ADT's Summary

A.N. Yzelman

6-7-2005

Inhoudsopgave

1 Preface **1**

2 Basic Data Types **1**
2.1 Position . . . 1
2.2 Entry . . . 1
2.3 Location-Aware Entry extends Entry implements Position . . . 2
2.4 Comparator . . . 2
2.5 Standard Methods . . . 2
2.6 Stacks . . . 2
2.7 Queue . . . 2
2.8 Deque . . . 3
2.9 Iterator . . . 3

3 Lists **3**
3.1 Vector . . . 3
3.2 List . . . 3
3.3 Sequence extends List, Vector . . . 4

4 Trees **4**
4.1 Standard Methods . . . 4
4.2 Tree . . . 4
4.3 BinaryTree extends Tree . . . 4
4.4 LinkedBinaryTree implements BinaryTree . . . 5
4.5 Complete Binary Tree extends Binairy Tree . . . 5
4.6 Heap extends Binary Tree . . . 5
4.7 Binary Search Tree extends Binary Tree . . . 5
4.8 AVL Tree extends Binary Search Tree . . . 6
4.9 Splay Trees extends Binary Search Tree . . . 6
4.10 (2,4) Tree . . . 6
4.11 Red-Black Tree extends Binary Search Tree . . . 6
4.12 (a,b) Tree extends (2,4) Tree . . . 7
4.13 B-Tree extends (a,b) Tree . . . 7

5 Priority Queues **7**
5.1 Priority Queue 7
5.2 Adaptable Priority Queue extends Priority Queue 7

6 Maps **7**
6.1 Map 7
6.2 Dictionairy 8
6.3 SkipList implements Dictionairy 8
6.4 Ordered Dictionairy extends Dictionairy 8

7 Sets **9**
7.1 Set 9
7.2 Partition 9

1 Preface

This is a summary containing various abstract data types, their methods and important properties. It is based on *Michael T. Goodrich's* and *Roberto Tamassia's "Data Structures and Algorithms in JAVA", 2004, John Wiley & Sons.* Study of that book or familiarity with the ADT's mentioned here is probably required before this summary is of any use.

2 Basic Data Types

2.1 Position

A position is always defined relatively; a position p will always be 'after' a position q and 'before' a position s.

base method:

element()

2.2 Entry

An entry is an key/object combination.

methods:
key()
value()

2.3 Location-Aware Entry extends Entry implements Position

Simply an entry which keeps track of its position.

2.4 Comparator

A comperator is a means of comparing two objects.

methods:

compare(object,object)

example:
compare(x,y) returns an integer < 0 if $x < y$, an integer > 0 if $x > y$, or a 0 if $x = y$.

2.5 Standard Methods

From now on, all ADT's (unless stated otherwise) support the following two methods:

size()
isEmpty()

2.6 Stacks

base methods:

push(object)
pop()
top()

2.7 Queue

base methods:

enqueue(object)
dequeue()
front()

2.8 Deque

base methods:

insertFirst(object)
insertLast(object)
removeFirst()
removeLast()
first()
last()

2.9 Iterator

methods:

object()
hasNext()
nextObject()
reset()

3 Lists

3.1 Vector

base methods:

elemAtRank(rank)
replaceAtRank(rank,element)
insertAtRank(rank,element)
removeAtRank(rank)

3.2 List

base methods (when returning, returns positions; not elements):

first()
last()
prev(position)
next(position)

update methods:

replace(position,element) - returns an element
insertFirst(element)
insertLast(element)
insertBefore(position,element)
insertAfter(position,element)
remove(position)

3.3 Sequence extends List, Vector

additional methods:

atRank(rank) : returns position
rankOff(position) : returns rank

4 Trees

4.1 Standard Methods

All trees support the following standard methods:

Iterator Elements()
Iterator Positions()
replace(position,object)

4.2 Tree

methods:

root()
parent(position)
positionIterator children(position)
isInternal(position)
isExternal(position)
isRoot(position)

4.3 BinaryTree extends Tree

additional methods:

left(position)
right(position)
hasLeft(position)

hasRight(position)

4.4 LinkedBinaryTree implements BinaryTree

added methods:

sibling(node)
addRoot(element)
insertLeft(node,element)
insertRight(node,element)
remove(node)
attach(node,tree,tree)

4.5 Complete Binary Tree extends Binairy Tree

A vector implementation of the Complete Binairy Tree is prefered to an linked list implementation.

