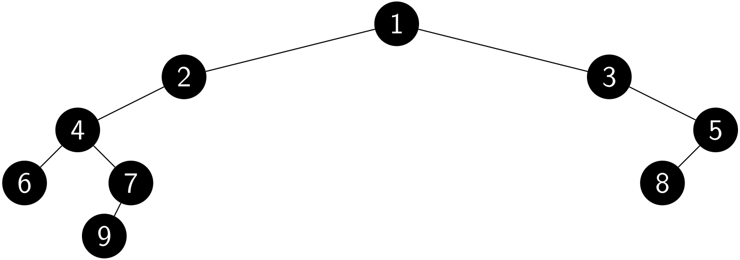
Enumeration and Succinct Encoding of AVL Trees

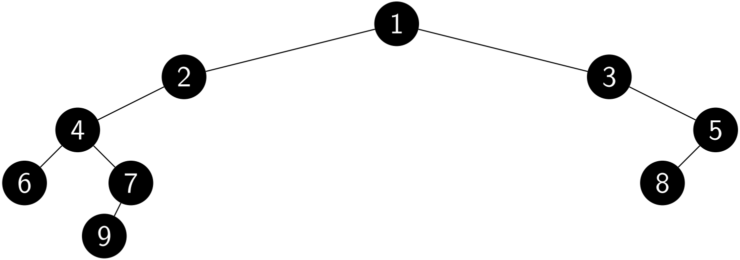
J. Chizewer, S. Melczer, J.I. Munro, A. Pun

June 17, 2024

Motivation - Storing Binary Trees



Storing Binary Trees



Naive Approach

Nodes	1	2	3	4	5	6	7	8	9
Pointers	2, 3	4, ∅	∅, 5	6, 7	8, ∅	∅, ∅	9, ∅	∅, ∅	∅, ∅

Storing Binary Search Trees

Naive approach storage requirements for n nodes

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

What is the minimum storage requirement?

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

What is the minimum storage requirement?

Let's enumerate!

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

What is the minimum storage requirement?

Let's enumerate!

$$B(z) = \frac{1 - \sqrt{1 - 4z}}{2z}$$

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

What is the minimum storage requirement?

Let's enumerate!

$$B(z) = \frac{1 - \sqrt{1 - 4z}}{2z}$$

Use analytic tools to get asymptotics of the coefficients:

Storing Binary Search Trees

Naive approach storage requirements for n nodes

- Each pointer requires $O(\log n)$ bits.
- There are $2n$ pointers.
- Total Requirement: $O(n \log n)$.

What is the minimum storage requirement?

Let's enumerate!

$$B(z) = \frac{1 - \sqrt{1-4z}}{2z}$$

Use analytic tools to get asymptotics of the coefficients:

$$[z^n]B(z) = \frac{4^n}{\sqrt{\pi n^3}}(1 + o(1))$$

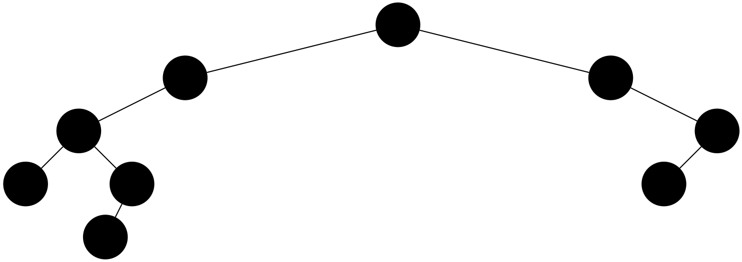
So $\log_2 \left(\frac{4^n}{\sqrt{\pi n^3}}(1 + o(1)) \right) = \boxed{2n + o(n)}$ bits are needed to give each tree a unique string

