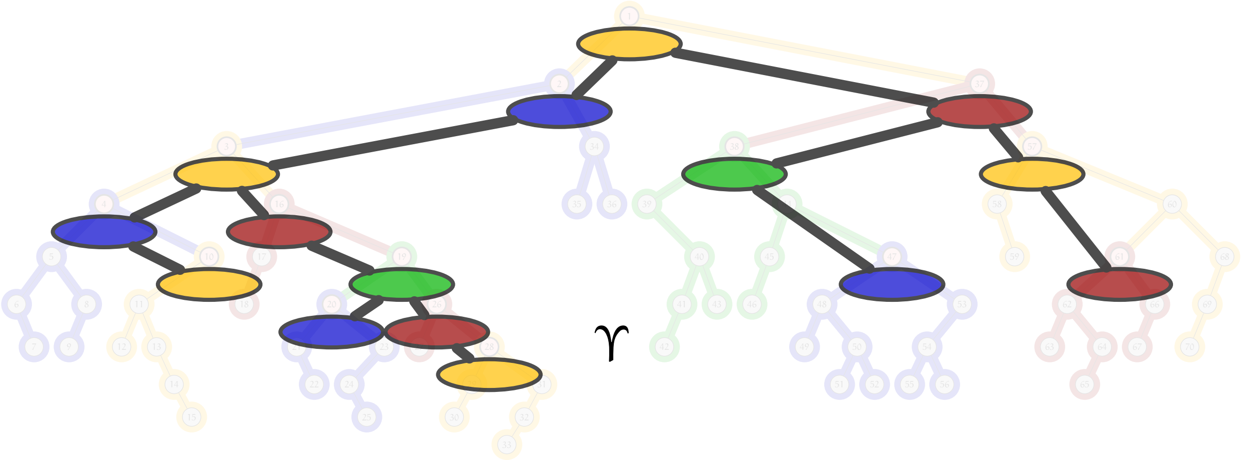
$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Example Partitioning



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

Example Partitioning

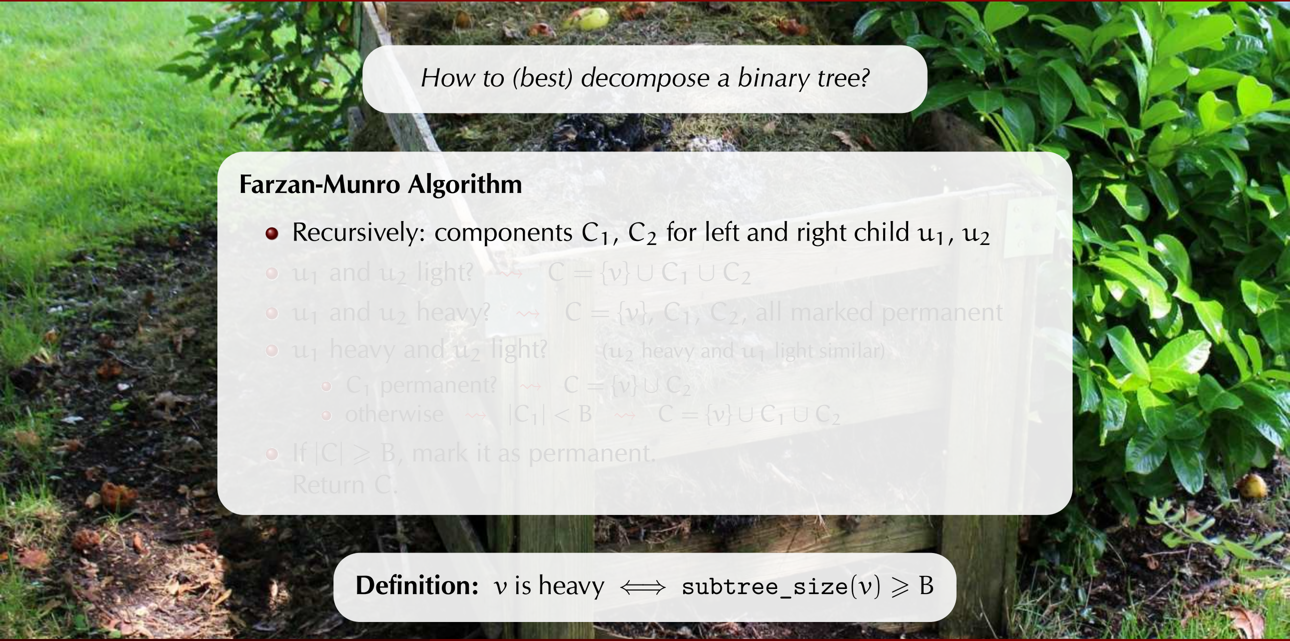


$n = 70$ nodes, $B = 6$ $\rightsquigarrow$ $m = 15$ micro trees

Farzan-Munro Algorithm

How to (best) decompose a binary tree?

Farzan-Munro Algorithm



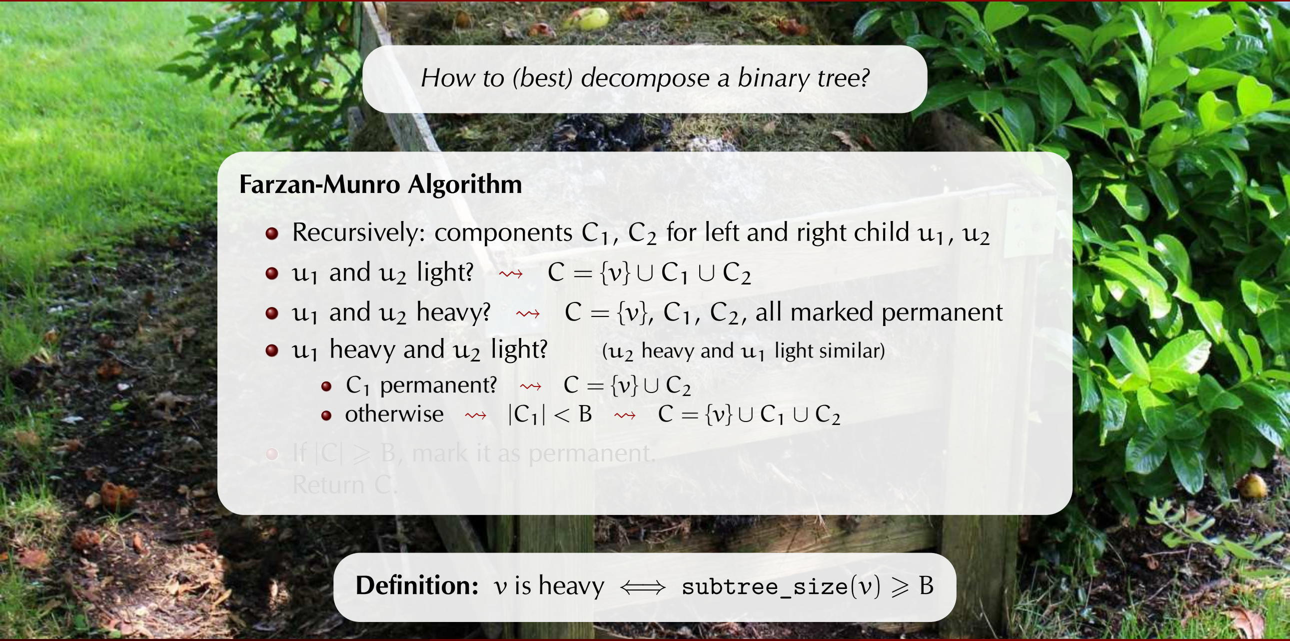
How to (best) decompose a binary tree?

Farzan-Munro Algorithm

- Recursively: components C_1, C_2 for left and right child u_1, u_2
- 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? (u_2 heavy and u_1 light similar)
  - 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 $\iff$ $\texttt{subtree\_size}(v) \geqslant B$

Farzan-Munro Algorithm



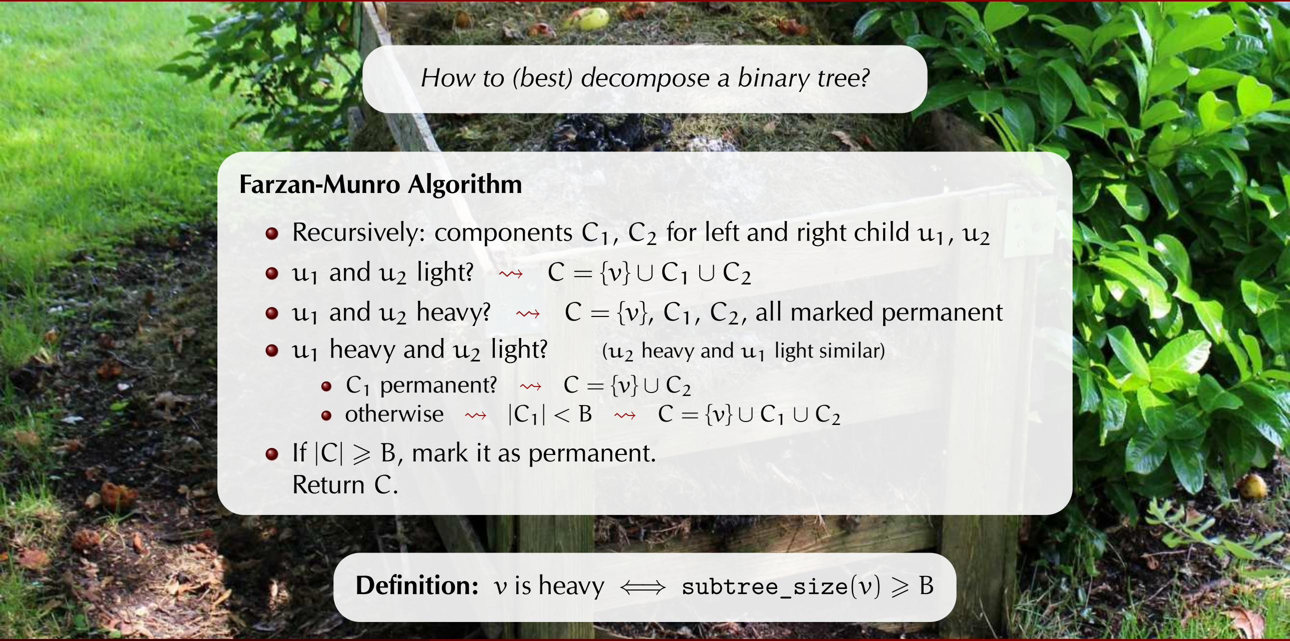
How to (best) decompose a binary tree?

