Astrid Manuel

3354122

March 28, 2017

COP-3540, Section U04, Assignment #3

**Problem #1:**

1. The preorder traversal output of a binary (min) heap is not sorted. A binary (min) heap is a binary tree that is complete from left to right, in which the values of the parent nodes are less than the values of the respective children nodes. Preorder traversal of a tree is a way of ordering the elements of a tree with the root first, most left-subtree second and most right-subtree last. Because there are no rules for specifically the left and/or right children of a binary (min) heap, a preorder traversal is very unlikely to provide a sorted output. For example, if you consider the following binary (min) heap:

1

/ \

3 2

/ \ / \

4 5 7 6

/ \

9 10

the preorder traversal output is: {1, 3, 4, 9, 10, 5, 2, 7, 6}. This output is not sorted.

1. The inorder and postorder traversals of a binary (min) heap are also not sorted. The inorder traversal orders the elements of a tree with the most left-subtree first, the node second, and the most right-subtree last. It is impossible to obtain an ordered output of a binary (min) heap for this traversal because in a binary (min) heap the element of the node is always less than the element of the children. This traversal would provide an ordered output for the binary search tree, however the binary (min) heap is not a binary search tree. The postorder traversal orders the elements of a tree with the most left-subtree first, the most right-subtree second and the node last. It is also impossible for this traversal to provide a sorted output in the case of a binary (min) heap because the node (which is the element of lesser value) is listed last. If you consider the binary (min) heap above, the inorder traversal output is: {9, 4, 10, 3, 5, 1, 7, 2, 6}, and the postorder traversal output is: {9, 10, 4, 5, 3, 7, 6, 2, 1}. Neither of these outputs are sorted.

**Problem #2:**

1. Max-heap resulting from inputting the following values: 11, 19, 23, 12, 13, 14, 17, 13, 18, and 33:

33

/ \

23 19

/ \ / \

14 18 17 13

/ \ /

11 13 12

1. Binary min heap that results from inputting the following values: 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2:

1

/ \

3 2

/ \ / \

6 7 5 4

/ \ / \ / \ / \

1. 14 12 9 10 11 13 8
2. Performing three deleteMin operations from tree in part (b):

First deleteMin:

2

/ \

3 4

/ \ / \

6 7 5 8

/ \ / \ / \ /

1. 14 12 9 10 11 13

Second deleteMin:

3

/ \

6 4

/ \ / \

13 7 5 8

/ \ / \ / \

1. 14 12 9 10 11

Third deleteMin:

4

/ \

6 5

/ \ / \

13 7 10 8

/ \ / \ /

1. 14 12 9 11

**Problem #3:**

1. Method for replacing key in MinHeap class:

public void replaceKey (Integer oldKey, Integer newKey){

for (int i = 1; i < minHeap.length; i++){

if (minHeap[i] = oldKey){

minHeap[i] = newKey;

while(minHeap[i] < minheap[i/2]){

perlocateUp();

}

return;

}

}

System.out.println(“The key you are trying to replace is not found.”);

}

1. The running time complexity of my replaceKey method is O(n^2). This is because there is a while loop of size n nested inside of a for loop, which is also of size n.

**Problem #4:**

1. Linear probing:

--------------------------------------------------------------------------------------------------------------------

| 9679 | 4371 | 1989 | 1323 | 6173 | 4344 | | | | 4199 |

--------------------------------------------------------------------------------------------------------------------

0 1 2 3 4 5 6 7 8 9

1. Quadratic probing:

--------------------------------------------------------------------------------------------------------------------

| 9679 | 4371 | | 1323 | 6173 | 4344 | | | 1989 | 4199 |

--------------------------------------------------------------------------------------------------------------------

0 1 2 3 4 5 6 7 8 9

1. Double hashing:

--------------------------------------------------------------------------------------------------------------------

| 1989 | 4371 | 1989 | 1323 | 6173 | 9679 | | 4344 | | 4199 |

--------------------------------------------------------------------------------------------------------------------

0 1 2 3 4 5 6 7 8 9

1. If S = {s1, s2, …, sm} and T = {t1, t2, …, tn}, m <= n, the following algorithm may be used to test whether set S is a subset of set T, using a hash table of size m:

* Create a hash table of size m:
  + Declare an array of linked lists of size m. This array will be your hash table. In this hash table the collision problem is solved with separate chaining by storing additional elements in the respective linked list.
* Create method store(LinkedList T[]) to store set T in hash table:
  + Use the hash function *h(key) = key mod TableSize* to store all elements of set T in the hash table created
* Create method testIfSubset(Integer S[]) to test whether S is a subset of T:
  + Declare a for loop to check each element in S with condition index < s.length-1 and an incrementing index
  + Check if the respective location of element in S is occupied or empty. If empty, S is not a subset of T, therefore print message and return to exit loop. If occupied:
  + Declare a for loop for a linear search within the respective linked list with the condition index < linkedList.length-1 and an incrementing index.
  + If the element is not found, S is not a subset of T, therefore print message and return to exit loop. If found
  + If all elements are found in respective linked lists in respective locations, the loops will run through and S is a subset of T. Print an appropriate message after loops.