Chapter 41

Tables

In this chapter we describe an interface and cost model for tables (also called maps or dictionaries). In addition to traditional functions such as insertion, deletion and search, we consider bulk functions that are better suited for parallelism.

1 Interface

The ability to map keys to associated values is crucial as a component of many algorithms and applications. Mathematically this corresponds to a map or function. In programming languages, data types to support such functionality are variously called maps, dictionaries, tables, key-value stores, and associative arrays. In this book we use the term table. Traditionally the focus has been on supporting insertion, deletion and search. In this book we are interested in parallelism so, as with sets, it is useful to have bulk functions, such as intersection and map.

Data Type 41.1 (Tables)**.** For a universe of keys $\mathbb{K}$ supporting equality, and a universe of values $\mathbb{V}$, the TABLE abstract data type is a type $\mathbb{T}$ representing the power set of $\mathbb{K} \times \mathbb{V}$

restricted so that each key appears at most once. The data type supports the following:

$$
\begin{array}{lcl}
\mathit{size} & : & \mathbb{T} \to \mathbb{N} \\
\mathit{empty} & : & \mathbb{T} \\
\mathit{singleton} & : & \mathbb{K} \times \mathbb{V} \to \mathbb{T} \\
\mathit{domain} & : & \mathbb{T} \to \mathbb{S} \\
\mathit{tabulate} & : & (\mathbb{K} \to \mathbb{V}) \to \mathbb{S} \to \mathbb{T} \\
\mathit{map} & : & (\mathbb{V} \to \mathbb{V}) \to \mathbb{T} \to \mathbb{T} \\
\mathit{filter} & : & (\mathbb{K} \times \mathbb{V} \to \mathbb{B}) \to \mathbb{T} \to \mathbb{T} \\
\mathit{intersection} & : & (\mathbb{V} \times \mathbb{V} \to \mathbb{V}) \to \mathbb{T} \to \mathbb{T} \to \mathbb{T} \\
\mathit{union} & : & (\mathbb{V} \times \mathbb{V} \to \mathbb{V}) \to \mathbb{T} \to \mathbb{T} \to \mathbb{T} \\
\mathit{difference} & : & \mathbb{T} \to \mathbb{T} \to \mathbb{T} \\
\mathit{find} & : & \mathbb{T} \to \mathbb{K} \to (\mathbb{V} \cup \bot) \\
\mathit{delete} & : & \mathbb{T} \to \mathbb{K} \to \mathbb{T} \\
\mathit{insert} & : & (\mathbb{V} \times \mathbb{V} \to \mathbb{V}) \to \mathbb{T} \to (\mathbb{K} \times \mathbb{V}) \to \mathbb{T} \\
\mathit{restrict} & : & \mathbb{T} \to \mathbb{S} \to \mathbb{T} \\
\mathit{subtract} & : & \mathbb{T} \to \mathbb{S} \to \mathbb{T}
\end{array}
$$

Throughout the set $\mathbb{S}$ denotes the powerset of the keys $\mathbb{K}$, $\mathbb{N}$ are the natural numbers (non-negative integers), and $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$.

Syntax 41.2 (Table Notation)**.** In the book we will write

$$\{k_1 \mapsto v_1, k_2 \mapsto v_2, \ldots\}$$

for a table that maps k_i to v_i.

The function size returns the size of the table, defined as the number of key-value pairs, i.e.,

$$\mathit{size}\ (a : \mathbb{T}) : \mathbb{N} = |a|.$$

The function empty generates an empty table, i.e.,

$$\mathit{empty} : \mathbb{T} = \emptyset.$$

The function $\mathit{singleton}$ generates a table consisting of a single key-value pair, i.e.,

$$\mathit{singleton}\ (k : \mathbb{K}, v : \mathbb{V}) : \mathbb{T} = \{k \mapsto v\}\,.$$

The function $\mathit{domain}(a)$ returns the set of all keys in the table a.

Larger tables can be created by using the $\mathit{tabulate}$ function, which takes a function and a set of key and creates a table by applying the function to each element of the set, i.e.,

$$\mathit{tabulate}\ (f : \mathbb{K} \to \mathbb{V})\ (a : \mathbb{S}) : \mathbb{T} = \{k \mapsto f(k) : k \in a\}\,.$$

The function map creates a table from another by mapping each key-value pair in a table to another by applying the specified function to the value while keeping the keys the same:

$$\mathit{map}\ (f : \mathbb{V} \to \mathbb{V})\ (a : \mathbb{T}) : \mathbb{T} = \{k \mapsto f(v) : (k \mapsto v) \in a\}\,.$$

