Data Structures

Practical Assignment #2
DHeap

מגישים: עומר ברק (305555070) omerbarak2, חגי בן יהודה (305237000) hagaib1

Documentation

Nested Classes

Modifier and Type	Class and Description
(package private) static class	`DHeap.SortResult` Auxiliary class used to hold the result of a DHeapSortMeasure operation.

Fields

Modifier and Type	Field and Description
private DHeap_Item[]	`array` The actually members of the heap.
(package private) int	`compare_count` Used for testing comparisons inside the heap, for measurements.
private int	`d` The number of children each node in the d-heap structure has.
private int	`max_size` Holds the maximum number of elements possible for the heap to hold.
private int	`size` Holds the number of elements currently in the heap.

Constructor

DHeap

DHeap(int m_d, int m_size)
Constructor for the DHeap class.

Runtime: O(max_size)

Parameters:

m_d - The number of children for each node in the heap

m_size - The maximum size the heap might get

Methods

arrayToHeap

public void arrayToHeap(DHeap_Item[] array1)
The function builds a new heap from the given array. Previous data of the heap will be erased.

Precondition: array1.length <= max_size
Postcondition: isHeap(), size = array.length()
Runtime: O(n)

Parameters:

array1 - An array of all the items to insert into the heap

arrayToHeap

public void arrayToHeap(int[] numbers)
Make a heap from an array of numbers. Erases the current contents of the heap.

Precondition: numbers.length <= max_size
Postcondition: isHeap(), size = array.length()
Runtime: O(n)

Parameters:

numbers - The numbers to insert into the heap

arrayToHeap

public void arrayToHeap(List<Integer> numbers)
Make a heap from a list of Integers. Erases the current contents of the heap.

Precondition: numbers.size() <= max_size
Postcondition: isHeap(), size = array.length()
Runtime: O(n)

Parameters:

numbers - The numbers to insert into the heap

checkHeap

void checkHeap()
An internal method, used to assert the internal array satisfies the properties of a D-ary heap, and a few more internal tests.

Runtime: O(n)

child

public int child(int i, int k)
Computes the index of the k-th child of node i in a complete D-ary tree stored in an array. 1 <= k <= d. Note that indices of arrays in Java start from 0.

Precondition: i >= 0
Runtime: O(1)

Parameters:

i - The node who's child is requested

k - The requested child's index in relations to the parent

Returns:

The index of the requested child

childrenCount

private int childrenCount(int i)
Returns the number of children a specific node has.

Precondition: i >= 0
Runtime: O(1)

Parameters:

i - The node we want to get the number of children it has

Returns:

The number of children the given node has

Decrease_Key

public void Decrease_Key(DHeap_Item item, int delta)
Decrease the key of the given item by the given amount.

Precondition: item.pos < size, item != null, isHeap()
Postcondition: isHeap()
Runtime: O(log(n)/log(d))

Parameters:

item - The item who's key we want to decrease

delta - The value by which we want to decrease the item's key

Delete_Min

public void Delete_Min()
Deletes the smallest element from the heap.

Precondition: size > 0, isHeap()
Postcondition: isHeap()
Runtime: O(d/log(d)*log(n))

Delete

public void Delete(DHeap_Item item)
Deletes the given item from the heap.

Precondition: item.pos < size, item != null, isHeap()
Postcondition: isHeap()
Runtime: O(d/log(d)*log(n))

Parameters:

item - The item we want to delete from the heap

DHeapSort

public static int[] DHeapSort(int[] array)
Return a sorted array containing the same integers in the input array (done using HeapSort with a d-ary heap where d=2).

Runtime: O(n*log(n)), n = array.length

Parameters:

array - The array to sort

Returns:

A new array with the same values as the received one but sorted

DHeapSort

static int[] DHeapSort(int[] array, int d)
Return a sorted array containing the same integers in the input array.

Runtime: O(d/log(d)*n*log(n)), n = array.length

Parameters:

array - The array to sort

d - The arity of the d-ary heap to use

Returns:

A new array with the same values as the received one but sorted

DHeapSortMeasure

static DHeap.SortResult DHeapSortMeasure(int[] array, int d)
Return a sorted array containing the same integers in the input array, and also count the amount of comparisons made to detect the order.

Runtime: O(d/log(d)*n*log(n)), n = array.length

Parameters:

array - The array to sort

d - The arity of the d-ary heap to use

Returns:

A new array with the same values as the received one but sorted

Get_Min

public DHeap_Item Get_Min()
Returns the element with the lowest key value in the heap.

