On Constructing 2-3 Trees

Ralf Hinze

Radboud University

December 2016

There are only 10 types of people in the world: those who understand binary, and those who don't.

In computer science, we are so accustomed to thinking about binary numbers, that we sometimes forget that other bases are possible.

Purely Functional Data Structures—Chris Okasaki

1 2-3 Trees

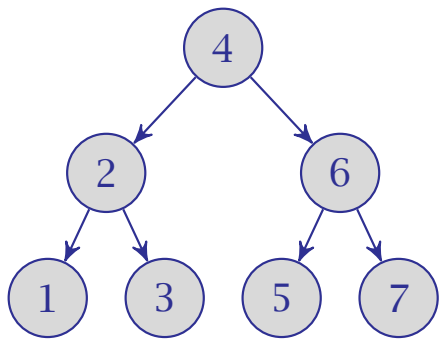
- one of my favourite balancing schemes
- 2-3 trees are an interesting blend of binary and ternary trees

```
data Tree23 a
  = L                                          — leaf
  | N2 (Tree23 a) a (Tree23 a)                 — 2-node
  | N3 (Tree23 a) a (Tree23 a) a (Tree23 a)    — 3-node
```

- *height condition:* all leaves must appear on the same level

1 2-3 Trees: Extremes

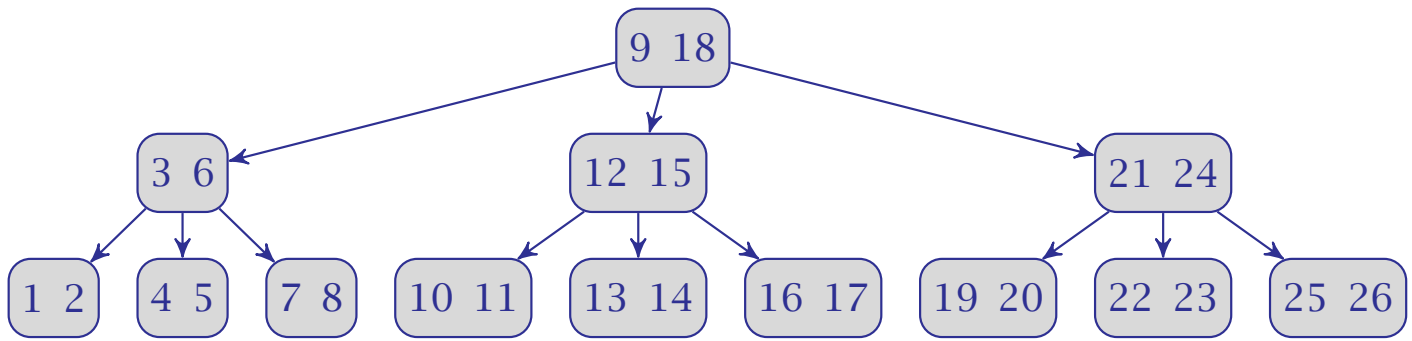
- full binary trees of size $2^h - 1$



$$7 = 2^3 - 1$$

1 2-3 Trees: Extremes—Continued

- full ternary trees of size $3^h - 1$



$$26 = 3^3 - 1$$

But is there a 2-3 tree for any given size?

1 Problem Specification

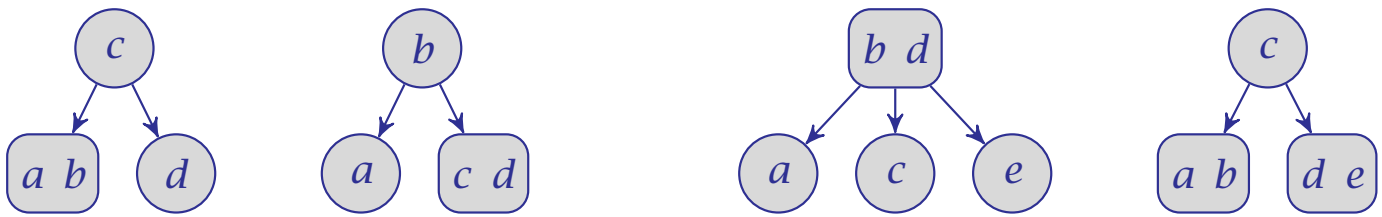
- given a sequence of elements we seek to build a 2-3 tree that contains the elements in symmetric order

$$inorder \cdot build = id$$

- the builder should run in linear time

1 Problem Specification—Continued

- the specification is ambiguous
- there are two trees of size 4 and two trees of size 5



- there are 727 trees of size 20
- which one to pick?

1 Fat and Slim 2-3 Trees

- *slim:* minimal number of 3-nodes for a given size
- *fat:* maximal number of 3-nodes for a given size

1 Overview

- top-down construction
- bottom-up construction: slim trees
- bottom-up construction: fat trees
- incremental construction: slim trees
- incremental construction: fat trees

Section 2

Top-down Construction

2 Initial Thoughts

- simple inductive approach: add one element after the other, starting with an empty tree: *foldr cons L*
- *too slow:* $\Theta(n \log n)$
- repeatedly traverses left spine of the 2-3 tree from the root to the leftmost leaf
- perhaps a divide-and-conquer scheme is more efficient?

