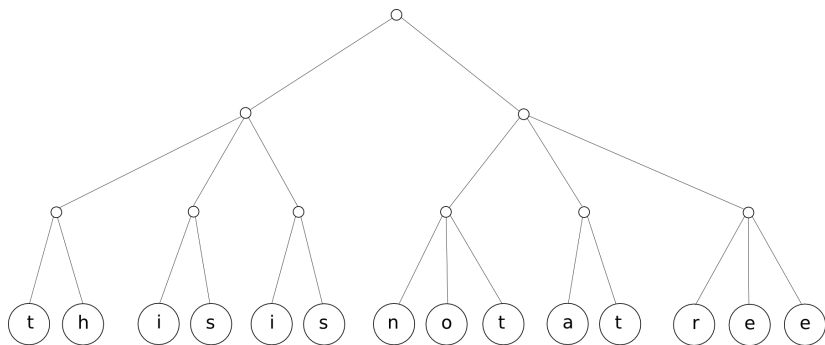
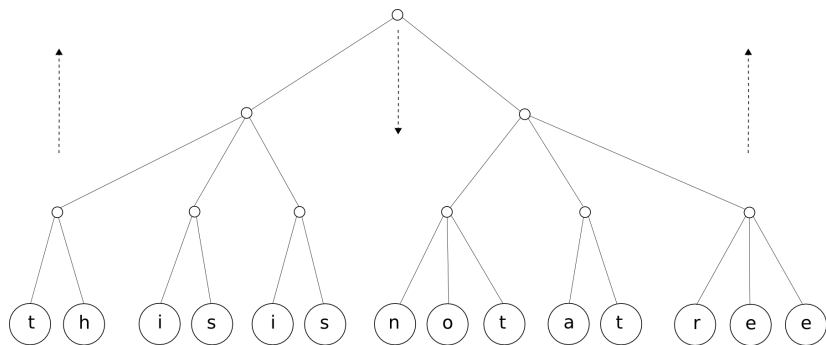


Tisztán funkcionális adatszerkezetek (folytatás)

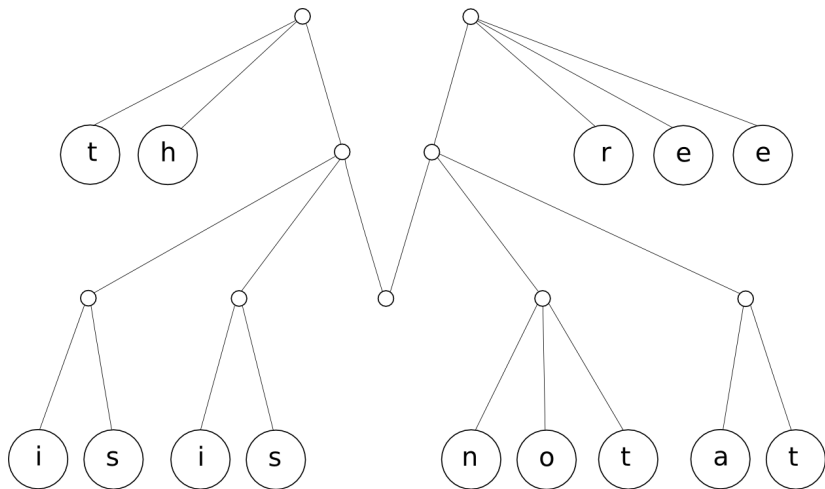
FingerTree (intuïció)



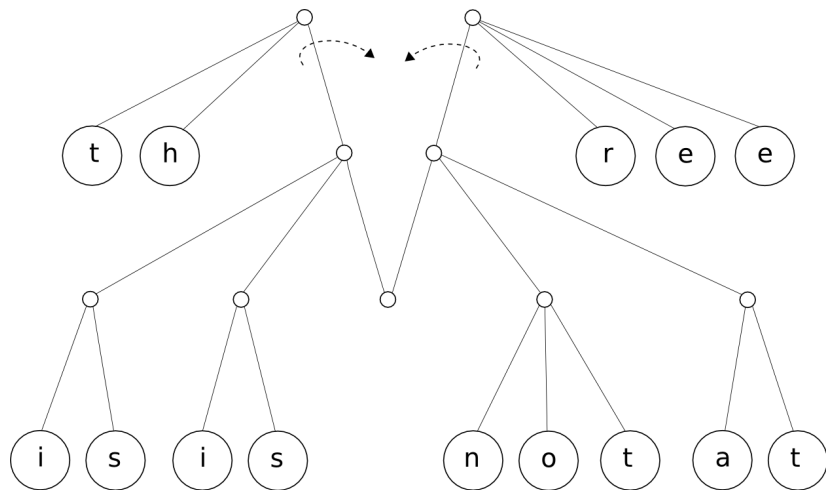
FingerTree (intuïció)



