**GIT Department of Computer Engineering CSE 222/505 - Spring 2021 Homework 4 # Report**

**Yeşim Yalçın**

**200104004094**

**1-) System Requirements**

**A-) Functional Requirements**

**-MaxHeap Requirements**

-MaxHeap can store elements that are comparable.

-MaxHeap can store maximum 7 elements.

-It behaves like a max heap. Stores the greatest element in the root and the children are always smaller than the root.

-MaxHeap stores the elements in an ArrayList. They are stored as nodes in the ArrayList.

-The nodes store the data of the element and the amount of the element.

-One element can be added more than once however, they will be stored in the same node meaning the amount field of the node will be incremented only.

-If an element wants to be deleted, the element node’s amount is checked first. If the amount is 1, the node is deleted. If not 1, the amount is decremented by 1.

-If the mode element of the heap is tried to be found and there are more than one elements with the same highest frequency, the last found element will be chosen as the mode element.

**-BSTHeapTree Requirements**

-BSTHeapTree can store elements inside MaxHeaps. Each node represents a MaxHeap and the MaxHeaps are stored in a BinarySearchTree.

-If an element is added to the BSTHeapTree, the element will be compared to the roots of the MaxHeap and will be placed in the first non full proper heap that will not break the order of the BinarySearchTree.

-If an element is removed from the BSTHeapTree the removal will be the same as the removal in MaxHeap however if a non leaf node becomes non full because of this, some elements from the leaf nodes are removed and added to that non leaf node to make it full again.

- If the mode element of BSTHeapTree is tried to be found and there are more than one elements with the same highest frequency, the last found element will be chosen as the mode element.

-It is possible to print out all the heaps in an array form in preorder. However it will not show which heap is the parent of which heaps etc. It is also possible to print the tree in proper preorder. However when this is done, only the roots of the heaps are printed out to represent the heaps.

**-HelperMethods Requirements**

-These methods are used in the driver code.

-They are generic methods that can do some simple operations to help.

**B-) Non-Functional Requirements**

**-**The system is written using Java jdk 15.0.1

-You can use the makefile then use java DriverCode command to run the program.

-DriverCode outputs are really long therefore running the program in an environment which shows all the outputs of the program is recommended.

If you are using terminal on Ubuntu, changing terminal’s scrolling setting is recommended. You can press Edit -> Profile Preferences -> Scrolling Tab -> Unlimited option when you are in terminal.

**2-) Class Diagram**

**-**The class diagram is added as ClassDiagram.png to the homework file.

**3-) Problem Solution Aproach**

-Firstly I decided to implement a functioning heap class that can be used to create a BSTHeapTree in a way the homework wanted. To be able to do that I decided to implement a max heap because BSTHeapTree stores max heaps inside. I also limited the maximum element a heap can store to 7, to be able to satisfy that condition of BSTHeapTree too.

-I decided to create nodes to store both the data and the amount of an element inside a heap to be able to track how many times an element is inserted to the BSTHeapTree later on. However I did not use the nodes to connect to eachother, instead I used an ArrayList to store the nodes because this is more efficient to reach the datas inside the heaps.

-After properly implementing the MaxHeap, I started implementing the BSTHeapTree. I chose to use the BinarySearchTree implementation in our book to be able to store the maxheaps inside a binary search tree. This made me have access to some methods like add or delete. Moreover, I added some additional methods to the BinarySearchTree.

-I combined the methods of the BinarySearchTree and MaxHeap also added some more additional operations to satisfy the conditions of BSTHeapTree and finished the implementation.

-To test the code I firstly tested MaxHeap class to make sure it is working fine and then the BSTHeapTree. After I was sure both were working fine, I proceeded to test to code one last time with the required implementation test.