2 Braun Trees

- but we'd like to avoid dealing with the tedious details of a binary sub-division scheme
- therefore we assume that the input is given by a *Braun tree*

$$\textbf{data}\ \mathit{Braun}\ a = \mathit{Empty} \mid \mathit{Node}\ (\mathit{Braun}\ a)\ a\ (\mathit{Braun}\ a)$$

- for any given node $\mathit{Node}\ l\ a\ r$, the left subtree l is either exactly the same size as the right subtree r, or one element smaller

$$\mathit{size}\ l \leqslant \mathit{size}\ r \leqslant \mathit{size}\ l + 1$$

- a functional idiom: a control structure is turned into a data structure, whose fold corresponds to the control structure

2 Top-down Construction

- *challenge:* ensure that the height condition is met
- we annotate each intermediate 2-3 tree with its height

$$
\begin{aligned}
&\textbf{type } \mathit{Height} = \mathit{Int}\\
&\mathit{create} :: \mathit{Braun}\ a \to (\mathit{Height}, \mathit{Tree23}\ a)\\
&\mathit{create}\ \mathit{Empty} = \mathit{empty}\\
&\mathit{create}\ (\mathit{Node}\ l\ a\ r) = \mathit{node}\ (\mathit{create}\ l)\ a\ (\mathit{create}\ r)\\
&\mathit{empty} = (0, L)\\
&\mathit{node}\ (h, t)\ a\ (k, u) \mid h == k = (h+1, N_2\ t\ a\ u)\\
&\mathit{node}\ (h, t)\ a\ (k, N_2\ u\ b\ v) \mid h+1 == k = (h+1, N_3\ t\ a\ u\ b\ v)
\end{aligned}
$$

- the smart constructor *node* creates a 2-node if both argument trees have the same height; otherwise, it makes the shorter tree the child of the taller one, creating a 3-node

2 Correctness

- the definition of *node* is deceptively simple
- it is not immediately clear that it works
- it seems to rely on three assumptions:
  - the left tree is at most as tall as the right tree
  - the height difference is at most one
  - in case of a height difference, the root of the right tree is necessarily a 2-node
- each of these assumptions requires justification

2 Correctness—continued

- let us express the height in terms of the input size
- *empty*: the height is 0
- *node*: the height of the resulting tree is only determined by the height of the *left* tree

$$\begin{aligned} \mathit{height}\ 0 &= 0 \\ \mathit{height}\ (2 \times m + 1) &= 1 + \mathit{height}\ m \\ \mathit{height}\ (2 \times m + 2) &= 1 + \mathit{height}\ m \end{aligned}$$

2 Correctness—continued

- let us express the height in terms of the input size
- *empty*: the height is 0
- *node*: the height of the resulting tree is only determined by the height of the *left* tree

$$\begin{aligned} \mathit{height}\ 0 &= 0 \\ \mathit{height}\ (2 \times m + 1) &= 1 + \mathit{height}\ m \\ \mathit{height}\ (2 \times m + 2) &= 1 + \mathit{height}\ m \end{aligned}$$

- the height h is given by the length of the binary representation of the input size n, *composed of the digits* 1 *and* 2

$$n = (d_0 \ldots d_{h-1})_2 = \sum_{i=0}^{h-1} d_i \times 2^i \qquad \text{with } d_i \in \{1, 2\}$$

2 The $\{1,2\}$-Binary Number System

- counting in the $\{1,2\}$-binary number system

  $()_2, (1)_2, (2)_2, (11)_2, (21)_2, (12)_2, (22)_2, (111)_2, (211)_2,$
  $(121)_2, (221)_2, (112)_2, (212)_2, (122)_2, (222)_2, (1111)_2, \ldots$

- binary increment

  $$()_2 + 1 = (1)_2$$
  $$(1\alpha)_2 + 1 = (2\alpha)_2$$
  $$(2\alpha)_2 + 1 = (1(\alpha + 1))_2$$

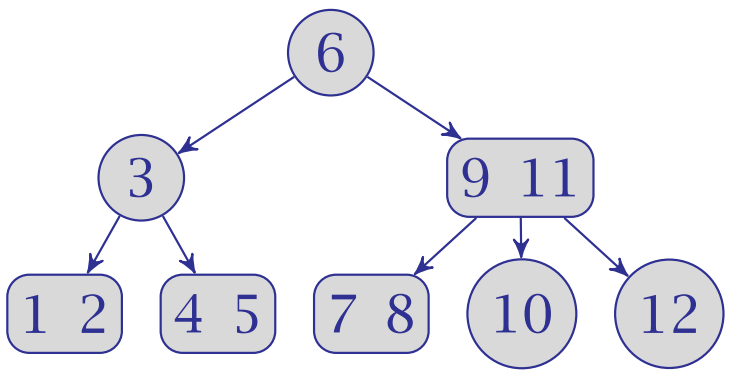
- cascading carry takes a sequence of 2s to a sequence of 1s
- a tree of size $(1^{\{h\}})_2 = 2^h - 1$ is a full *binary* tree

2 Top-down Construction

- the top-down construction of 2-3 trees combines unfolding and folding Braun trees

$$\begin{aligned} &\mathit{top\text{-}down} :: [a] \to \mathit{Tree23}\ a \\ &\mathit{top\text{-}down} = \mathit{snd} \cdot \mathit{create} \cdot \mathit{braun} \end{aligned}$$

- for the list $[1..12]$ of size $12 = (212)_2$ the builder creates



$$\begin{gathered} 12 \\ = \\ (5) + 1 + (6) \\ = \\ ((2) + 1 + (2)) + 1 + ((2) + 1 + (3)) \end{gathered}$$

- the running time of *top-down* is clearly linear

Section 3

Bottom-up Construction: Slim Trees

3 Initial Thoughts

- height condition of 2-3 trees dictates that all leaves appear on the same level, suggesting, in fact, a bottom-up approach
- we represent a level by an alternating sequence of 2-3 trees and elements

$$t_0 \mathbin{:\!/} a_1 \mathbin{:\!\backslash} t_1 \mathbin{:\!/} a_2 \mathbin{:\!\backslash} t_2 \cdots t_{n-1} \mathbin{:\!/} a_n \mathbin{:\!\backslash} t_n \,\square$$

  where all trees have the same height
- the builder repeatedly passes over the sequence combining adjacent 2-3 trees and elements until a single tree remains

3 Bottom-up Construction: Slim Trees

- bottom layer from the given list of elements

$$\begin{aligned} \mathit{bottom}\ [\,] &= L\,\square \\ \mathit{bottom}\ (a : x) &= L :\!\diagup\ a :\!\diagdown\ \mathit{bottom}\ x \end{aligned}$$