Farzan-Munro Algorithm

- Recursively: components C_1, C_2 for left and right child u_1, u_2
- u_1 and u_2 light? ⇝ $C = \{v\} \cup C_1 \cup C_2$
- u_1 and u_2 heavy? ⇝ $C = \{v\}$, C_1, C_2, all marked permanent
- u_1 heavy and u_2 light? (u_2 heavy and u_1 light similar)
  - C_1 permanent? ⇝ $C = \{v\} \cup C_2$
  - otherwise ⇝ $|C_1| < B$ ⇝ $C = \{v\} \cup C_1 \cup C_2$
- 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, u_2
- 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? (u_2 heavy and u_1 light similar)
    - 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 $\iff$ $\texttt{subtree\_size}(v) \geqslant B$

Hypersuccinct code

i.e. compression algorithm

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

$$\rightsquigarrow \boxed{\begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\le m \cdot \big(\mathcal{H}(\mu_1 \ldots \mu_m) + 1\big) + o(n) \end{aligned}}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm ↙

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

$$\rightsquigarrow \boxed{\begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\leqslant m \cdot \big(\mathcal{H}(\mu_1 \ldots \mu_m) + 1\big) + o(n) \end{aligned}}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm ↙

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

$$\rightsquigarrow \boxed{\begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\leqslant m \cdot (\mathcal{H}(\mu_1 \ldots \mu_m) + 1) + o(n) \end{aligned}}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm

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)

$$\rightsquigarrow \quad \begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\leqslant m \cdot (\mathcal{H}(\mu_1 \ldots \mu_m) + 1) + o(n) \end{aligned}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm

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

- 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)

$$\rightsquigarrow \quad \begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\le m \cdot (\mathcal{H}(\mu_1 \dots \mu_m) + 1) + o(n) \end{aligned}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm

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)

$$\rightsquigarrow \quad \begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\leqslant m \cdot \bigl(\mathcal{H}(\mu_1 \ldots \mu_m) + 1\bigr) + o(n) \end{aligned}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Hypersuccinct code

i.e. compression algorithm

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)

$$\rightsquigarrow \quad \begin{aligned} |H(t)| &= \sum_{i=1}^{m} |C(\mu_i)| + o(n) \\ &\leqslant m \cdot \big(\mathcal{H}(\mu_1 \ldots \mu_m) + 1\big) + o(n) \end{aligned}$$

where $\mathcal{H}$ is the **(empirical) entropy** of the micro trees

? For what tree distributions is $|H(t)| \sim \lg(1/\mathbb{P}[t])$?

Outline

1 Hypersuccinct Trees

2 Two Favorite Trees

3 Beyond Trees

4 Bonus: Range-Minimum Queries

5 Bonus: Succinct Bitvectors

2 Two Favorite Trees

Two Examples

Here: two representative examples

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

2. (uniform) **Weight-Balanced BSTs** (BB$[\alpha]$)
   - 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 $\{1, \ldots, n\}$
   - Successively insert $\pi_1, \ldots, \pi_n$ 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 $\{1,\ldots,n\}$
- Successively insert $\pi_1,\ldots,\pi_n$ into initially empty (unbalanced) BST.
- Challenge: highly non-uniform distribution

② (uniform) **Weight-Balanced BSTs** (BB$[\alpha]$)

- 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 $\{1, \ldots, n\}$
- Successively insert $\pi_1, \ldots, \pi_n$ into initially empty (unbalanced) BST.
- Challenge: highly non-uniform distribution

② (uniform) **Weight-Balanced BSTs** (BB$[\alpha]$)

- 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

❶ **Random BSTs**

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

❷ (uniform) **Weight-Balanced BSTs** (BB$[\alpha]$)

- 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

⇝ 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 $\ell \in \{0, \ldots, n-1\})$

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

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

Step 3
Use properties of $\mathcal{S}$ 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

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

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

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

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

Step 3
Use properties of $\mathcal{S}$ 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 ⇝ every possible split equally likely

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

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

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

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

Step 3
Use properties of $\mathcal{S}$ 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
$D_{\mathcal{S}} : \{\mu_1, \dots, \mu_m\} \to \{0,1\}^\star$
Goal: $|D_{\mathcal{S}}(\mu_i)| \approx \lg(1/\mathbb{P}[\mu_i])$

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

Step 3
Use properties of $\mathcal{S}$ 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 $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

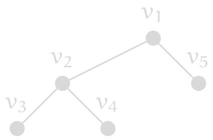
- know `subtree_size`$(v_2) = \ell_1 = 3$

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

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

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



2b **Arithmetic coding:**

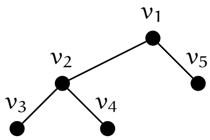
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I \quad (l, m \in \mathbb{N})$
Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
l-bit binary representation of m.
Here: 10110
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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 $l \leqslant \lg(1/|I|) + 2$

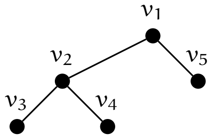
Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

- `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:**

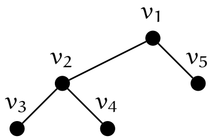
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: 10110
- Always have $l \leq \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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 $l \leq \lg(1/|I|) + 2$

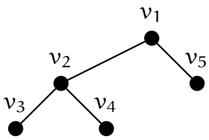
Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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:**

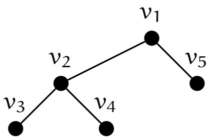
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
l-bit binary representation of m.
Here: 10110
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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:**

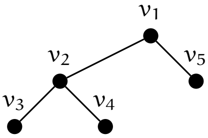
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: 10110
- Always have $l \le \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_\mathcal{S}$ for $\mu_1, \ldots, \mu_m$ with $|D_\mathcal{S}(\mu_i)| \sim \lg(1/|\mathbb{P}_\mathcal{S}[\mu_i])$

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 $l \leqslant \lg(1/|I|) + 2$

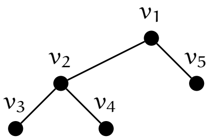
Random BSTs – Source-specific code

$\boxed{\textbf{Step 1}}$ Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

- `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:**

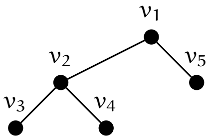
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: 10110
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code D_S for $\mu_1, \ldots, \mu_m$ 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: 10110
- Always have $l \leqslant \lg(1/|I|) + 2$

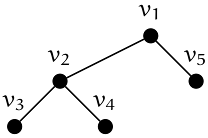
Random BSTs – Source-specific code

$\boxed{\textbf{Step 1}}$ Code $D_{\mathcal{S}}$ for $\mu_1, \dots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

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

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

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

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

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

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

(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 $l \le \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i]|)$

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

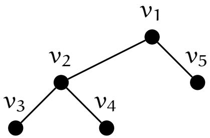
- know `subtree_size`$(v_2) = \ell_1 = 3$

⇝ 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:**

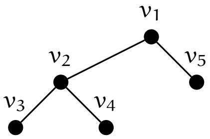
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
l-bit binary representation of m.
Here: 10110
- Always have $l \le \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

- `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: 10110
- Always have $l \leq \lg(1/|I|) + 2$

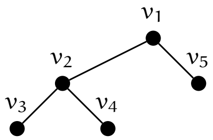
Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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)

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

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

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

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

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

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

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

(2b) **Arithmetic coding:**

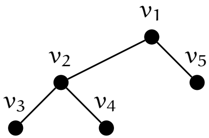
- Find interval $[\frac{m}{2^l}, \frac{m+1}{2^l}) \subseteq I$ $(l, m \in \mathbb{N})$
  Here: $[\frac{22}{32}, \frac{23}{32})$
- encode I by
  l-bit binary representation of m.
  Here: 10110
- Always have $l \leqslant \lg(1/|I|) + 2$

Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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: `10110`
- Always have $l \le \lg(1/|I|) + 2$