Succinct Encodings

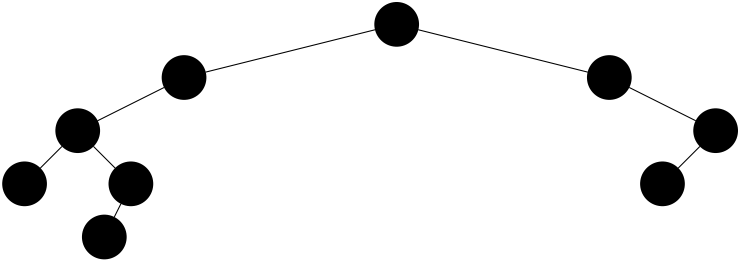
Definition

We call an encoding *succinct* if it uses the information-theoretic minimum number of bits

Succinct Encoding – Binary Trees

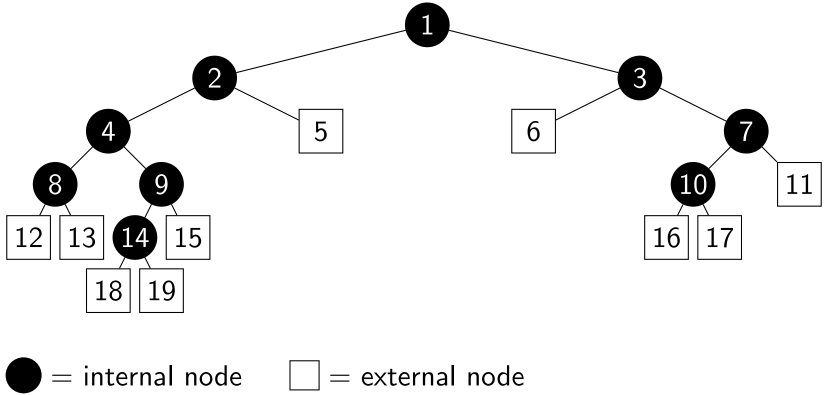


Succinct Encoding – Binary Trees

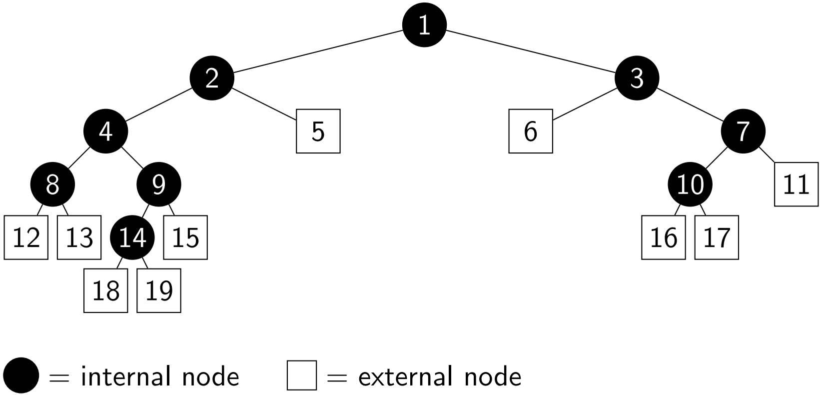


Step 1: Make sure every internal node has exactly two children by adding external nodes.

Succinct Encoding – Binary Trees

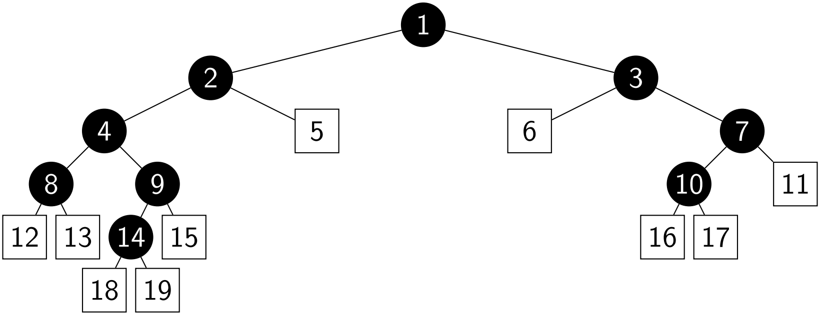


Succinct Encoding – Binary Trees



Step 2: Traverse the tree in level order, writing 1 for internal nodes and 0 for external nodes.

