15-210
Parallel and Sequential Algorithms and Data Structures

Lecture 5

Data Abstraction and Sequences

Synopsis

- Abstractions and Implementations
  - Meldable Priority Queues
- The Sequence ADT
- The `scan` operation
- Introduction to contraction

Abstractions and Implementations

	Abstraction	**Implementation**
Functions	Problem	Algorithm
Data	Abstract Data Type	Data Structure

Meldable Priority Queues

- Priority Queues
  - Insert an item – `insert`
  - Return and delete the item with the minimum priority – `deleteMin`
- Meldable Priority Queue
  - Join two priority queues into one – `meld`

Meldable Priority Queues

- $\mathbb{S}$ is a totally ordered set (integers, strings, reals, ...).
- $\mathbb{T}$ is a type representing *subsets* of $\mathbb{S}$.

$$
\begin{array}{lcll}
\texttt{empty} & : & \mathbb{T} & = \{\} \\
\texttt{insert}(S, e) & : & \mathbb{T} \times \mathbb{S} \to \mathbb{T} & = S \cup \{e\} \\
\texttt{deleteMin}(S) & : & \mathbb{T} \to \mathbb{T} \times (\mathbb{S} \cup \{\bot\}) & = \begin{cases} (S, \bot) & S = \{\} \\ (S \setminus \{\min S\}, \min S) & \textit{otherwise} \end{cases} \\
\texttt{meld}(S_1, S_2) & : & \mathbb{T} \times \mathbb{T} \to \mathbb{T} & = S_1 \cup S_2
\end{array}
$$

MPQ Definition in SML

```
signature MPQ
 sig
   struct S : ORD
   type t
   val empty : t
   val insert : t * S.t -> t
   val deleteMin : t -> t * S.t option
   val meld : t * t -> t
 end
```

- No semantics, only the types.

MPQ: Cost Specifications

- Implementation 1:

Operation	Work
`insert`(S, e)	$O(\|S\|)$
`deleteMin`(S)	$O(1)$
`meld`(S_1, S_2)	$O(\|S_1\| + \|S_2\|)$

- What is the underlying data structure? Sorted Array
- `meld` is actually an array merge.

MPQ: Cost Specifications

- Implementation 2:

Operation	Work
`insert`(S, e)	$O(\log \lvert S \rvert)$
`deleteMin`(S)	$O(\log \lvert S \rvert)$
`meld`(S_1, S_2)	$O(\lvert S_1 \rvert + \lvert S_2 \rvert)$

- What is the underlying data structure? Heaps

MPQ: Cost Specifications

- Implementation 3:

Operation	Work
`insert`(S, e)	$O(\log \lvert S \rvert)$
`deleteMin`(S)	$O(\log \lvert S \rvert)$
`meld`(S_1, S_2)	$O(\log(\lvert S_1 \rvert + \lvert S_2 \rvert))$

- Later!

Abstractions and Implementations

- The Abstract Data Type
  - Functionality
  - Correctness
- The Cost Specification
  - Multiple Cost Specifications
  - We only need these to do cost analysis.
- Underlying Data Structure
  - Multiple Data Structures

The Sequence ADT - Some Basics

- A *relation* is a set of ordered pairs.
  - First from set A, second from set B
- A relation $\rho \subseteq A \times B$.
- A *function* is a relation ρ, where for every $a \in A$ there is only one b such that $(a, b) \in \rho$.
- A *sequence* is a function where $A = \{0, \ldots, n-1\}$ for some $n \in \mathbb{N}$.

The Sequence ADT – Functionality

- A *sequence* is a type $\mathbb{S}_\alpha$ representing functions from $\{0, \ldots, n-1\}$ to α.

`empty`	:	$\mathbb{S}_\alpha$	=	$\{\}$
`length`(A)	:	$\mathbb{S}_\alpha \to \mathbb{N}$	=	$\lvert A \rvert$
`singleton`(v)	:	$\alpha \to \mathbb{S}_\alpha$	=	$\{(0, v)\}$
`nth`(A, i)	:	$\mathbb{S}_\alpha \to \alpha$	=	$A(i)$
`map`(f, A)	:	$(\alpha \to \beta) \times \mathbb{S}_\alpha \to \mathbb{S}_\beta$	=	$\{(i, f(v)) : (i, v) \in A\}$
`tabulate`(f, n)	:	$(\mathbb{N} \to \alpha) \times \mathbb{N} \to \mathbb{S}_\alpha$	=	$\{(i, f(i)) : i \in \{0, \ldots, n-1\}\}$
`take`(A, n)	:	$\mathbb{S}_\alpha \times \mathbb{N} \to \mathbb{S}_\alpha$	=	$\{(i, v) \in A \mid i < n\}$
`drop`(A, n)	:	$\mathbb{S}_\alpha \times \mathbb{N} \to \mathbb{S}_\alpha$	=	$\{(i - n, v) : (i, v) \in A \mid i \geq n\}$
`append`(A, B)	:	$\mathbb{S}_\alpha \times \mathbb{S}_\alpha \to \mathbb{S}_\alpha$	=	$A \cup \{(i + \lvert A \rvert, v) : (i, v) \in B\}$