**4-) Test Cases and Running Results**

**1-) Driver Code Tests:**

A-) MaxHeap Class Driver Code

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Scenario** | **Expected Result** | **Real Result** | **Pass/Fail** |
| **1** | Creating a MaxHeap and inserting elements | Successfuly adding | The same | Pass |
| **2** | Removing elements | Successfuly removing | The same | Pass |
| **3** | Creating and merging heaps | Successfuly merging | The same | Pass |
| **4** | Removing n’th greatest elements | Successfuly removing | The same | Pass |
| **5** | Finding the heap’s mode | Successfuly finding | The same | Pass |

B-) BSTHeapTree Driver Code

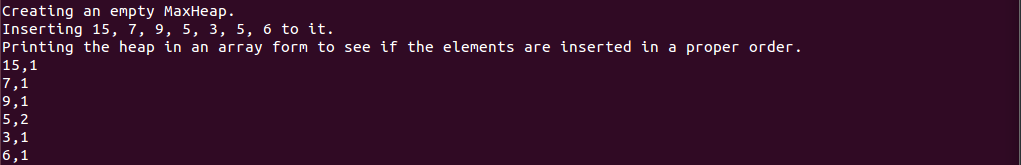
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Scenario** | **Expected Result** | **Real Result** | **Pass/Fail** |
| **6** | Creating 4 max heaps | Successfuly creating | The same | Pass |
| **7** | Inserting the heaps inside an empty BSTHeapTree | Successfuly adding the heaps in an order | The same | Pass |
| **8** | Adding new elements 1 | Successfuly adding (increasing the amounts) | The same | Pass |
| **9** | Adding new elements 2 | Successfuly adding (adding new nodes) | The same | Pass |
| **10** | Adding new elements 3 | Successfuly adding (adding new heaps) | The same | Pass |
| **11** | Removing elements 1 | Successfuly removing (decrementing amounts) | The same | Pass |
| **12** | Removing elements 2 | Successfuly removing (also removing some heaps) | The same | Pass |
| **13** | Searching for elements | Successfuly finding their amount in the tree | The same | Pass |

**2-) Implementation Test**

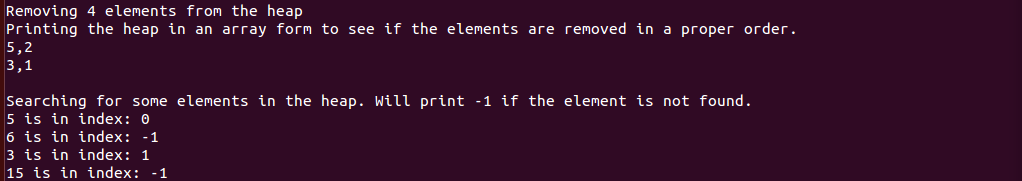
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Scenario** | **Expected Result** | **Real Result** | **Pass/Fail** |
| **14** | Inserting 3000 random elements into an array and BSTHeapTree | Successfuly adding | The same | Pass |
| **15** | Searching for 100 existing elements | Successfuly finding (with the same amounts in both) | The same | Pass |
| **16** | Searching for 10 non existing elements | Cannot find because does not exist | The same | Pass |
| **17** | Finding the mode | Successfuly finding the mode (the sam efor both the array and the tree) | The same | Pass |
| **18** | Removing 10 non existing elements | Cannot remove because does not exist | The same | Pass |
| **19** | Removing 100 elements | Successfuly removing | The same | Pass |

**The Results**

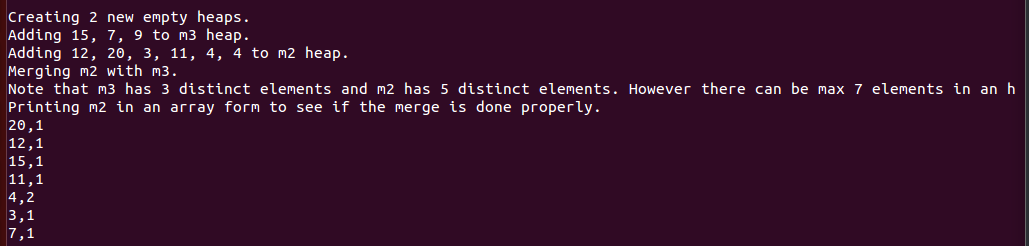
1-) Creating an empty MaxHeap and inserting elements to it.