Succinct Encoding – Binary Trees



● = internal node   □ = external node

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

Succinct Encoding Analysis

How many bits did we use?

Succinct Encoding Analysis

How many bits did we use?

- We started with n nodes.

Succinct Encoding Analysis

How many bits did we use?

- We started with n nodes.
- We added $n + 1$ nodes.

Succinct Encoding Analysis

How many bits did we use?

- We started with n nodes.
- We added $n + 1$ nodes.
- We used 1 bit per node in the new tree.

Succinct Encoding Analysis

How many bits did we use?

- We started with n nodes.
- We added $n+1$ nodes.
- We used 1 bit per node in the new tree.

That is $2n+1$ bits which is succinct.

What if we care about more than just storage costs?

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.
- That means $O(\sqrt{n})$ comparisons on average to find a node.

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.
- That means $O(\sqrt{n})$ comparisons on average to find a node.
- Average depth in binary search tree is $O(\log n)$

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.
- That means $O(\sqrt{n})$ comparisons on average to find a node.
- Average depth in binary search tree is $O(\log n)$
  - But worst case $O(n)$.

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.
- That means $O(\sqrt{n})$ comparisons on average to find a node.
- Average depth in binary search tree is $O(\log n)$
  - But worst case $O(n)$.

Can we construct a tree with shorter depth?

What if we care about more than just storage costs?

In general, binary trees can be very inefficient.

- A random binary tree on n nodes has average depth $\sqrt{n}$.
- That means $O(\sqrt{n})$ comparisons on average to find a node.
- Average depth in binary search tree is $O(\log n)$
  - But worst case $O(n)$.

Can we construct a tree with shorter depth?

AVL Trees

AVL Trees

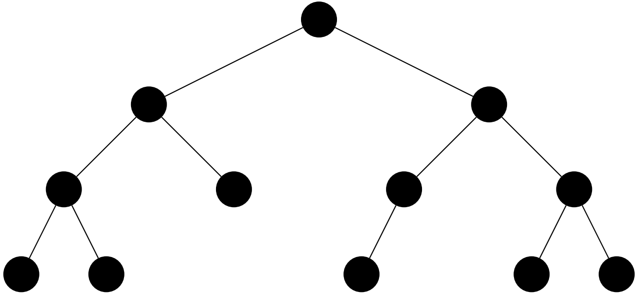
Definition

An *AVL tree* is a binary tree where every node satisfies the property that its left and right subtrees differ in height by at most one

AVL Trees

Definition

An *AVL tree* is a binary tree where every node satisfies the property that its left and right subtrees differ in height by at most one



Enumerating AVL Trees

Is our encoding for binary trees still succinct for AVL trees?

Enumerating AVL Trees

Is our encoding for binary trees still succinct for AVL trees?

Let's enumerate!

Enumerating AVL Trees

Start with a symbolic class:

Enumerating AVL Trees

Start with a symbolic class:

$$\mathcal{A}_0 = \bullet,\ \mathcal{A}_1 = \bullet \times \bullet,\ \mathcal{A}_{h+2} = \mathcal{A}_{h+1} \times (\mathcal{A}_{h+1} + 2\mathcal{A}_h)$$

$$\mathcal{A} = \bigcup_{h \geq 0} \mathcal{A}_h$$

Where $\mathcal{A}_h$ is the class of AVL trees with height h.

Enumerating AVL Trees

Start with a symbolic class:

$$\mathcal{A}_0 = \bullet,\ \mathcal{A}_1 = \bullet \times \bullet,\ \mathcal{A}_{h+2} = \mathcal{A}_{h+1} \times (\mathcal{A}_{h+1} + 2\mathcal{A}_h)$$

$$\mathcal{A} = \bigcup_{h \geq 0} \mathcal{A}_h$$

Where $\mathcal{A}_h$ is the class of AVL trees with height h.

