Evaluation Example 2

Let us take the first two numbers from the string. This is aided by:

Evaluation

The evaluation:

\[ \text{take } n: (x:xs) \rightarrow \begin{cases} \text{take } (n-1) \times x & \text{if } n > 0 \\ \emptyset & \text{if } n = 0 \end{cases} \]

First case:

Let \( n = \sqrt{x} \). To square a thousand numbers and then just keep the

Evaluation Example 1

Reuse breaks amortization here. There are only 2n operations, but the time to amortize over it is \( O(m + n) \).

1. \( O(n) \text{ time each) } \)
2. \( \sum_{i=1}^{n} i \text{ time each) } \)
3. \( \sum_{i=1}^{n} 2i \text{ time each) } \)

The cost amortization is persistent, the cost amortization will break.
Now we can consider queues with lazy evaluation in mind:

If we consider queues with lazy evaluation in mind:

... where partial results are possible

- one at a time, pay as you go

- fold to map, take explicit incremental laziness

- fold to map, take explicit incremental laziness

- all or nothing, fail fast

- fold to map, take explicit monothetic laziness

Let us summarize what we learned from the examples:

Lazy Evaluation Summary

Evaluation Example 4

We sometimes want to reduce + with reverse, which in turn is defined as

(x + y) = foldr (+) y x

We just ask for the first few elements:

There is no deep recursion, and the actual combined list is not constructed.

Now we know maybe it's a good thing do [1,2,3] ++ [4,5,6] and just

Recall that ++ and foldr are not tail recursive. In fact we often define:

Evaluation Example 3
Performance of Lazy Queue

Quickly of unvisited graph's ++, but note that its depth is logarithmic and decreases worst thing that could happen is a lot of moves that lead to a tree.

Note: (++) is incremental, so it adds only O(1) cost to each T. The
exceeded. So we have m + I operations to share the O(m) cost. Good.

If we do not store a reversal will be created but let unexecuted due
as if there is at least another T's before the O(m) reversal.

Performance of Lazy Queue

Data structure: a pop, Int [a]. Int [a]

Lazy Queue for Persistence

Lazy Queue for Persistence