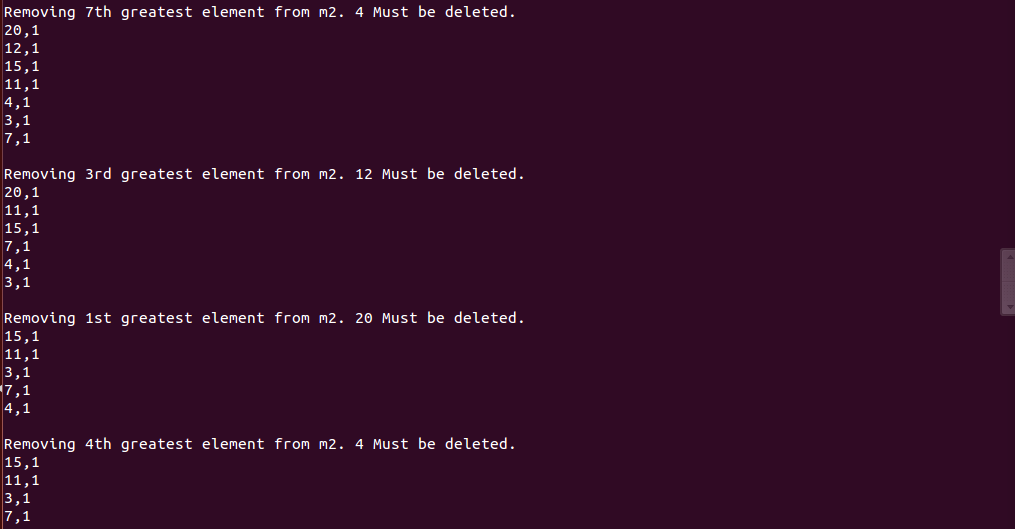
2-) Removing elements from the heap and then searching for some elements inside it.



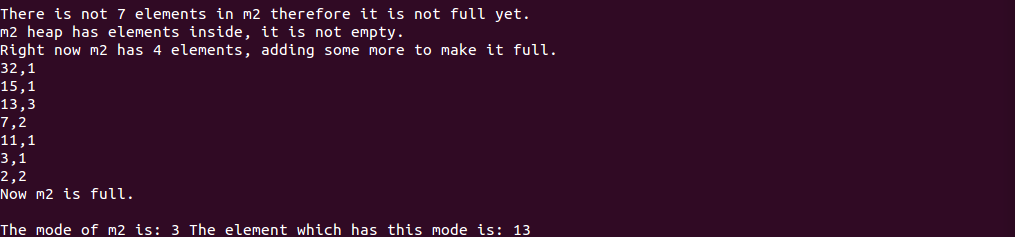
3-) Creating new heaps and merging them.



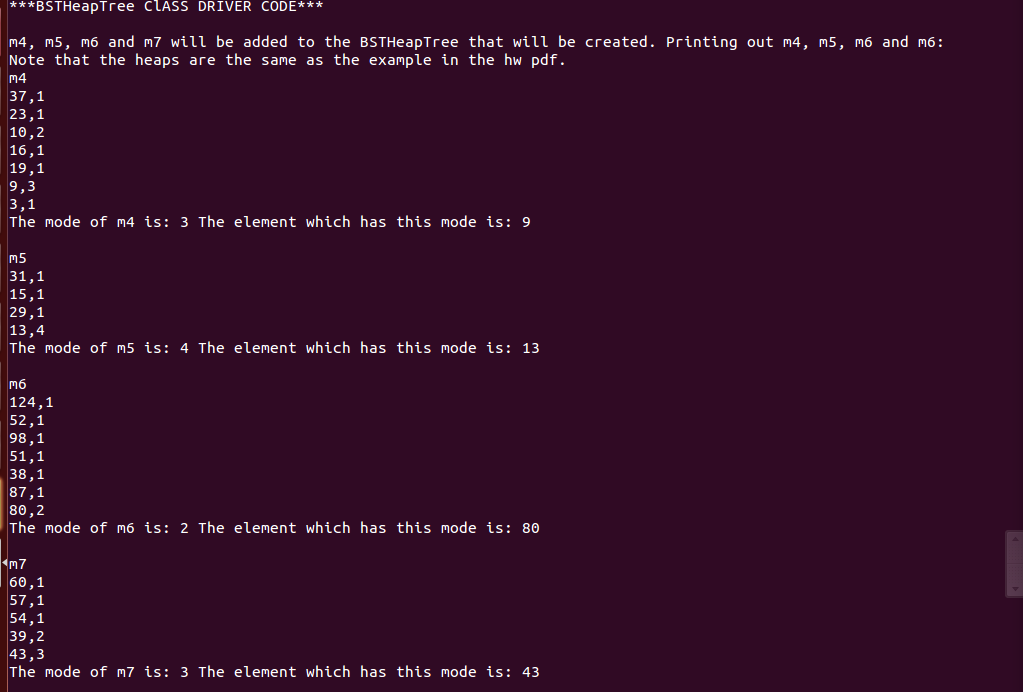
4-) Removing some n’th greatest elements from the heap.