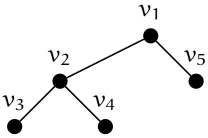
Random BSTs – Source-specific code

Step 1 Code $D_{\mathcal{S}}$ for $\mu_1, \ldots, \mu_m$ with $|D_{\mathcal{S}}(\mu_i)| \sim \lg(1/|\mathbb{P}_{\mathcal{S}}[\mu_i])$

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 length

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

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

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

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

Random BSTs – Source-specific code length

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

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

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

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

Random BSTs – Source-specific code length

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

- For node v with $\begin{aligned}\texttt{subtree\_size}(v) &= n_v,\\ \texttt{subtree\_size}(v\texttt{.left}) &= \ell_v\\ \texttt{subtree\_size}(v\texttt{.right}) &= r_v\end{aligned}$
  - 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" $D_{\mathcal{S}}$

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

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

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

Random BSTs – Source-specific code length

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

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

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

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

$\boxed{\textbf{Step 2}}$ 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 $\begin{aligned}\texttt{subtree\_size}(v) &= n_v,\\ \texttt{subtree\_size}(v.\texttt{left}) &= \ell_v\\ \texttt{subtree\_size}(v.\texttt{right}) &= r_v\end{aligned}$
  - shrink interval by factor $p(\ell_v, r_v)$

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

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

Step 2 Huffman optimality

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

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

Random BSTs – Source-specific code length

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

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

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

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

$\boxed{\textbf{Step 2}}$ Huffman optimality

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

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

Random BSTs – Source-specific code length

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

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

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

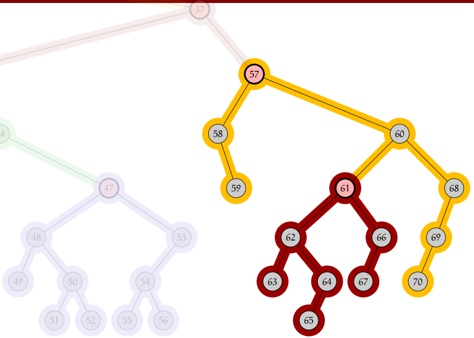
$\rightsquigarrow$ $|D_{\mathcal{S}}(\mu_i)| \leqslant \lg(1/\mathbb{P}_{\mathcal{S}}[\mu_i]) + 2$

$\boxed{\textbf{Step 2}}$ Huffman optimality

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

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

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!

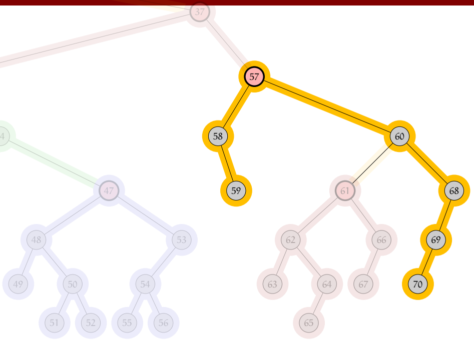
But:
- ℓ_v, r_v in μ_i only smaller, and
- $\boxed{p(\ell+1, r) \leqslant p(\ell, r) \text{ and } p(\ell, r+1) \leqslant p(\ell, r)}$ (monotonic source)

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

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

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

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
- $\boxed{p(\ell+1, r) \leqslant p(\ell, r) \text{ and } p(\ell, r+1) \leqslant p(\ell, r)}$

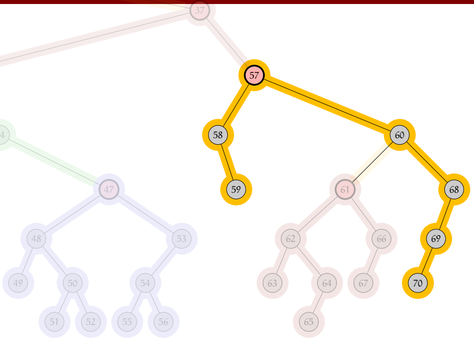
(monotonic source)

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

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

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

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!

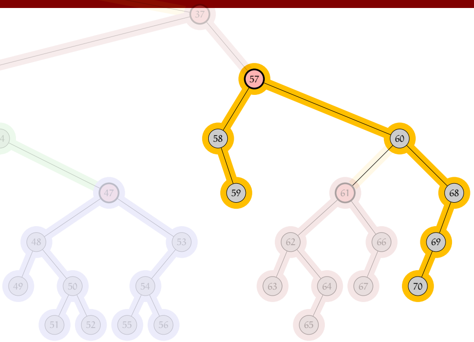
But:

- 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)

$$\begin{aligned}\rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] &= \prod_{i=1}^{m} \prod_{v\in\mu_i} p\big(\mathtt{subtree\_size}_{\mu_i}(v.\mathtt{left}), \mathtt{subtree\_size}_{\mu_i}(v.\mathtt{right})\big) \\ &\geqslant \prod_{i=1}^{m} \prod_{v\in\mu_i} p\big(\mathtt{subtree\_size}_{t}(v.\mathtt{left}), \mathtt{subtree\_size}_{t}(v.\mathtt{right})\big) \\ &= \prod_{v\in t} p\big(\mathtt{subtree\_size}_{t}(v.\mathtt{left}), \mathtt{subtree\_size}_{t}(v.\mathtt{right})\big) \;=\; \mathbb{P}[t]\end{aligned}$$

Random BSTs – Monotonicity



$\boxed{\textbf{Step 3}}$ 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

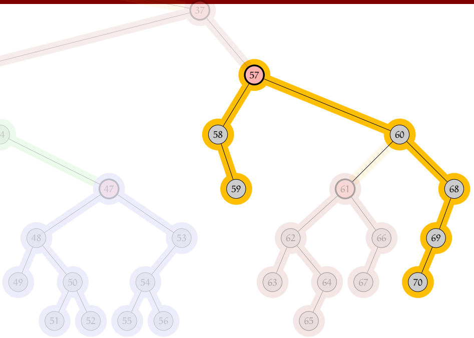
  ⇝ uses wrong subtree sizes!

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)

$$\begin{aligned} \rightsquigarrow \prod_{i=1}^{m} \mathbb{P}[\mu_i] &= \prod_{i=1}^{m} \prod_{v \in \mu_i} p\bigl(\mathtt{subtree\_size}_{\mu_i}(v.\mathtt{left}), \mathtt{subtree\_size}_{\mu_i}(v.\mathtt{right})\bigr) \\ &\geqslant \prod_{i=1}^{m} \prod_{v \in \mu_i} p\bigl(\mathtt{subtree\_size}_{t}(v.\mathtt{left}), \mathtt{subtree\_size}_{t}(v.\mathtt{right})\bigr) \\ &= \prod_{v \in t} p\bigl(\mathtt{subtree\_size}_{t}(v.\mathtt{left}), \mathtt{subtree\_size}_{t}(v.\mathtt{right})\bigr) = \mathbb{P}[t] \end{aligned}$$

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!

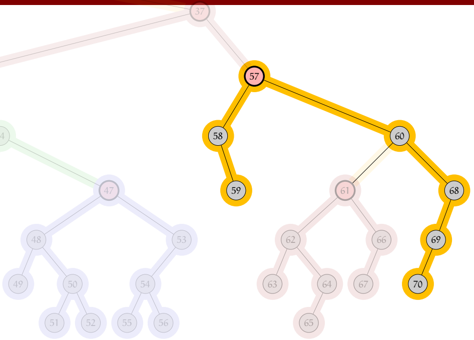
But:
- ℓ_v, r_v in μ_i only smaller, and
- $\boxed{p(\ell+1, r) \leqslant p(\ell, r) \text{ and } p(\ell, r+1) \leqslant p(\ell, r)}$ (**monotonic** source)

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

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

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

Random BSTs – Monotonicity



$\boxed{\textbf{Step 3}}$ 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 – Conclusion

Step 4 Only have to put things together now:

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

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

Random BSTs

- $\displaystyle\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\mathtt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\displaystyle\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim \underset{\nwarrow\ \sum_{k=1}^{\infty} \frac{2\lg(k)}{(k+1)(k+2)}}{1.736}n$

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: $\prod_{i=1}^{m} \mathbb{P}[\mu_i] \;\geqslant\; \mathbb{P}[t]$

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

Random BSTs

- $\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\mathtt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim \underset{\uparrow\; \sum_{k=1}^{\infty} \frac{2\lg(k)}{(k+1)(k+2)}}{1.736n}$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

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

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

Random BSTs

- $\displaystyle\lg(1/\mathbb{P}[t]) = \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - This is also the splay tree potential!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim \underset{\displaystyle\uparrow\atop\displaystyle\sum_{k=1}^{\infty}\frac{2\lg(k)}{(k+1)(k+2)}}{1.736}n$

Random BSTs – Conclusion

Step 4 Only have to put things together now:

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

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

$$\leqslant\; \sum_{i=1}^{m} \lg(1/\mathbb{P}[\mu_i]) \;+\; o(n)$$

$$\leqslant\; \lg(1/\mathbb{P}[t]) \;+\; o(n)$$

Random BSTs

- $\displaystyle\lg(1/\mathbb{P}[t]) \;=\; \sum_{v \in t} \lg(\texttt{subtree\_size}(v))$
  - 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: $\sum_{i=1}^{m} |C(\mu_i)| \leqslant \sum_{i=1}^{m} \left(\lg(1/\mathbb{P}[\mu_i]) + 2 \right)$
- 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!
- $\mathbb{E}[\lg(1/\mathbb{P}[t])] \sim \underset{\uparrow\ \sum_{k=1}^{\infty} \frac{2\lg(k)}{(k+1)(k+2)}}{1.736}n$