- a single pass

$$\begin{aligned} &\mathit{pass\text{-}slim}\ (t :\!\diagup\ a :\!\diagdown\ u\,\square) = N_2\ t\ a\ u\,\square \\ &\mathit{pass\text{-}slim}\ (t :\!\diagup\ a :\!\diagdown\ u :\!\diagup\ b :\!\diagdown\ v\,\square) = N_3\ t\ a\ u\ b\ v\,\square \\ &\mathit{pass\text{-}slim}\ (t :\!\diagup\ a :\!\diagdown\ u :\!\diagup\ b :\!\diagdown\ v :\!\diagup\ c :\!\diagdown\ q) \\ &\quad = N_2\ t\ a\ u :\!\diagup\ b :\!\diagdown\ \mathit{pass\text{-}slim}\ (v :\!\diagup\ c :\!\diagdown\ q) \end{aligned}$$

- we have to be careful that the sequence passed to the recursive call contains at least two trees
- each level contains at most one 3-node

3 Bottom-up Construction: Slim Trees

- repeated passes

$$\begin{aligned} &\mathit{loop\text{-}slim}\ (t\,\square) &&= t \\ &\mathit{loop\text{-}slim}\ (t :\!\diagup\ a :\!\diagdown\ q) &&= \mathit{loop\text{-}slim}\ (\mathit{pass\text{-}slim}\ (t :\!\diagup\ a :\!\diagdown\ q)) \end{aligned}$$

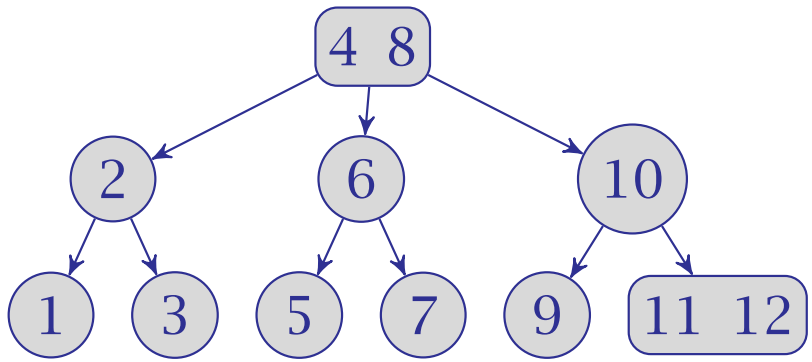
- bottom-up construction

$$\begin{aligned} &\mathit{bottom\text{-}up\text{-}slim} :: [a] \to \mathit{Tree23}\ a \\ &\mathit{bottom\text{-}up\text{-}slim} = \mathit{loop\text{-}slim} \cdot \mathit{bottom} \end{aligned}$$

- each pass roughly halves the length of the alternating sequence, resulting in a linear running time overall

3 Example

- an example tree of size 12



$$13 = 2 + 2 + 2 + 2 + 2 + 3$$
$$6 = 2 + 2 + 2$$
$$3$$

- *claim:* the tree is slim

Section 4

Bottom-up Construction: Fat Trees

4 Bottom-up Construction: Fat Trees

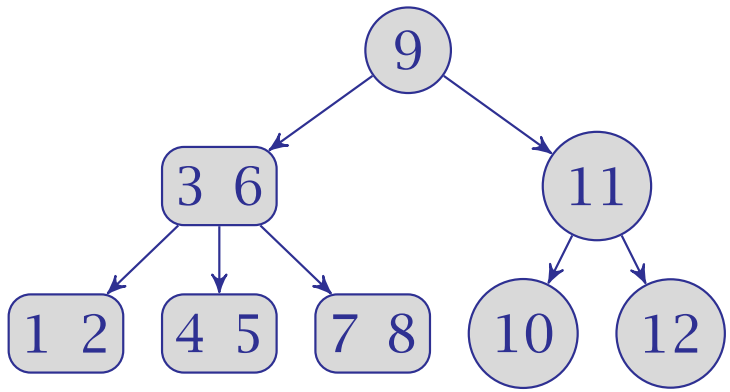
- bottom-up approach can be easily modified to build fat trees
- it suffices to adapt *pass-slim*

$$
\begin{aligned}
&\mathit{pass\text{-}fat}\ (t :\nearrow a :\nwarrow u\ \square) &&= N_2\ t\ a\ u\ \square \\
&\mathit{pass\text{-}fat}\ (t :\nearrow a :\nwarrow u :\nearrow b :\nwarrow v\ \square) &&= N_3\ t\ a\ u\ b\ v\ \square \\
&\mathit{pass\text{-}fat}\ (t :\nearrow a :\nwarrow u :\nearrow b :\nwarrow v :\nearrow c :\nwarrow w\ \square) &&= N_2\ t\ a\ u :\nearrow b :\nwarrow N_2\ v\ c\ w\ \square \\
&\mathit{pass\text{-}fat}\ (t :\nearrow a :\nwarrow u :\nearrow b :\nwarrow v :\nearrow c :\nwarrow w :\nearrow d :\nwarrow q) \\
&\quad = N_3\ t\ a\ u\ b\ v :\nearrow c :\nwarrow \mathit{pass\text{-}fat}\ (w :\nearrow d :\nwarrow q)
\end{aligned}
$$

- we are are careful to pass at least two trees to the recursive call
- each level contains at most two 2-nodes