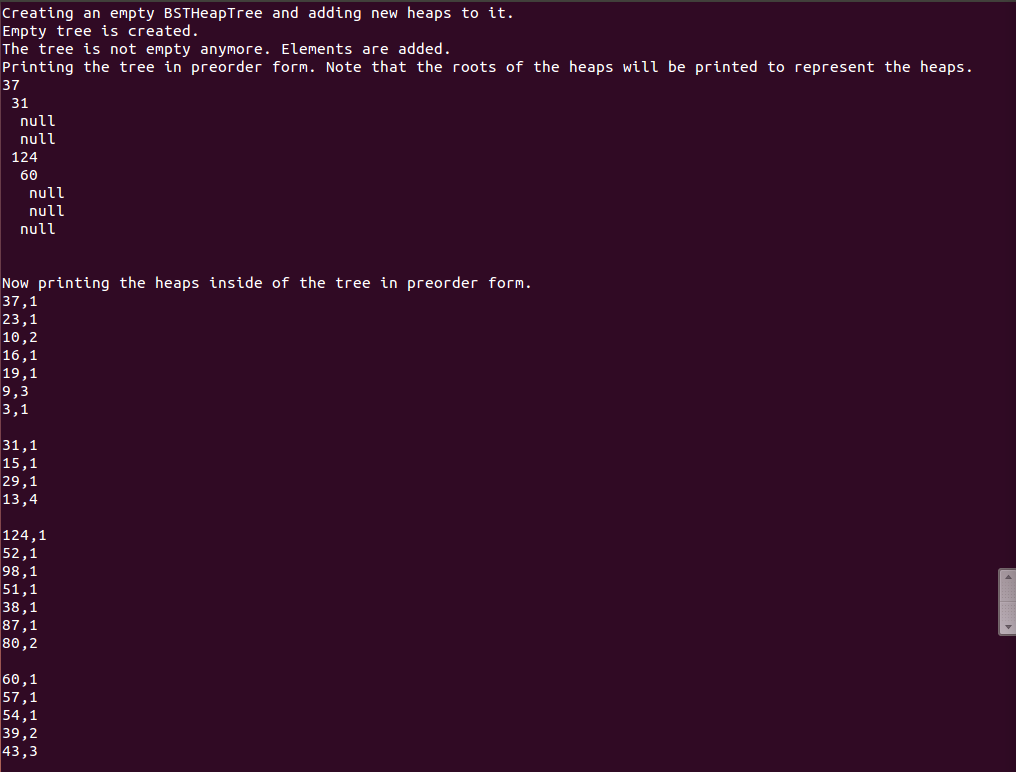
5-) Checking if a heap is empty, full and then finding it’s mode.

