OrderedSymbolTable.mesa

OrderedSymbolTable is a package for maintaining symbol tables (key to value maps) with an ordering among keys. The ordering allows the table to perform searches such as "find the smallest item in the table that is larger than this one," as well as exact-match searches. The package's implementation uses a binary tree representation of 2-3-4 trees, called red-black trees; this means that any search, insertion, or deletion from a table of n items takes O(n log n) time.

A table stores items of type Node with keys of type Key, where Node and Key are defined in a definitions module that parameterizes the OrderedSymbolTable interface. A user of the package creates a suitable definitions module to parameterize OrderedSymbolTable, then compiles OrderedSymbolTable and RedBlackTreeImpl.

Last edited by:

MBrown on November 10, 1982 6:38 pm

```
DIRECTORY
  Environment USING [Comparison],
  ParticularTable USING [Node, NodeRecord, Key, GetKey, CompareKeyToNode];

OrderedSymbolTable: CEDAR DEFINITIONS IMPORTS ParticularTable = BEGIN
  Node: TYPE = ParticularTable.Node;
  NodeRecord: TYPE = ParticularTable.NodeRecord;
  Key: TYPE = ParticularTable.Key;
  GetKey: PRIVATE PROC [n: Node] RETURNS [Key]
    = INLINE { RETURN [ParticularTable.GetKey[n]] };
  Compare: PRIVATE PROC [k: Key, n: Node] RETURNS [Environment.Comparison]
    = INLINE { RETURN [ParticularTable.CompareKeyToNode[k, n]] };
  Table: TYPE = RECORD [Node];
    The RECORD type constructor allows most procedures below to be invoked using object
      notation.
  Error: ERROR [ec: ErrorType];
  ErrorType: TYPE = {notInitialized, badTable};
    All procedures below with a self: Table parameter raise Error[badTable] if self appears to be
      malformed (it might for instance be a normal Node that was passed by mistake.)
```

Procedures

```
Initialize: PROC [sentinel1, sentinel2: Node];
  This procedure must be called before calling any procedures below. Caller supplies the package
    instance with two nodes for its internal use. (These nodes are gone forever; we provide no way to
    reclaim their space.)
CreateTable: PROC [header: Node] RETURNS [Table];
  ! Error[notInitialized] (if Initialize has not been called.)
  Caller supplies a node to represent a table. The procedure creates an empty table and returns it.
DestroyTable: PROC [self: Table];
  Makes table empty, as it was just after CreateTable.

Insert: PROC [self: Table, nodeToInsert: Node, insertKey: Key];
  ! DuplicateKey (if insertKey already present in table).
  Caller asserts that Compare[insertKey, nodeToInsert] = equal. Inserts nodeToInsert into table.
    ERRORs DuplicateKey if Compare[insertKey, x] = equal for some node x already in the table.
DuplicateKey: ERROR;
```

Delete : PROC [self: Table, deleteKey: Key] RETURNS [deletedNode: Node];

Removes the (unique) node x in table such that Compare[deleteKey, x] = equal, and returns it, or returns NIL if no such node exists.

LookupProc : TYPE = PROC [self: Table, lookupKey: Key] RETURNS [Node];

Procedures Lookup, LookupNextLarger, LookupNextSmaller can all be assigned to a variable of this type.

Lookup : PROC [self: Table, lookupKey: Key] RETURNS [equalNode: Node];

Returns the (unique) node x in table such that Compare[lookupKey, x] = equal, or NIL if no such node exists.

LookupSmallest : PROC [self: Table] RETURNS [smallestNode: Node];

Returns the item in table with smallest key, or NIL if t is empty.

LookupNextLarger : PROC [self: Table, lookupKey: Key] RETURNS [largerNode: Node];

Returns the node in table with the smallest key strictly larger than lookupKey, or NIL if no such item exists.

LookupLargest : PROC [self: Table] RETURNS [largestNode: Node];

Returns the item in table with largest key, or NIL if t is empty.

LookupNextSmaller : PROC [self: Table, lookupKey: Key] RETURNS [smallerNode: Node];

Returns the node in table with the largest key strictly smaller than lookupKey, or NIL if no such node exists.