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

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

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

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

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

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

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

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

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

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

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

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

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)
  - $\boxed{\text{only } O(n/B) \text{ nodes have subtree size} \geqslant B}$ (not obvious, but not hard to prove)

    Keep this in mind!
  - Can be generated with a fixed-size source using

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

general recipe for uniform distributions

but **not monotonic**

Weight-Balanced BSTs – Problems

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

Weight-Balanced BSTs – Problems

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

⇝ Cannot possibly hope to show $$\prod_{i=1}^{m} \mathbb{P}[\underset{\nwarrow\ \text{potentially } 0}{\mu_i}] \;\geqslant\; \mathbb{P}[t]$$

Weight-Balanced BSTs – Problems

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

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

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 $\displaystyle\prod_{i=1}^{m}\mathbb{P}[\mu_i] \geqslant \mathbb{P}[t]$

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 $D_{\mathcal{S}}$, break up micro trees into

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

⇝ $D_{\mathcal{S}}$ 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

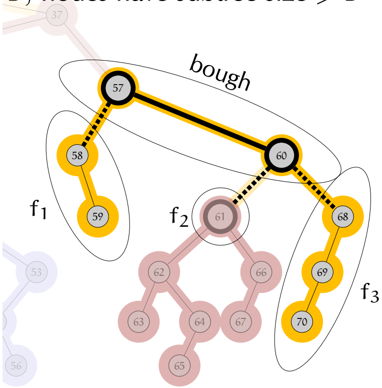
⇝ Only boughs stored suboptimally and these are a vanishing fraction of t.

Weight-Balanced BSTs – Great-Branching Code

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

Inside $D_{\mathcal{S}}$, break up micro trees into

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



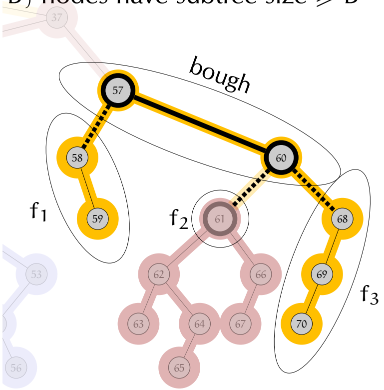
⇝ $D_{\mathcal{S}}$ stores

- 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

⇝ 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 $D_{\mathcal{S}}$, break up micro trees into

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

⇝ $D_{\mathcal{S}}$ 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

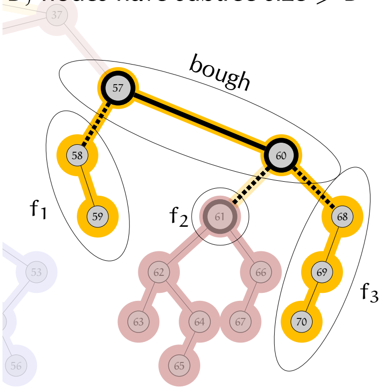
⇝ Only boughs stored suboptimally and these are a vanishing fraction of t.

Weight-Balanced BSTs – Great-Branching Code

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

Inside $D_{\mathcal{S}}$, break up micro trees into

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



⇝ $D_{\mathcal{S}}$ 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

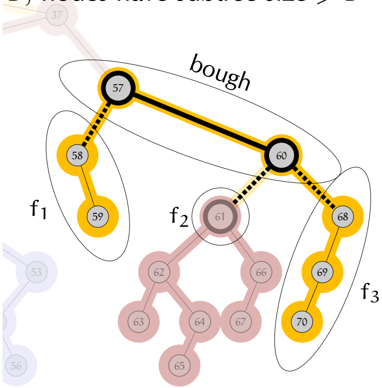
⇝ Only boughs stored suboptimally and these are a vanishing fraction of t.

Weight-Balanced BSTs – Great-Branching Code

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

Inside $D_{\mathcal{S}}$, break up micro trees into

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



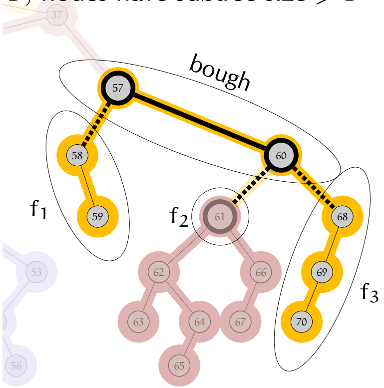
⇝ $D_{\mathcal{S}}$ stores

- 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

⇝ 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 $D_{\mathcal{S}}$, break up micro trees into

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

⇝ $D_{\mathcal{S}}$ 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

⇝ Only boughs stored suboptimally and these are a vanishing fraction of t.

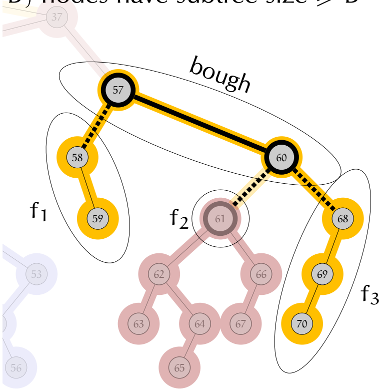
Weight-Balanced BSTs – Great-Branching Code

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

Inside $D_{\mathcal{S}}$, break up micro trees into

① "boughs" of heavy nodes

② fringe-subtrees $f_{i,j}$ ("twigs") hanging off boughs



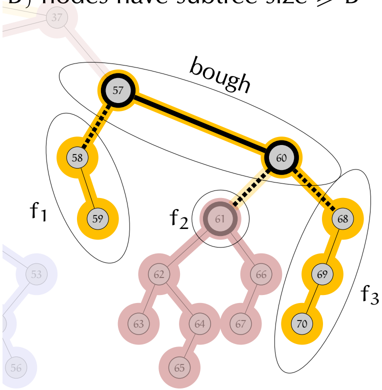
⇝ $D_{\mathcal{S}}$ stores

- 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

⇝ 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 $D_{\mathcal{S}}$, break up micro trees into

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

⇝ $D_{\mathcal{S}}$ 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

⇝ Only boughs stored suboptimally and these are a vanishing fraction of t.

Outline

1 **Hypersuccinct Trees**

2 **Two Favorite Trees**

3 **Beyond Trees**

4 **Bonus: Range-Minimum Queries**

5 **Bonus: Succinct Bitvectors**

3 Beyond Trees

How about graphs?

❶ **Does efficient computation and/or distributed storage have an intrinsic space cost?**

largely missing:

- an information theory of (graph-)structured data
  How much space is needed to store a graph?
- When and how can we achieve such space with
  - efficient queries?
  - distributed storage?

Besta, Hoefler: *Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations*, arXiv **2018**

Spinrad: *Efficient graph representations*, Fields monographs **2003**

② Work towards a **hypersuccinct** graph representation

- succinct representations of *distributions* of graphs
- universal codes?

Graphs here are

- undirected
- unlabeled
- static

How about graphs?

① **Does efficient computation and/or distributed storage have an intrinsic space cost?**

largely missing:

- an information theory of (graph-)structured data
  How much space is needed to store a graph?
- When and how can we achieve such space with
  - efficient queries?
  - distributed storage?

📘 **Besta, Hoefler**: *Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations*, arXiv **2018**

📘 **Spinrad**: *Efficient graph representations*, Fields monographs **2003**

② Work towards a **hypersuccinct** graph representation

- succinct representations of *distributions* of graphs
- universal codes?

Graphs here are

- undirected
- unlabeled
- static

How about graphs?

① **Does efficient computation and/or distributed storage have an intrinsic space cost?**

largely missing:

- an information theory of (graph-)structured data
  How much space is needed to store a graph?
- When and how can we achieve such space with
  - efficient queries?
  - distributed storage?

Besta, Hoefler: *Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations*, arXiv **2018**

Spinrad: *Efficient graph representations*, Fields monographs **2003**

Graphs here are

- undirected
- unlabeled
- static

② Work towards a **hypersuccinct** graph representation

- succinct representations of *distributions* of graphs
- universal codes?

How about graphs?

① **Does efficient computation and/or distributed storage have an intrinsic space cost?**

largely missing:

- an information theory of (graph-)structured data
  How much space is needed to store a graph?
- When and how can we achieve such space with
  - efficient queries?
  - distributed storage?

Besta, Hoefler: *Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations*, arXiv **2018**

Spinrad: *Efficient graph representations*, Fields monographs **2003**

② Work towards a **hypersuccinct** graph representation

- succinct representations of *distributions* of graphs
- universal codes?

Graphs here are

- undirected
- unlabeled
- static

How about graphs?

❶ **Does efficient computation and/or distributed storage have an intrinsic space cost?**

largely missing:

- an information theory of (graph-)structured data
  How much space is needed to store a graph?
- When and how can we achieve such space with
  - efficient queries?
  - distributed storage?

Besta, Hoefler: *Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations*, arXiv **2018**

Spinrad: *Efficient graph representations*, Fields monographs **2003**

(❷ Work towards a **hypersuccinct** graph representation

- succinct representations of *distributions* of graphs
- universal codes?)