The Sequence ADT – Cost Specs

	ArraySequence	
	Work	*Span*
`length`(T)	$O(1)$	$O(1)$
`nth`(T)	$O(1)$	$O(1)$
`append`(S_1, S_2)	$O(\|S_1\| + \|S_2\|)$	$O(1)$

The Sequence ADT – Cost Specs

ArraySequence

	Work	*Span*
`tabulate` $f\ n$	$O\left(\sum_{i=0}^{n} W(f(i))\right)$	$O\left(\max_{i=0}^{n} S(f(i))\right)$
`map` $f\ S$	$O\left(\sum_{s \in S} W(f(s))\right)$	$O\left(\max_{s \in S} S(f(s))\right)$

The Sequence ADT – Cost Specifications

	TreeSequence	
	Work	*Span*
`length`(T)	$O(1)$	$O(1)$
`nth`(T)	$O(\log n)$	$O(\log n)$
`append`(S_1, S_2)	$O(\log(\|S_1\| + \|S_2\|))$	$O(\log(\|S_1\| + \|S_2\|))$

The Sequence ADT – Cost Specifications

TreeSequence

	Work	*Span*
`tabulate` f n	$O\left(\sum_{i=0}^{n} W(f(i))\right)$	$O\left(\log n + \max_{i=0}^{n} S(f(i))\right)$
`map` f S	$O\left(\sum_{s \in S} W(f(s))\right)$	$O\left(\log \lvert S \rvert + \max_{s \in S} S(f(s))\right)$

Some Notational Conventions

S_i	The i^{th} element of sequence S
$\|S\|$	The length of sequence S
$\langle\ \rangle$	The empty sequence
$\langle\, v \,\rangle$	A sequence with a single element v
$\langle\, i, \ldots, j \,\rangle$	A sequence of integers starting at i and ending at $j \geq i$.
$\langle\, e : p \in S \,\rangle$	**Map** the expression e to each element p of sequence S. The same as "`map` (**fn** $p \Rightarrow e$) S" in ML.
$\langle\, p \in S \mid e \,\rangle$	**Filter** out the elements p in S that satisfy the predicate e. The same as "`filter` (**fn** $p \Rightarrow e$) S" in ML.

- More examples are given in the "Syntax and Costs" document.

The Scan Operation

- Related to `reduce`.

$$\texttt{scan}\ f\ I\ S : (\alpha \times \alpha \to \alpha) \to \alpha \to \alpha\ \texttt{seq}$$

$$\to (\alpha\ \texttt{seq} \times \alpha)$$

- I is the identity value
- f is an (associative) function
- S is a sequence
- Produces $\langle I, f(I, S_0), f(f(I, S_0), S_1), \ldots \rangle$ and $\texttt{reduce}\ f\ I\ S$
  - $\texttt{scan} + 0 \langle 2, 1, 4, 6 \rangle = (\langle 0, 2, 3, 7 \rangle, 13)$

The Scan Operation

- `scan` computes prefix sums.

```
1  fun scan f I S =
2      (⟨ reduce f I (take(S, i)) : i ∈ ⟨0, ... n − 1⟩⟩,
3       reduce f I S)
```

- S has n elements
- Apply `reduce` to each prefix of S of i elements, $0 \leq i \leq n-1$
  - Gives you the α `seq` part
- Apply `reduce` to S
  - Gives you the α part
- So you get $(\alpha\ \texttt{seq} \to \alpha)$

The Scan Operation

$$
\begin{aligned}
\mathtt{scan}\ +\ 0\ \langle 2,1,3\rangle &= (\langle\ \mathtt{reduce}\ +\ 0\ \langle\,\rangle,\\
&\qquad \mathtt{reduce}\ +\ 0\ \langle 2\rangle,\\
&\qquad \mathtt{reduce}\ +\ 0\ \langle 2,1\rangle\ \rangle\\
&\qquad \mathtt{reduce}\ +\ 0\ \langle 2,1,3\rangle)\\
&= (\langle 0,2,3\rangle, 6)
\end{aligned}
$$

- This is obviously not efficient!
- We will see how to do this with

$$
\begin{aligned}
W(\mathtt{scan}\ f\ I\ S) &= O(|S|)\\
S(\mathtt{scan}\ f\ I\ S) &= O(\log|S|)
\end{aligned}
$$

The Inclusive Scan Operation

- `reduce` all prefixes ending at position i, $0 \leq i \leq n-1$

$$\mathit{scanI}\ +\ 0\ \langle 2,1,3\rangle\ =\ \langle 2,3,6\rangle$$

Using Scan in the MCSS Prob.