4 Example

- an example tree of size 12



$$13 = 3 + 3 + 3 + 2 + 2$$
$$5 = 3 + 2$$
$$2$$

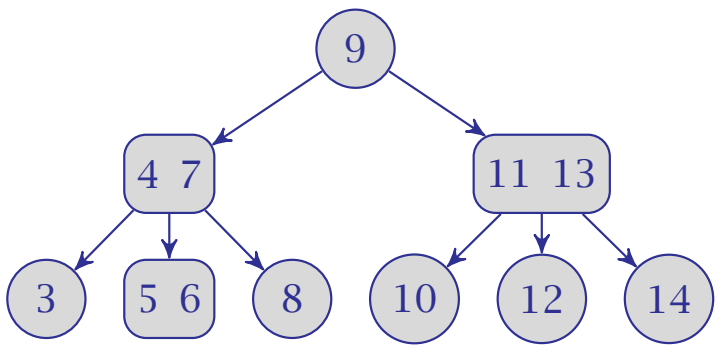
- *claim:* the tree is fat

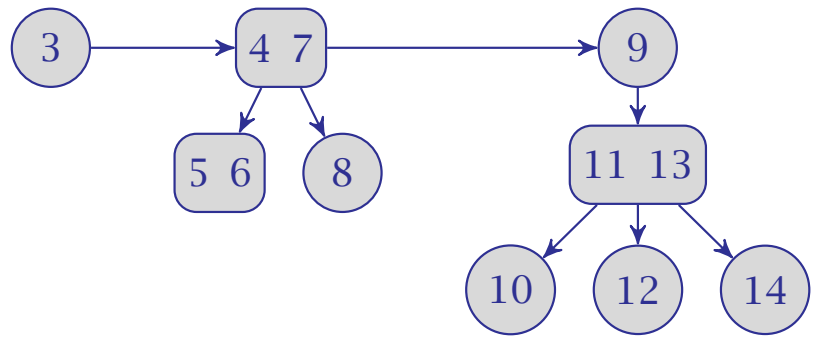
Section 5

Incremental Construction: Slim Trees

5 Initial Thoughts

- (why incremental? construction can be interleaved with other operations, for example, queries)
- we rejected the “simple inductive approach: add one element after the other, starting with an empty tree”
- we can, however, improve the performance if we reverse the left spine exposing the “hot spot”, the leftmost leaf
- meet the *left-spine view*





- the left spine amounts to a sequence of *pennants*

5 Left-spine View

- datatype for the left-spine view

```
data Spine23 a
  = Nil                                                  — empty
  | D1 a (Tree23 a)                    (Spine23 a)       — 1-pennant
  | D2 a (Tree23 a) a (Tree23 a)       (Spine23 a)       — 2-pennant
```

- similar to the type of 2-3 trees except that for each node the "downward pointer" to the leftmost child has been replaced by an "upward pointer" to the parent
- *height condition:* the height of the i-th pennant is i

5 Conversion

- to convert a tree to the left-spine view, we additionally pass in the "pointer" to the parent

$$
\begin{aligned}
&\mathit{to\text{-}spine} :: \mathit{Tree23}\ a \to \mathit{Spine23}\ a \to \mathit{Spine23}\ a \\
&\mathit{to\text{-}spine}\ L \qquad\qquad\quad\ z = z \\
&\mathit{to\text{-}spine}\ (N_2\ t\ a\ u) \quad z = \mathit{to\text{-}spine}\ t\ (D_1\ a\ u \qquad z) \\
&\mathit{to\text{-}spine}\ (N_3\ t\ a\ u\ b\ v)\ z = \mathit{to\text{-}spine}\ t\ (D_2\ a\ u\ b\ v\ z)
\end{aligned}
$$

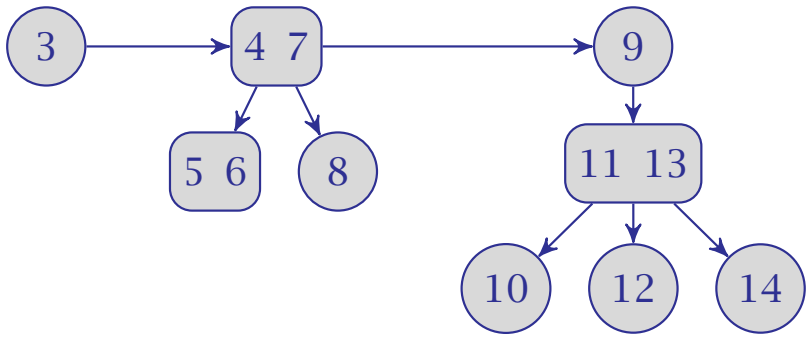
- conversely, to turn a spine into a tree we supply a "pointer" to the leftmost child

$$
\begin{aligned}
&\mathit{from\text{-}spine} :: \mathit{Tree23}\ a \to \mathit{Spine23}\ a \to \mathit{Tree23}\ a \\
&\mathit{from\text{-}spine}\ t\ \mathit{Nil} \qquad\qquad\quad = t \\
&\mathit{from\text{-}spine}\ t\ (D_1\ a\ u \qquad z) = \mathit{from\text{-}spine}\ (N_2\ t\ a\ u) \qquad z \\
&\mathit{from\text{-}spine}\ t\ (D_2\ a\ u\ b\ v\ z) = \mathit{from\text{-}spine}\ (N_3\ t\ a\ u\ b\ v)\ z
\end{aligned}
$$

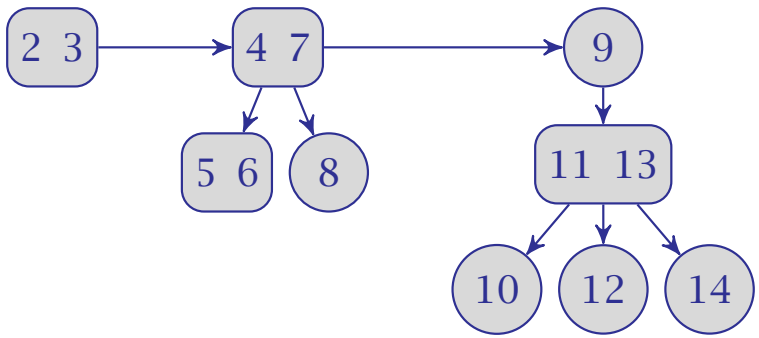
- both functions run in $\Theta(\log n)$ time

