Finger Trees

Custom Persistent Collections

Hack around with them during the talk:
`http://tinyurl.com/fingertree`

Chris Houser
a.k.a. Chouser

Clojure Conj, Oct. 22 2010, Durham NC

Finger Trees

Invented by Ralf Hinze and Ross Paterson

Another persistent collection type

Complements existing Clojure collections

Customizable

double-list

```
(def dl (double-list 4 5 6 7))

dl
;=> (4 5 6 7)

[(first dl) (rest dl)]
;=> [4 (5 6 7)]

[(pop dl) (peek dl)]
;=> [(4 5 6) 7]
```

new “conj” functions

conjr *always* adds on the right

```
(conjr dl 'x)
;=> (4 5 6 7 x)
```

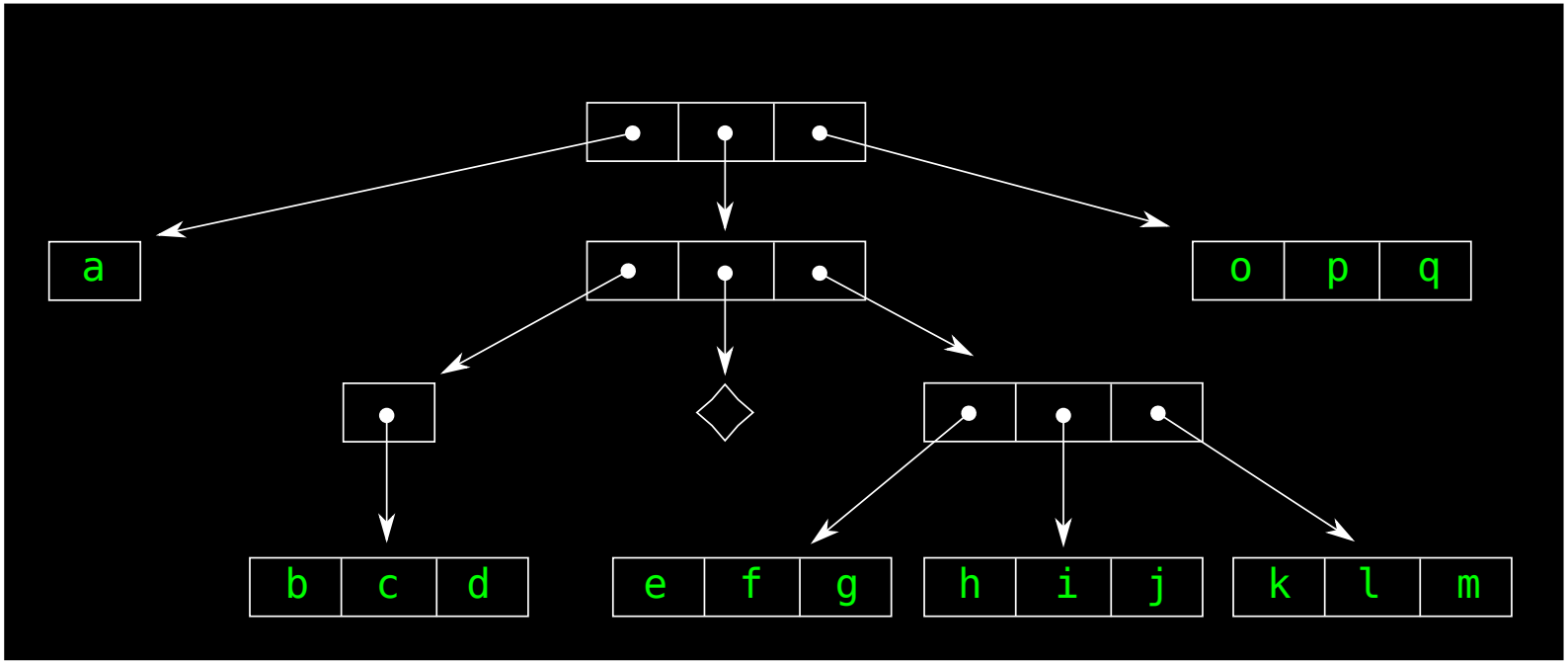
consl adds on the left and keeps the collection type

```
(consl dl 'x)
;=> (x 4 5 6 7)
```

