50001 - Algorithm Analysis and Design - Lecture 9

Oliver Killane

17/11/21

Lecture Recording

Lecture recording is available here

Amortization

The complexity of **tail** is an example of **Amortized analysis**, where operation's wider context are considered when calculating the complexity.

$$xs_0 \overset{op_0}{\rightsquigarrow} xs_1 \overset{op_1}{\rightsquigarrow} xs_2 \overset{op_2}{\rightsquigarrow} xs_3 \overset{op_3}{\rightsquigarrow} \ldots \overset{op_{n-1}}{\rightsquigarrow} xs_n$$

We defined 3 parts:

1. **Cost Function**
   $C_{op_i}(xs_i)$ determines the cost of operation op_i on data xs_i. Estimating how many steps it takes for each operation to execute.
2. **Amortized Cost Function**
   $A_{op_i}(xs_i)$ for each operation op_i on data xs_i.
3. **Size Function**
   $S(xs)$ that calculates the size of data xs

We define these functions with the goal to show that:

$$C_{op_i}(xs_i) \leq A_{op_i}(xs_i) + S(xs_i) - S(xs_{i+1})$$

The cost of the operation is smaller than the amortized cost, plus the difference in size of the data structure before and after the operation.

Once this is shown, we can infer that:

$$\sum_{i=0}^{n-1} C_{op_i}(xs_i) \leq \sum_{i=0}^{n-1} A_{op_i}(xs_i) + S(xs_0) - S(xs_n)$$

Furthermore when $S(xs_0) = 0$ this implies:

$$\sum_{i=0}^{n-1} C_{op_i}(xs_i) \leq \sum_{i=0}^{n-1} A_{op_i}(xs_i) - S(xs_n) \Rightarrow \sum_{i=0}^{n-1} C_{op_i}(xs_i) \leq \sum_{i=0}^{n-1} A_{op_i}(xs_i)$$

This means the cost of the operations is less than the sum of the amortized costs.

For example, if $A_{op_i}(xs) = 1$ then the total cost will be bounded by $O(n)$.

Tail example

$$C_{cons}(xs) = 1 \quad C_{snoc}(xs) = 1 \quad C_{head}(xs) = 1 \quad C_{last}(xs) = 1$$

For tail we can do the following:

(1)	$C_{tail}(Dequeue\ us\ sv) = length\ sv$	(Create a cost function of **tail**.)
(2)	$A_{op}(xs) = 2$	(Create an arbitrary cost function.)
(3)	$S(Dequeue\ us\ sv) = \lvert length\ us - length\ sv \rvert$	(Create a size function for **dequeue**.)
(4)	$Dequeue\ us\ sv$ where $length\ sv = k$	(Worst case where us is a singleton)
(5)	$S(Dequeue\ us\ sv) = k - 1$ $S(Dequeue\ us'\ sv') = 1$	(Size of the next data structure can be at most 1.)
(6)	$C_{tail}(Dequeue\ us\ sv) = k$	(Calculate the worst case cost of **tail**.)
(7)	$k \leq 2 + (k - 1) - 1 = k + 2$	(As this inequality holds, the time complexity of all n instructi

As the time complexity of all n instructions together is $O(n)$, the amortized cost of a single instruction is $O(1)$.

About the size function

We want to balance the size function such that:

- The size function is 0 to start with.
- The size between operations is large enough to prove the inequality.

The size function is arbitrary, if you cannot choose a size function that satisfied the goal inequality, then you're probably making a mistake

Peano Numbers

```haskell
data Peano = Zero | Succ Peano

-- analogous to (:) Cons
incr :: Peano -> Peano
incr = Succ

-- analogous to tail
decr :: Peano -> Peano
decr (Succ n) = n
decr Zero = error "Cannot decrement zero"

-- analogous to (++) concatenate
add :: Peano -> Peano -> Peano
add a Zero = a
add a (Succ b) = Succ (add a b)

-- tail recursive version for extra goodness!
add a b = add (incr a) (decr b)
```

This shows how similar operations of similarly structured data can be.

Binary Numbers

```haskell
data Bit = I | O

-- [LSB, ..., MSB]
type Binary = [Bit]

```

```
incr :: Binary -> Binary
incr [] = [I]
incr (O:bs) = I:bs
incr (I:bs) = O:incr bs
```

we can do amortized analysis on incr:

(1)	$C_{incr}(bs) = t + 1$ where $t = length\ (takeWhile\ (== I)\ bs)$	(Create a cost function.)
(2)	$A_{incr}(bs) = 2$	(Create Amortized Cost)
(3)	$S(bs) = length.filter\ (== I)\ \$\ bs$	(Create size function.)
(4)	Given $bs' = incr\ bs$ we show $C_{incr}(bs) \leq A_{incr}bs + S(bs) - S(bs')$	(Setup inequality)
(5)	$t + 1 \leq 2 + S(bs) - (S(bs) - t + 1)$	(Subsitiute in inequality)
(6)	$t + 1 \leq 1 + t$	(Hence inequality holds)
(7)	$S(start) = 0$	(Start size is zero.)

Hence The sum of C is smaller than the sum of A, as this is over n operations and $\sum A = 2n$, $C_{incr}(bs) \in O(1)$.