

15-210: Parallel and Sequential Data Structures and Algorithms

Syntax and Costs for Sequences, Sets and Tables

1 Pseudocode Syntax

The pseudocode we use in the class will use the following notation for operations on sequences, sets and tables. In the translations e, e_1, e_2 represent expressions, and p, p_1, p_2, k, k_1, k_2 represent patterns. The syntax described here is not meant to be complete, but hopefully sufficient to figure out any missing rules. Warning: Since we have been refining the notation as we go, this notation is not completely consistent across the lectures.

Sequences

S_i	<code>nth S i</code>
$ S $	<code>length(S)</code>
$\langle \rangle$	<code>empty()</code>
$\langle v \rangle$	<code>singleton(v)</code>
$\langle i, \dots, j \rangle$	<code>tabulate (fn k \Rightarrow i + k) (j - i + 1)</code>
$\langle e : p \in S \rangle$	<code>map (fn p \Rightarrow e) S</code>
$\langle e : i \in \langle 0, \dots, n-1 \rangle \rangle$	<code>tabulate (fn i \Rightarrow e) n</code>
$\langle p \in S \mid e \rangle$	<code>filter (fn p \Rightarrow e) S</code>
$\langle e_1 : p \in S \mid e_2 \rangle$	<code>map (fn p \Rightarrow e₁) (filter (fn p \Rightarrow e₂) S)</code>
$\langle e : p_1 \in S_1, p_2 \in S_2 \rangle$	<code>flatten(map (fn p₁ \Rightarrow map (fn p₂ \Rightarrow e) S₂) S₁)</code>
$\langle e_1 : p_1 \in S_1, p_2 \in S_2 \mid e_2 \rangle$	<code>flatten(map (fn p₁ \Rightarrow $\langle e_1 : p_2 \in S_2 \mid e_2 \rangle$) S₁)</code>
$\sum_{p \in S} e$	<code>reduce add 0 (map (fn p \Rightarrow e) S)</code>
$\sum_{i=k}^n e$	<code>reduce add 0 (map (fn i \Rightarrow e) $\langle k, \dots, n \rangle$)</code>
$\operatorname{argmax}_{p \in S}(e)$	<code>argmax compare (map (fn p \Rightarrow e) S)</code>

The meaning of add, 0, and compare in the reduce and argmax will depend on the type. The \sum can be replaced with min, max, \cup and \cap with the presumed meanings. The function $\operatorname{argmax} f S : (\alpha \times \alpha \rightarrow \text{order}) \rightarrow (\alpha \text{ seq}) \rightarrow \text{int}$ returns the index in S which has the maximum value with respect to the order defined by the function f . $\operatorname{argmin}_{p \in S} e$ can be defined by reversing the order of compare.

Sets

$ S $	<code>size(S)</code>
$\{\}$	<code>empty</code>
$\{v\}$	<code>singleton(v)</code>
$\{v \in S \mid e\}$	<code>filter (fn v => e) S</code>
$S_1 \cup S_2$	<code>union(S₁, S₂)</code>
$S_1 \cap S_2$	<code>intersection(S₁, S₂)</code>
$S_1 \setminus S_2$	<code>different(S₁, S₂)</code>
$\sum_{k \in S} e$	<code>reduce add 0 (Table.tabulate (fn k => e) S)</code>

Tables

$ T $	<code>size(T)</code>
$\{\}$	<code>empty()</code>
$\{k \mapsto v\}$	<code>singleton(k, v)</code>
$\{e : v \in T\}$	<code>map (fn v => e) T</code>
$\{k \mapsto e : (k \mapsto v) \in T\}$	<code>mapk (fn (k, v) => e) T</code>
$\{k \mapsto e : k \in S\}$	<code>tabulate (fn k => e) S</code>
$\{v \in T \mid e\}$	<code>filter (fn v => e) T</code>
$\{(k \mapsto v) \in T \mid e\}$	<code>filterk (fn (k, v) => e) T</code>
$\{e_1 : v \in T \mid e_2\}$	<code>map (fn v => e₁) (filter (fn v => e₂) T)</code>
$\{k : (k \mapsto _) \in T\}$	<code>domain(T)</code>
$\{v : (_ \mapsto v) \in T\}$	<code>range(T)</code>
$T_1 \cup T_2$	<code>merge (fn (v₁, v₂) => v₂) (T₁, T₂)</code>
$T \cap S$	<code>extract(T, S)</code>
$T \setminus S$	<code>erase(T, S)</code>
$\sum_{v \in T} e$	<code>reduce add 0 (map (fn v => e) T)</code>
$\sum_{(k \mapsto v) \in T} e$	<code>reduce add 0 (mapk (fn (k, v) => e) T)</code>
$\operatorname{argmax}_{(k \mapsto v) \in T} (e)$	<code>argmax max (mapk (fn (k, v) => e) T)</code>

2 Function Costs

ArraySequence	Work	Span
length(T)		
singleton(v)	$O(1)$	$O(1)$
nth S i		
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	$O\left(\sum_{s \in S} W(f(s))\right)$	$O\left(\max_{s \in S} S(f(s))\right)$
filter f S	$O\left(\sum_{s \in S} W(f(s))\right)$	$O\left(\log S + \max_{s \in S} S(f(s))\right)$
reduce f i S	$O\left(S + \sum_{f(a,b) \in \mathcal{O}_r(f,i,S)} W(f(a,b))\right)$	$O\left(\log S \max_{f(a,b) \in \mathcal{O}_r(f,i,S)} S(f(a,b))\right)$
scan f i S	$O\left(S + \sum_{f(a,b) \in \mathcal{O}_s(f,i,S)} W(f(a,b))\right)$	$O\left(\log S \max_{f(a,b) \in \mathcal{O}_s(f,i,S)} S(f(a,b))\right)$
showt S	$O(S)$	$O(1)$
hidet NODE(l, r)	$O(l + r)$	$O(1)$
append(S_1, S_2)	$O(S_1 + S_2)$	$O(1)$
flatten(S)	$O(S + \sum_{s \in S} s)$	$O(\log S)$
partition I S	$O(I + S)$	$O(1)$
inject I S	$O(S)$	$O(1)$
merge f $ S_1 $ $ S_2 $	$O(S_1 + S_2)$	$O(\log(S_1 + S_2))$
sort f S	$O(S \log S)$	$O(\log^2 S)$
collect f S	$O(S \log S)$	$O(\log^2 S)$
Single Threaded Array Sequence		
nth S i		
update (i, v) S	$O(1)$	$O(1)$
inject I S	$O(I)$	$O(1)$
fromSeq S toSeq S	$O(S)$	$O(1)$

For reduce, $\mathcal{O}_r(f, i, S)$ represents the set of applications of f as defined in the documentation. For scan, $\mathcal{O}_s(f, i, S)$ represents the applications of f defined by the implementation of scan in the lecture notes. For merge, sort, and collect the costs assume that the work and span of the application of f is constant.

Tree Sets and Tables	Work	Span
size(T)	$O(1)$	$O(1)$
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	$O(\lg T)$	$O(\lg T)$
delete k T		
merge f (T_1, T_2)		
extract (T, S)	$O(m \lg(\frac{n+m}{m}))$	$O(\lg(n+m))$
erase (T, S)		
domain T		
range T	$O(T)$	$O(\lg T)$
toSeq T		
collect S	$O(S \lg S)$	$O(\lg^2 S)$
fromSeq S		
union (S_1, S_2)		
intersection (S_1, S_2)	$O(m \lg(\frac{n+m}{m}))$	$O(\lg(n+m))$
difference (S_1, S_2)		

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.