So far these are all amortized constant time

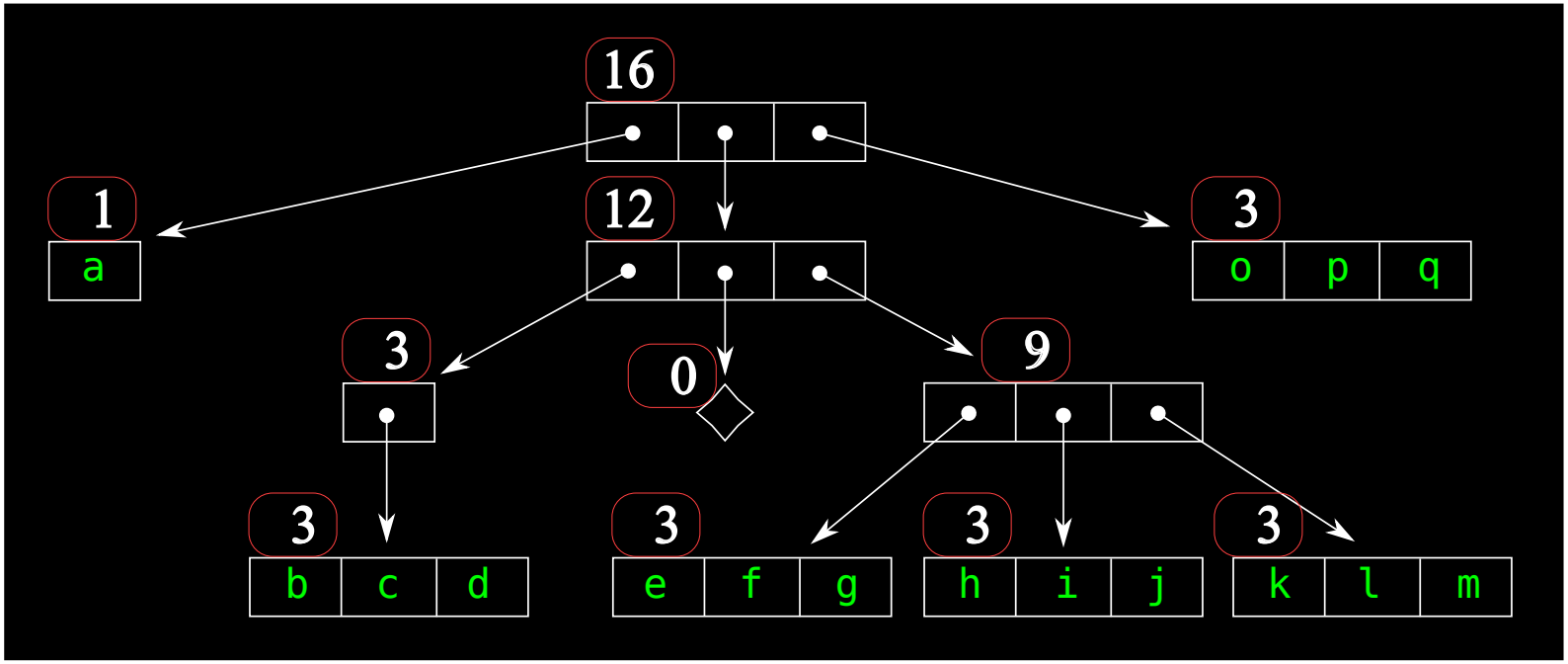
double-list

```
(apply double-list '[a b c d e f g h i j k l m])
```