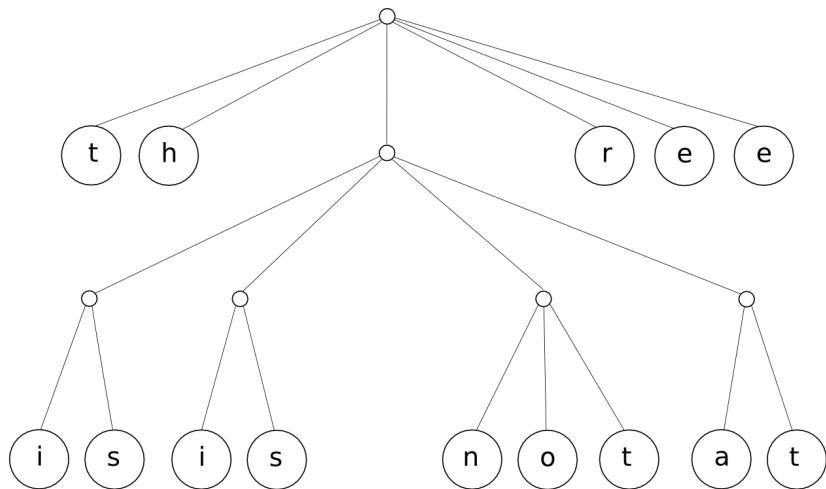
FingerTree (intuïció)



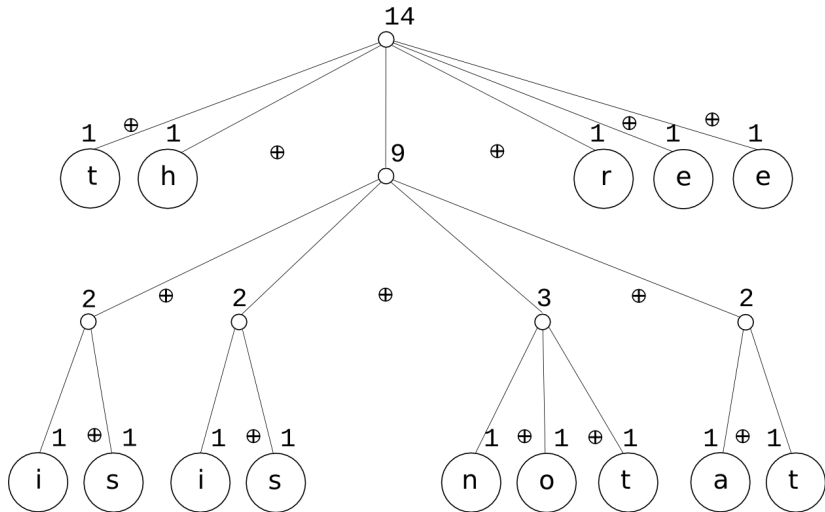
FingerTree (intuïció)



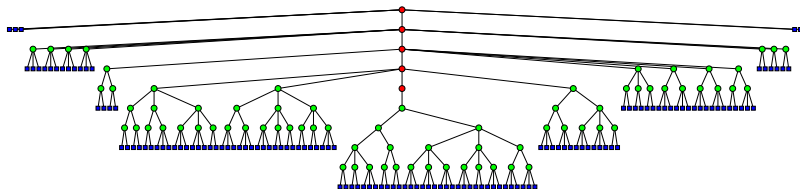
FingerTree (intuïció)



FingerTree (intuïció)



FingerTree (intuïció)