Graphs here are

- undirected
- unlabeled
- static
- uniformly taken from a ground set

How to count graphs?

How many graphs of size n are there in a family $\mathcal{F}$?

$\mathcal{F}^n$: set of **labeled graphs** of size n

$\mathcal{F}_n$: set equivalence classes (under graph isomorphisms) of $\mathcal{F}^n$;
set of **unlabeled graphs** of size n

- $\mathcal{F}$ = all complete graphs
  $|\mathcal{F}^n| = |\mathcal{F}_n| = 1$ (the single complete graph over $[n]$)
- $\mathcal{F}$ = at most one edge in total
  $|\mathcal{F}^n| = 1 + \binom{n}{2}$ (1 empty graph plus $\binom{n}{2}$ ways to pick a pair for the single edge)
  $|\mathcal{F}_n| = 2$

Expert note: Concatenating all labels of a labeling scheme allows to reconstruct the *equivalence class* (under graph isomorphisms), but not the *labeled graph* – unless the original labels are made part of $\ell(v)$.

$\rightsquigarrow$ need to pay attention

Space-efficient graph representations

Given a (hereditary) graph family $\mathcal{F}$, we define

$\mathcal{F}$ is a hereditary graph family if

- $\mathcal{F}$ closed under isomorphism
- $\mathcal{F}$ closed under taking **induced subgraphs**

⇝ $\mathcal{F}^n$: graphs $G \in \mathcal{F}$ with vertex set $V(G) = [n]$

① A **succinct encoding** of $\mathcal{F}$:

$encode : \mathcal{F} \to \{0,1\}^*, \quad decode : \{0,1\}^* \to \mathcal{F}$

- lossless: $decode(encode(G)) = G$ for $G \in \mathcal{F}$.
- **succinct**: $G \in \mathcal{F}^n$ ⇝ $|encode(G)| = \log_2(|\mathcal{F}^n|) \cdot (1 + o(1))$
- **efficient**: *encode*, *decode* efficiently computable (say polytime)

② A **succinct data structure** for $\mathcal{F}$ (for adjacency):

succinct encoding plus adjacency-list queries

- $adjacent(v,u)$: 1 if $vu \in E(G)$ else 0
- $nextNeigbor(v,u)$: successor of u in v's adj list
- computable efficiently on word-RAM say $o(\log(|\mathcal{F}^n|))$ time; often $O(1)$
- (potentially more queries)

③ A **succinct labeling scheme** for $\mathcal{F}$ (for adjacency):

$\ell : V(G) \to \{0,1\}^*$,
$labelAdj : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$

- $labelAdj(\ell(v), \ell(u)) = adjacent(v,u)$ for $v,u \in V(G)$.
- **succinct**: $G \in \mathcal{F}^n$ ⇝ $|\ell(v)| \leqslant \frac{1}{n}\log_2(|\mathcal{F}^n|)(1 + o(1))$
- weaker version: **compact**: $|\ell(v)| = O(\frac{1}{n}\log_2(|\mathcal{F}^n|))$
- (labels and decoder can differ for each $G \in \mathcal{F}$, but heredity puts limits on that)

Space-efficient graph representations

Given a (hereditary) graph family $\mathcal{F}$, we define

① A **succinct encoding** of $\mathcal{F}$:

$encode : \mathcal{F} \to \{0,1\}^*$, $decode : \{0,1\}^* \to \mathcal{F}$

- lossless: $decode(encode(G)) = G$ for $G \in \mathcal{F}$.
- **succinct**: $G \in \mathcal{F}^n$ ⇝ $|encode(G)| = \log_2(|\mathcal{F}^n|) \cdot (1 + o(1))$
- **efficient**: *encode*, *decode* efficiently computable (say polytime)

② A **succinct data structure** for $\mathcal{F}$ (for adjacency):

succinct encoding plus adjacency-list queries

- $adjacent(v, u)$: 1 if $vu \in E(G)$ else 0
- $nextNeigbor(v, u)$: successor of u in v's adj list
- computable efficiently on word-RAM say $o(\log(|\mathcal{F}^n|))$ time; often $O(1)$
- (potentially more queries)

$\mathcal{F}$ is a hereditary graph family if

- $\mathcal{F}$ closed under isomorphism
- $\mathcal{F}$ closed under taking **induced subgraphs**

⇝ $\mathcal{F}^n$: graphs $G \in \mathcal{F}$ with vertex set $V(G) = [n]$

③ A **succinct labeling scheme** for $\mathcal{F}$ (for adjacency):

$\ell : V(G) \to \{0,1\}^*$,
$labelAdj : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$

- $labelAdj(\ell(v), \ell(u)) = adjacent(v, u)$ for $v, u \in V(G)$.
- **succinct**: $G \in \mathcal{F}^n$ ⇝ $|\ell(v)| \leqslant \frac{1}{n} \log_2(|\mathcal{F}^n|)(1 + o(1))$
- weaker version: **compact**: $|\ell(v)| = O(\frac{1}{n} \log_2(|\mathcal{F}^n|))$
- (labels and decoder can differ for each $G \in \mathcal{F}$, but heredity puts limits on that)

Space-efficient graph representations

Given a (hereditary) graph family $\mathcal{F}$, we define

① A **succinct encoding** of $\mathcal{F}$:

$encode : \mathcal{F} \to \{0,1\}^*,\ \ decode : \{0,1\}^* \to \mathcal{F}$

- lossless: $decode(encode(G)) = G$ for $G \in \mathcal{F}$.
- **succinct**: $G \in \mathcal{F}^n \rightsquigarrow$ $|encode(G)| = \log_2(|\mathcal{F}^n|) \cdot (1 + o(1))$
- **efficient**: *encode*, *decode* efficiently computable (say polytime)

② A **succinct data structure** for $\mathcal{F}$ (for adjacency):

succinct encoding plus adjacency-list queries

- $adjacent(v,u)$: 1 if $vu \in E(G)$ else 0
- $nextNeigbor(v,u)$: successor of u in v's adj list
- computable efficiently on word-RAM say $o(\log(|\mathcal{F}^n|))$ time; often $O(1)$
- (potentially more queries)

$\mathcal{F}$ is a hereditary graph family if

- $\mathcal{F}$ closed under isomorphism
- $\mathcal{F}$ closed under taking **induced subgraphs**

$\rightsquigarrow$ $\mathcal{F}^n$: graphs $G \in \mathcal{F}$ with vertex set $V(G) = [n]$

③ A **succinct labeling scheme** for $\mathcal{F}$ (for adjacency):

$\ell : V(G) \to \{0,1\}^*,$
$labelAdj : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$

- $labelAdj(\ell(v), \ell(u)) = adjacent(v,u)$ for $v, u \in V(G)$.
- **succinct**: $G \in \mathcal{F}^n \rightsquigarrow$ $|\ell(v)| \leqslant \frac{1}{n}\log_2(|\mathcal{F}^n|)(1 + o(1))$
- weaker version: **compact**: $|\ell(v)| = O(\frac{1}{n}\log_2(|\mathcal{F}^n|))$
- (labels and decoder can differ for each $G \in \mathcal{F}$, but heredity puts limits on that)

Space-efficient graph representations

Given a (hereditary) graph family $\mathcal{F}$, we define

❶ A **succinct encoding** of $\mathcal{F}$:

$\mathit{encode} : \mathcal{F} \to \{0,1\}^*$, $\mathit{decode} : \{0,1\}^* \to \mathcal{F}$

- lossless: $\mathit{decode}(\mathit{encode}(G)) = G$ for $G \in \mathcal{F}$.
- **succinct**: $G \in \mathcal{F}^n \rightsquigarrow$ $|\mathit{encode}(G)| = \log_2(|\mathcal{F}^n|) \cdot (1 + o(1))$
- **efficient**: *encode*, *decode* efficiently computable (say polytime)

❷ A **succinct data structure** for $\mathcal{F}$ (for adjacency):

succinct encoding plus adjacency-list queries

- $\mathit{adjacent}(v,u)$: 1 if $vu \in E(G)$ else 0
- $\mathit{nextNeigbor}(v,u)$: successor of u in v's adj list
- computable efficiently on word-RAM say $o(\log(|\mathcal{F}^n|))$ time; often $O(1)$
- (potentially more queries)

$\mathcal{F}$ is a hereditary graph family if

- $\mathcal{F}$ closed under isomorphism
- $\mathcal{F}$ closed under taking **induced subgraphs**

$\rightsquigarrow$ $\mathcal{F}^n$: graphs $G \in \mathcal{F}$ with vertex set $V(G) = [n]$

❸ A **succinct labeling scheme** for $\mathcal{F}$ (for adjacency):

$\ell : V(G) \to \{0,1\}^*$,
$\mathit{labelAdj} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$

- $\mathit{labelAdj}(\ell(v), \ell(u)) = \mathit{adjacent}(v,u)$ for $v, u \in V(G)$.
- **succinct**: $G \in \mathcal{F}^n \rightsquigarrow$ $|\ell(v)| \leqslant \frac{1}{n} \log_2(|\mathcal{F}^n|)(1 + o(1))$
- weaker version: **compact**: $|\ell(v)| = O(\frac{1}{n} \log_2(|\mathcal{F}^n|))$
- (labels and decoder can differ for each $G \in \mathcal{F}$, but heredity puts limits on that)

Questions

① Do all graph families have a succinct encoding?

