S8.2: Purely Functional Data Structures, Amortization, Chapter 5
CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2018

Amortization

Focus on cost of sequence of many operations, not each one at a time.

- for m operations
- bound total cost by $O(m)$
- without requiring that each is $O(1)$
- some can be longer, if others are shorter

Amortized vs actual costs

$$\sum_{i=1}^{m} a_i \geq \sum_{i=1}^{m} t_i$$

- a_i - amortized cost of operation i
- t_1 - actual cost of operation i

But we usually show $\forall j$,

$$\sum_{i=1}^{j} a_i \geq \sum_{i=1}^{j} t_i$$

- difference is *accumulated savings*
- some operations (whose actual costs are greater than amortized costs) are *expensive*, some *cheap*.

Proving amortized bounds

Need to show that expensive operations don't happen too often.

They only occur when we've accumulated enough savings to pay for them.

- think of accumulating credits - Banker's method
- measure "potential" of data - Physicist's method

Banker's method

- Credits are associated with locations in the data.
  They pay for future access.
- $a_i = t_i + c_i - \bar{c}_i$
  - c_i - credits allocated during the operation
  - $\bar{c}_i$ - credits spent
- Credits are allocated before they are spent and can only be spent once.
- $\sum c_i \geq \sum \bar{c}_i$

Physicist's method

- Potential function Φ over data.
- Initially 0, always non-negative.
- Sets a lower bound on accumulated savings.
- Let d_i be result of i^{th} operation, input for $(i+1)^{th}$
- $a_i = t_i + \Phi(d_i) - \Phi(d_{i-1})$

Accumulated actual costs

Recall, $a_i = t_i + \Phi(d_i) - \Phi(d_{i-1})$

$$\begin{aligned}\sum_{i=1}^{j} t_i &= \sum_{i=1}^{j}(a_i + \Phi(d_{i-1}) - \Phi(d_i)) \\ &\sum_{i=1}^{j} a_i + \sum_{i=1}^{j}(\Phi(d_{i-1}) - \Phi(d_i)) \\ &\sum_{i=1}^{j} a_i + \Phi(d_0) - \Phi(d_j)\end{aligned}$$

- Banker's and Physicist's methods are similar and we can convert between them.

Queues

- See Figure 5.1
- Often represented as a “front” and “rear” pair of lists.
- Elements 1..6 represented as f = [1,2,3], r = [6,5,4]
- Thus, **type** α Queue = α list $\times$ α list
- `fun head (x :: f, r) = x`
- `fun tail (x :: f, r) = (f, r)`
- `fun snoc = ((f, r), x) = (f, x :: r)`
- How do we migrate values from r to f?

Migration

- Reverse r, to become f, whenever f would be empty.
- Invariant: f is empty only if r is empty.
- Otherwise, accessing first element requires getting last element from r, an $O(n)$ operation.
-
```
fun snoc ( ([], _), x ) = ( [x], [] )
  | snoc ( (f, r), x ) = ( f, x :: r )

fun tail ( [x], r ) = ( rev r, [] )
  | tail ( x :: f, r ) = ( f, r )
```
- See Figure 5.2, `tail` is potentially $O(n)$ actual cost

Amortized cost - Banker's method

- credit invariant: every rear list element has a single credit
- every `snoc` (into non-empty list) takes 1 actual step,
  allocates 1 credit for new element
  amortized cost is 2
- every `tail` that doesn't reverse take one actual step,
  allocates 0 credits, uses 0 credits
- every `tail` that does reverse, take $n + 1$ actual steps
  it spends n credits for the elements in the list
  amortized cost: $m + 1 - m = 1$

Amortized costs - Physicist's method

In pairs, work out the argument using the Physicist's method.

That is, how does each operation change Φ? Why is it always positive? Why does `tail` have an amortized cost of $O(1)$?

Recall,

- Potential function Φ over data.
- Initially 0, always non-negative.
- Sets a lower bound on accumulated savings.
- Let d_i be result of i^{th} operation, input for $(i + 1)^{th}$
- $a_i = t_i + \Phi(d_i) - \Phi(d_{i-1})$

Binomial Heaps, revisited

- Recall Figure 3.4
- `insert` has $O(1)$ amortized cost
- How?
- Using Physicist's method, $\Phi(h) =$ number of trees
- `insert` takes $k + 1$ step, with k calls to `link`
- How many trees after an `insert` with k calls to `link`?
- Then, what is the change in potential?
- What is the amortized cost?

That is ...

- initially, 0 trees, $\Phi(h) = 0$
- a call to `insert` takes $k + 1$ steps
  with k calls to link
- each call to link reduces the number of trees by 1
- so, if we start with t trees
  we are inserting 1 and doing some linking
  So we get $t - k + 1$ trees after the insertion
- So change in potential $(\Phi(h_i) - \Phi(h_{i-1}))$ is
  $(t - k + 1) - t = -k + 1 = 1 - k$
- So amortized cost is
  $(k + 1)$ (the actual cost) $+ (1 - k)$ (the diff of potentials)
  This is 2.

Ephemeral vs. Persistent

- So, why do we lose $O(1)$ amortized costs when queues are used persistently?
- Consider adding n elements to an empty queue? Call it `q1`.
  What is in the front list? The rear list?
-

```
let q2 = tail q1
let q3 = tail q2
let q4 = tail q3
...
```

  n calls to `tail`, each on result of the previous.
  What behavior do we see?

But ...

- What about this?


```
let q2 = tail q1
let q3 = tail q1
let q4 = tail q1
...
```

  n calls to `tail`, each on **the original** `q1`.
- What is the cost of constructing `q1` and these n calls to `tail`?
- What went wrong?

Banker's method violation

- We stored up $n-1$ credits in the rear list and then spent them on the first call to `tail`.
- We then spent them again on the second call.
- We can't spend credits more than once.

Physicist's method violation

- This one assumes the result of an operation is only used as input to the following operation.
- But here we used the result (`q1`) many times.