Compute generating functions:

Enumerating AVL Trees

Start with a symbolic class:

$$\mathcal{A}_0 = \bullet,\ \mathcal{A}_1 = \bullet \times \bullet,\ \mathcal{A}_{h+2} = \mathcal{A}_{h+1} \times (\mathcal{A}_{h+1} + 2\mathcal{A}_h)$$

$$\mathcal{A} = \bigcup_{h \geq 0} \mathcal{A}_h$$

Where $\mathcal{A}_h$ is the class of AVL trees with height h.

Compute generating functions:

$$A_0(z) = z,\ A_1(z) = z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Where $A_h(z)$ is the GF for AVL trees with height h.

Enumerating AVL Trees

Start with a symbolic class:

$$\mathcal{A}_0 = \bullet,\ \mathcal{A}_1 = \bullet \times \bullet,\ \mathcal{A}_{h+2} = \mathcal{A}_{h+1} \times (\mathcal{A}_{h+1} + 2\mathcal{A}_h)$$

$$\mathcal{A} = \bigcup_{h \geq 0} \mathcal{A}_h$$

Where $\mathcal{A}_h$ is the class of AVL trees with height h.

Compute generating functions:

$$A_0(z) = z,\ A_1(z) = z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Where $A_h(z)$ is the GF for AVL trees with height h.
Note: enumeration is harder because we are recursing on height but enumerating by size.

Enumerating AVL Trees

$$A_0(z) = z,\ A_1(z) = 2z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Perform singularity analysis

When does $A(z)$ diverge?

Enumerating AVL Trees

$$A_0(z) = z,\ A_1(z) = 2z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Perform singularity analysis

When does $A(z)$ diverge? Find the fixed point for the recurrence:

$$C = C(C + 2C) \implies C = 1/3$$

Enumerating AVL Trees

$$A_0(z) = z,\ A_1(z) = 2z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Perform singularity analysis

When does $A(z)$ diverge? Find the fixed point for the recurrence:

$$C = C(C + 2C) \implies C = 1/3$$

$A(z)$ diverges when $A_h(z) \geq 1/3$ and $A_{h+1}(z) \geq 1/3$.

Enumerating AVL Trees

$$A_0(z) = z,\ A_1(z) = 2z^2,\ A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z))$$

$$A(z) = \sum_{h \geq 0} A_h(z)$$

Perform singularity analysis

When does $A(z)$ diverge? Find the fixed point for the recurrence:

$$C = C(C + 2C) \implies C = 1/3$$

$A(z)$ diverges when $A_h(z) \geq 1/3$ and $A_{h+1}(z) \geq 1/3$.

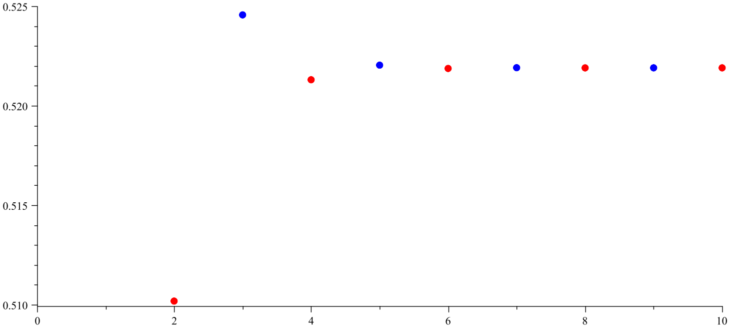
When does $A(z)$ converge?

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$



Values α_h for even h (red) and odd h (blue).

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Proof sketch.

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Proof sketch.

1. Show that α_h is strictly decreasing for odd h, strictly increasing for even h.

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Proof sketch.

1. Show that α_h is strictly decreasing for odd h, strictly increasing for even h.
2. Show for all h that $\alpha_{2h+1} > \alpha_{2h}$.

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Proof sketch.