② Do all graph families have a succinct data structure (with adjacency queries)?

③ Do all graph families have a compact (adjacency) labeling scheme?

Questions

① **Do all graph families have a succinct encoding?**

- No, if we have to recognize graphs in the family.
- If *encode* is not required to work correctly on $G \notin \mathcal{F}$??

Spinrad: *Efficient graph representations,* Fields monographs **2003**

② **Do all graph families have a succinct data structure (with adjacency queries)?**

③ **Do all graph families have a compact (adjacency) labeling scheme?**

Questions

① **Do all graph families have a succinct encoding?**

- No, if we have to recognize graphs in the family.
- If *encode* is not required to work correctly on $G \notin \mathcal{F}$??

Spinrad: *Efficient graph representations,* Fields monographs **2003**

② **Do all graph families have a succinct data structure (with adjacency queries)?**

- No, for graphs with m edges and $n^{\delta} < m < n^{2-\delta}$, ($\delta > 0$ constant) when we want *adjacent* and *nextNeighbor* in $O(1)$ time

  But this is not a hereditary property.
- For hereditary classes?? Only faster-than-decompress queries??

Farzan, Munro: *Succinct encoding of arbitrary graphs,* TCS **2013**

③ **Do all graph families have a compact (adjacency) labeling scheme?**

Questions

1. **Do all graph families have a succinct encoding?**
    - No, if we have to recognize graphs in the family.
    - If *encode* is not required to work correctly on $G \notin \mathcal{F}$??

Spinrad: *Efficient graph representations,* Fields monographs **2003**

2. **Do all graph families have a succinct data structure (with adjacency queries)?**
    - No, for graphs with m edges and $n^\delta < m < n^{2-\delta}$, ($\delta > 0$ constant) when we want *adjacent* and *nextNeighbor* in $O(1)$ time

      But this is not a hereditary property.
    - For hereditary classes?? Only faster-than-decompress queries??

Farzan, Munro: *Succinct encoding of arbitrary graphs,* TCS **2013**

3. **Do all graph families have a compact (adjacency) labeling scheme?**
    - **Recent news**: Resounding No!
    - Which families do??

Hatami, Hatami: *The Implicit Graph Conjecture is False,* FOCS **2022**

Summary

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
- isolated other combinatorial structures

- Entropy of micro-tree distribution is an interesting parameter
  - direct implication for succinct data structures
  - yields some twists in the analysis
- Is the hypersuccinct code asymptotically optimal for more tree distributions?

Summary

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
- isolated other combinatorial structures

- Entropy of micro-tree distribution is an interesting parameter
  - direct implication for succinct data structures
  - yields some twists in the analysis
- Is the hypersuccinct code asymptotically optimal for more tree distributions?

Summary

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
- isolated other combinatorial structures

- Entropy of micro-tree distribution is an interesting parameter
  - direct implication for succinct data structures
  - yields some twists in the analysis
- Is the hypersuccinct code asymptotically optimal for more tree distributions?

Outline


1 Hypersuccinct Trees


2 Two Favorite Trees


3 Beyond Trees


4 Bonus: Range-Minimum Queries


5 Bonus: Succinct Bitvectors

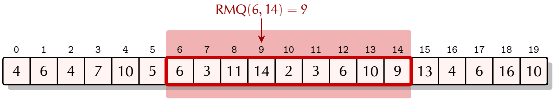
4 Bonus: Range-Minimum Queries

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

RMQ$(6, 14) = 9$

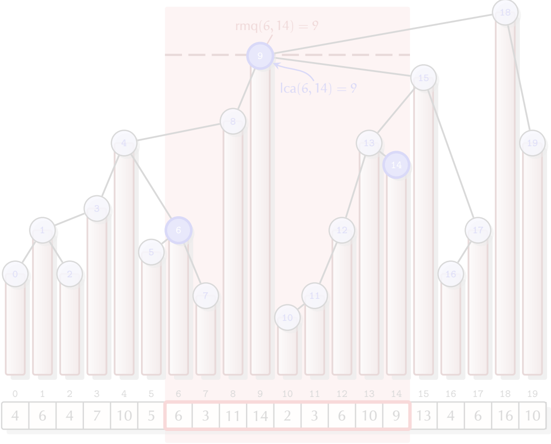


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

- 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:



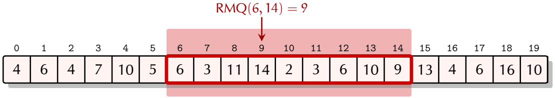
rmq$(6, 14) = 9$

lca$(6, 14) = 9$

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



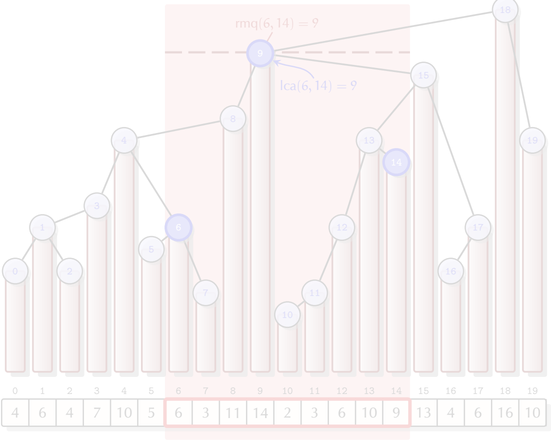
RMQ(6, 14) = 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

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

lowest common ancestor

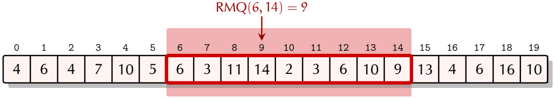
RMQ is **equivalent** to LCA in binary trees:



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

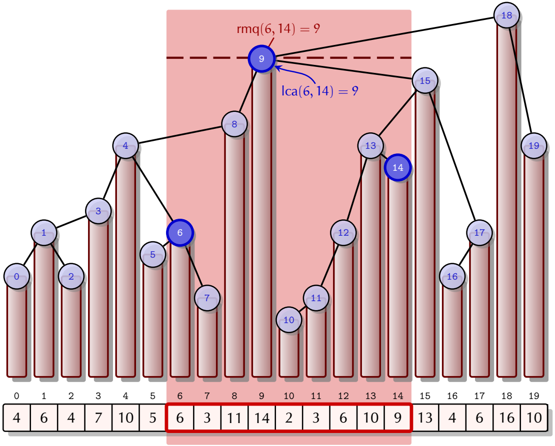


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

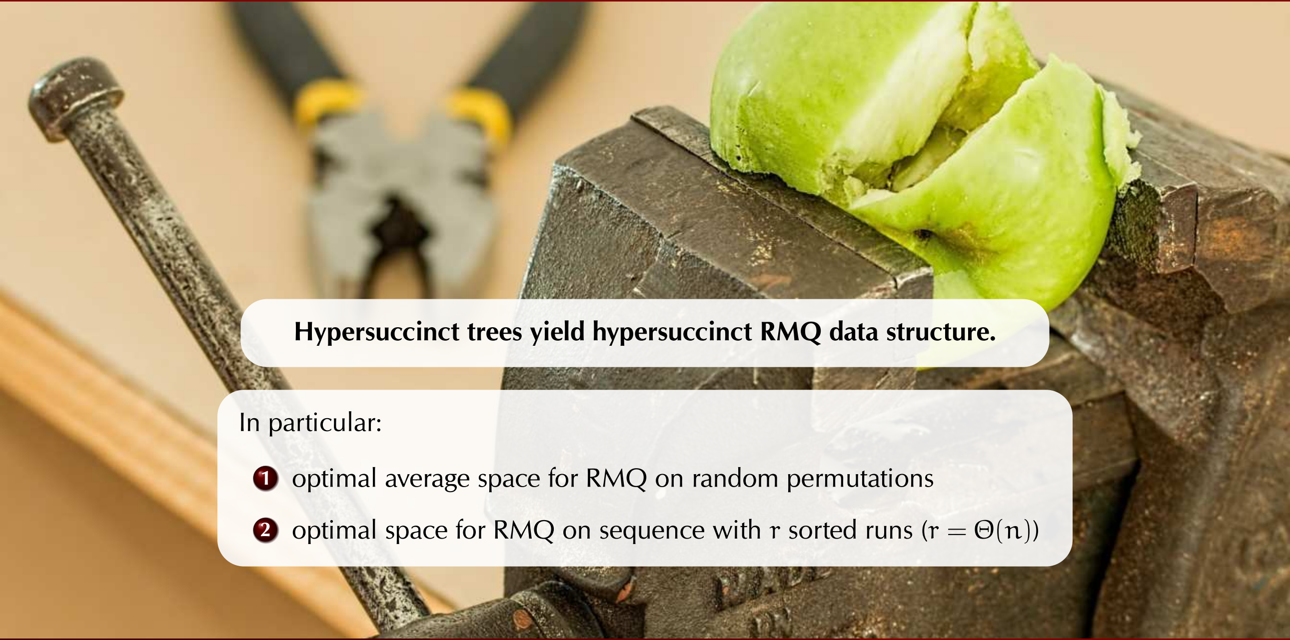
- **Nitpicks:**
  - Report **index** of maximum, not its value
  - Report **leftmost** position in case of ties

