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 might not be completely consistent across the lectures.

Sequences

S_i	`nth` S i
$\|S\|$	`length`(S)
$\langle\,\rangle$	`empty`$()$
$\langle\, v\,\rangle$	`singleton`(v)
$\langle\, i,\ldots,j\,\rangle$	`tabulate` $(\textbf{fn}\, k \Rightarrow i+k)$ $(j-i+1)$
$\langle\, e : p \in S\,\rangle$	`map` $(\textbf{fn}\, p \Rightarrow e)$ S
$\langle\, e : i \in \langle 0,\ldots,n-1\rangle\,\rangle$	`tabulate` $(\textbf{fn}\, i \Rightarrow e)$ n
$\langle\, p \in S \mid e\,\rangle$	`filter` $(\textbf{fn}\, p \Rightarrow e)$ S
$\langle\, e_1 : p \in S \mid e_2\,\rangle$	`map` $(\textbf{fn}\, p \Rightarrow e_1)$ (`filter` $(\textbf{fn}\, p \Rightarrow e_2)$ S)
$\langle\, e : p_1 \in S_1, p_2 \in S_2\,\rangle$	`flatten(map` $(\textbf{fn}\, p_1 \Rightarrow$ `map` $(\textbf{fn}\, p_2 \Rightarrow e)$ $S_2)$ $S_1)$
$\langle\, e_1 : p_1 \in S_1, p_2 \in S_2 \mid e_2\,\rangle$	`flatten(map` $(\textbf{fn}\, p_1 \Rightarrow \langle\, e_1 : p_2 \in S_2 \mid e_2\,\rangle)$ $S_1)$
$\displaystyle\sum_{p\in S} e$	`reduce add 0 (map` $(\textbf{fn}\, p \Rightarrow e)$ $S)$
$\displaystyle\sum_{i=k}^{n} e$	`reduce add 0 (map` $(\textbf{fn}\, i \Rightarrow e)$ $\langle\, k,\ldots,n\,\rangle)$
$\displaystyle\operatorname*{argmax}_{p\in S}(e)$	`argmax compare (map` $(\textbf{fn}\, p \Rightarrow e)$ $S)$

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 `argmax f S : ` $(\alpha \times \alpha \rightarrow$ `order`$) \rightarrow (\alpha$ `seq`$) \rightarrow$ `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

$\|S\|$	`size`(S)
$\{\}$	`empty`
$\{v\}$	`singleton`(v)
$\{v \in S \mid e\}$	`filter` $(\textbf{fn}\ v \Rightarrow e)$ S
$S_1 \cup S_2$	`union`(S_1, S_2)
$S_1 \cap S_2$	`intersection`(S_1, S_2)
$S_1 \setminus S_2$	`different`(S_1, S_2)
$\sum_{k \in S} e$	`reduce add 0 (Table.tabulate` $(\textbf{fn}\ k \Rightarrow e)$ S`)`

Tables

$\|T\|$	`size`(T)
$\{\}$	`empty()`
$\{k \mapsto v\}$	`singleton`(k, v)
$\{e : v \in T\}$	`map` $(\textbf{fn}\ v \Rightarrow e)$ T
$\{k \mapsto e : (k \mapsto v) \in T\}$	`mapk` $(\textbf{fn}\ (k, v) \Rightarrow e)$ T
$\{k \mapsto e : k \in S\}$	`tabulate` $(\textbf{fn}\ k \Rightarrow e)$ S
$\{v \in T \mid e\}$	`filter` $(\textbf{fn}\ v \Rightarrow e)$ T
$\{(k \mapsto v) \in T \mid e\}$	`filterk` $(\textbf{fn}\ (k, v) \Rightarrow e)$ T
$\{e_1 : v \in T \mid e_2\}$	`map` $(\textbf{fn}\ v \Rightarrow e_1)$ `(filter` $(\textbf{fn}\ v \Rightarrow e_2)$ T`)`
$\{k : (k \mapsto \_) \in T\}$	`domain`(T)
$\{v : (\_ \mapsto v) \in T\}$	`range`(T)
$T_1 \cup T_2$	`merge` $(\textbf{fn}\ (v_1, v_2) \Rightarrow v_2)$ (T_1, T_2)
$T \cap S$	`extract`(T, S)
$T \setminus S$	`erase`(T, S)
$\sum_{v \in T} e$	`reduce add 0 (map` $(\textbf{fn}\ v \Rightarrow e)$ T`)`
$\sum_{(k \mapsto v) \in T} e$	`reduce add 0 (mapk` $(\textbf{fn}\ (k, v) \Rightarrow e)$ T`)`
$\operatorname{argmax}_{(k \mapsto v) \in T}(e)$	`argmax max (mapk` $(\textbf{fn}\ (k, v) \Rightarrow e)$ T`)`