Complete Binary Tree property:
A tree T with height h is a *complete* binairy tree if levels $0, 1, ..., h-1$ of T have the maximum number of nodes possible. In level
$h-1$, all the internal nodes are to the left of the external nodes and there is at most one node with one child, which must be a left child.

methods:

add(object) - output: position
remove() - output: object

4.6 Heap extends Binary Tree

Heap-order property:
In a heap T, for every node other than the root, the key stored is greater than or equal to the parent's key.

4.7 Binary Search Tree extends Binary Tree

Binary Search Tree properties:
All keys in entries in the left subtree of an internal node are less than the key of that internal node.
All keys in entries in the right subtree of an internal node are greater then the

key of that internal node.

4.8 AVL Tree extends Binary Search Tree

Height-Balance property:
Let u, v be the children of an internal node. Then $|\text{height}(u)\text{-height}(v)| \leq 1$.

4.9 Splay Trees extends Binary Search Tree

Splay Tree property:
After the search, insertion, or deletion algorithm finishes, we splay the last node visited (or the parent thereof in case of removal).

4.10 (2,4) Tree

Size property:
All internal nodes have at most four children, and at least two children (excluding the root).

Depth property:
All external nodes have the same depth.

Multi-way search tree:
An internal node with n children stores $n-1$ ordered entries.
Let us denote the i'th subtree of an internal node v with T_i, and let k_i be the key stored at the i'th entry of v. Then, for all keys k_L in T_i, and all keys k_R in T_{i+1}: $k_L \leq k_i \leq k_R$, for $1 \leq i < n$ (where n equals the number of children of v).

4.11 Red-Black Tree extends Binary Search Tree

Root property:
The root is black

External property:
Every external node is black

Internal property:
The children of a red node are black

Depth property:
All external nodes have the same black depth

4.12 (a,b) Tree extends (2,4) Tree

Size property:
Each internal node, excluding the root, has at least a children. Every internal node has at most b children.

4.13 B-Tree extends (a,b) Tree

A B-Tree is a (b/2,b) tree.

5 Priority Queues

A heap is preferred to implement both the priority queue and the adaptable priority queue.

5.1 Priority Queue

methods (returns entries):

min()
insert(key,value)
removeMin()

5.2 Adaptable Priority Queue extends Priority Queue

methods:

remove(entry)
replaceKey(entry,key)
replaceValue(entry,object)

6 Maps

6.1 Map

methods (returns objects):

get(key)
put(key,object)

remove(key)

iterators:

keys() – returns keys
values() – returns objects

6.2 Dictionairy

Unlike a map, a dictionairy supports entries with the same key

methods (returns entries):

find(key)
insert(key,object)
remove(entry)

iterators:

findAll(key) – returns entries
entries() – returns entries

6.3 SkipList implements Dictionairy

additional methods (returns positions):

next(position)
prev(position)
below(position)
above(position)
skipSearch(key)
skipInsert(key,object)
skipRemove(key) – returns an entry

6.4 Ordered Dictionairy extends Dictionairy

additional methods (returns entries):

first()
last()

additional iterators (returns entries):

successors(key)
predecessors(key)

7 Sets

7.1 Set

A set is a collection of objects (not neccesarily of the same type)

methods:

union(set)
intersect(set)
subtract(set)

7.2 Partition

methods:

makeSet(x) – returns position
union(set,set) – returns set
find(position) – returns set