Implementáció: társított (indexelt) típuszinonimák

```
{-# LANGUAGE TypeFamilies, KindSignatures #-}
```

```
class Collects  $\alpha$  where
```

```
  type Elem  $\alpha$  ::  $\star$ 
```

```
  empty ::  $\alpha$ 
```

```
  insert :: Elem  $\alpha \rightarrow \alpha \rightarrow \alpha$ 
```

```
  ...
```

```
instance Eq (Elem  $[\varepsilon]$ )  $\Rightarrow$  Collects  $[\varepsilon]$  where
```

```
  type Elem  $[\varepsilon]$  =  $\varepsilon$ 
```

```
  empty      = []
```

```
  insert e xs = (e : xs)
```

```
  ...
```

Implementáció: nézetminták

```
{-# LANGUAGE ViewPatterns #-}
```

```
type Typ = ...
```

```
data TypView = Unit | Arrow Typ Typ
```

```
view :: Typ → TypView
```

```
view = ...
```

```
size :: Typ → Integer
```

```
size t = case (view t) of
```

```
    Unit           → 1
```

```
    Arrow t1 t2 → size t1 + size t2
```

Nézetminták segítségével pedig:

```
size (view → Unit) = 1
```

```
size (view → Arrow t1 t2) = size t1 + size t2
```

Implementáció: mohón kiértékelt adatkonstruktorok

data $T = T !Int !Int$

- ▶ A konstruktor ! segítségével megjelölt paramétereit (strictness annotation) normálformára kell hozni, mielőtt azt alkalmazzuk.
- ▶ Körültekintéssel kell alkalmazni, mivel ez automatikusan nem vezet a teljesítmény növekedéséhez.
- ▶ Sőt, ronthatja a teljesítményt: ha az adott mezőt már egyszer kiértékeltük, akkor lényegében még egyszer kiértékeltetjük (feleslegesen).
- ▶ A helyzet tisztázásában a fordító nem mindig tud a segítségünkre lenni.

Implementáció: egymásba ágyazott típusok

-- alternáló lista

data *AList* $\alpha \beta = Nil \mid Cons \alpha (AList \beta \alpha)$

alist :: *AList* Int Char

alist = Cons 1 (Cons 'A' (Cons 2 (Cons 'B' Nil)))

-- ciklikus lista

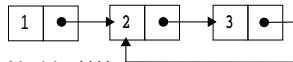
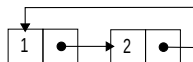
data *Void*

data *CList* $\alpha \beta = Var \beta \mid Nil \mid RCons \alpha (CList (Maybe \beta))$

clist₁, *clist₂* :: *CList* Int *Void*

clist₁ = *RCons* 1 (*RCons* 2 (*Var* Nothing))

clist₂ = *RCons* 1 (*RCons* 2 (*RCons* 3 (*Var* (Just Nothing))))



- ▶ A rekurzív részben az adattípust nem a deklaráció szerint alkalmazzuk: nested, non-regular, non-uniform, heterogenous data type
- ▶ Adattípusokon belüli invariánsok (típusozott) megtartására alkalmazható.

FingerTree: definíció (1)

class (*Monoid* (*Measure* α)) \Rightarrow *Measured* α **where**

type *Measure* α

measure :: $\alpha \rightarrow \text{Measure } \alpha$

data *FingerTree* α

= *Empty*

| *Single* α

| *Deep* !(*Measure* α) !(*Digit* α) (*FingerTree* (*Node* α)) !(*Digit* α)

deep :: *Measured* α

\Rightarrow *Digit* $\alpha \rightarrow \text{FingerTree} (\text{Node } \alpha) \rightarrow \text{Digit } \alpha \rightarrow \text{FingerTree } \alpha$

deep pr m sf = *Deep* (*measure pr* \oplus *measure m* \oplus *measure sf*) *pr m sf*

data *Node* α = *Node2* (*Measure* α) α α | *Node3* (*Measure* α) α α α

node2 :: *Measured* $\alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \text{Node } \alpha$

node2 x y = *Node2* (*measure x* \oplus *measure y*) *x y*

node3 :: *Measured* $\alpha \Rightarrow \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \text{Node } \alpha$

node3 x y z = *Node3* (*measure x* \oplus *measure y* \oplus *measure z*) *x y z*

