Chapter 8

Dictionary Types

8.1 Dictionaries (dictionary)

1. Definition

An instance D of the parameterized data type *dictionary*<K, I> is a collection of items (*dic_item*). Every item in D contains a key from the linearly ordered data type K, called the key type of D, and an information from the data type I, called the information type of D. IF K is a user-defined type, you have to provide a compare function (see Section 2.3). The number of items in D is called the size of D. A dictionary of size zero is called the empty dictionary. We use $\langle k, i \rangle$ to denote an item with key k and information i (i is said to be the information associated with key k). For each $k \in K$ there is at most one $i \in I$ with $\langle k, i \rangle \in D$.

#include < *LEDA/core/dictionary.h* >

2. Types

dictionary<K, I>::*item* the item type.

dictionary<K, I>::*key_type* the key type.

dictionary<K, I>::*inf_type* the information type.

3. Creation

dictionary<K, I> D;

creates an instance D of type *dictionary*<K, I> based on the linear order defined by the global *compare* function and initializes it with the empty dictionary.

dictionary<*K*, *I*> *D*(*int* (∗*cmp*)(*const K*& , *const K*&));

creates an instance D of type *dictionary*<*K*, *I*> based on the linear order defined by the compare function *cmp* and initializes it with the empty dictionary.

4. Operations

const K&	*D*.key(*dic_item it*)	returns the key of item *it*. *Precondition*: *it* is an item in D.
const I&	*D*.inf(*dic_item it*)	returns the information of item *it*. *Precondition*: *it* is an item in D.
I&	*D*[*dic_item it*]	returns a reference to the information of item *it*. *Precondition*: *it* is an item in D.
dic_item	*D*.insert(*const K*& *k*, *const I*& *i*)	associates the information i with the key k. If there is an item $<k, j>$ in D then j is replaced by i, else a new item $\langle k, i\rangle$ is added to D. In both cases the item is returned.
dic_item	*D*.lookup(*const K*& *k*)	returns the item with key k (nil if no such item exists in D).
I	*D*.access(*const K*& *k*)	returns the information associated with key k. *Precondition*: there is an item with key k in D.
void	*D*.del(*const K*& *k*)	deletes the item with key k from D (null operation, if no such item exists).
void	*D*.del_item(*dic_item it*)	removes item *it* from D. *Precondition*: *it* is an item in D.
bool	*D*.defined(*const K*& *k*)	returns true if there is an item with key k in D, false otherwise.
void	*D*.undefine(*const K*& *k*)	deletes the item with key $k$from D (null operation, if no such item exists).
void	*D*.change_inf(*dic_item it*, *const I*& *i*)	makes i the information of item *it*. *Precondition*: *it* is an item in D.
void	*D*.clear( )	makes D the empty dictionary.

int	*D*.size( )	returns the size of D.
bool	*D*.empty( )	returns true if D is empty, false otherwise.

Iteration

forall_items(it, D) { "the items of D are successively assigned to it" }

forall_rev_items(it, D) { "the items of D are successively assigned to it in reverse order" }

forall(i, D) { "the informations of all items of D are successively assigned to i" }

forall_defined(k, D) { "the keys of all items of D are successively assigned to k" }

STL compatible iterators are provided when compiled with `-DLEDA_STL_ITERATORS` (see *LEDAROOT/demo/stl/dic.c* for an example).

5. Implementation

Dictionaries are implemented by $(2, 4)$-trees. Operations insert, lookup, del_item, del take time $O(\log n)$, key, inf, empty, size, change_inf take time $O(1)$, and clear takes time $O(n)$. Here n is the current size of the dictionary. The space requirement is $O(n)$.

6. Example

We count the number of occurrences of each string in a sequence of strings.

```
#include <LEDA/core/dictionary.h>

main()
{ dictionary<string,int> D;
  string s;
  dic_item it;

  while (cin >> s)
  { it = D.lookup(s);
    if (it==nil) D.insert(s,1);
    else D.change_inf(it,D.inf(it)+1);
  }

  forall_items(it,D) cout << D.key(it) << " : " <<  D.inf(it) << endl;

}
```