Lookup3 : PROC [self: Table, lookupKey: Key] RETURNS [leftNode, equalNode, rightNode: Node];

This Lookup procedure returns three nodes in table: [LookupNextSmaller[self, lookupKey], Lookup[self, lookupKey], LookupNextLarger[self, lookupKey]]. Note that in a nonempty table, one of the results is guaranteed # NIL. (This procedure was Howard Sturgis' idea.)

EnumerateIncreasing : PROC [self: Table, procToApply: PROC [Node] RETURNS [--*stop*--BOOL]];

For each item x in the table, in increasing order by key value, executes procToApply[x]. Returns when procToApply returns TRUE or when all items have been enumerated.

procToApply is restricted: it cannot call other procedures in this interface. In particular it cannot examine or modify the table being enumerated. Such enumerations can be implemented using LookupNextLarger or LookupNextSmaller.

CheckTable : PROC [self: Table];

ERRORs Error[badTable] if the red-black tree representing the table is not well-formed. Used by package test programs; may be used for client debugging of unsafe programs.

RootNode : PRIVATE PROC [self: Table] RETURNS [rootNode: Node];

Returns the the root of the red-black tree representing the table. Used by package test programs only.

END .

How to use

Compiling the package

This package is compile-time tailorable to a particular application. This tailoring is done without editing the source code of the package's interface or implementation. This is easy if only one version of the package is to be part of the application, and somewhat more involved if two or more versions of the package are to be part of the application. In the former case the procedure is:

(1) create a ParticularTable definitions module, which must define Node, NodeRecord, Key, GetKey, and CompareKeyToNode as follows:

```
DIRECTORY Environment USING [Comparison], ... ;
ParticularTable: DEFINITIONS = BEGIN
  Node: TYPE = REF NodeRecord;
  NodeRecord: TYPE = ... ;
    Must contain fields "rbLLink", "rbRLink", of type Node, and "rbColor", of type BOOL.
    It is likely that type NodeRecord is simply equated to a record type from some other
    defs module.
  Key: TYPE = ... ;
    Any old type that allows allows the following procedure to be defined:
  GetKey: PROC [n: Node] RETURNS [k: Key];
    Extracts the key from a node. This is called only from the procedure CheckTable below,
    so it need not be implemented if CheckTable will not be called.
  CompareKeyToNode: PROC [k: Key, n: Node] RETURNS [Environment.Comparison] ... ;
    Compares a bare key to the key embedded in a node.
    Result = less means k < n's Key, etc. May be defined inline.
  END.
```

(2) Compile the ParticularTable module created in step 1.

(3) Compile OrderedSymbolTable (this module).

(4) Compile RedBlackTreeImpl (the implementation of this module).

(5) Clients of the package use the OrderedSymbolTable.bcd created in step 3, and the application binds in the RedBlackTreeImpl.bcd created in step 4.

In case of multiple versions of the package within a single application, the different versions of modules ParticularTable, OrderedSymbolTable, and RedBlackTreeImpl must have distinct bcd names. The different ParticularTable source files must also have distinct names. Since in this case the module name <-> file name correspondence is not one-to-one, compiler command-line parameterization controls the different versions, as in:

(3') xxxOrderedSymbolTable _ OrderedSymbolTable[ParticularTable: xxxParticularTable]

(4') xxxRedBlackTreeImpl _ RedBlackTreeImpl[OrderedSymbolTable: xxxOrderedSymbolTable]

A version of this package that does its own storage allocation, stores REF ANY items, and can be used for different applications without recompiling is available as OrderedSymbolTableRef .

Concurrency

The implementation of this package is a module monitor. Hence for each instance of this package, at most one table operation may be in progress at a time.

Object notation

Clients of this package may use object notation to call any procedure whose first parameter is "self: Table". For instance, write "table.Insert[node, key]" instead of "OrderedSymbolTable.Insert[table, node, key]".

Change Log

Created by MBrown on March 2, 1982 4:14 pm

By editing OrderedSymbolTableRef.

Changed by MBrown on March 8, 1982 12:09 pm

Defined LookupProc, and removed named results to conform. Added comment about object notation.

Changed by MBrown on June 28, 1982 11:11 am

Make interface CEDAR.

Changed by MBrown on August 26, 1982 10:49 pm

Use Environment.Comparison.