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 ($r = \Theta(n)$)

Outline

1 Hypersuccinct Trees

2 Two Favorite Trees

3 Beyond Trees

4 Bonus: Range-Minimum Queries

5 Bonus: Succinct Bitvectors

5 Bonus: Succinct bitvectors

Computing over compressed data

- traditionally:
  - compression for **archiving** data, minimize size of representation
  - computation/analysis: minimize time; use extra data structures

  ⇝ always decompress data first!
  - reaches limits of fast memory for large datasets

- Approach in space-efficient data structures:
  - Represent data in compressed form
  - Augment with **small** index data structures to enable fast queries **directly on compressed representation**
  - **succinct** $= (1 + o(1)) \cdot$ information-theoretic lower bound

Computing over compressed data

- traditionally:
  - compression for **archiving** data, minimize size of representation
  - computation/analysis: minimize time; use extra data structures
  - $\rightsquigarrow$ always decompress data first!
  - reaches limits of fast memory for large datasets
- Approach in space-efficient data structures:
  - Represent data in compressed form
  - Augment with **small** index data structures to enable fast queries **directly on compressed representation**
  - **succinct** $= (1 + o(1)) \cdot$ information-theoretic lower bound

A motivating example

- Suppose you store a text $T[1..n]$ compressed with a Huffman code
- C stores concatenation of all codewords
- Would like to allow random access to $T[i]$

each char encoded separately;
codewords of variable length

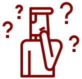
Huffman code for
Alice in Wonderland

A	1110
B	010110
C	01010
D	11111
E	100
F	110010
G	00001
H	0111
I	1011
J	000111011
K	000110
L	11110
M	110011
N	1010
O	1101
P	010111
Q	000111010
R	0100
S	0110
T	001
U	11000
V	0001111
W	00010
X	000111001
Y	00000
Z	000111000

A motivating example

- Suppose you store a text $T[1..n]$ compressed with a Huffman code
- C stores concatenation of all codewords
- Would like to allow random access to $T[i]$

each char encoded separately;
codewords of variable length



How to know where ith character starts?

Huffman code for
Alice in Wonderland

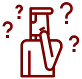
A	1110
B	010110
C	01010
D	11111
E	100
F	110010
G	00001
H	0111
I	1011
J	000111011
K	000110
L	11110
M	110011
N	1010
O	1101
P	010111
Q	000111010
R	0100
S	0110
T	001
U	11000
V	0001111
W	00010
X	000111001
Y	00000
Z	000111000

A motivating example

- Suppose you store a text $T[1..n]$ compressed with a Huffman code

each char encoded separately; codewords of variable length

- C stores concatenation of all codewords
- Would like to allow random access to $T[i]$

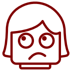


How to know where ith character starts?

unless we decode from start

- We don't. But we can store it!
  - Naive way: Store starting index for ith char in T in $S[1..n]$

$\rightsquigarrow$ n numbers in $[n]$ $\rightsquigarrow$ $n \lg n$ bits.
That's much more than the (compressed) text!



  - Can we do better?

Huffman code for *Alice in Wonderland*

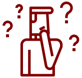
A	1110
B	010110
C	01010
D	11111
E	100
F	110010
G	00001
H	0111
I	1011
J	000111011
K	000110
L	11110
M	110011
N	1010
O	1101
P	010111
Q	000111010
R	0100
S	0110
T	001
U	11000
V	0001111
W	00010
X	000111001
Y	00000
Z	000111000

A motivating example

- Suppose you store a text $T[1..n]$ compressed with a Huffman code

each char encoded separately; codewords of variable length

- C stores concatenation of all codewords
- Would like to allow random access to $T[i]$

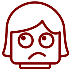


How to know where ith character starts?

unless we decode from start

- We don't. But we can store it!
  - Naive way: Store starting index for ith char in T in $S[1..n]$

$\rightsquigarrow$ n numbers in $[n]$ $\rightsquigarrow$ $n \lg n$ bits.
That's much more than the (compressed) text!



  - Can we do better?

Yes! With $o(n)$ extra bits, we can support constant(!)-time random access!

Huffman code for
Alice in Wonderland

A	1110
B	010110
C	01010
D	11111
E	100
F	110010
G	00001
H	0111
I	1011
J	000111011
K	000110
L	11110
M	110011
N	1010
O	1101
P	010111
Q	000111010
R	0100
S	0110
T	001
U	11000
V	0001111
W	00010
X	000111001
Y	00000
Z	000111000

Bitvectors

- $B[1..n]$ static array of n bits (Boolean array).
  - trivial to store using n bits of space
- $\mathrm{rank}_B(i) = \#\,1\mathrm{s}$ in $B[1..i]$ (first i positions) (= prefix sum)
- $\mathrm{select}_B(i) =$ position of ith 1 in B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0

- $\mathrm{rank}_B(12) = \mathrm{rank}_B(13) = 4$
- $\mathrm{select}_B(3) = 11$
- $\mathrm{select}_B(4) = 12$

Goal: Support rank and select on bitvector using $n + o(n)$ bits of space. [Jacobson 1988], [Clark 1996]

⇝ Will show how to do rank; select is similar

Bitvectors

- $B[1..n]$ static array of n bits (Boolean array).
  - trivial to store using n bits of space
- $\mathrm{rank}_B(i) = \#\,1\mathrm{s}$ in $B[1..i]$ (first i positions) (= prefix sum)
- $\mathrm{select}_B(i) =$ position of ith 1 in B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0

- $\mathrm{rank}_B(12) = \mathrm{rank}_B(13) = 4$
- $\mathrm{select}_B(3) = 11$
- $\mathrm{select}_B(4) = 12$

Goal: Support rank and select on bitvector using $n + o(n)$ bits of space. [Jacobson 1988], [Clark 1996]

⇝ Will show how to do rank; select is similar

Bitvectors

- $B[1..n]$ static array of n bits (Boolean array).
  - trivial to store using n bits of space
- $\mathrm{rank}_B(i) = \#\,1\mathrm{s}$ in $B[1..i]$ (first i positions) (= prefix sum)
- $\mathrm{select}_B(i) =$ position of ith 1 in B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	1	0	0	0	0	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0

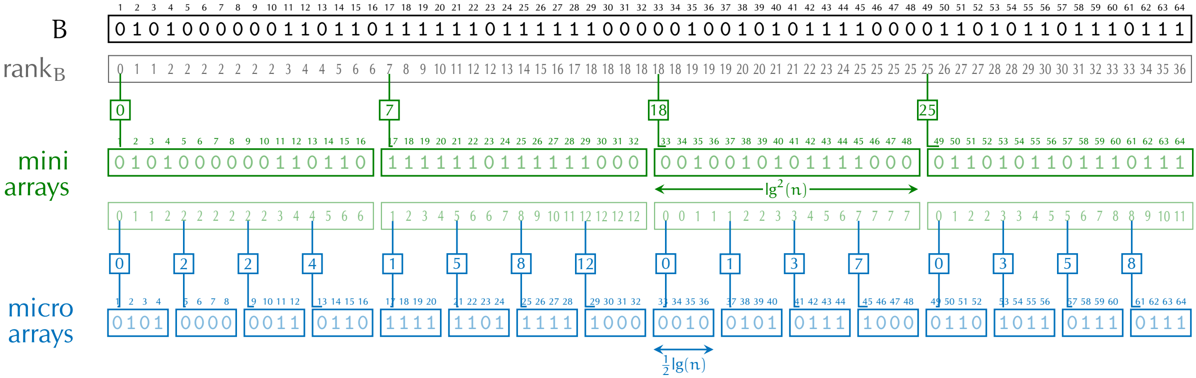
- $\mathrm{rank}_B(12) = \mathrm{rank}_B(13) = 4$
- $\mathrm{select}_B(3) = 11$
- $\mathrm{select}_B(4) = 12$

Goal: Support rank and select on bitvector using $n + o(n)$ bits of space. [Jacobson 1988], [Clark 1996]

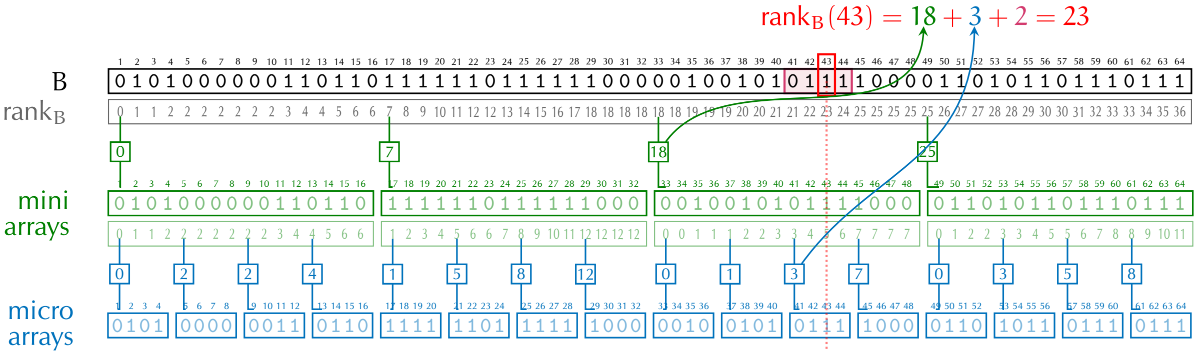
$\rightsquigarrow$ Will show how to do rank; select is similar