The function *filter* selects the key-value pairs in a table that satisfy a given function:

$$\mathit{filter}\ (f : \mathbb{K} \times \mathbb{V} \to \mathbb{B})\ (a : \mathbb{T}) : \mathbb{T} = \{(k \mapsto v) \in a \mid f(k, v)\}\,.$$

The function $\mathit{intersection}$ takes the intersection of two tables to generate another table. To handle the case for when the key is already present in the table, the function takes a ***combining*** function f of type $\mathbb{V} \times \mathbb{V} \to \mathbb{V}$ as an argument. The combining function f is applied to the two values with the same key to generate a new value. We specify intersection as

$$\begin{aligned}&\mathit{intersection}\ (f : \mathbb{V} \times \mathbb{V} \to \mathbb{V})\ (a : \mathbb{T})\ (b : \mathbb{T}) : \mathbb{T} \\ &\quad = \{k \mapsto f(\mathit{find}\ a\ k, \mathit{find}\ b\ k) : k \in (\mathit{domain}(a) \cap\ \mathit{domain}(b))\}\,.\end{aligned}$$

The function $\mathit{difference}$ subtracts one table from another by throwing away all entries in the first table whose key appears in the second.

$$\begin{aligned}&\mathit{difference}\ (a : \mathbb{T})\ (b : \mathbb{T}) : \mathbb{T} \\ &\quad = \{(k \mapsto v) \in a \mid k \notin \mathit{domain}(b)\}\,.\end{aligned}$$

The function union unions the key value pairs in two tables into a single table. As with $\mathit{intersection}$, the function takes a combining function to determine the ultimate value of a key that appears is both tables. We specify union in terms of the $\mathit{intersection}$ and $\mathit{difference}$ functions.

$$\begin{aligned}&\mathit{union}\ (f : \mathbb{V} \times \mathbb{V} \to \mathbb{V})\ (a : \mathbb{T})\ (b : \mathbb{T}) : \mathbb{T} \\ &\quad = (\mathit{intersection}\ f\ a\ b)\ \cup\ (\mathit{difference}\ a\ b)\ \cup\ \ (\mathit{difference}\ b\ a)\end{aligned}$$

The function *find* returns the value associated with the key k. As it may not find the key in the table, its result may be bottom ($\bot$).

$$\mathit{find}\ (a : \mathbb{T})\ (k : \mathbb{K}) : \mathbb{B} = \begin{cases} v & \text{if } (k \mapsto v) \in a \\ \bot & \text{otherwise} \end{cases}$$

Given a table, the function delete deletes a key-value pair for a specified key from the table:

$$\mathit{delete}\ (a : \mathbb{T})\ (k : \mathbb{K}) = \{(k' \mapsto v') \in a \mid k \neq k'\}\,.$$

The function insert inserts a key-value pair into a given table. It can be thought as a singleton version of union and specified as such:

$$\begin{aligned}&\mathit{insert}\ (f : \mathbb{V} * \mathbb{V} \to \mathbb{V})\ (a : \mathbb{T})\ (k : \mathbb{K}, v : \mathbb{V}) : \mathbb{T} \\ &\quad = \mathit{union}\ f\ a\ (\mathit{singleton}\ (k, v)).\end{aligned}$$

The function *restrict* restricts the domain of the table to a given set:

$$restrict\ (a : \mathbb{T})\ (b : set) : \mathbb{T} = \{k \mapsto v \in a \mid k \in b\}\,.$$

It is similar to *intersection*, and in fact we have the equivalence:

$$intersection\ first\ a\ b = restrict\ a\ (domain\ b)$$

where $first(x, y) = x$. It can also be viewed as a bulk version of *find* since it finds all the keys of b that appear in a.

The function *subtract* deletes from a table the entries that belong a specified set:

$$\begin{aligned} &subtract\ (a : \mathbb{T})\ (b : set) : \mathbb{T} \\ &\quad = \{(k \mapsto v) \in a \mid k \notin b\}\,. \end{aligned}$$

It is similar to *difference*, and in fact we can define

$$difference\ a\ b = subtract\ a\ (domain\ b)$$

It can also be viewed as a bulk version of *delete* since it deletes all the keys of b from a.

In addition to these functions, we can also provide a *collect* function that takes a sequence a of key-value pairs and produces a table that maps every key in a to all the values associated with it in a, gathering all the values with the same key together in a sequence. Such a function can be implemented in several ways. For example, we can use the *collect* function that operate on sequences (Chapter 18) as discussed previously and then use tabulate to make a table out of this sequence. We can also implement it more directly using *reduce* as follows.

