Data Structure Assignment 3

Report on Binomial Heap

2013-Jan-12

By 洪茂榮 (0016110)

I. Introduction

Comparison of Efficiency

Procedure	Binary (worst-case)	Binomial (worst-case)
Make-Heap	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(\lg n)$	$O(\lg n)$
Minimum	$\Theta(1)$	$O(\lg n)$
Extract-Min	$\Theta(\lg n)$	$\Theta(\lg n)$
Union	$\Theta(n)$	$O(\lg n)$
Decrease-Key	$\Theta(\lg n)$	$\Theta(\lg n)$
Delete	$\Theta(\lg n)$	$\Theta(\lg n)$

Binomial heap is a one of heap (min-heap) data structures. A binomial heap consists of one or more Binomial tree.

Binomial tree B_k is defined as follows:

- B_0 consists of a single node.
- For k>=1, B_k is a pair of B_{k+1} trees, where the root of one B_{k+1} becomes the leftmost child of the other.

Binomial-heap properties:

1. No two binomial trees in the collection have the same size.
2. Each node in each tree has a key.
3. Each binomial tree in the collection is heap-ordered in the sense that each non-root has a key strictly less than the key of its parent.

II. Objectives

The goal of this project is to implement a Binomial Heap class together with the definition of some basic member functions. Then test whether the heap built by the class constructor and all the functions defined in the class is working properly.

For the testing purpose, another function *postOrder* and *inOrder* member function are defined to transverse and display each nodes' data with the appropriate printing sequence.

III. Implementation Descriptions

Make a data structure, Binomial heap that has all the properties mentioned above.

Data Structure used in this program:

- BinoHeap which private data members are data pointer-to BinoNode, function bLink(), function mergeHeap.
- BinoNode which public data fields are integer data, integer degree, pointer to left child, pointer to right sibling, pointer to parent.
- User-defined deque, merely just to help in transversing and some other reversing order need.

The explanations for the implementation of the functions are described as comments in the each project file.

The Binomial Heap class should also provide some public member functions:

1. **getMin** // return the min value of binomial heap

   From the head pointer, compare the root with root right siblings at the root lists until the right sibling is null.

2. **insert** // insert a node into binomial heap

   Make a new binomial heap that consist of a node then union it to the heap you want to insert.

3. **deleteMin** // delete min

   - By getMin function, we know the position of the minimum node.
   - Then, take out the minimum node together with the tree.
   - The nodes at 1 level below the min-node root are make linked to each other in reversed order.
   - Lastly, call the *bHeapUnion* to join the reversed rootlist with the subtracted Heap (which have no min-node).

4. **size** // return the number of nodes in binomial heap

   Note that the nodes at Bk is 2^k. so the total nodes is the sum 2^ki where ki is the degree of the root-list of the heap.

5. **postorder** // output the postorder traversal of the binoheap

   Using recursive with some modification to reach the rSiblings where the tree has more than 2 children.

6. **inorder** // output the postorder traversal of the bino heap

   Using combination of iterative (helped by deque) and recursive approach to transverse the whole heap.

7. **bHeapUnion** // merge 2 binomial heap

Basically, the algorithm is already provided in the hand-out. If the two heaps is not empty it merge those two and then manage the binomial properties by dividing into 4 cases. And call the *bLink()* to link two binomial tree if needed.

For more details about each function please go to the codes and see the comments on the right side of the codes.

IV. Results

INPUTS:

Input myHeap1: 5 4 10 2 5 7

Input myHeap2: 7 12 8 11 6 1 9 3

myHeap3 := bHeapUnion(myHeap1,myHeap2)

OUTPUTS:

```
Test postOrder myHeap1: 7 10 5 4 2
Test inOrder myHeap1:   7 10 4 5 2
Size:   5

Test postOrder myHeap2: 3 9 1 12 11 8 6
Test Inorder myHeap2:   3 9 1 12 8 11 6
Size: 7

Test postOrder myHeap1 union myHeap:
7 9 3 1 12 11 8 10 5 4 6 2
Test Inorder myHeap1 union myHeap2:
7 3 9 1 12 8 11 6 10 4 5 2
Size: 12
getMin(): 1

After delete min:
Test myHeap1 union myHeap2 postOrder after delete:
9 7 3 12 11 8 10 5 4 6 2
Test myHeap1 union myHeap2 Inorder after delete:
9 7 3 12 8 11 6 10 4 5 2
size: 11
Press any key to continue . . .
```

V. How to Run this Program

- Run it using Visual Studio 2010 or other suitable version.
- Create a project.
- Place the .cpp file and the header files containing classes to the project folder directories.
- Run the project.

Remark: if any modifications of inputs needed to be made, user should go through main function and carefully make some adjustments.

VI. Other

Actually, the rather confusing part is the post-order and in-order transversal. Since the tree is not a binary tree, the usual algorithm does not work. Furthermore, the representation of the (multiway) Binomial tree is left child right sibling which can go back (to the left) if we have transverse the right sibling pointer to the right.