Precondition: heap-size > 0, isHeap(), size > 0
Postcondition: isHeap()
Runtime: O(1)

Returns:

The element with the lowest key value in the heap

getItems

List<DHeap_Item> getItems()
Returns a list of all the items in the heap.

Runtime: O(n)

Returns:

```
All the items in the heap, as a list
```

getSize

public int getSize()
Returns the size of the heap.

Runtime: O(1)

Returns:

```
The current number of elements in the heap
```

hasChildren

private boolean hasChildren(int i)
Says if a certain node has any children.

Precondition: i >= 0
Runtime: O(1)

Parameters:

```
i - The node to check
```

Returns:

```
true iff the node has any children
```

heapify

private void heapify()
Makes sure the internal array is structured like a heap, moving elements where necessary.

Runtime: O(n)

heapifyDown

private void heapifyDown(int i)
Makes sure the elements in the underlying array maintain the heap property, under a specific node in the heap, swapping elements if needed.

Runtime: O(d/log(d)*log(n))

Parameters:

i - The node from which to start the verification

heapifyUp

private void heapifyUp(int i)
Makes sure the elements in the underlying array maintain the heap property, under a specific node in the heap, swapping elements if needed.

Runtime: O(log(n)/log(d))

Parameters:

i - The heap index to start the process from

Insert

public void Insert(DHeap_Item item)
Inserts an item into the heap.

Precondition: item != null, isHeap(), size < max_size
Postcondition: isHeap()
Runtime: O(log(n)/log(d))

Parameters:

item - The heap item to insert

Insert

void Insert(int number)
Inserts a new item from a number into the heap.

Precondition: item != null, isHeap(), size < max_size
Postcondition: isHeap()
Runtime: O(log(n)/log(d))

Parameters:

number - The number to insert to the heap (as a heap item)

isHeap

public boolean isHeap()
public boolean isHeap() The function returns true if and only if the D-ary tree rooted at array[0] satisfies the heap property or size == 0.

Runtime: O(n)

Returns:

```
If the array satisfies the properties of a D-ary tree
```

lastChildIndex

private int lastChildIndex(int i)
Returns the index of the last child of the checked node.

Precondition: i >= 0
Precondition: hasChildren(i) == true
Runtime: O(1)

Parameters:

```
i - The node to check
```

Returns:

```
The index of the last child of the tested node
```

minChildIndex

private int minChildIndex(int i)
Returns the index of the child with the lowest key from the given node's children.

Precondition: hasChildren(i) == true
Runtime: O(d)

Parameters:

```
i - The node who's child we want
```

Returns:

```
The index of the child of the given node with the lowest key
```

newItem

private DHeap_Item newItem(int number)
Creates a new DHeap_Item from a number, to insert it into the heap.

Runtime: O(1)

Parameters:

number - The number to wrap

Returns:

The DHeap_Item wrapping the number

parent

public int parent(int i)
Computes the index of the parent for the heap element at the given index.

Precondition: i > 0
Runtime: O(1)

Parameters:

i - The index of the element who's parent is requested

Returns:

The index of the parent for the given element

popMin

private DHeap_Item popMin()
Returns the element with the lowest key value in the heap, and deletes it from the heap.

Precondition: heap-size > 0, isHeap(), size > 0
Postcondition: isHeap()
Runtime: O(d/log(d)*log(n))

Returns:

The element with the lowest key value in the heap

printTree

void printTree()
Prints the heap as a tree. Used internally for testing.

Runtime: O(n)

printTree

private void printTree(int i, String prefix, boolean isTail)
The internal method that prints a part of the heap as a subtree.

Note: adapted from http://stackoverflow.com/a/8948691
Runtime: O(n), n = sub-heap size.

Parameters:

i - `The index from which to start printing`

prefix - `The prefix used in order to align all all the prints`

isTail - `Says weather the current node is it's father's last child`

setItem

private void setItem(int i, DHeap_Item item)
Set the specified array slot to the given item, and update the item's position.

Runtime: O(1)

Parameters:

i - `The index at which to set the item`

item - `The item to put in the internal array`

swapItems

private void swapItems(int i, int j)
Swaps two items' position in the internal representation if the Heap.

Runtime: O(1)

Parameters:

i - `The index of the first item`

j - `The index of the second item`

Measurements

Part 1

Average comparisons for the arrayToHeap+DHeapSort operations:

m\d	2	3	4
1000	16859.8	16431.4	17608.8
10000	235320.4	226530.5	242759.0
100000	3018566.7	2896628.0	3076698.0

Part 2

Average comparisons for the Decrease-Key tests:

x\d	2	3	4
1	99999.0	99999.0	99999.0
100	153092.6	130732.9	122941.5
1000	303370.5	212932.7	181124.3