FingerTree: definíció (2)

newtype *Digit* α = *D* { *unD* :: [α] }

prependDigit :: $\alpha \rightarrow \text{Digit } \alpha \rightarrow \text{Digit } \alpha$
prependDigit *x* (*D xs*) = *D* (*x* : *xs*)

appendDigit :: *Digit* $\alpha \rightarrow \alpha \rightarrow \text{Digit } \alpha$
appendDigit (*D xs*) *x* = *D* (*xs* ++ [*x*])

breakDigit :: *Digit* $\alpha \rightarrow (\alpha, \text{Digit } \alpha)$
breakDigit (*D* (*x* : *xs*)) = (*x*, *D xs*)

kaerbDigit :: *Digit* $\alpha \rightarrow (\alpha, \text{Digit } \alpha)$
kaerbDigit (*D xs*) = (*last xs*, *D* (*init xs*))

FingerTree: definíció (3)

instance *Measured* $\alpha \Rightarrow \text{Measured (Node } \alpha)$ **where**
type *Measure* (Node α) = *Measure* α

measure (Node2 *m* __) = *m*

measure (Node3 *m* ___) = *m*

instance *Measured* $\alpha \Rightarrow \text{Measured (Digit } \alpha)$ **where**
type *Measure* (Digit α) = *Measure* α

measure (D *xs*) = *foldr* (\oplus) *mempty* (*map measure xs*)

instance *Measured* $\alpha \Rightarrow \text{Measured (FingerTree } \alpha)$ **where**
type *Measure* (FingerTree α) = *Measure* α

measure *Empty* = *mempty*

measure (Single *x*) = *measure x*

measure (Deep *m* ___) = *m*

FingerTree: létrehozás

-- O(1)

empty :: *Measured* $\alpha \Rightarrow$ *FingerTree* α

empty = *Empty*

singleton :: *Measured* $\alpha \Rightarrow \alpha \rightarrow$ *FingerTree* α

singleton = *Single*

infixr 5 \triangleleft

(\triangleleft) :: *Measured* $\alpha \Rightarrow \alpha \rightarrow$ *FingerTree* $\alpha \rightarrow$ *FingerTree* α

$x \triangleleft$ *Empty* = *Single* x

$x \triangleleft$ (*Single* y) = *deep* (*D* $[x]$) *Empty* (*D* $[y]$)

$x \triangleleft$ (*Deep* _ (*D* $[y, z, u, w]$) m *sf*) = *deep* (*D* $[x, y]$) (*node3* $z\ u\ w\ \triangleleft\ m$) *sf*

$x \triangleleft$ (*Deep* _ *pr* m *sf*) = *deep* (*prependDigit* $x\ pr$) $m\ sf$

-- O(n)

(\trianglelefteq) :: *Measured* $\alpha \Rightarrow [\alpha] \rightarrow$ *FingerTree* $\alpha \rightarrow$ *FingerTree* α

$xs \trianglelefteq ys$ = *foldr* (\triangleleft) $ys\ xs$

fromList :: *Measured* $\alpha \Rightarrow [\alpha] \rightarrow$ *FingerTree* α

fromList xs = $xs \trianglelefteq$ *Empty*

digitToTree :: *Measured* $\alpha \Rightarrow$ *Digit* $\alpha \rightarrow$ *FingerTree* α

digitToTree (*D* xs) = *fromList* xs

Ugyanígy megadható a \triangleright (végére illesztés) művelete is.

FingerTree: elérés

infixr 5 \ll

data $\text{ViewL } \alpha = \text{EmptyL} \mid \alpha \ll (\text{FingerTree } \alpha)$

-- $O(1)$

$\text{viewl} :: \text{Measured } \alpha \Rightarrow \text{FingerTree } \alpha \rightarrow \text{ViewL } \alpha$

$\text{viewl } \text{Empty} = \text{EmptyL}$

$\text{viewl } (\text{Single } x) = x \ll \text{Empty}$

$\text{viewl } (\text{Deep_pr } m \text{ sf}) = p \ll (\text{deepl } lpr \text{ m sf})$

where $(p, lpr) = \text{breakDigit } pr$

$\text{deepl} :: \text{Measured } \alpha$

$\Rightarrow \text{Digit } \alpha \rightarrow \text{FingerTree } (\text{Node } \alpha) \rightarrow \text{Digit } \alpha \rightarrow \text{FingerTree } \alpha$

$\text{deepl } (D []) (\text{viewl} \rightarrow \text{EmptyL}) \text{ sf} = \text{digitToTree sf}$