1. Show that α_h is strictly decreasing for odd h, strictly increasing for even h.
2. Show for all h that $\alpha_{2h+1} > \alpha_{2h}$.
3. Show

$$\lim_{h\to\infty} \alpha_{2h+1} = \lim_{h\to\infty} \alpha_{2h} \qquad \square$$

Enumerating AVL Trees

Let α_h denote the unique positive solution to $A_h(z) = 1/3$

Lemma (C., Melczer, Munro, Pun, 2023)

The limit $\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$ *exists.*

Proof sketch.

1. Show that α_h is strictly decreasing for odd h, strictly increasing for even h.
2. Show for all h that $\alpha_{2h+1} > \alpha_{2h}$.
3. Show

$$\lim_{h\to\infty} \alpha_{2h+1} = \lim_{h\to\infty} \alpha_{2h}$$

□

Lemma (C., Melczer, Munro, Pun, 2023)

$A(z)$ *converges in the disk* $|z| < \alpha$

Enumerating AVL Trees

Theorem (C., Melczer, Munro, Pun, 2023)

There are $\alpha^{-n}\,\theta(n)$ *AVL trees on* n *nodes where* $\theta(n)$ *is sub-exponential.*

Enumerating AVL Trees

Theorem (C., Melczer, Munro, Pun, 2023)

There are $\alpha^{-n}\,\theta(n)$ AVL trees on n nodes where $\theta(n)$ is sub-exponential.

Proof sketch.

Enumerating AVL Trees

Theorem (C., Melczer, Munro, Pun, 2023)

There are $\alpha^{-n}\,\theta(n)$ AVL trees on n nodes where $\theta(n)$ is sub-exponential.

Proof sketch.

1. Apply the previous lemmas to conclude α is the singularity of smallest modulus

Enumerating AVL Trees

Theorem (C., Melczer, Munro, Pun, 2023)

There are $\alpha^{-n}\,\theta(n)$ AVL trees on n nodes where $\theta(n)$ is sub-exponential.

Proof sketch.

1. Apply the previous lemmas to conclude α is the singularity of smallest modulus
2. Conclude $1/\alpha$ is the exponential growth of a_n. $\square$

Generalizing the Main Theorem

Let $\mathcal{F} = \bigsqcup_{h=0}^{\infty} \mathcal{F}_h$ such that $F_h(z)$ are non-constant and

$$F_h(z) = f(F_{h-1}(z), F_{h-2}(z), \ldots, F_{h-c}(z)) \quad \text{for all} \quad h \geq c$$

where c is a positive integer and f has non-negative coefficients.

[1] f must satisfy an additional technical condition

Generalizing the Main Theorem

Let $\mathcal{F} = \bigsqcup_{h=0}^{\infty} \mathcal{F}_h$ such that $F_h(z)$ are non-constant and

$$F_h(z) = f(F_{h-1}(z), F_{h-2}(z), \ldots, F_{h-c}(z)) \quad \text{for all} \quad h \geq c$$

where c is a positive integer and f has non-negative coefficients.

Theorem (C., Melczer, Munro, Pun, 2023)

If f has positive fixed point B and β_h is the positive root of $B = F_h(z)$ then the limit $\beta = \lim_{h\to\infty} \beta_h$ exists[1] *and there are $\beta^{-n}\theta(n)$ objects in $\mathcal{F}$ of size n where $\theta(n)$ is sub-exponential.*

[1] f must satisfy an additional technical condition

Back to Encoding AVL Trees

Using the enumeration result, we know that

$$\log_2 \left(\alpha^{-n}\,\theta(n)\right) = (0.938\ldots)\,n + o(n)$$

bits are needed.

Back to Encoding AVL Trees

Using the enumeration result, we know that

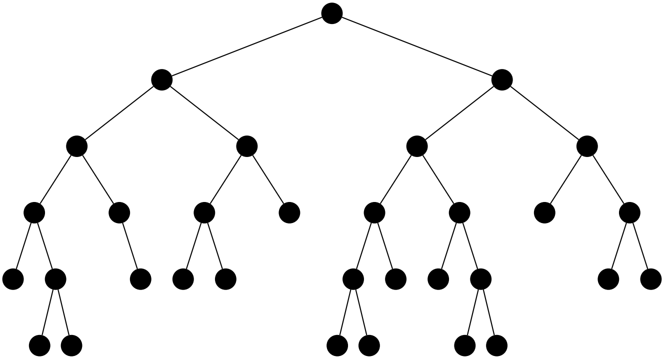
$$\log_2\left(\alpha^{-n}\,\theta(n)\right) = (0.938\ldots)\,n + o(n)$$

bits are needed.