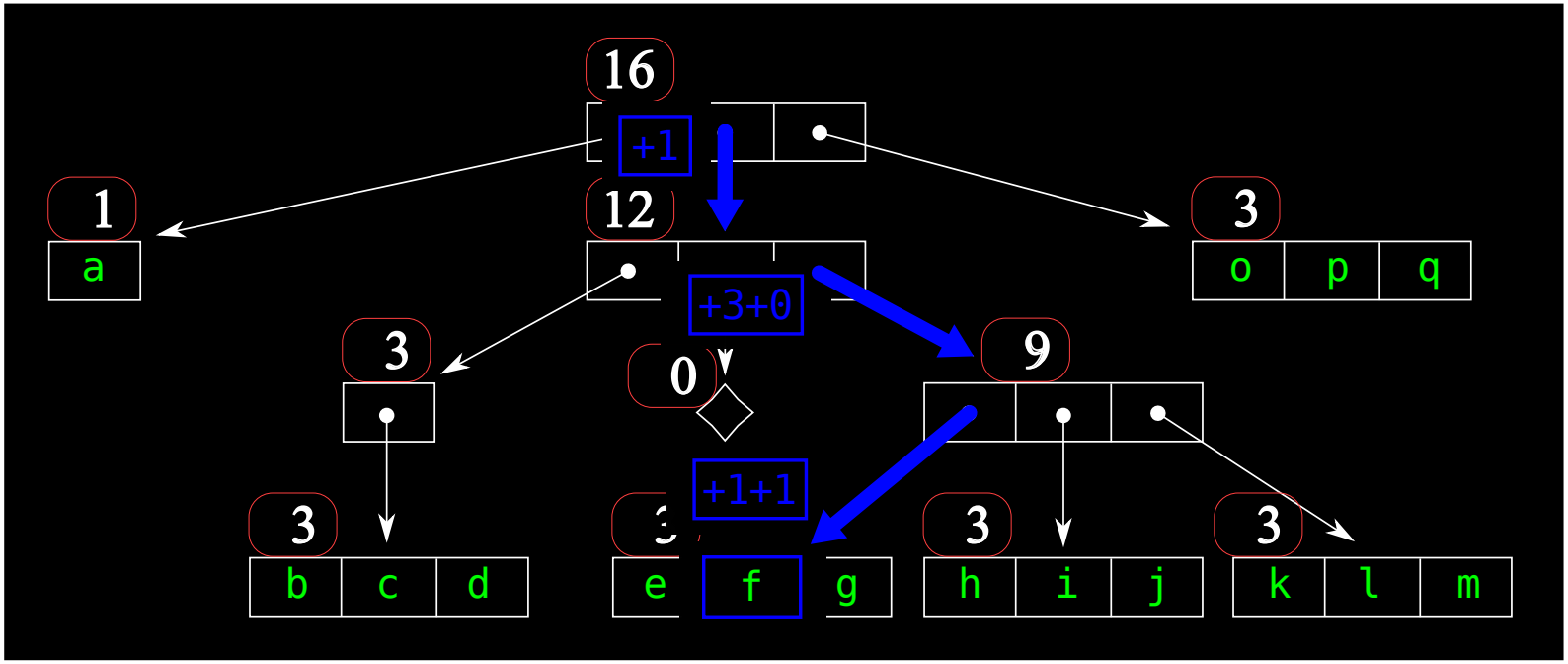
counted-double-list

```
(apply counted-double-list '[a b c d e f g h i j k l m]))
```