$\text{deepl } (D []) (\text{viewl} \rightarrow x \ll m) \text{ sf} = \text{deep } (\text{nodeToDigit } x) \text{ m sf}$

$\text{deepl } pr \text{ m sf} = \text{deep } pr \text{ m sf}$

$\text{nodeToDigit} :: \text{Measured } \alpha \Rightarrow \text{Node } \alpha \rightarrow \text{Digit } \alpha$

$\text{nodeToDigit } (\text{Node2_} x \text{ y}) = D [x, y]$

$\text{nodeToDigit } (\text{Node3_} x \text{ y } z) = D [x, y, z]$

Ugyanígy megadható a `viewr` (végéről, jobbról balra bejárás) művelete is.

FingerTree: elérés (alkalmazás)

$isEmpty :: Measured\ \alpha \Rightarrow FingerTree\ \alpha \rightarrow Bool$
 $isEmpty\ (viewl \rightarrow EmptyL) = True$
 $isEmpty\ _ = False$

$headl :: Measured\ \alpha \Rightarrow FingerTree\ \alpha \rightarrow \alpha$
 $headl\ (viewl \rightarrow x \ll _) = x$

$tail :: Measured\ \alpha \Rightarrow FingerTree\ \alpha \rightarrow FingerTree\ \alpha$
 $tail\ (viewl \rightarrow _ \ll x) = x$

Hasonlóan létezik `headr` és `tailr` is.

FingerTree: összekapcsolás

infixr 5 \bowtie

-- $O(\log(\min(n, m)))$

$(\bowtie) :: \text{Measured } \alpha \Rightarrow \text{FingerTree } \alpha \rightarrow \text{FingerTree } \alpha \rightarrow \text{FingerTree } \alpha$

$xs \bowtie ys = f\ xs\ []\ ys$ **where**

$f :: \text{Measured } \alpha \Rightarrow \text{FingerTree } \alpha \rightarrow [\alpha] \rightarrow \text{FingerTree } \alpha \rightarrow \text{FingerTree } \alpha$

$f\ \text{Empty} \quad \quad \quad ts\ xs \quad \quad \quad = ts \trianglelefteq xs$

$f\ xs \quad \quad \quad ts\ \text{Empty} \quad \quad \quad = xs \trianglerighteq ts$

$f\ (\text{Single } x) \quad \quad \quad ts\ xs \quad \quad \quad = x \triangleleft (ts \trianglelefteq xs)$

$f\ xs \quad \quad \quad ts\ (\text{Single } x) \quad \quad \quad = (xs \trianglerighteq ts) \triangleright x$

$f\ (\text{Deep_pr}_1\ m_1\ sf_1)\ ts\ (\text{Deep_pr}_2\ m_2\ sf_2) =$
 $\text{deep } pr_1\ (f\ m_1\ (\text{nodes } (unD\ sf_1\ ++\ ts\ ++\ unD\ pr_2))\ m_2)\ sf_2$

$\text{nodes} :: \text{Measured } \alpha \Rightarrow [\alpha] \rightarrow [\text{Node } \alpha]$

$\text{nodes } [x, y] \quad \quad \quad = [\text{node2 } x\ y]$

$\text{nodes } [x, y, z] \quad \quad \quad = [\text{node3 } x\ y\ z]$

$\text{nodes } [x, y, z, u] \quad \quad \quad = [\text{node2 } x\ y, \text{node2 } z\ u]$

$\text{nodes } (x : y : z : xs) = \text{node3 } x\ y\ z : \text{nodes } xs$

FingerTree: felbontás (1)

data $\text{Split } \phi \alpha = \text{Split } (\phi \alpha) \alpha (\phi \alpha)$

i : kezdőérték, $P(\cdot)$: monoton predikátum, csak \wedge és \vee

$\neg P(i) \wedge P(i \oplus \|d\|) \implies$

let $\text{Split } l \ x \ r = \text{splitDigit } p \ i \ d$

in $\text{toList } l \ ++ \ [x] \ ++ \ \text{toList } r = \text{toList } d$

$\wedge \neg P(i \oplus \|l\|) \wedge P(i \oplus \|l\| \oplus \|x\|)$

$\text{splitDigit} :: \text{Measured } \alpha$

$\implies (\text{Measure } \alpha \rightarrow \text{Bool}) \rightarrow \text{Measure } \alpha \rightarrow \text{Digit } \alpha \rightarrow \text{Split Digit } \alpha$