2 Function Costs

ArraySequence	*Work*	*Span*
`length`(T) `singleton`(v) `nth` $S\ i$ `empty()`	$O(1)$	$O(1)$
`tabulate` $f\ n$	$O\left(\sum_{i=0}^{n-1} W(f(i))\right)$	$O\left(\max_{i=0}^{n-1} S(f(i))\right)$
`map` $f\ S$	$O\left(\sum_{e \in S} W(f(e))\right)$	$O\left(\max_{e \in S} S(f(e))\right)$
`map2` $f\ S_1\ S_2$	$O\left(\sum_{i=0}^{\min(\|S_1\|,\|S_2\|)-1} W(f(S_{1i}, S_{2i}))\right)$	$O\left(\max_{i=0}^{\min(\|S_1\|,\|S_2\|)-1} S(f(S_{1i}, S_{2i}))\right)$
`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` $f\ b\ S$	$O\left(\|S\| + \sum_{f(x,y) \in \mathcal{O}_r(f,b,S)} W(f(x,y))\right)$	$O\left(\log \|S\| \max_{f(x,y) \in \mathcal{O}_r(f,b,S)} S(f(x,y))\right)$
`scan` $f\ b\ S$	$O\left(\|S\| + \sum_{f(x,y) \in \mathcal{O}_s(f,b,S)} W(f(x,y))\right)$	$O\left(\log \|S\| \max_{f(x,y) \in \mathcal{O}_s(f,b,S)} S(f(x,y))\right)$
`iter` $f\ b_0\ S$	$O\left(\sum_{i=0}^{\|S\|-1} W(f(b_i, S_i))\right)$	$O\left(\sum_{i=0}^{\|S\|-1} S(f(b_i, S_i))\right)$
`iterh` $f\ b_0\ S$	$O\left(\sum_{i=0}^{\|S\|-1} W(f(b_i, S_i))\right)$	$O\left(\sum_{i=0}^{\|S\|-1} S(f(b_i, S_i))\right)$
`showt` S `showti` $S\ f$	$O(\|S\|)$	$O(1)$
`showl` S	$O(\|S\|)$	$O(1)$
`hidet(NODE(`L, R`))`	$O(\|L\| + \|R\|)$	$O(1)$
`hidel(CONS(`x, xs`))`	$O(\|S\|)$	$O(1)$

