Programación Funcional Avanzada

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Finger Trees

Finger Trees

Estructura de datos de propósito general
Puede ser usada como:

- secuencia (partir, concatenar, acceso a los extremos)
- cola de prioridad (encontrar el mínimo)
- árbol de búsqueda (encontrar un elemento)

Las estructuras de datos especializadas suelen ser un poco más eficientes, pero los finger trees son competitivos

Disponible en *Data.Sequence*

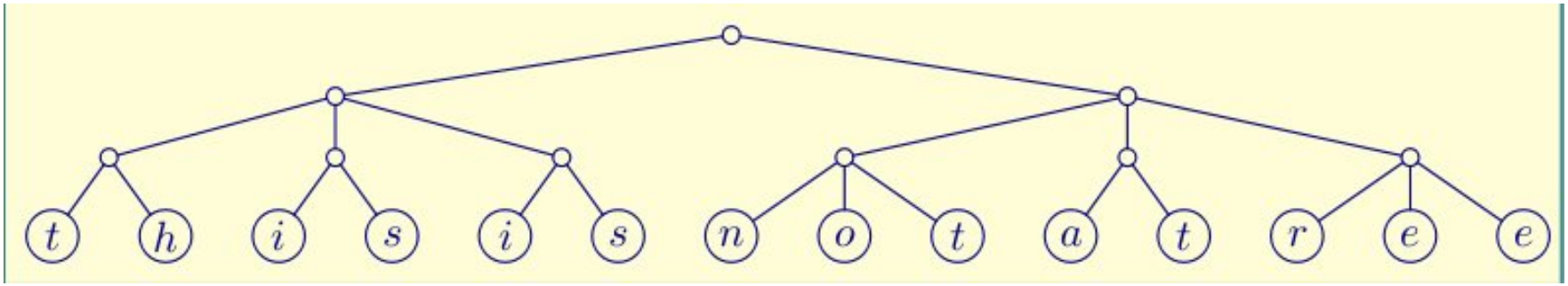
Finger Trees - Complejidad

	regular lists	*Seq* worst-case	*Seq* amortized
head	$O(1)$	$O(1)$	$O(1)$
cons	$O(1)$	$O(\log n)$	$O(1)$
tail	$O(1)$	$O(\log n)$	$O(1)$
last	$O(n)$	$O(1)$	$O(1)$
snoc	$O(n)$	$O(\log n)$	$O(1)$
init	$O(n)$	$O(\log n)$	$O(1)$
$(+\!\!+)$	$O(n)$	$O(\log n)$	$O(\log n)$
length	$O(n)$	$O(1)$	$O(1)$
$(!!)$	$O(n)$	$O(\log n)$	$O(\log n)$

2-3-Trees

Árboles balanceados

- valores se encuentran sólo en las hojas
- cada nodo tiene dos o tres hijos



data *Tree a* $=$ *Zero a* $\mid$ *Succ* (*Tree* (*Node a*))
data *Node a* $=$ *Node2 a a* $\mid$ *Node3 a a a*

Tipo de datos **anidado**: tiene recursión no regular

2-3-Trees (2)

$$\textbf{data}\ \mathit{Tree}\ a\ = \mathit{Zero}\ a \mid \mathit{Succ}\ (\mathit{Tree}\ (\mathit{Node}\ a))$$
$$\textbf{data}\ \mathit{Node}\ a = \mathit{Node2}\ a\ a \mid \mathit{Node3}\ a\ a\ a$$

Los constructores *Zero* y *Succ* codifican el número de niveles del árbol

Sólo se pueden construir árboles balanceados

$\mathit{Zero}\ 0$ -- 1 nivel
$\mathit{Succ}\ (\mathit{Zero}\ (\mathit{Node2}\ 0\ 1))$ -- 2 niveles
$\mathit{Succ}\ (\mathit{Zero}\ (\mathit{Node3}\ 0\ 1\ 2))$ -- 2 niveles
$\mathit{Succ}\ (\mathit{Succ}\ (\mathit{Zero}\ (\mathit{Node2}\ (\mathit{Node2}\ 0\ 1)\ (\mathit{Node2}\ 2\ 3))))$ -- 3 niveles

En general $\mathit{Succ}^n\ (\mathit{Zero}\ t)$
fuerza que t sea un árbol de tipo $\mathit{Node}^n\ a$

