J. Functional Programming **7** (3): 349–354, May 1997. Printed in the United Kingdom 349

FUNCTIONAL PEARL

On merging and selection

RICHARD S. BIRD
Programming Research Group, Oxford University,
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

1 Introduction

Given two ascending lists xs and ys of combined length greater than n, consider the computation of

$$merge\,(xs, ys)\;!!\;n$$

The standard function *merge* merges two ascending sequences and (!!) denotes list indexing. With a lazy evaluator the computation takes $O(n)$ steps; with an eager one it takes $O(p+q)$ steps, where $p = length\;xs$ and $q = length\;ys$. Now in functional programming it is more efficient to index a tree than a list, so the question arises: can we find a faster solution if xs and ys are each represented by a tree? Somewhat surprisingly the answer is yes: if xs and ys are each represented by balanced binary search trees, then the computation can be reduced to $O(\log p + \log q)$ steps. This is despite the fact that there is no known method for merging two binary search trees in better than linear time. The details, presented below, depend on a subtle relationship between merging and indexing.

2 Specification

We begin by specifying the problem more precisely. Consider the type

$$\textbf{data}\;Tree\;a \quad = \quad Null \mid Node\;Int\;(Tree\;a)\;a\;(Tree\;a)$$

of binary trees under the restriction that each element xt of $Tree\,a$ satisfies the following datatype invariants:

1. The sequence $flatten\;xt$ is ascending, where

$$\begin{aligned} flatten\;Null &= [\,] \\ flatten\,(Node\;p\;xt\;x\;yt) &= flatten\;xt \mathbin{+\!\!+} [x] \mathbin{+\!\!+} flatten\;yt \end{aligned}$$

   In words, xt is a binary search tree.

2. If $xt = Node\;p\;yt\;y\;zt$, then $p = size\;yt$, where

$$\begin{aligned} size\;Null &= 0 \\ size\,(Node\;p\;xt\;x\;yt) &= size\;xt + 1 + size\;yt. \end{aligned}$$

https://doi.org/10.1017/S0956796897002736 Published online by Cambridge University Press

In words, each node is labelled with the size of its left subtree.

In particular, the following equations hold, referred to subsequently by the hint 'datatype invariant':

$$\begin{array}{rcl} \mathit{take}\ p\ (\mathit{flatten}\ (\mathit{Node}\ p\ xt\ x\ yt)) & = & \mathit{flatten}\ xt \\ \mathit{flatten}\ (\mathit{Node}\ p\ xt\ x\ yt)\ !!\ p & = & x \\ \mathit{drop}\ (p+1)\ (\mathit{flatten}\ (\mathit{Node}\ p\ xt\ x\ yt)) & = & \mathit{flatten}\ yt \end{array}$$

Now define

$$\mathit{select}\ (xt, yt)\ n \quad = \quad \mathit{merge}\ (\mathit{flatten}\ xt, \mathit{flatten}\ yt)\ !!\ n$$

Our aim is to show that $\mathit{select}\ (xt, yt)\ n$ can be computed in $O(\mathit{height}\ xt + \mathit{height}\ yt)$ steps, where *height* is defined by

$$\begin{array}{lcl} \mathit{height}\ \mathit{Null} & = & 0 \\ \mathit{height}\ (\mathit{Node}\ p\ xt\ x\ yt) & = & 1 + \mathit{max}\ (\mathit{height}\ xt, \mathit{height}\ yt) \end{array}$$

3 Derivation

It is instructive to consider first a simpler problem. Define *index* by

$$\mathit{index}\ xt\ n \quad = \quad (\mathit{flatten}\ xt)\ !!\ n$$

To synthesise a more efficient program for *index* we need the following relationship between concatenation and indexing:

$$\begin{array}{rcll} (xs \mathbin{+\!\!+} ys)\ !!\ n & = & xs\ !!\ n, & \text{if } n < p \\ & = & ys\ !!\ (n-p), & \text{otherwise} \\ & & \text{where } p = \mathit{length}\ xs & \end{array}$$

It is now an easy task to synthesise the following alternative program for *index*:

$$\begin{array}{rcll} \mathit{index}\ (\mathit{Node}\ p\ xt\ x\ yt)\ n & = & \mathit{index}\ xt\ n, & \text{if } n < p \\ & = & x, & \text{if } n = p \\ & = & \mathit{index}\ yt\ (n-p-1), & \text{if } n > p \end{array}$$

Evaluation of $\mathit{index}\ xt\ n$ takes $O(\mathit{height}\ xt)$ steps. In particular, if xt is balanced, then the cost is $O(\log(\mathit{size}\ xt))$ steps.