Rank Index for Bitvector

- Apart from B, we store:
  - Rank of first element of each mini array ⇝ $\frac{n}{\lg(n)^2} \cdot \lg(n) = \frac{n}{\lg n} = o(n)$ bits
  - Rank of first element in micro array *relative* to its mini array
    ⇝ ranks are numbers in $[\lg^2(n)]$ ⇝ $2 \lg \lg n$ bits suffice for each
    ⇝ $\frac{n}{\frac{1}{2}\lg n} \cdot 2 \lg \lg n = \frac{4n \lg \lg n}{\lg n} = o(n)$ bits



How to find rank


$\mathrm{rank}_B(43) = 18 + 3 + 2 = 23$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64

B: 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1 0 1 1 1

rank_B: 0 1 1 2 2 2 2 2 2 2 3 4 4 5 6 6 7 8 9 10 11 12 12 13 14 15 16 17 18 18 18 18 18 18 19 19 19 20 20 21 21 22 23 24 25 25 25 25 25 26 27 27 28 28 29 30 30 31 32 33 33 34 35 36

mini arrays: 0 | 7 | 18 | 25

1–16: 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1 0 — 0 1 1 2 2 2 2 2 2 2 3 4 4 5 6 6

17–32: 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 — 1 2 3 4 5 6 6 7 8 9 10 11 12 12 12 12

33–48: 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 — 0 0 1 1 1 2 2 3 3 4 5 6 7 7 7 7

49–64: 0 1 1 0 1 0 1 1 0 1 1 1 0 1 1 1 — 0 1 2 2 3 3 4 5 5 6 7 8 8 9 10 11

micro arrays: 0 2 2 4 | 1 5 8 12 | 0 1 3 7 | 0 3 5 8

1–4: 0101 | 5–8: 0000 | 9–12: 0011 | 13–16: 0110 | 17–20: 1111 | 21–24: 1101 | 25–28: 1111 | 29–32: 1000 | 33–36: 0010 | 37–40: 0101 | 41–44: 0111 | 45–48: 1000 | 49–52: 0110 | 53–56: 1011 | 57–60: 0111 | 61–64: 0111

- How to compute $\mathrm{rank}_B(i)$?
  - find rank up to element's mini array
  - add rank up to element's micro array (mini-array local)
  - add micro-array-local rank of position
    - can either do this naively by scanning $\frac{1}{2}\lg n$ bits $\rightsquigarrow$ $O(\log n)$ time
    - or use bit marks and pop-count instructions on CPUs $\rightsquigarrow$ $O(1)$ time
    - or use exhaustive *lookup table!* $\frac{1}{2}\lg n$ bits $\rightsquigarrow$ only $2^{\frac{1}{2}\lg n} = \sqrt{n}$ different micro arrays

Huffman application

- Recall the motivating toy problem: random access to Huffman coded text
  - Idea: Use a bitvector to mark beginning of codewords

  $\rightsquigarrow$ Can use select to find ith codeword

Example: `abananaandanapple`

Huffman code: $\mathtt{a} = 0, \mathtt{b} = 11100, \mathtt{d} = 11101, \mathtt{e} = 11110, \mathtt{l} = 11111, \mathtt{n} = 10, \mathtt{p} = 110$

$C = 01110001001000101110101001101101111111110$ concatenation of codewords

$B = 11000011011011101000011011001001000010000$ bitvector of codeword start

Huffman application

- Recall the motivating toy problem: random access to Huffman coded text
  - Idea: Use a bitvector to mark beginning of codewords
  - ⇝ Can use select to find ith codeword

Example: `abananaandanapple`
Huffman code: $\mathtt{a} = 0, \mathtt{b} = 11100, \mathtt{d} = 11101, \mathtt{e} = 11110, \mathtt{l} = 11111, \mathtt{n} = 10, \mathtt{p} = 110$
$C = 0111000100100010111010100110110111111110$ concatenation of codewords
$B = 1100001101101110100001101100100100010000$ bitvector of codeword start

Huffman application

- Recall the motivating toy problem: random access to Huffman coded text
  - Idea: Use a bitvector to mark beginning of codewords

⇝ Can use select to find ith codeword

Example: `abananaandanapple`

Huffman code: $\mathtt{a} = \mathtt{0}, \mathtt{b} = \mathtt{11100}, \mathtt{d} = \mathtt{11101}, \mathtt{e} = \mathtt{11110}, \mathtt{l} = \mathtt{11111}, \mathtt{n} = \mathtt{10}, \mathtt{p} = \mathtt{110}$

$C = \mathtt{011100010010001011101010011011011111111110}$ concatenation of codewords

$B = \mathtt{110000110110111010000110110010010000010000}$ bitvector of codeword start

Huffman application

- Recall the motivating toy problem: random access to Huffman coded text
  - Idea: Use a bitvector to mark beginning of codewords
  - ⇝ Can use select to find ith codeword

Example: `abananaandanapple`
Huffman code: $\mathtt{a} = 0, \mathtt{b} = 11100, \mathtt{d} = 11101, \mathtt{e} = 11110, \mathtt{l} = 11111, \mathtt{n} = 10, \mathtt{p} = 110$
$C = 01110001001000101110101001101101111111110$ concatenation of codewords
$B = 11000011011011101000011011001001000010000$ bitvector of codeword start

- Note: support for constant-time rank/select only needs $o(n)$ bits on top of B.
- But would ideally not want to store B! (only C)

Huffman application

- Recall the motivating toy problem: random access to Huffman coded text
  - Idea: Use a bitvector to mark beginning of codewords
  - ⇝ Can use select to find ith codeword

Example: `abananaandanapple`
Huffman code: $\texttt{a} = 0, \texttt{b} = 11100, \texttt{d} = 11101, \texttt{e} = 11110, \texttt{l} = 11111, \texttt{n} = 10, \texttt{p} = 110$
$C = 0111000100100010111010100110110111111110$ concatenation of codewords
$B = 1100001101101110100001101100100100010000$ bitvector of codeword start

- Note: support for constant-time rank/select only needs $o(n)$ bits on top of B.
- But would ideally not want to store B! (only C)

Can compute micro-array contents of B on-the-fly!

- store number of leading 0s in micro array ⇝ $\lg\lg n$ bits per micro array ⇝ $o(n)$
- when we need a micro array, reconstruct, from C and Huffman code (skipping suffix of first codeword) can be done in $O(1)$ via a lookup table

Other Succinct Data Structures

- flourishing field
- succinct data structures exist for various other objects
  - sequences
  - permutations
  - some classes of trees
  - some classes of graphs
  - some geometric data structures
- found wide adoption in practice through programming libraries

Outline

1 Hypersuccinct Trees

2 Two Favorite Trees

3 Further interests

4 Bonus: Range-Minimum Queries

5 Bonus: Succinct Bitvectors

6 Full results

6 Full results

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
  - ⇝ widely applicable compression method

⇝ *(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:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\quad\mathrm{type}(v) \in \{$  $\}$
- *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

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\mathtt{subtree\_size}(v.\mathtt{left}), \mathtt{subtree\_size}(v.\mathtt{right}))$
- ***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:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\qquad$ $\mathrm{type}(v) \in \{$  $\}$
- *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

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\mathtt{subtree\_size}(v.\mathtt{left}), \mathtt{subtree\_size}(v.\mathtt{right}))$
- ***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:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\qquad \mathrm{type}(v) \in \{$  $\}$
- 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

  $\rightsquigarrow$ $\mathbb{P}[t] = \prod_{v \in t} p(\mathtt{subtree\_size}(v.\mathtt{left}), \mathtt{subtree\_size}(v.\mathtt{right}))$
- ***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:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\qquad \mathrm{type}(v) \in \{$  $\}$
- 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 $\mathcal{S}$ $\quad = \quad$ prob. distribution over tree shapes with a filter, e. g., tree of size n

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

Studied sources:

- *memoryless **type** process:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\qquad \mathrm{type}(v) \in \{$  $\}$
- *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

  $\rightsquigarrow \mathbb{P}[t] = \prod_{v \in t} p(\texttt{subtree\_size}(v.\texttt{left}), \texttt{subtree\_size}(v.\texttt{right}))$
- ***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:* $\mathbb{P}[t] = \prod_{v \in t} p(\text{type}(v))$ $\qquad$ $\text{type}(v) \in \{$  $\}$
- 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:* $\mathbb{P}[t] = \prod_{v \in t} p(\mathrm{type}(v))$ $\mathrm{type}(v) \in \{$  $\}$
- *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

  $$\rightsquigarrow\ \mathbb{P}[t] = \prod_{v \in t} p(\mathtt{subtree\_size}(v.\mathtt{left}), \mathtt{subtree\_size}(v.\mathtt{right}))$$
- ***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.

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

Vector graphics from Pressfoto, brgfx, macrovector and Jannoon028 on freepik.com
Other photos from `www.pixabay.com`.