5 Adding an Element to the Front: Example

- consider adding first 2 and then 1 to the 12-element spine

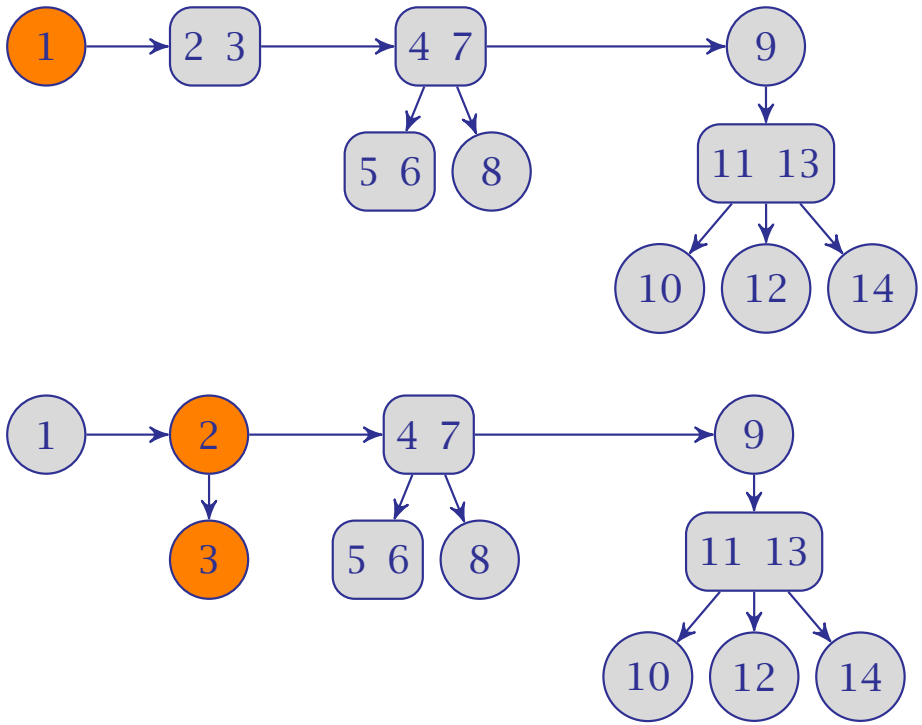


- adding 2 is easy: we turn the initial 2-node into a 3-node



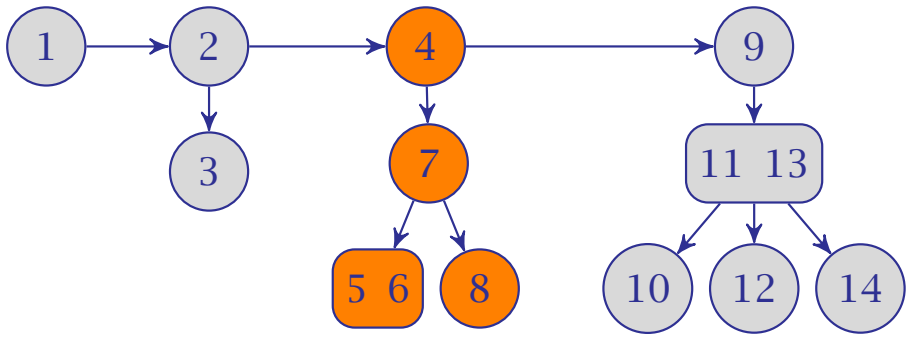
5 Adding an Element to the Front: Example

- adding 1 is harder as we cannot further grow the 3-node
- generalize task from adding an element to adding a pennant

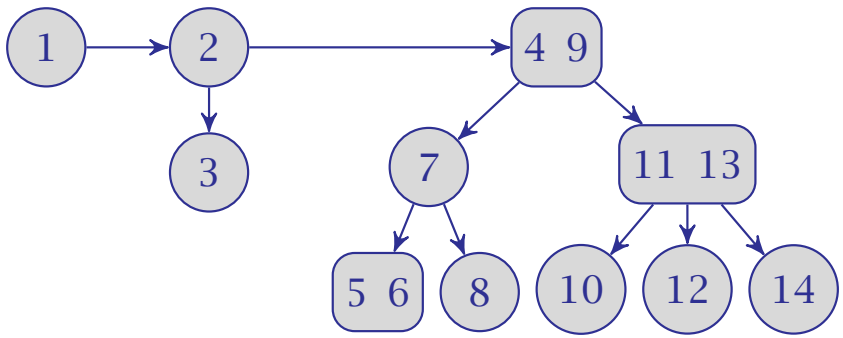


5 Adding an Element to the Front: Example

- continuing . . .



- the final 1-pennant can be grown into a 2-pennant



5 Incremental Construction: Slim Trees

- we first build a spine and then convert it to a tree

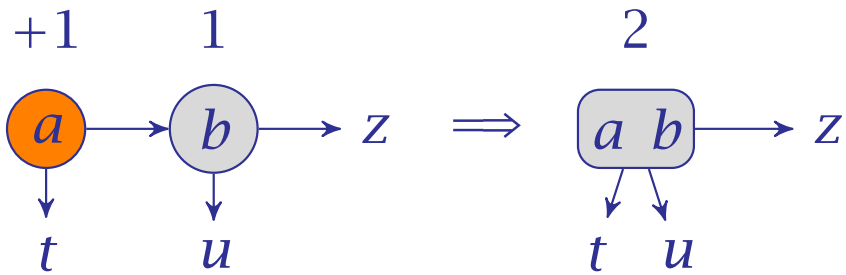
$$\begin{aligned}&\mathit{incremental\text{-}slim} :: [a] \to \mathit{Tree23}\ a\\&\mathit{incremental\text{-}slim} = \mathit{from\text{-}spine}\ L \cdot \mathit{foldr}\ \mathit{cons}\ \mathit{Nil}\end{aligned}$$