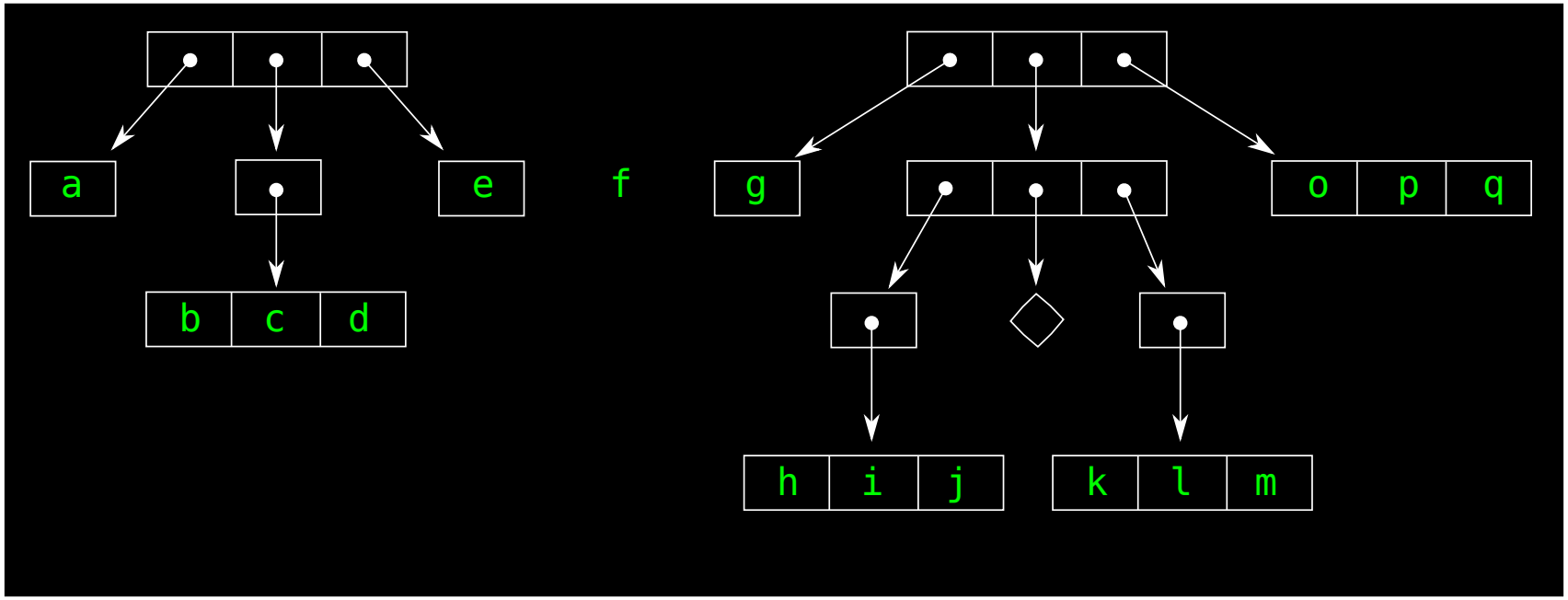
Lookup by count

```
(nth cdl 5)
```



Split

```
(ft-split-at cdl 5)
;=> [(a b c d e) f (g h i j k l m)]
```



Assoc

```
(def parts
  (let [[left _ right] (ft-split-at cdl 5)]
    {:left left, :right right}))

parts
;=> {:left (a b c d e), :right (g h i j k l m)}

(ft-concat (conjr (:left parts) 'XX) (:right parts))
;=> (a b c d e XX g h i j k l m)
```

Remove and Insert

```
(ft-concat (:left parts) (:right parts))
;=> (a b c d e g h i j k l m)
               ^-- missing f

(ft-concat (into (:left parts) '[X Y Z])
           (:right parts))
;=> (a b c d e X Y Z g h i j k l m)
```

Meter for counted-double-list

```
(finger-tree (meter
              (constantly 1)  ; measure
              0               ; measure of empty
              +))             ; combine
```

“measure of empty” and “combine” together form a *monoid*

```
instance Reduce FingerTree where
  reducer (≺) Empty          z = z
  reducer (≺) (Single x)     z = x ≺ z
  reducer (≺) (Deep pr m sf) z = pr ≺' (m ≺'' (sf ≺' z))
    where (≺')  = reducer (≺)
          (≺'') = reducer (reducer (≺))

  reducel (≻) z Empty          = z
  reducel (≻) z (Single x)     = z ≻ x
  reducel (≻) z (Deep pr m sf) = ((z ≻' pr) ≻'' m) ≻' sf
    where (≻')  = reducel (≻)
          (≻'') = reducel (reducel (≻))
```