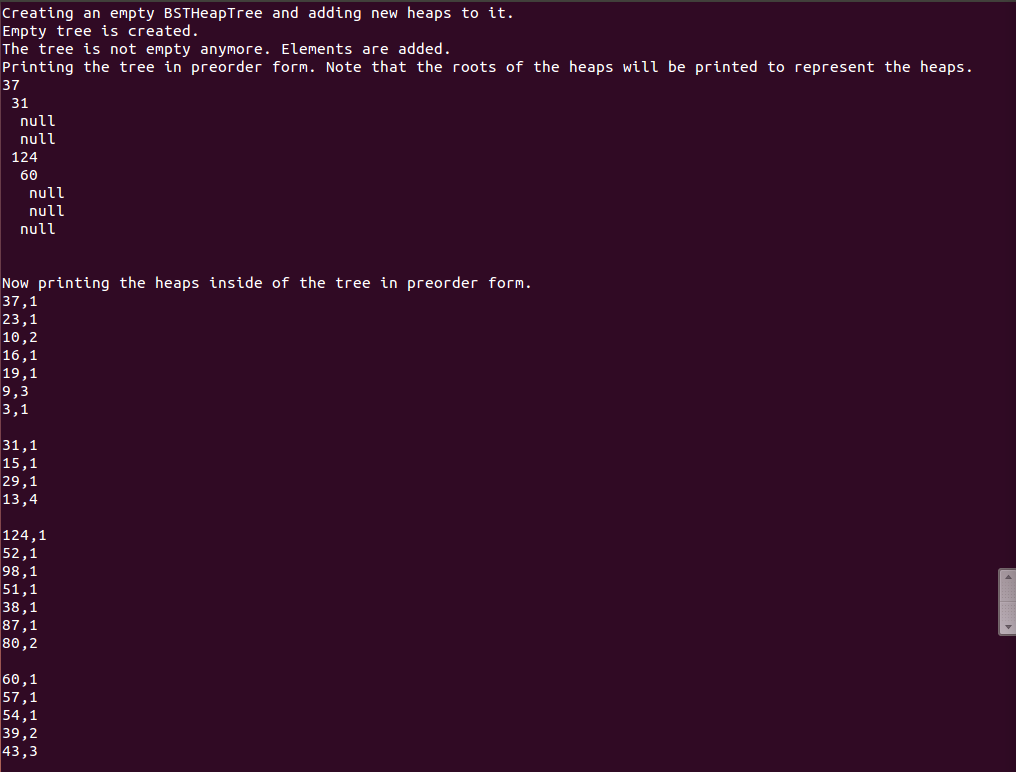


6-) Creating 4 max heaps.

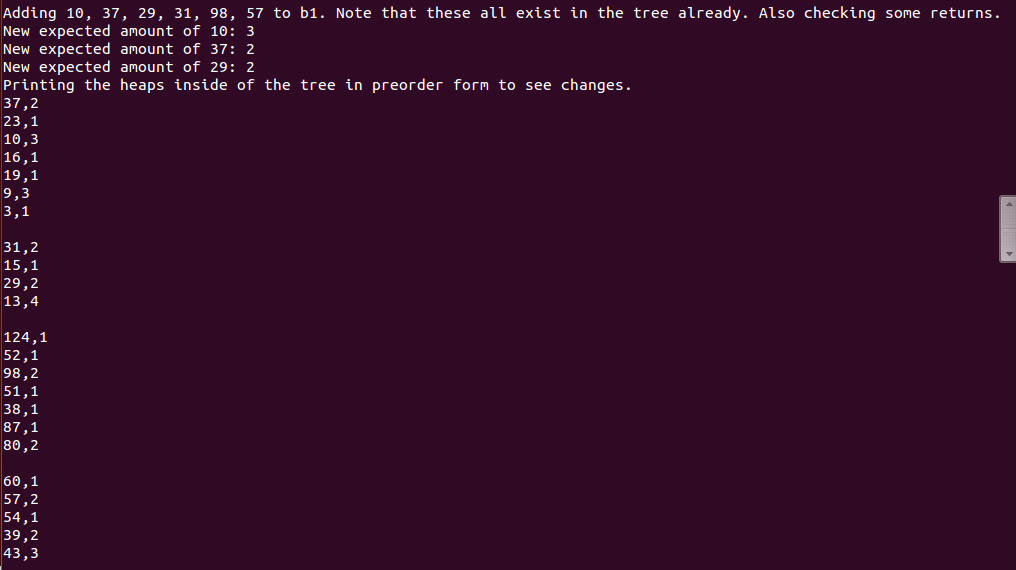


7-) Creating an empty BSTHeapTree, inserting the 4 max heaps and printing the tree.