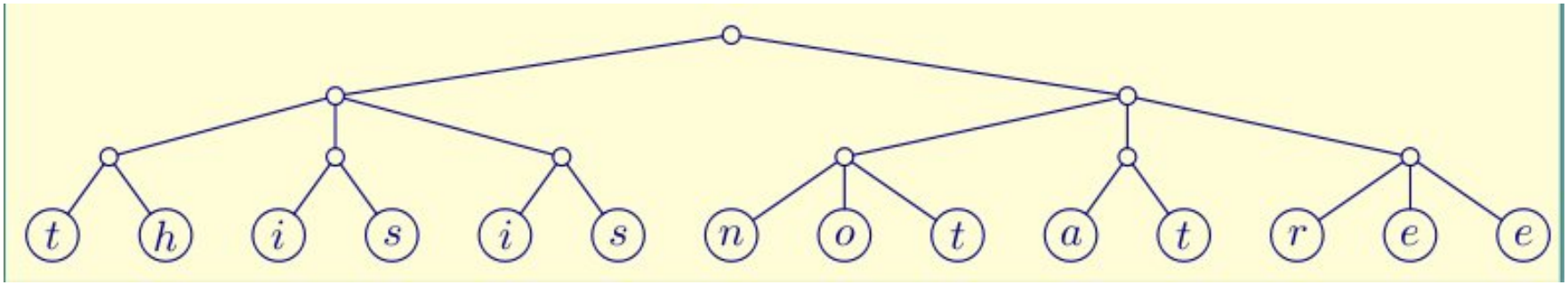
Finger Trees

2-3-Trees dan acceso logarítmico a todos los elementos

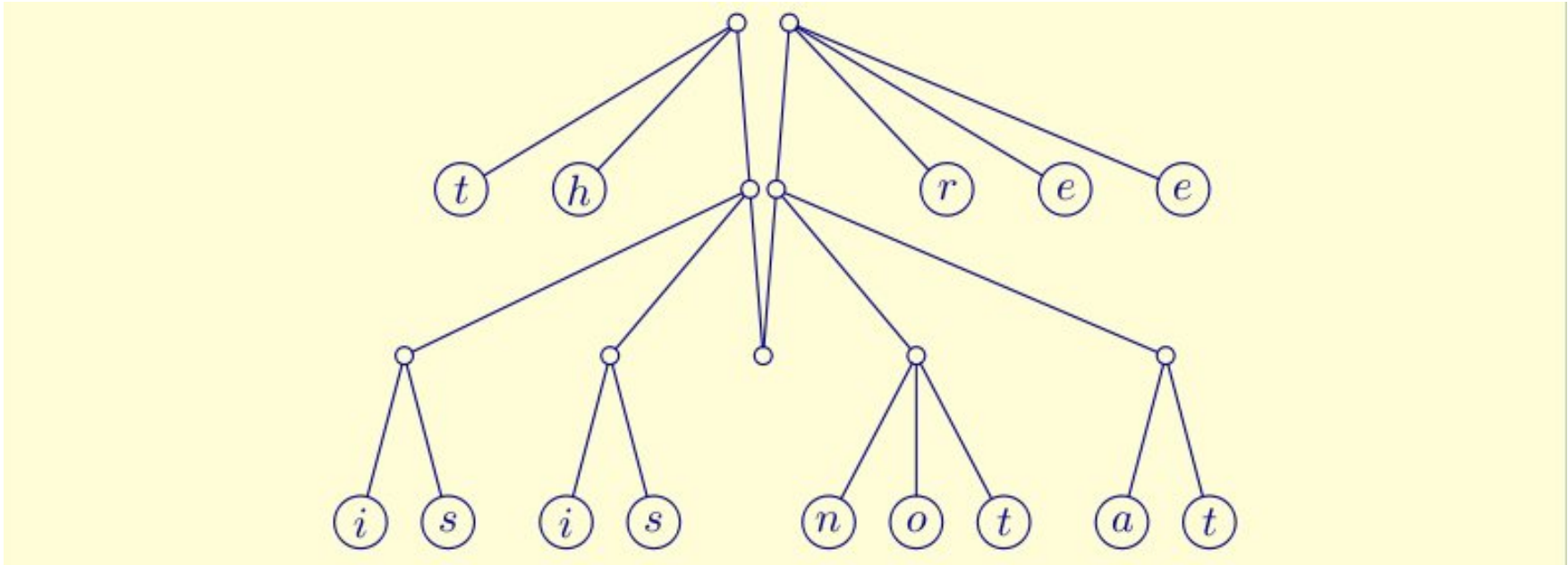
Para operaciones de secuencia también queremos acceso en tiempo constante a los dos extremos