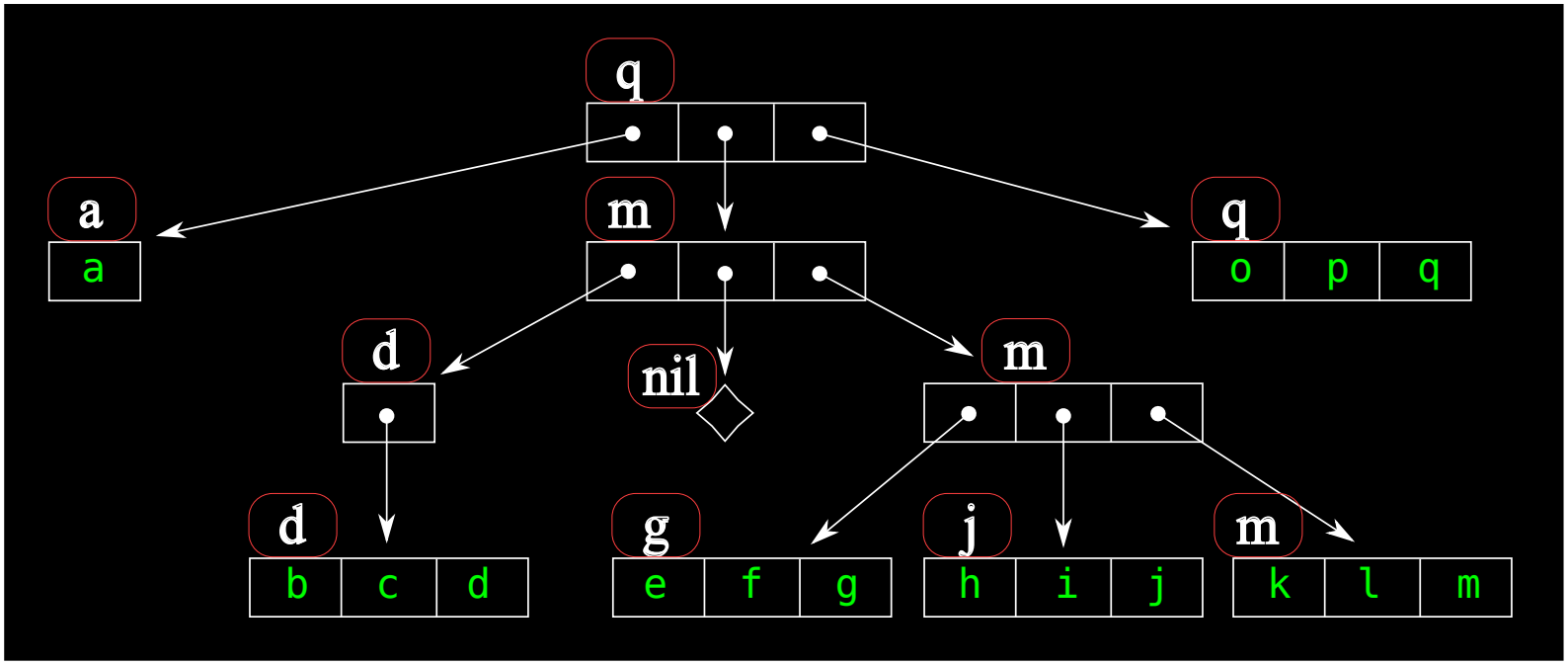
Split for counted-double-list

Split uses a predicate, splits where the predicate changes from false to true

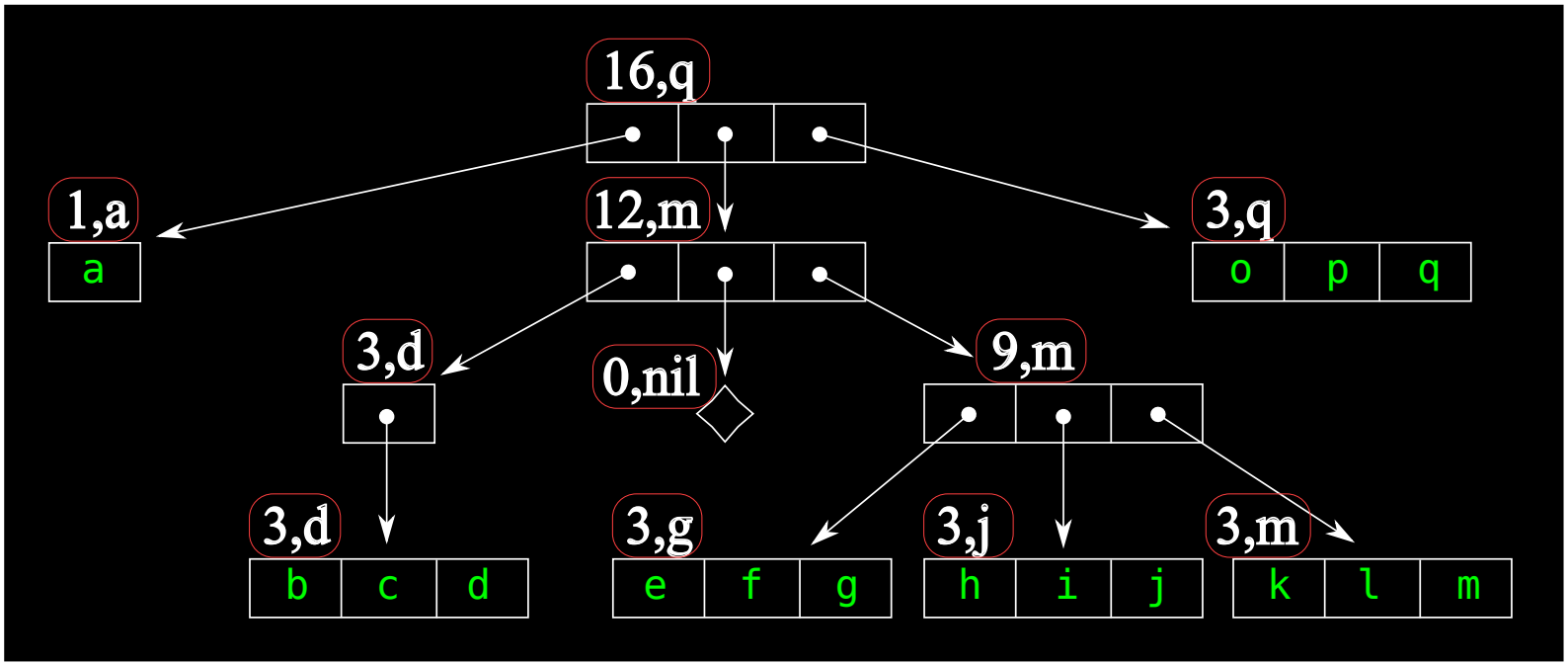
```
(split-tree tree #(> % 5))
```

Meter for _____

```
(finger-tree (meter
               identity          ; measure
               nil               ; measure of empty
               #(or %2 %1)))     ; combine
```



counted-sorted-set



counted-sorted-set

```
(def css (apply counted-sorted-set
                '[m j i e d a f k b c f g h l]))
css
;=> (a b c d e f g h i j k l m)

(get css 'e)      ; O(log(n))
;=> e

(get css 'ee)     ; O(log(n))
;=> nil

(nth css 5)       ; O(log(n))
;=> f

(count css)       ; O(1)
;=> 13
```

Build-your-own finger tree

```
(def empty-cost-tree (finger-tree (meter :cost 0 +)))

(def ct (conj empty-cost-tree
              {:id :h, :cost 5} {:id :i, :cost 1}
              {:id :j, :cost 2} {:id :k, :cost 3}
              {:id :l, :cost 4}))

(measured ct)
;=> 15

(next (split-tree ct #(> % 7)))
;=> ({:cost 2, :id :j}
     ({:cost 3, :id :k} {:cost 4, :id :l}))

(next (split-tree (rest ct) #(> % 7)))
;=> ({:cost 4, :id :l} ())
```

Summary of `clojure.data.finger-tree`

double-list (add/remove on left/right)

counted-double-list (double-list plus nth)

counted-sorted-set (like sorted-set plus nth)

tools for building your own finger-tree

Future work

Implement metadata, equality, etc.

Adjust Clojure's abstractions to allow `cons conj split-at concat` instead of `consl conjr ft-split-at ft-concat`

Tests for correctness, complexity

Performance

Primitives

Questions?

http://tinyurl.com/fingertree