$\text{splitDigit } p \ i \ (\text{breakDigit} \rightarrow (x, D [])) = \text{Split } (D []) \ x \ (D [])$

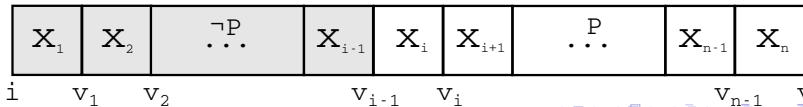
$\text{splitDigit } p \ i \ (\text{breakDigit} \rightarrow (x, xs))$

| $p \ j \quad = \text{Split } (D []) \ x \ xs$

| $\text{otherwise} = \text{let } \text{Split } l \ y \ r = \text{splitDigit } p \ j \ xs$

in $\text{Split } (\text{prependDigit } x \ l) \ y \ r$

where $j = i \oplus \text{measure } x$



FingerTree: felbontás (2)

$\neg P(i) \wedge P(i \oplus \|t\|) \implies$

```
let Split l x r = splitTree p i t
in toList l ++ [x] ++ toList r = toList t
```

$\wedge \neg P(i \oplus \|l\|) \wedge P(i \oplus \|l\| \oplus \|x\|)$

-- $O(\log(\min(n_l, n_r)))$

splitTree :: *Measured* α

$\implies (\text{Measure } \alpha \rightarrow \text{Bool}) \rightarrow \text{Measure } \alpha \rightarrow \text{FingerTree } \alpha \rightarrow \text{Split FingerTree } \alpha$

splitTree p i (Single x) = Split Empty x Empty

splitTree p i (Deep _pr m sf)

| p vpr = let Split l x r = splitDigit p i pr

in Split (digitToTree l) x (deepl r m sf)

| p vm = let Split ml xs mr = splitTree p vpr m

Split l x r = splitDigit p (vpr \oplus measure ml) (nodeToDigit xs)

in Split (deepr pr ml l) x (deepl r mr sf)

| otherwise = let Split l x r = splitDigit p vm sf

in Split (deepr pr m l) x (digitToTree r)

where (vpr, vm) = (i \oplus measure pr, vpr \oplus measure m)

FingerTree: felbontás (3)

$t \neq \text{Empty} \implies$

let $\text{Split } l \times r = \text{splitTree } p \ i \ t$
in $\text{toList } l \ ++ \ [x] \ ++ \ \text{toList } r = \text{toList } t$

$\wedge (l = \text{Empty} \vee \neg P(i \oplus \|l\|)) \wedge (r = \text{Empty} \vee P(i \oplus \|l\| \oplus \|x\|))$

$\text{split} :: \text{Measured } \alpha$

$\implies (\text{Measure } \alpha \rightarrow \text{Bool}) \rightarrow \text{FingerTree } \alpha \rightarrow (\text{FingerTree } \alpha, \text{FingerTree } \alpha)$

$\text{split } p \ \text{Empty} = (\text{Empty}, \text{Empty})$

$\text{split } p \ t$

| $p(\text{measure } t)$ $= (l, x \triangleleft r)$

| *otherwise* $= (t, \text{Empty})$

where $\text{Split } l \times r = \text{splitTree } p \ \text{mempty } t$

FingerTree: véletlen elérésű sorozatok (alkalmazás)

```
newtype Elem  $\nu$   $\alpha$  = Elem { getElem ::  $\alpha$  } --  $\nu$  : fantomtípus  
newtype Size      = Size { getSize :: Int } deriving (Eq, Ord)  
newtype Seq  $\alpha$     = Seq (FingerTree (Elem Size  $\alpha$ ))
```

```
instance Monoid Size where
```

```
    mempty                = Size 0  
    mappend (Size m) (Size n) = Size (m + n)
```

```
instance Measured (Elem Size  $\alpha$ ) where
```

```
    type Measure (Elem Size a) = Size  
    measure (Elem _) = Size 1
```

```
fromList :: [ $\alpha$ ]  $\rightarrow$  Seq  $\alpha$   
fromList xs = Seq (FT.fromList (map Elem xs))
```

```
toList :: Seq  $\alpha$   $\rightarrow$  [ $\alpha$ ]  
toList (Seq t) = map getElem (FT.toList t)
```