Can we improve on the $2n+1$ bits used in the general binary tree encoding?

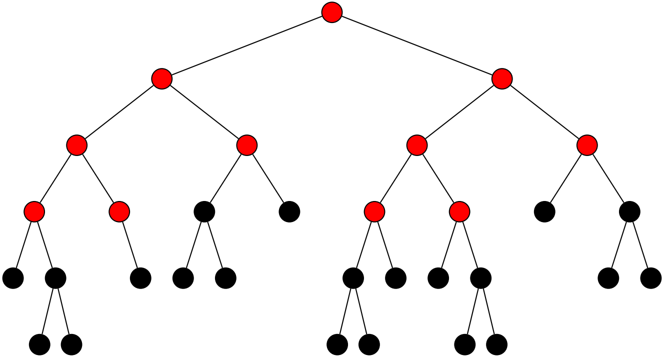
Succinct Encoding for AVL Trees

$n = 32$



Succinct Encoding for AVL Trees

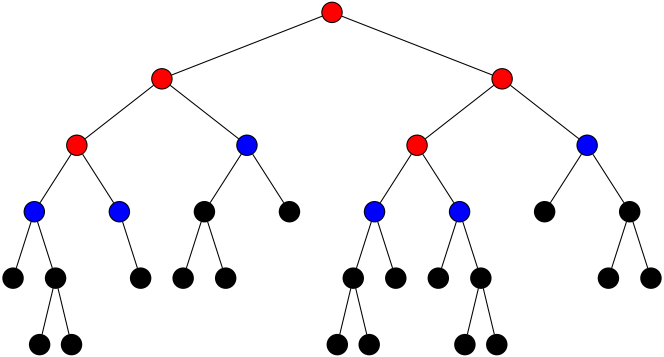
$n = 32$



Identify subtree τ (in red) of all nodes whose parents are the roots of large subtrees.

Succinct Encoding for AVL Trees

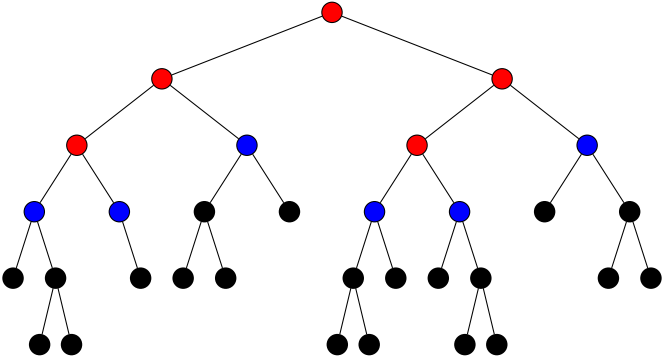
$n = 32$

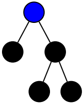


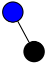
Name the shapes of subtrees rooted at leaves (in blue) of τ (in red/blue)

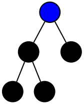
Succinct Encoding for AVL Trees

$n = 32$



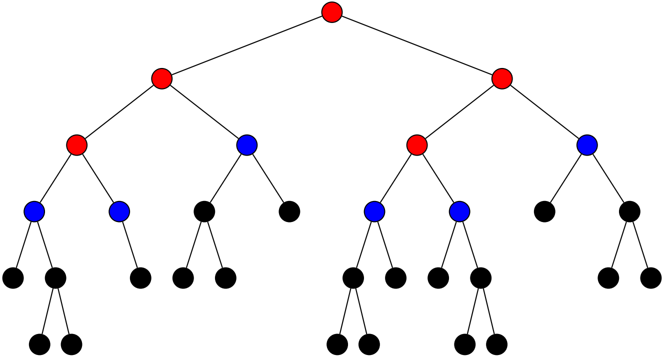
00 = 

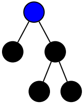
01 = 

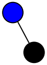
10 = 

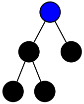
Succinct Encoding for AVL Trees

$n = 32$



$00 =$ 

$01 =$ 

$10 =$ 

Write τ, codeword mapping, and names of leaf trees in leaf order.