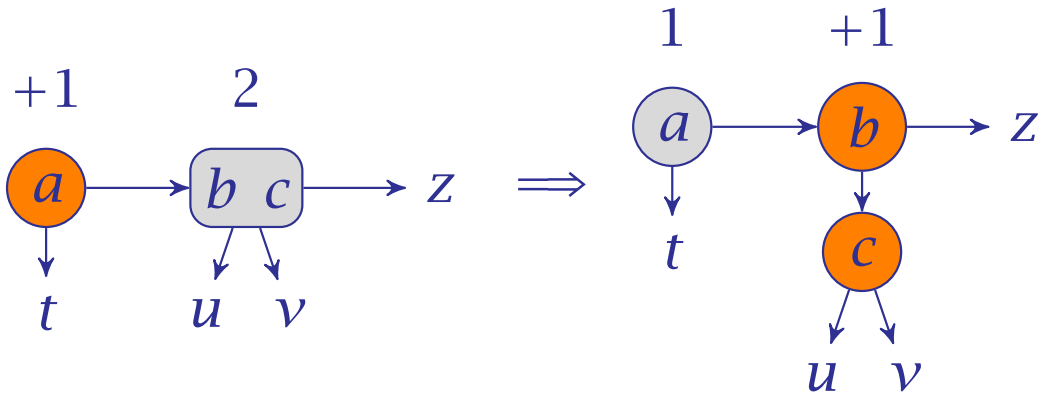
- adding an element to the front (c_1 like *c*arry)

$$\begin{aligned}&\mathit{cons} :: a \to \mathit{Spine23}\ a \to \mathit{Spine23}\ a\\&\mathit{cons}\ a\ z = c_1\ a\ L\ z\\&c_1 :: a \to \mathit{Tree23}\ a \to \mathit{Spine23}\ a \to \mathit{Spine23}\ a\\&c_1\ a\ t\ \mathit{Nil} = D_1\ a\ t\ \mathit{Nil}\\&c_1\ a\ t\ (D_1\ b\ u\ z) = D_2\ a\ t\ b\ u\ z\\&c_1\ a\ t\ (D_2\ b\ u\ c\ v\ z) = D_1\ a\ t\ (c_1\ b\ (N_2\ u\ c\ v)\ z) \quad \text{— new 2-node}\end{aligned}$$

5 Analysis

- the carry function c_1 only creates 2-nodes below the spine





- if we grow a spine using calls to *cons* only, then the trees hanging off the spine are *full binary trees*

5 Analysis—Continued

- a 1-pennant $D_1\ a\ t$ has size 1×2^h
- a 2-pennant $D_2\ a\ t\ b\ u$ has size 2×2^h
- thus, the shape of the 2-3 spine is determined by the binary representation of its size

5 Analysis—Continued

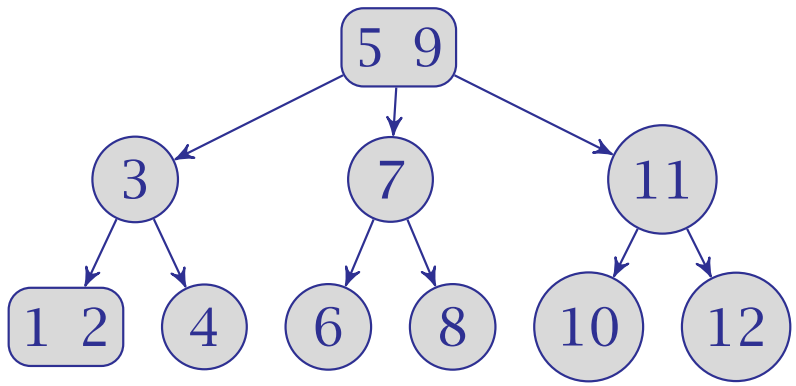
- a 1-pennant $D_1\ a\ t$ has size 1×2^h
- a 2-pennant $D_2\ a\ t\ b\ u$ has size 2×2^h
- thus, the shape of the 2-3 spine is determined by the binary representation of its size
- the $\{1, 2\}$-binary number system makes a second appearance:
  - a 1-pennant $D_1\ a\ t$ corresponds to the digit 1
  - a 2-pennant $D_2\ a\ t\ b\ u$ corresponds to the digit 2
  - consing an element corresponds to incrementing a binary number
- pennants witness *weighted digits*
- since the binary increment runs in amortized constant time, the overall running time is linear

5 Analysis—Continued

- a 1-pennant $D_1\ a\ t$ has size 1×2^h
- a 2-pennant $D_2\ a\ t\ b\ u$ has size 2×2^h
- thus, the shape of the 2-3 spine is determined by the binary representation of its size
- the $\{1, 2\}$-binary number system makes a second appearance:
  - a 1-pennant $D_1\ a\ t$ corresponds to the digit 1
  - a 2-pennant $D_2\ a\ t\ b\ u$ corresponds to the digit 2
  - consing an element corresponds to incrementing a binary number
- pennants witness *weighted digits*
- since the binary increment runs in amortized constant time, the overall running time is linear
- the generated trees are *slim* as 3-nodes only occur *on* the left spine, and each natural number has a unique representation in the $\{1, 2\}$-binary number system

5 Example

- an example tree of size 12



$$12 = (2\ 1\ 2)_2$$

Section 6

Incremental Construction: Fat Trees

6 Initial Thoughts

- can we use the same approach to create fat 2-3 trees?
- easy!