Algorithm 41.3 (*collect* on Tables)**.**

$$\begin{aligned} collect\ a\ \ &= Sequence.reduce \\ &\quad (Table.union\ Sequence.append) \\ &\quad \{\} \\ &\quad \langle\, \{k \mapsto \langle\, v \,\rangle\} : (k, v) \in a \,\rangle \end{aligned}$$

Syntax 41.4 (Tables (continued))**.** We will also use the following shorthands:

$$\begin{array}{ll} a[k] & \equiv find\ A\ k \\ \{k \mapsto f(x) : (k \mapsto x) \in a\} & \equiv map\ f\ a \\ \{k \mapsto f(x) : k \in a\} & \equiv tabulate\ f\ a \\ \{(k \mapsto v) \in a \mid p(k, v)\} & \equiv filter\ p\ a \\ a \setminus m & \equiv subtract\ a\ m \\ a \cup b & \equiv union\ second\ a\ b \end{array}$$

where $second(a, b) = b$.

Example 41.1. Define tables a and b and set S as follows.

$$\begin{array}{lcl} a & = & \{\texttt{'a'} \mapsto 4, \texttt{'b'} \mapsto 11, \texttt{'c'} \mapsto 2\} \\ b & = & \{\texttt{'b'} \mapsto 3, \texttt{'d'} \mapsto 5\} \\ c & = & \{3, 5, 7\}\,. \end{array}$$

The examples below show some functions, also using our syntax.

$$
\begin{array}{lrcl}
\mathit{find}: & a[\texttt{'b'}] & = & 11 \\
\mathit{filter}: & \{k \mapsto x \in a \mid x < 7\} & = & \{\texttt{'a'} \mapsto 4, \texttt{'c'} \mapsto 2\} \\
\mathit{map}: & \{k \mapsto 3 \times v : k \mapsto v \in b\} & = & \{\texttt{'b'} \mapsto 9, \texttt{'d'} \mapsto 15\} \\
\mathit{tabulate}: & \{k \mapsto k^2 : k \in c\} & = & \{3 \mapsto 9, 5 \mapsto 25, 9 \mapsto 81\} \\
\mathit{union}: & a \cup b & = & \{\texttt{'a'} \mapsto 4, \texttt{'b'} \mapsto 3, \texttt{'c'} \mapsto 2, \texttt{'d'} \mapsto 5\} \\
\mathit{union}: & \mathit{union} + (a, b) & = & \{\texttt{'a'} \mapsto 4, \texttt{'b'} \mapsto 14, \texttt{'c'} \mapsto 2, \texttt{'d'} \mapsto 5\} \\
\mathit{subtract}: & a \setminus \{\texttt{'b'}, \texttt{'d'}, \texttt{'e'}\} & = & \{\texttt{'a'} \mapsto 4, \texttt{'c'} \mapsto 2\}
\end{array}
$$

2 Cost Specification for Tables

The costs of the table functions are very similar to those for sets. As with sets there is a symmetry between the three functions *restrict*, *union*, and *subtract*, and the three functions *find*, *insert*, and *delete*, respectively, where the prior three are effectively "parallel" versions of the earlier three.

Cost Specification 41.5 (Tables)**.**

Operation	Work	Span
size a	1	1
singleton (k, v)	1	1
domain a	$\lvert a\rvert$	$\lg \lvert a\rvert$
filter p a	$\sum_{(k \mapsto v) \in a} W(p(k, v))$	$\lg \lvert a\rvert + \max_{(k \mapsto v) \in a} S(f(k, v))$
map f a	$\sum_{(k \mapsto v) \in a} W(f(v))$	$\lg \lvert a\rvert + \max_{(k \mapsto v) \in a} S(f(v))$
find a k	$\lg \lvert a\rvert$	$\lg \lvert a\rvert$
delete a k	$''$	$''$
insert f a (k, v)	$''$	$''$
intersection f a b	$m \lg(1 + \frac{n}{m})$	$\lg(n + m)$
difference a b	$''$	$''$
union f a b	$''$	$''$
restrict a c	$''$	$''$
subtract a c	$''$	$''$

where $n = \max(|a|, |b|)$ and $m = \min(|a|, |b|)$ or $n = \max(|a|, |c|)$ and $m = \min(|a|, |c|)$ as applicable. For *insert*, *union* and *intersection*, we assume that $W(f(\cdot, \cdot)) = S(f(\cdot, \cdot)) = O(1)$.

Remark. Abstract data types that support mappings of some form are referred to by various names including mappings, maps, tables, dictionaries, and associative arrays. They are perhaps the most studied of any data type. Most programming languages have some form of mappings either built in (e.g. dictionaries in Python, Perl, and Ruby), or have libraries to support them (e.g. map in the C++ STL library and the Java collections framework).

Remark. Tables are similar to sets: they extend sets so that each key now carries a value. Their cost specification and implementations are also similar.