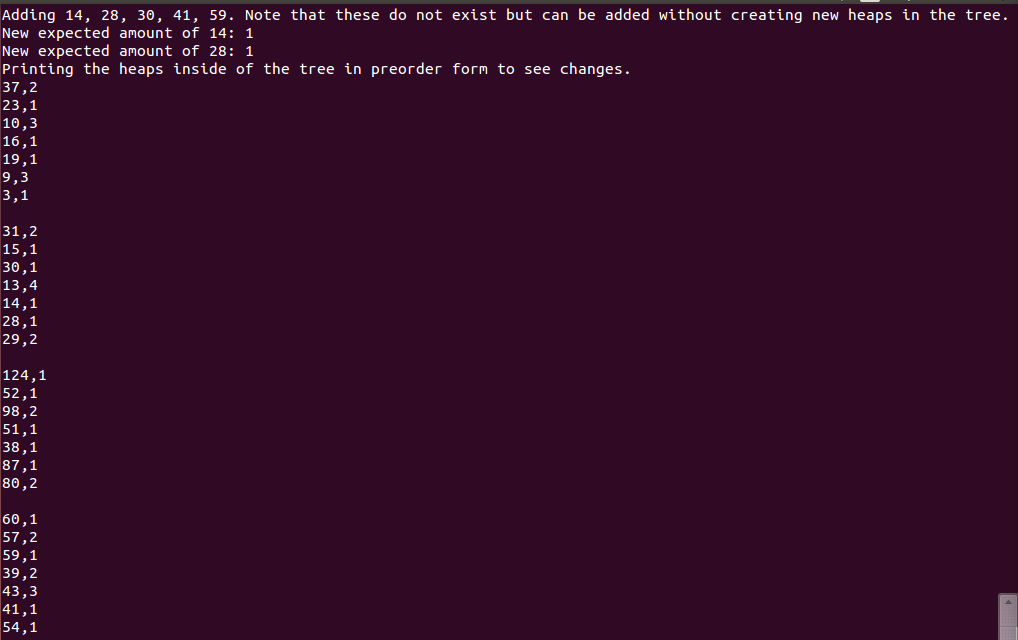




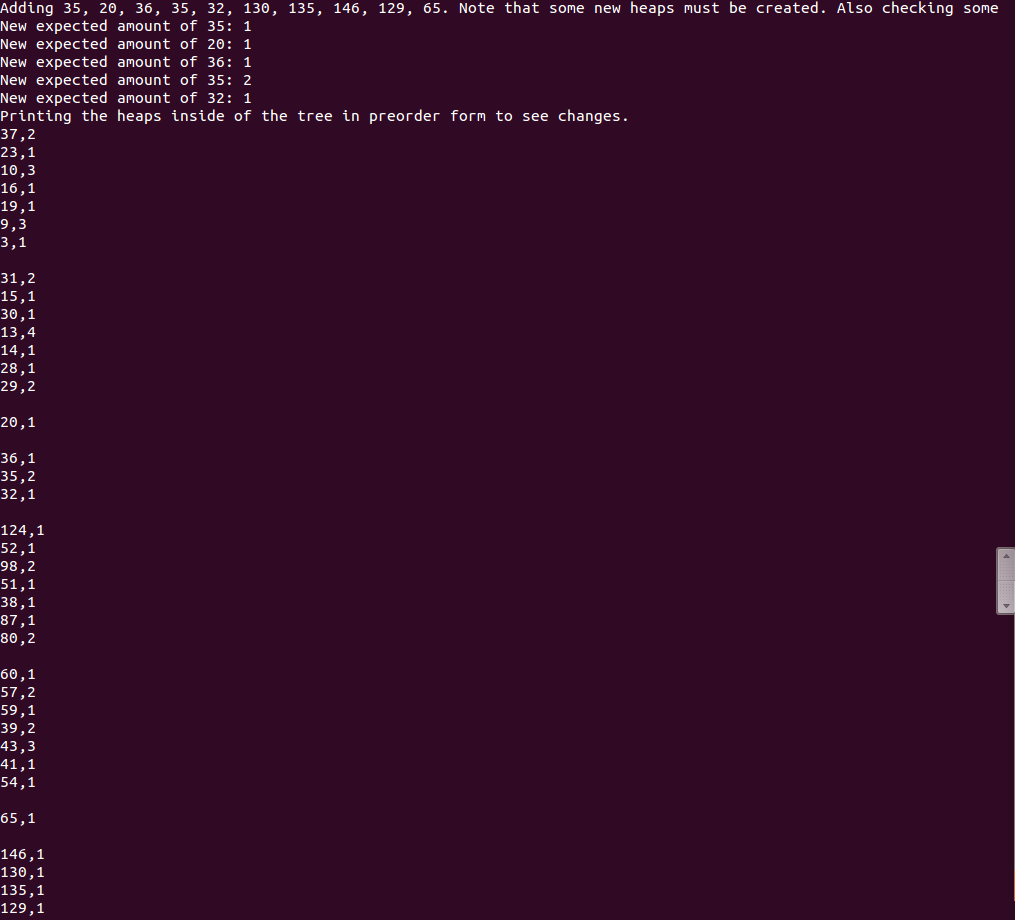
8-) Adding new elements to the tree (only amounts increases).