The efficient program for *select* is derived in an analogous fashion, and depends on the relationship between merging and indexing. Suppose xs and ys are two ascending sequences of combined length greater than n, and let p and q be two natural numbers satisfying $0 \leq p < \mathit{length}\ xs$ and $0 \leq q < \mathit{length}\ ys$. The facts we need are:

1. If $n \leq p + q$, then

$$\begin{array}{rcll} \mathit{merge}\ (xs, ys)\ !!\ n & = & \mathit{merge}\ (xs, \mathit{take}\ q\ ys)\ !!\ n, & \text{if } xs\ !!\ p \leq ys\ !!\ q \\ & = & \mathit{merge}\ (\mathit{take}\ p\ xs, ys)\ !!\ n, & \text{if } ys\ !!\ q \leq xs\ !!\ p \end{array}$$

2. If $p + q < n$, then

$$\begin{array}{ll} \multicolumn{2}{l}{\mathit{merge}\,(xs, ys)\,!!\,n} \\ = & \mathit{merge}\,(\mathit{drop}\,(p+1)\,xs, ys)\,!!\,(n-p-1), \quad \text{if } xs\,!!\,p \le ys\,!!\,q \\ = & \mathit{merge}\,(xs, \mathit{drop}\,(q+1)\,ys)\,!!\,(n-q-1), \quad \text{if } ys\,!!\,q \le xs\,!!\,p \end{array}$$

Given these facts, the synthesis of the program for *select* is straightforward. We give the details for just one case. For brevity, introduce the functions

$$\begin{array}{lcl} \mathit{label}\,(\mathit{Node}\ p\ xt\ x\ yt) & = & p \\ \mathit{left}\,(\mathit{Node}\ p\ xt\ x\ yt) & = & xt \\ \mathit{value}\,(\mathit{Node}\ p\ xt\ x\ yt) & = & x \\ \mathit{right}\,(\mathit{Node}\ p\ xt\ x\ yt) & = & yt \end{array}$$

Now, with $p = \mathit{label}\ xt$ and $q = \mathit{label}\ yt$ we argue:

$$\begin{array}{cl} & \mathit{select}\,(xt, yt)\,n \\ = & \quad \{\text{definition}\} \\ & \mathit{merge}\,(\mathit{flatten}\ xt, \mathit{flatten}\ yt)\,!!\,n \\ = & \quad \{\text{property (1), assuming } n \le p + q \text{ and } \mathit{value}\ xt \le \mathit{value}\ yt\} \\ & \mathit{merge}\,(\mathit{flatten}\ xt, \mathit{take}\ q\,(\mathit{flatten}\ yt))\,!!\,n \\ = & \quad \{\text{datatype invariant}\} \\ & \mathit{merge}\,(\mathit{flatten}\ xt, \mathit{flatten}\,(\mathit{left}\ yt))\,!!\,n \\ = & \quad \{\text{definition of } \mathit{select}\} \\ & \mathit{select}\,(xt, \mathit{left}\ yt)\,n \end{array}$$

In full, the improved program for *select* is:

$$\begin{array}{lcll} \mathit{select}\,(xt, yt)\,n & = & \mathit{index}\ yt\ n, & \text{if } xt = \mathit{Null} \\ & = & \mathit{index}\ xt\ n, & \text{if } yt = \mathit{Null} \\ & = & \mathit{select}\,(xt, \mathit{left}\ yt)\,n, & \text{if } n \le p + q \wedge x \le y \\ & = & \mathit{select}\,(\mathit{left}\ xt, yt)\,n, & \text{if } n \le p + q \wedge y \le x \\ & = & \mathit{select}\,(\mathit{right}\ xt, yt)\,(n-p-1), & \text{if } p + q < n \wedge x \le y \\ & = & \mathit{select}\,(xt, \mathit{right}\ yt)\,(n-q-1), & \text{if } p + q < n \wedge y \le x \\ & & \text{where} \quad p = \mathit{label}\ xt & \\ & & \qquad\quad\ q = \mathit{label}\ yt & \\ & & \qquad\quad\ x = \mathit{value}\ xt & \\ & & \qquad\quad\ y = \mathit{value}\ yt & \end{array}$$

It is clear that *select* has the desired time complexity: at each step, one or other of the two trees is reduced in height by at least one.