Succinct Encoding for AVL Trees

Algorithm Summary

Succinct Encoding for AVL Trees

Algorithm Summary

1. Identify subtree τ of all nodes whose parents are the roots of large subtrees.

Succinct Encoding for AVL Trees

Algorithm Summary

1. Identify subtree τ of all nodes whose parents are the roots of large subtrees.
2. The leaves of τ are the roots of small subtrees.

Succinct Encoding for AVL Trees

Algorithm Summary

1. Identify subtree τ of all nodes whose parents are the roots of large subtrees.
2. The leaves of τ are the roots of small subtrees.
3. Write a unique code word for each subtree in a lookup table.

Succinct Encoding for AVL Trees

Algorithm Summary

1. Identify subtree τ of all nodes whose parents are the roots of large subtrees.
2. The leaves of τ are the roots of small subtrees.
3. Write a unique code word for each subtree in a lookup table.
4. Store τ, the lookup table, and the codewords in order.

Succinct Encoding for AVL Trees

Why does it work?

Succinct Encoding for AVL Trees

Why does it work?

1. The lookup table has size $o(n)$ because there are few distinct shapes

Succinct Encoding for AVL Trees

Why does it work?

1. The lookup table has size $o(n)$ because there are few distinct shapes
2. The tree τ has size $o(n)$

Succinct Encoding for AVL Trees

Why does it work?

1. The lookup table has size $o(n)$ because there are few distinct shapes
2. The tree τ has size $o(n)$
3. The names of the codewords are asymptotically optimal in bits per node used.

Generalizing the Encoding

Definition

A class $\mathcal{T}$ of trees is *weakly tame*[2] if

1. All subtrees of a tree in the class are also in the class
2. The subtree τ defined previously satisfies $|\tau| = o(n)$
3. $\log \mathcal{T}_n = c \cdot n + o(n)$ for some constant c.

[2] Following the work of J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild.

Generalizing the Encoding

Definition

A class $\mathcal{T}$ of trees is *weakly tame*[2] if

1. All subtrees of a tree in the class are also in the class
2. The subtree τ defined previously satisfies $|\tau| = o(n)$
3. $\log \mathcal{T}_n = c \cdot n + o(n)$ for some constant c.

Theorem (C., Melczer, Munro, Pun, 2023)

There exists a static succinct encoding for any weakly tame class of trees.

[2] Following the work of J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild.

Conclusion

Theorem (C., Melczer, Munro, Pun, 2023)

If f has positive fixed point B and β_h is the positive root of $B = F_h(z)$ then the limit $\beta = \lim_{h\to\infty} \beta_h$ exists[3] and there are $\beta^{-n}\theta(n)$ objects in $\mathcal{F}$ of size n where $\theta(n)$ is sub-exponential.

[3] f must satisfy an additional technical condition

Conclusion

Theorem (C., Melczer, Munro, Pun, 2023)

If f has positive fixed point B and β_h is the positive root of $B = F_h(z)$ then the limit $\beta = \lim_{h\to\infty} \beta_h$ exists[3] *and there are $\beta^{-n}\,\theta(n)$ objects in $\mathcal{F}$ of size n where $\theta(n)$ is sub-exponential.*

Theorem (C., Melczer, Munro, Pun, 2023)

There exists a static succinct encoding for any weakly tame class of trees.

[3] f must satisfy an additional technical condition

Future Work

1. Construct dynamic succinct encodings

Future Work

1. Construct dynamic succinct encodings
2. Apply enumeration tools to other balanced data structures.

Thank you!