Recitation 7 – *Sets, Tables and Exam I Debrief*

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2015)

February 24th, 2015

1 Announcements

- How did the exam go?
- *BabbleLab* will be released soon!
- Questions from lecture?

2 Sets and Tables

Now that you're comfortable with sequences, we'll be introducing new ADT's for you to learn. The Set ADT behaves just like a mathematical set – it's an unordered collection of values. We'll write the pseudocode for sets just like in math: $\{v_1, v_2, \cdots\}$. The Table ADT, formally, is just a set of key-value pairs, $\{(k_1, v_1), (k_2, v_2), \cdots\}$. Since we're treating tables as a separate ADT, we'll write their pseudocode as $\{k_1 \mapsto v_1, k_2 \mapsto v_2, \cdots\}$. With both sets and tables, we support intersection, union, and difference operations with other sets and tables. The cost bounds for sets and tables are shown in the back of this recitation; make sure to look at them later!

It's easy to implement tables as sets. How can you implement sets as tables?

3 Table.Collect

3.1 Cost of Table.collect

One special operation on tables is `Table.collect`. The following code implements `Table.collect`(S).

`Seq.reduce` (`Table.merge` `Seq.append`) $\langle\,\rangle$ $\langle \{k \mapsto \langle v \rangle\} : (k, v) \in S \rangle$

What does this code actually do?

Derive (tight) asymptotic upper bounds on the work and span for the code in terms of $n = |S|$. You can assume the comparison function used by the table takes constant work, and that the Sequence is a array sequence.

3.2 Making a Histogram

Implement a function `histogram` that, given a sequence of integers, returns a table that maps each unique element to the number of times it appears in the sequence. For example

$$\texttt{histogram}(\langle 7, 2, 4, 2, 3, 2, 1, 3 \rangle)$$

would return $\{1 \mapsto 1, 2 \mapsto 3, 3 \mapsto 2, 4 \mapsto 1, 7 \mapsto 1\}$.

```
fun histogram(S : int Seq) :  int T.table =
```

What is the (tight) asymptotic upper bound for work and span for your implementation in terms of $n = |S|$?

3.3 Working for the Registrar

Suppose you are working on Academic Audit. An academic advisor comes to you asking for a table mapping each of her students to a set of the courses they've taken in their time at CMU, and you need to do this fast. What you have right now is a sequence of tuples, where each tuple is a string (the student's name), and a set of integers (the course numbers for their schedule in one semester). Write a function, `bulkAudit`, to do this for her.

```
fun bulkAudit (S : (string * (int Set)) Seq) :  (int Set) T.table =
```

4 Exam

Exam debrief. Any questions from the exam?

Table/Set Operations	*Work*	*Span*
`size`(T) `singleton`(k, v)	$O(1)$	$O(1)$
`filter` f T	$O\left(\sum_{(k,v)\in T} W(f(v))\right)$	$O\left(\lg\|T\| + \max_{(k,v)\in T} S(f(v))\right)$
`map` f T	$O\left(\sum_{(k,v)\in T} W(f(v))\right)$	$O\left(\max_{(k,v)\in T} S(f(v))\right)$
`tabulate` f S	$O\left(\sum_{k\in S} W(f(k))\right)$	$O\left(\max_{k\in S} S(f(k))\right)$
`find` T k `insert` f (k,v) T `delete` k T	$O(\lg\|T\|)$	$O(\lg\|T\|)$
`extract` (T_1, T_2) `merge` f T_1 T_2 `erase` (T_1, T_2)	$O\left(m \lg(\frac{n+m}{m})\right)$	$O\left(\lg(n+m)\right)$
`domain` T `range` T `toSeq` T	$O(\|T\|)$	$O(\lg\|T\|)$
`collect` S `fromSeq` S	$O(\|S\|\lg\|S\|)$	$O(\lg^2\|S\|)$
`intersection` (S_1, S_2) `union` (S_1, S_2) `difference` (S_1, S_2)	$O\left(m \lg(\frac{n+m}{m})\right)$	$O\left(\lg(n+m)\right)$

where $n = \max(|T_1|, |T_2|)$ and $m = \min(|T_1|, |T_2|)$. For `reduce` you can assume the cost is the same as `Seq.reduce f init (range(T))`. In particular `Seq.reduce` defines a balanced tree over the sequence, and `Table.reduce` will also use a balanced tree. For `merge` and `insert` the bounds assume the merging function has constant work.