The Maximum Contiguous Subsequence Sum Problem

- Given a sequence of numbers $S = \langle s_1, \ldots, s_n \rangle$,
- Find

$$\text{mcss}(S) = \max_{1 \le i \le j \le n} \left\{ \sum_{k=i}^{j} s_k \right\}$$

- $S = \langle 0, -1, \mathbf{2}, -\mathbf{1}, \mathbf{4}, -1, 0 \rangle$, $mcss(S) = 5$

Using Scan in the MCSS Prob.

- Consider $S = \langle 1, -2, 3, -1, 2, -3 \rangle$
- Let $X = \texttt{scanI}\ +\ 0\ S = \langle 1, -1, 2, 1, 3, 0 \rangle$
- What is $X_j - X_i$ for $j > i$?
  - $\sum_{k=i+1}^{j} S_k$
  - $X_4 - X_0 = 3 - 1 = 2$

Using Scan in the MCSS Prob.

- Define R_j as the maximum sum that starts at some i and ends at $j \geq i$.

$$
\begin{aligned}
R_j &= \max_{i=0}^{j} \sum_{k=i}^{j} S_k \\
&= \max_{i=0}^{j} (X_j - X_{i-1}) \\
&= X_j + \max_{i=0}^{j} (-X_{i-1}) \\
&= X_j + \max_{i=0}^{j-1} (-X_i) = X_j - \min_{i=0}^{j-1} X_i \text{ (Why?)}
\end{aligned}
$$

Using Scan in the MCSS Prob.

$$R_j = X_j - \min_{i=0}^{j-1} X_i$$

- You need X_j and the minimum previous X_i, $i < j$
  - can be done by a minimum `scan`

$$(M, \_) = \texttt{scan}\ \min\ 0\ X = (\langle\, 0, 0, -1, -1, -1, -1 \,\rangle, -1)$$

$$R = \langle\, X_j - M_j : 0 \le j < |S| \,\rangle = \langle\, 1, -1, 3, 2, 4, 1 \,\rangle$$

Let's Recap

- Given $S = \langle 1, -2, 3, -1, 2, -3 \rangle$
- We computed X with a $+$ *scanI*.
  - $X = \langle 1, -1, 2, 1, 3, 0 \rangle$
- We computed M with a `min` *scan*
  - $M = \langle 0, 0, -1, -1, -1, -1 \rangle$
- We computed $R = \langle X_j - M_j : 0 \leq j < |S| \rangle$
  - $R = \langle 1, -1, 3, 2, 4, 1 \rangle$
- A final max `reduce` on R gives us the MCSS, 4.

Using Scan in the MCSS Prob.

```
1  fun MCSS(S) =
2  let
3      X = scanI + 0 S
4      (M, _) = scan min 0 X
5  in
6      max⟨ X_j − M_j : 0 ≤ j < |S| ⟩
7  end
```

- Work? $O(n)$
- Span? $O(\log n)$

Copy Scan

- Scan can also be used to pass information along a sequence.

$$\langle \mathit{NONE}, \mathit{SOME}(7), \mathit{NONE}, \mathit{NONE}, \mathit{SOME}(3), \mathit{NONE} \rangle$$

$$\downarrow$$

$$\langle \mathit{NONE}, \mathit{NONE}, \mathit{SOME}(7), \mathit{SOME}(7), \mathit{SOME}(7), \mathit{SOME}(3) \rangle$$

- Each element receives the nearest previous `SOME()` value.
- Easy to do sequentially with *`iter`*.

Copy Scan

- Can we do this with `scan`?
- $f : \alpha$ *option* $\times \alpha$ *option* $\rightarrow \alpha$ *option*

```
1   fun copy(a, b) =
2       case b of
3           SOME( _ ) ⇒ b
4         | NONE ⇒ a
```

- Passes its right argument if it is `SOME`, else passes its left argument.
- How do you show `copy` is associative.

Implementing Scan – Contraction

- `scan` looks inherently sequential.
  - Naive implementation needs $O(n^2)$ work.
  - Slightly clever sequential implementation needs $O(n)$ work.
  - Divide an Conquer approaches do not break the sequentiality. (Why?)
- Contraction
  1. Construct a much smaller instance of the problem
  2. Solve the smaller instance recursively
  3. Construct solution to the original instance.