FingerTree: véletlen elérésű sorozatok (alkalmazás)

```
length :: Seq  $\alpha$   $\rightarrow$  Int  
length (Seq xs) = getSize (FT.measure xs)
```

```
splitAt :: Int  $\rightarrow$  Seq  $\alpha$   $\rightarrow$  (Seq  $\alpha$ , Seq  $\alpha$ )  
splitAt i (Seq xs) = (Seq l, Seq r)  
  where (l, r) = FT.split (Size i <) xs
```

```
(!) :: Seq  $\alpha$   $\rightarrow$  Int  $\rightarrow$   $\alpha$   
(Seq xs) ! i = getElem x  
  where Split _ x _ = FT.splitTree (Size i <) (Size 0) xs
```


FingerTree: prioritásos sor (alkalmazás)

```
newtype PElem  $\pi$   $\alpha$  = PE ( $\pi$ ,  $\alpha$ )  
data Prio  $\alpha$  = Prio  $\alpha$  | NoPrio deriving (Eq, Ord)  
instance Ord  $\alpha \Rightarrow$  Monoid (Prio  $\alpha$ ) where  
  empty = NoPrio  
  mappend (Prio x) (Prio y) = Prio (min x y)  
  mappend (Prio x) _ = Prio x  
  mappend _ (Prio y) = Prio y  
  mappend _ _ = NoPrio  
instance Monoid (Prio  $\pi$ )  $\Rightarrow$  Measured (PElem (Prio  $\pi$ )  $\alpha$ ) where  
  type Measure (PElem (Prio  $\pi$ )  $\alpha$ ) = Prio  $\pi$   
  measure (PE (i, _)) = i  
newtype PQueue  $\pi$   $\alpha$  = PQ (FingerTree (PElem (Prio  $\pi$ )  $\alpha$ ))  
  
empty :: Ord  $\pi \Rightarrow$  PQueue  $\pi$   $\alpha$   
empty = PQ FT.empty  
  
null :: Ord  $\pi \Rightarrow$  PQueue  $\pi$   $\alpha \rightarrow$  Bool  
null (PQ t) = FT.isEmpty t
```

FingerTree: prioritásos sor (alkalmazás)

$singleton :: Ord\ \pi \Rightarrow \pi \rightarrow \alpha \rightarrow PQueue\ \pi\ \alpha$
 $singleton\ k\ v = PQ\ (FT.singleton\ (PE\ (Prio\ k,\ v)))$

$extractMin :: Ord\ \pi \Rightarrow PQueue\ \pi\ \alpha \rightarrow Maybe\ (\alpha,\ PQueue\ \pi\ \alpha)$
 $extractMin\ (PQ\ t)$
 | $FT.isEmpty\ t = Nothing$
 | $otherwise = Just\ (x,\ PQ\ (l\ \bowtie\ r))$
 where $Split\ l\ (PE\ (_,\ x))\ r = FT.splitTree\ (FT.measure\ t\ \geq)\ mempty\ t$

$union :: Ord\ \pi \Rightarrow PQueue\ \pi\ \alpha \rightarrow PQueue\ \pi\ \alpha \rightarrow PQueue\ \pi\ \alpha$
 $union\ (PQ\ p)\ (PQ\ q) = PQ\ (p\ \bowtie\ q)$

$insert :: Ord\ \pi \Rightarrow \pi \rightarrow \alpha \rightarrow PQueue\ \pi\ \alpha \rightarrow PQueue\ \pi\ \alpha$
 $insert\ k\ v\ q = union\ (singleton\ k\ v)\ q$

$add :: Ord\ \pi \Rightarrow \pi \rightarrow \alpha \rightarrow PQueue\ \pi\ \alpha \rightarrow PQueue\ \pi\ \alpha$
 $add\ k\ v\ q = union\ q\ (singleton\ k\ v)$

$fromList :: Ord\ \pi \Rightarrow [(\pi,\ \alpha)] \rightarrow PQueue\ \pi\ \alpha$
 $fromList = foldr\ (\lambda\ (x,\ y) \rightarrow insert\ x\ y)\ mempty$