4 Merging and indexing

The relationship between merging and indexing on which the fast algorithm for *select* depends is none too obvious I would say. Certainly the proof seems quite tricky.

In what follows it is convenient to imagine that all ascending sequences are extended on the left by $-\infty$ values, and on the right by ∞ values. Thus, we assume that

$$\begin{array}{rcll} xs \mathbin{!!} k & = & -\infty, & \text{if } k < 0 \\ & = & \infty, & \text{if } \mathit{length}\ xs \le k \end{array}$$

The following lemma will be useful.

Lemma 1
Let xs and ys be two ascending sequences and n a natural number. Then there exist unique natural numbers i and j with $i + j = n$ and

$$xs \mathbin{!!} (i-1) \le ys \mathbin{!!} j \quad \text{and} \quad ys \mathbin{!!} (j-1) < xs \mathbin{!!} i$$

Furthermore, $\mathit{merge}\,(xs, ys) \mathbin{!!} n = \mathit{min}\,(xs \mathbin{!!} i, ys \mathbin{!!} j)$.

Proof
The proof is by induction on n. For the base case the unique assignment is $(i,j) = (0,0)$. For the induction step, suppose (i,j) are the values associated with case n. If $xs \mathbin{!!} i \le ys \mathbin{!!} j$, then $(i+1, j)$ is the unique assignment in case $n+1$, while if $ys \mathbin{!!} j < xss \mathbin{!!} i$, the assignment is $(i, j+1)$.

With the definition

$$\begin{array}{lcll} \mathit{merge}\,([\,], ys) & = & ys & \\ \mathit{merge}\,(x : xs, [\,]) & = & x : xs & \\ \mathit{merge}\,(x : xs, y : ys) & = & x : \mathit{merge}\,(xs, y : ys), & \text{if } x \le y \\ & = & y : \mathit{merge}\,(x : xs, ys), & \text{otherwise} \end{array}$$

of *merge* it is easy to show, with i and j as given above, that

$$\mathit{drop}\ n\,(\mathit{merge}\,(xs, ys)) \quad = \quad \mathit{merge}\,(\mathit{drop}\ i\ xs, \mathit{drop}\ j\ ys)$$

Hence we can argue:

$$\begin{array}{cl} & \mathit{merge}\,(xs, ys) \mathbin{!!} n \\ = & \quad \{\text{since } zs \mathbin{!!} k = \mathit{head}\,(\mathit{drop}\ k\ zs)\} \\ & \mathit{head}\,(\mathit{drop}\ n\,(\mathit{merge}\,(xs, ys))) \\ = & \quad \{\text{above}\} \\ & \mathit{head}\,(\mathit{merge}\,(\mathit{drop}\ i\ xs, \mathit{drop}\ j\ ys)) \\ = & \quad \{\text{definition of } \mathit{merge}\} \\ & \mathit{min}\,(xs \mathbin{!!} i, ys \mathbin{!!} j) \end{array}$$

□

Since $\mathit{merge}\,(xs, ys) = \mathit{merge}\,(ys, xs)$ the lemma has a dual version in which the roles of xs and ys are interchanged. Thus, for property (1) it is sufficient to show that if $n \le p + q$ and $xs \mathbin{!!} p \le ys \mathbin{!!} q$, then

$$\mathit{merge}\,(xs, ys) \mathbin{!!} n \quad = \quad \mathit{merge}\,(xs, \mathit{take}\ q\ ys) \mathbin{!!} n$$

Let i and j be the numbers associated with xs and ys as specified in the lemma. If

$q < j$, then $i = n - j < n - q \leq p$, and so

$$xs \mathbin{!!} i \leq xs \mathbin{!!} p \leq ys \mathbin{!!} q \leq ys \mathbin{!!} (j-1) \tag{1}$$

This contradicts the definition of i and j, so $j \leq q$. Furthermore, since

$$\begin{aligned} xs \mathbin{!!} (i-1) \leq ys \mathbin{!!} j &\leq (take\ q\ ys) \mathbin{!!} j \\ (take\ q\ ys) \mathbin{!!} (j-1) = ys \mathbin{!!} (j-1) &< xs \mathbin{!!} i \end{aligned}$$

the numbers i and j are also the numbers associated with xs and $take\ q\ ys$.