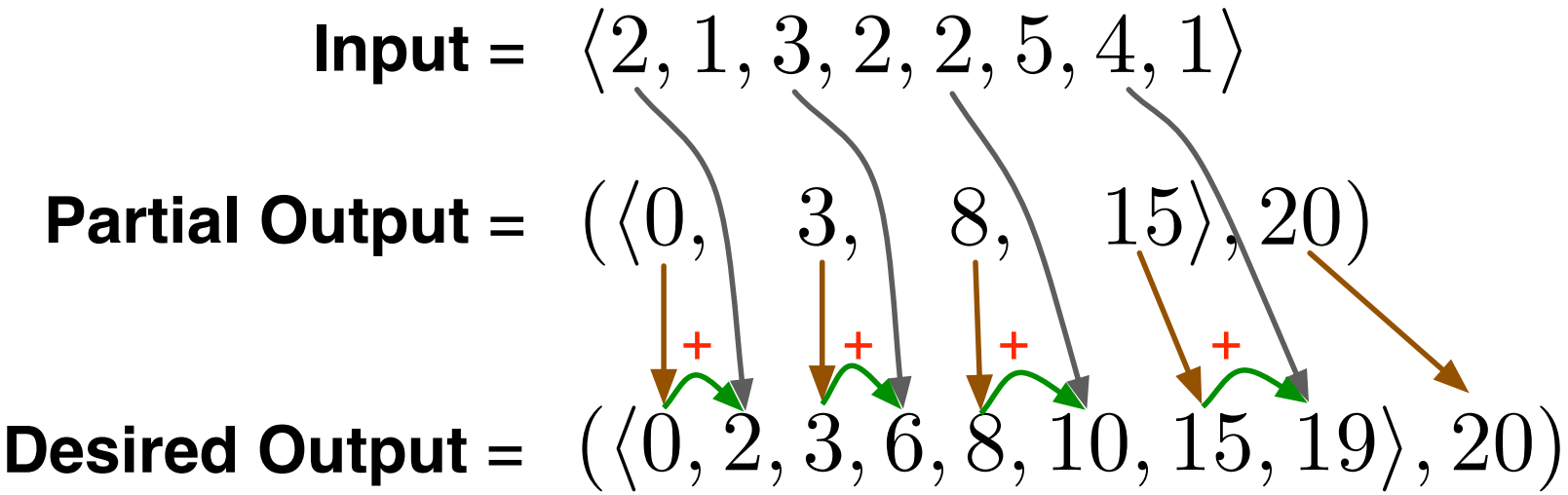
Implementing Reduce with Contraction

- Given $\langle 2, 1, 3, 2, 2, 5, 4, 1 \rangle$
- Apply $+$ pairwise and (in parallel) to get $\langle 3, 5, 7, 5 \rangle$
  - This is the contracted instance!
- Apply $+$ pairwise to get $\langle 8, 12 \rangle$
- Finally apply $+$ pairwise to get $\langle 20 \rangle$
- The 3^{rd} step of the contraction does nothing in this case.

Implementing Scan with Contraction

- Given $S = \langle 2, 1, 3, 2, 2, 5, 4, 1 \rangle$
  - $scan\ +\ 0\ S = (\langle 0, 2, 3, 6, 8, 10, 15, 19 \rangle, 20)$
- First do pairwise $+$ on S to get $\langle 3, 5, 7, 5 \rangle$
- Now (recursively) do scan on this to get $(\langle 0, 3, 8, 15 \rangle, 20)$
  - What is the relation to the final scan?
- We have every other element of the final scan!
- How do we fill in the rest?

Implementing Scan with Contraction



Implementing Scan with Contraction

```
1  % implements: the Scan problem on sequences that have a power of 2 length
2  fun scanPow2 f i s =
3     case |s| of
4        0 ⇒ (⟨⟩, i)
5      | 1 ⇒ (⟨i⟩, s[0])
6      | n ⇒
7          let
8              s' = ⟨f(s[2i], s[2i + 1]) : 0 ≤ i < n/2⟩
9              (r, t) = scanPow2 f i s'
10         in
11             (⟨p_i : 0 ≤ i < n⟩, t),  where p_i = { r[i/2]          if even(i)
                                                    { f(r[i/2], s[i − 1])  otherwise.
12         end
```

- General case is in the course notes.

Summary

- Abstractions and Implementations
  - Meldable Priority Queues
- The Sequence ADT
- The *scan* operation
- Introduction to contraction
- Implementing `scan` with contraction.