6 Initial Thoughts

- can we use the same approach to create fat 2-3 trees?
- easy!
- we simply replace the binary number system by a ternary one

6 2-3-4 Spines

- we add the digit 3

$$
\begin{aligned}
&\textbf{data}\ \mathit{Spine234}\ a \\
&\quad = \mathit{Nil} \\
&\quad \mid D_1\ a\ (\mathit{Tree23}\ a) && (\mathit{Spine234}\ a) \\
&\quad \mid D_2\ a\ (\mathit{Tree23}\ a)\ a\ (\mathit{Tree23}\ a) && (\mathit{Spine234}\ a) \\
&\quad \mid D_3\ a\ (\mathit{Tree23}\ a)\ a\ (\mathit{Tree23}\ a)\ a\ (\mathit{Tree23}\ a) && (\mathit{Spine234}\ a)
\end{aligned}
$$

- and adapt the carry function

$$
\begin{aligned}
&c_1 :: a \to \mathit{Tree23}\ a \to \mathit{Spine234}\ a \to \mathit{Spine234}\ a \\
&c_1\ a\ t\ \mathit{Nil} && = D_1\ a\ t\ \mathit{Nil} \\
&c_1\ a\ t\ (D_1\ b\ u\ \ \ \ z) && = D_2\ a\ t\ b\ u\ z \\
&c_1\ a\ t\ (D_2\ b\ u\ c\ v\ \ z) && = D_3\ a\ t\ b\ u\ c\ v\ z \\
&c_1\ a\ t\ (D_3\ b\ u\ c\ v\ d\ w\ z) \\
&\quad = D_1\ a\ t\ (c_1\ b\ (N_3\ u\ c\ v\ d\ w)\ z) && \text{— new 3-node}
\end{aligned}
$$

6 Afterthoughts

- that was easy indeed!

6 Afterthoughts

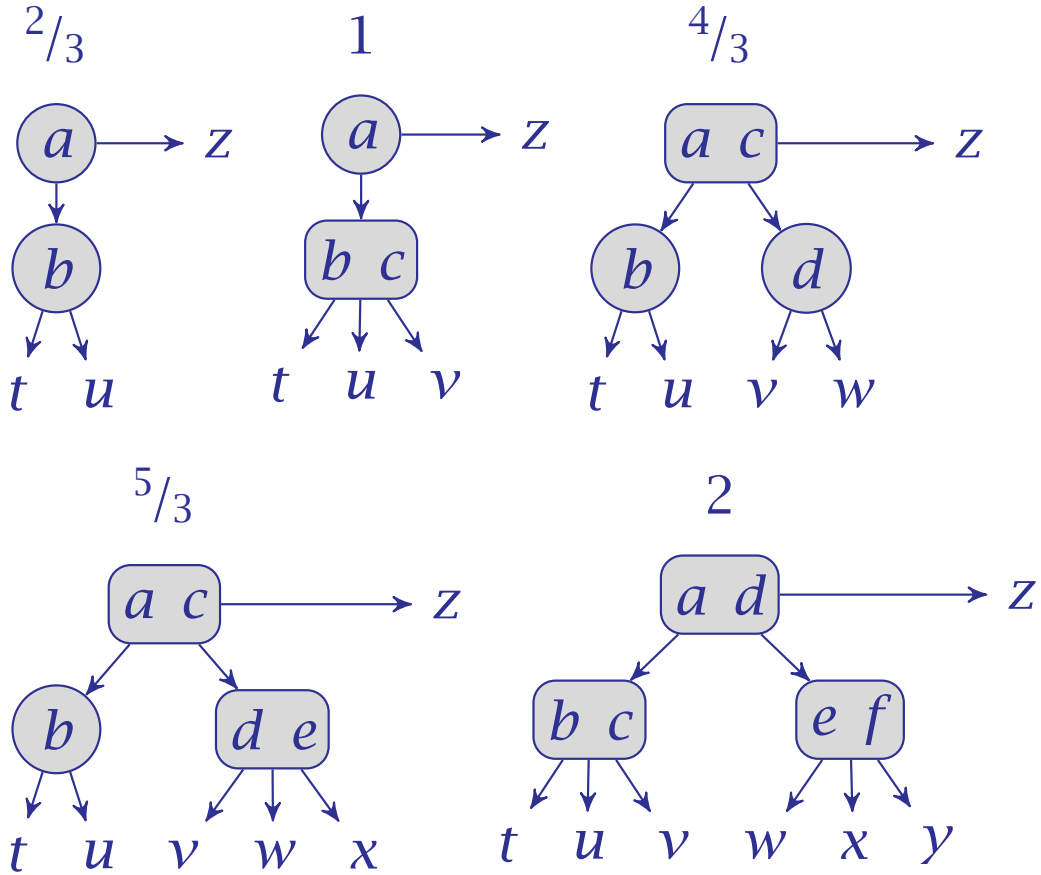
- that was easy indeed!
- except it doesn't work: we cannot convert the 2-3-4 spine to a 2-3 tree as 2-3 trees do not feature 4-nodes
- how to proceed?

6 Afterthoughts

- that was easy indeed!
- except it doesn't work: we cannot convert the 2-3-4 spine to a 2-3 tree as 2-3 trees do not feature 4-nodes
- how to proceed?
- perhaps, we can draw inspiration from the bottom-up approach?
- fat trees contain at most *two* 2-nodes per level, which suggests to consider pennants whose children are 2-nodes

6 Fractional Digits

- meet *fractional digits*



- the subtrees t, u etc are full ternary trees of height $h-1$
- size of $^2/_3$-pennant is $2 \times (3^{h-1} - 1) + 2 = {}^2/_3 \times 3^h$

6 2-3 Spines Revisited

- we revise the datatype of 2-3 spines accordingly

$$
\begin{aligned}
&\textbf{data}\ \mathit{Spine23}\ a \\
&\quad = \mathit{Nil} \\
&\quad \mid D_{2/3}\ a\ (\mathit{Tree23}\ a) \qquad\qquad\qquad\quad (\mathit{Spine23}\ a) \\
&\quad \mid D_{1}\ \ \ a\ (\mathit{Tree23}\ a) \qquad\qquad\qquad\quad (\mathit{Spine23}\ a) \\
&\quad \mid D_{5/3}\ a\ (\mathit{Tree23}\ a)\ a\ (\mathit{Tree23}\ a)\ (\mathit{Spine23}\ a) \\
&\quad \mid D_{2}\ \ \ a\ (\mathit{Tree23}\ a)\ a\ (\mathit{Tree23}\ a)\ (\mathit{Spine23}\ a)
\end{aligned}
$$