Finger Trees son una reorganización de los 2-3-Trees

Finger Trees (2)

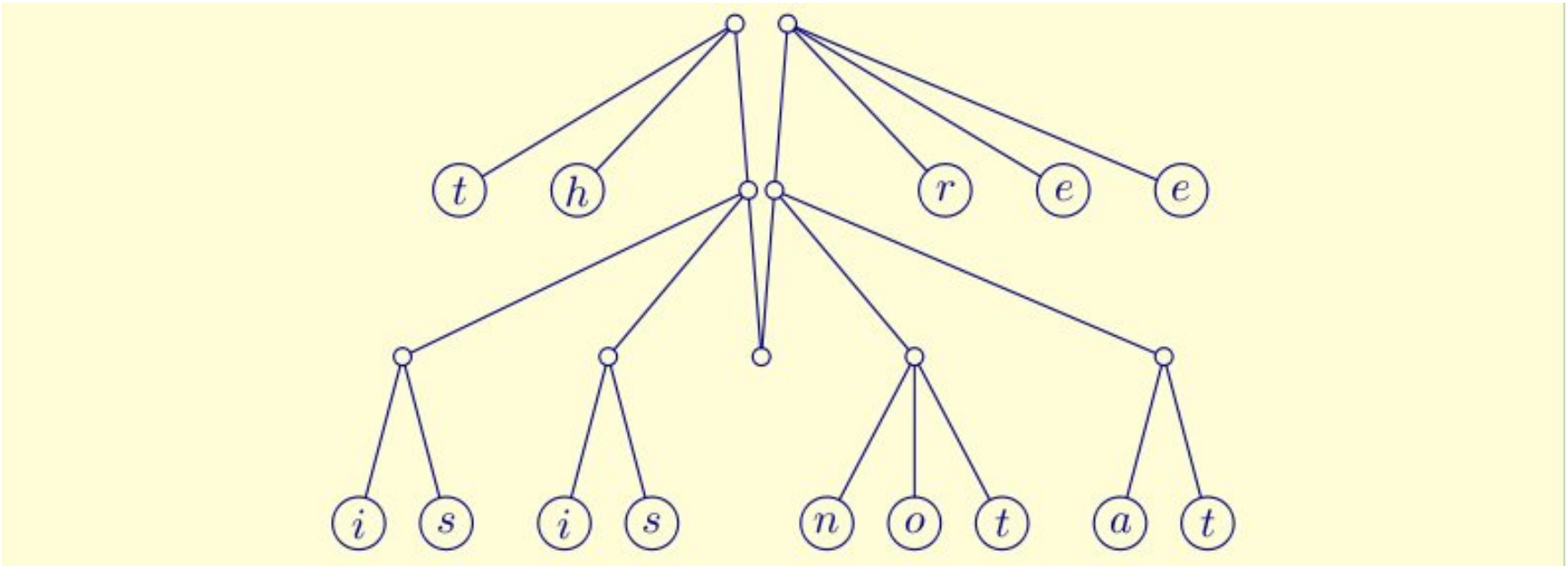


Tomar los nodos de los extremos y levantarlos juntos



Cada par de nodos en la columna central se junta en un solo nodo (llamado *Deep*)

Tipo de datos FingerTree



$$\textbf{data}\ \mathit{FingerTree}\ a = \mathit{Empty} \mid \mathit{Single}\ a \mid \mathit{Deep}\ (\mathit{Digit}\ a)\ (\mathit{FingerTree}\ (\mathit{Node}\ a))\ (\mathit{Digit}\ a)$$