`hidel(NIL)`
`hidet(ELT` e`)`
`hidet(EMPTY)`
	$O(1)$	$O(1)$

ArraySequence	*Work*	*Span*
`append`(S_1, S_2)	$O\left(\lvert S_1\rvert + \lvert S_2\rvert\right)$	$O(1)$
`take`(S, n) `drop`(S, n) `subseq` $S\ (s, n)$	$O(n)$	$O(1)$
`rake` $S\ (a, b, s)$	$O\left(\frac{\lvert b-a\rvert}{s}\right)$	$O(1)$
`splitMid`(S, i)	$O(\lvert S\rvert)$	$O(1)$
`flatten` S	$O\left(\lvert S\rvert + \sum_{e\in S}\lvert e\rvert\right)$	$O\left(\log\lvert S\rvert\right)$
`inject` $I\ S$	$O(\lvert I\rvert + \lvert S\rvert)$	$O(1)$
`partition` $I\ S$	$O(\lvert I\rvert + \lvert S\rvert)$	$O(1)$
`argmax` $f\ S$	$O(\lvert S\rvert)$	$O\left(\log\lvert S\rvert\right)$
`merge` $f\ S_1\ S_2$	$O\left(\lvert S_1\rvert + \lvert S_2\rvert\right)$	$O\left(\log(\lvert S_1\rvert + \lvert S_2\rvert)\right)$
`sort` $f\ S$	$O\left(\lvert S\rvert\log\lvert S\rvert\right)$	$O\left(\log^2\lvert S\rvert\right)$
`collate` $f\ (S_1, S_2)$	$O\left(\lvert S_1\rvert + \lvert S_2\rvert\right)$	$O\left(\log(\min(\lvert S_1\rvert, \lvert S_2\rvert))\right)$
`collect` $f\ S$	$O\left(\lvert S\rvert\log\lvert S\rvert\right)$	$O\left(\log^2\lvert S\rvert\right)$
`fromList`(S) `%`(S)	$O(\lvert S\rvert)$	$O(\lvert S\rvert)$
`toString` $f\ S$	$O\left(\sum_{e\in S} W(f(e))\right)$	$O\left(\sum_{e\in S} S(f(e))\right)$
`fields` $f\ S$ `tokens` $f\ S$	$O(\lvert S\rvert)$	$O\left(\log\lvert S\rvert\right)$

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 `iter` and `iterh`, $b_i = f(b_{i-1}, S_{i-1})$. For `showti`, `argmax`, `merge`, `sort`, `collate`, `collect`, `fields`, and `tokens` the given costs assume that the work and span of the application of f is constant.

TreeSequence	*Work*	*Span*
`nth` S i	$O(\log n)$	$O(\log n)$
`tabulate` f n	$---$	$O\left(\log n + \max_{i=0}^{n} S(f(i))\right)$
`map` f S	$---$	$O\left(\log \lvert S\rvert + \max_{s\in S} S(f(s))\right)$
`showt` S	$O(\log \lvert S\rvert)$	$O(\log \lvert S\rvert)$
`hidet(NODE(`L,R`))`	$O(\log(\lvert L\rvert + \lvert R\rvert))$	$O(\log(\lvert L\rvert + \lvert R\rvert))$
`append(`S_1, S_2`)`	$O(\log(\lvert S_1\rvert + \lvert S_2\rvert))$	$O(\log(\lvert S_1\rvert + \lvert S_2\rvert))$

`take(`S,n`)`
`drop(`S,n`)`
`subseq` S (s,n)
	$O(\log n)$	$O(\log n)$

`partition` I S	$O\left(\sum_{p\in S} p\right)$	$O(\log(\lvert I\rvert + \lvert S\rvert))$
`inject` I S	$O(\lvert I\rvert \lg(\lvert I\rvert + \lvert S\rvert))$	$O\left(\lg^2 \lvert I\rvert + \log \lvert S\rvert\right)$
`merge` f $\lvert S_1\rvert$ $\lvert S_2\rvert$	$O\left(m \lg(\frac{n+m}{m})\right)$	$O(\lg(n+m))$
`sort` f S	$O(\lvert S\rvert \log \lvert S\rvert)$	$O\left(\log^2 \lvert S\rvert\right)$
`collect` f S	$O(\lvert S\rvert \log \lvert S\rvert)$	$O\left(\log^2 \lvert S\rvert\right)$

where $n = \max(|S_1|, |S_2|)$ and $m = \min(|S_1|, |S_2|)$. For `singleton`, `length`, `filter`, `reduce`, `scan`, `sort` and `collect` the costs are the same as in ArraySequence. All $---$ entries are the same as ArraySequence. For `merge`, `sort`, and `collect` the costs assume that the work and span of the application of f is constant.

Single Threaded ArraySequence	*Work*	*Span*
`nth` S i `update` (i,v) S	$O(1)$	$O(1)$
`inject` I S	$O(\lvert I\rvert)$	$O(1)$
`fromSeq` S `toSeq` S	$O(\lvert S\rvert)$	$O(1)$

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

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.