- NB $^{4}/_{3}$-pennants are not needed

6 Conversion

- *goal:* eliminate the digit 3
- convert a ternary number whose digits are drawn from $\{1, 2, 3\}$ to a ternary number with digits in $\{{}^2/_3, 1, {}^5/_3, 2\}$

$$
\begin{aligned}
&\mathit{convert} :: \mathit{Spine234}\ a \to \mathit{Spine23}\ a \\
&\mathit{convert}\ \mathit{Nil} = \mathit{Nil} \\
&\mathit{convert}\ (D_1\ a\ u\ z) = D_1\ a\ u\ (\mathit{convert}\ z) \\
&\mathit{convert}\ (D_2\ a\ u\ b\ v\ z) = D_2\ a\ u\ b\ v\ (\mathit{convert}\ z) \\
&\mathit{convert}\ (D_3\ a\ u\ b\ v\ c\ w\ z) = D_1\ a\ u\ (c_{2/3}\ b\ (N_2\ v\ c\ w)\ z)
\end{aligned}
$$

- we basically apply the identity $3 = 1 + {}^2/_3 \times 3$

$$
\begin{aligned}
&c_{2/3} :: a \to \mathit{Tree23}\ a \to \mathit{Spine234}\ a \to \mathit{Spine23}\ a \\
&c_{2/3}\ a\ t\ \mathit{Nil} = D_{2/3}\ a\ t\ \mathit{Nil} \\
&c_{2/3}\ a\ t\ (D_1\ b\ u\ z) = D_{5/3}\ a\ t\ b\ u\ (\mathit{convert}\ z) \\
&c_{2/3}\ a\ t\ (D_2\ b\ u\ c\ v\ z) = D_{2/3}\ a\ t\ (c_{2/3}\ b\ (N_2\ u\ c\ v)\ z) \\
&c_{2/3}\ a\ t\ (D_3\ b\ u\ c\ v\ d\ w\ z) = D_{5/3}\ a\ t\ b\ u\ (c_{2/3}\ c\ (N_2\ v\ d\ w)\ z)
\end{aligned}
$$

6 Incremental Construction: Fat Trees

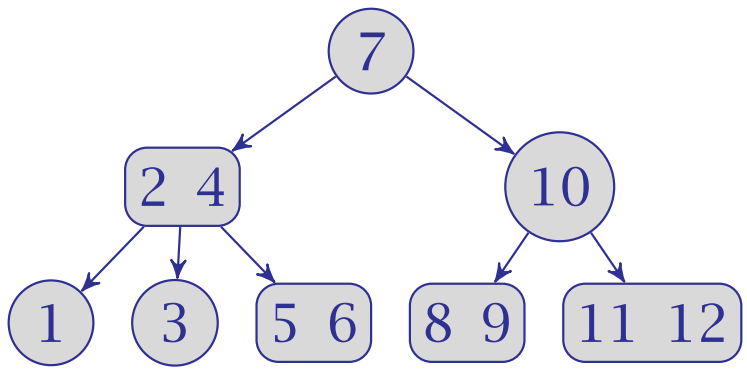
- we first build a 2-3-4 spine, then convert it to a 2-3 spine, which is then converted to a 2-3 tree

  $$\begin{aligned}&\mathit{incremental\text{-}fat} :: [a] \to \mathit{Tree23}\ a\\&\mathit{incremental\text{-}fat} = \mathit{from\text{-}spine}\ L \cdot \mathit{convert} \cdot \mathit{foldr}\ (\backslash a\ z \to c_1\ a\ L\ z)\ \mathit{Nil}\end{aligned}$$

- *from-spine* maps the digits $D_{2/3}$ and D_1 to 2-nodes, and the digits $D_{5/3}$ and D_2 to 3-nodes

6 Example

- an example tree of size 12



$$12 = (3\ 3)_3 = (1\ {}^{5}/_{3}\ {}^{2}/_{3})_3$$

6 Afterthoughts

- the $\{{}^2/_3, 1, {}^5/_3, 2\}$-number system can be used to show that the generated trees are fat
- argument more complicated as number system is redundant
- for example, $(1\,2)_3 = 7 = (2\,{}^5/_3)_3$; in general

$$(\alpha\, a\, b\, \beta)_3 = (\alpha\, (a-1)\, (b+\tfrac{1}{3})\, \beta)_3$$

- corresponding tree rewrites do not change the number of 2-nodes, *unless* the digit ${}^4/_3$ is involved

6 Afterthoughts

- the $\{{}^{2}/_{3}, 1, {}^{5}/_{3}, 2\}$-number system can be used to show that the generated trees are fat
- argument more complicated as number system is redundant
- for example, $(1\,2)_3 = 7 = (2\,{}^{5}/_{3})_3$; in general

$$(\alpha\, a\, b\, \beta)_3 = (\alpha\, (a-1)\, (b+\tfrac{1}{3})\, \beta)_3$$

- corresponding tree rewrites do not change the number of 2-nodes, *unless* the digit ${}^{4}/_{3}$ is involved
- first building a 2-3-4 spine and then turning it into a 2-3 spine is a roundabout way
- the intermediate step can be eliminated ...

7 Conclusion

- the bottom-up method produces the same trees as the incremental approach based on the *right-spine* view
- the top-down and the incremental approach are related by Horner's method for evaluating polynomials

$$1 \times 2^0 + 2 \times 2^1 + 2 \times 2^2 = 11 = 1 + 2 \times (2 + 2 \times (2 + 2 \times 0))$$

- formula on the left dictates the shape of the left-spine view; formula on the right guides the top-down construction

7 Conclusion

- the bottom-up method produces the same trees as the incremental approach based on the *right-spine* view
- the top-down and the incremental approach are related by Horner's method for evaluating polynomials

$$1 \times 2^0 + 2 \times 2^1 + 2 \times 2^2 = 11 = 1 + 2 \times (2 + 2 \times (2 + 2 \times 0))$$

- formula on the left dictates the shape of the left-spine view; formula on the right guides the top-down construction
- *challenge:* top-down construction of slim and fat 2-3 trees