Se relaja la condición 2-3 en *Digit*, permitiendo de 1 a 4 elementos

$$\textbf{type}\ \mathit{Digit}\ a = [a]$$ -- representacion simplificada

Agregar un elemento

Agregar un elemento en el extremo izquierdo:

$$
\begin{aligned}
&\textbf{infixr } 5 \lhd \\
&(\lhd) :: a \to \mathit{FingerTree}\ a \to \mathit{FingerTree}\ a \\
&a \lhd \mathit{Empty} &&= \mathit{Single}\ a \\
&a \lhd \mathit{Single}\ b &&= \mathit{Deep}\ [a]\ \mathit{Empty}\ [b] \\
&a \lhd \mathit{Deep}\ [b, c, d, e]\ m\ sf &&= \mathit{Deep}\ [a, b]\ (\mathit{Node3}\ c\ d\ e \lhd m)\ sf \\
&a \lhd \mathit{Deep}\ pr\ m\ sf &&= \mathit{Deep}\ ([a] +\!\!+ pr)\ m\ sf
\end{aligned}
$$

Notar que $(\lhd)$ usa recursión polimórfica

- no se soporta inferencia de tipos

Obtener y quitar el primer elemento

Vista de la izquierda de una secuencia s

$$\textbf{data}\ \mathit{ViewL}\ s\ a = \mathit{NilL} \mid \mathit{ConsL}\ a\ (s\ a)$$
$$\mathit{viewL} :: \mathit{FingerTree}\ a \to \mathit{ViewL}\ \mathit{FingerTree}\ a$$

Usando esas definiciones se pude definir:

$$\begin{aligned}
&\mathit{isEmpty} :: \mathit{FingerTree}\ a \to \mathit{Bool}\\
&\mathit{isEmpty}\ x = \textbf{case}\ \mathit{viewL}\ x\ \textbf{of}\ \mathit{NilL} \to \mathit{True}\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{ConsL}\ _\ _ \to \mathit{False}\\
&\mathit{headL} :: \mathit{FingerTree}\ a \to a\\
&\mathit{headL}\ x = \textbf{case}\ \mathit{viewL}\ x\ \textbf{of}\ \mathit{ConsL}\ a\ _ \to a\\
&\mathit{tailL} :: \mathit{FingerTree}\ a \to \mathit{FingerTree}\ a\\
&\mathit{tailL}\ x = \textbf{case}\ \mathit{viewL}\ x\ \textbf{of}\ \mathit{ConsL}\ _\ y \to y
\end{aligned}$$

Todas estas operaciones (incluso $\triangleleft$) tienen $O\ (1)$ amortizado

Análisis Amortizado

El **análisis amortizado** se diferencia del análisis en el caso promedio en que no involucra probabilidades, y en que garantiza el tiempo en el peor caso de una serie de N operaciones

Se puede ver como un **crédito** distribuído entre operaciones

- Tiempo de una operación es T
- Si una operación termina en un tiempo t antes que T, entonces junta crédito $T - t$
- Si otra operación lleva más tiempo, puede pagar con el crédito acumulado hasta el momento

En un contexto de evaluación perezosa y estructuras persistentes, hay que refinar el análisis

Análisis de las operaciones de deque

Se clasifican los *Digit* de 2 y 3 elementos como **seguros** y los de 1 y 4 como **inseguros**

Una operación podría propagar al siguiente nivel sólo en el caso de un *Digit* inseguro, pero al mismo tiempo lo vuelve seguro
Como mucho la mitad de las operaciones descienden un nivel, como mucho un cuarto dos niveles, etc.

Entonces para m operaciones tenemos que:
$T = m + \frac{1}{2}m + \frac{1}{4}m + \frac{1}{8}m + \frac{1}{16}m + ... = \sum_{i=0}^{n} m(\frac{1}{2})^i = 2m$
Siendo constante para cada operación.

En el caso persistente la **evaluación perezosa** asegura que las operaciones costosas se demoren y sean forzadas sólo luego de un número suficiente de operaciones que paguen el costo

Otras operaciones en Finger Trees

Algunas otras cosas que se pueden hacer con Finger Trees

```
data Seq a   -- abstracto
(><) :: Seq a → Seq a → Seq a              -- O(min(m, n))
length :: Seq a → Int                      -- O(1)
index :: Seq a → Int → a                   -- O(log n)
update :: Int → a → Seq a → Seq a          -- O(log n)
splitAt :: Int → Seq a → (Seq a, Seq a)    -- O(log n)
reverse :: Seq a → Seq a                   -- O(n)
```