We now need a case analysis. In the case $j = q$ we have $i \leq p$, and so

$$\begin{array}{ll} & merge\,(xs, ys) \mathbin{!!} n \\ = & \quad \{\text{lemma, and assumption } j = q\} \\ & min\,(xs \mathbin{!!} i, ys \mathbin{!!} q) \\ = & \quad \{\text{since } xs \mathbin{!!} i \leq xs \mathbin{!!} p \leq ys \mathbin{!!} q\} \\ & xs \mathbin{!!} i \\ = & \quad \{\text{since } (take\ q\ ys) \mathbin{!!} q = \infty\} \\ & min\,(xs \mathbin{!!} i, (take\ q\ ys) \mathbin{!!} q) \\ = & \quad \{\text{lemma}\} \\ & merge\,(xs, take\ q\ ys) \mathbin{!!} n \end{array}$$

In the case $j < q$ we reason

$$\begin{array}{ll} & merge\,(xs, ys) \mathbin{!!} n \\ = & \quad \{\text{lemma}\} \\ & min\,(xs \mathbin{!!} i, ys \mathbin{!!} j) \\ = & \quad \{\text{assumption } j < q\} \\ & min\,(xs \mathbin{!!} i, (take\ q\ ys) \mathbin{!!} j) \\ = & \quad \{\text{lemma}\} \\ & merge\,(xs, take\ q\ ys) \mathbin{!!} n \end{array}$$

For property (2) it is sufficient to prove that if $p + q < n$ and $xs \mathbin{!!} p \leq ys \mathbin{!!} q$, then

$$merge\,(xs, ys) \mathbin{!!} n \quad = \quad merge\,(drop\,(p+1)\,xs, ys) \mathbin{!!} (n-p-1)$$

Again, let i and j be the numbers associated with xs and ys. If $i \leq p$, then $j = n - i \geq n - p > q$ and (1) holds, a contradiction. Thus $i > p$. Since

$$\begin{aligned} (drop\,(p+1)\,xs) \mathbin{!!} (i-p-2) \leq xs \mathbin{!!} (i-1) &\leq ys \mathbin{!!} j \\ ys \mathbin{!!} (j-1) < xs \mathbin{!!} i &= (drop\,(p+1)\,xs) \mathbin{!!} (i-p-1) \end{aligned}$$

the unique numbers associated with the sequences $drop\,(p+1)\,xs$ and ys are $i-p-1$ and j. Hence

$$\begin{array}{ll} & merge\,(xs, ys) \mathbin{!!} n \\ = & \quad \{\text{lemma}\} \end{array}$$

https://doi.org/10.1017/S0956796897002736 Published online by Cambridge University Press

$$
\begin{array}{ll}
 & min\,(xs \mathbin{!!} i, ys \mathbin{!!} j) \\
= & \quad \{\text{since } i > p\} \\
 & min\,((drop\,(p+1)\,xs) \mathbin{!!} (i-p-1), ys \mathbin{!!} j) \\
= & \quad \{\text{lemma}\} \\
 & merge\,(drop\,(p+1)\,xs, ys) \mathbin{!!} (n-p-1)
\end{array}
$$

This concludes the proof.

5 Postscript

The problem treated in this pearl arose out of a tutorial exercise on divide and conquer set by my colleague, Bill McColl. Students were asked for an algorithm to find the median element of a set represented by two sorted lists, each of length $n > 0$, in $O(\log n)$ steps. Equivalently, the problem is to compute $merge\,(xs, ys) \mathbin{!!} n$ in $O(\log n)$ steps, assuming that list indexing takes constant time and xs and ys are both strictly increasing and have no elements in common. This variation is left as an exercise. Another variation, also left as an exercise, is to compute the same expression in the same time, assuming constant-time list indexing and that xs and ys are infinite ascending lists (not necessarily increasing, nor necessarily disjoint).

Acknowledgement

I would like to thank two anonymous referees for pointing out errors in previous drafts of this pearl.