9-) Adding new elements to the tree (new nodes are created).

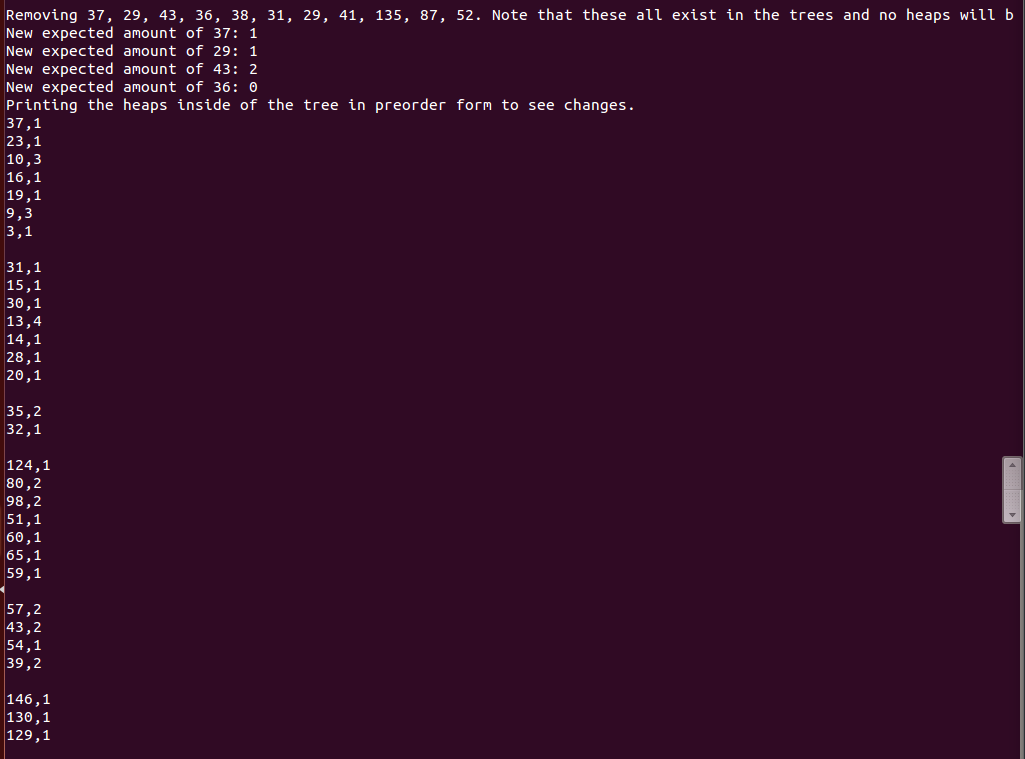


10-) Adding new elements to the tree (new heaps are created).

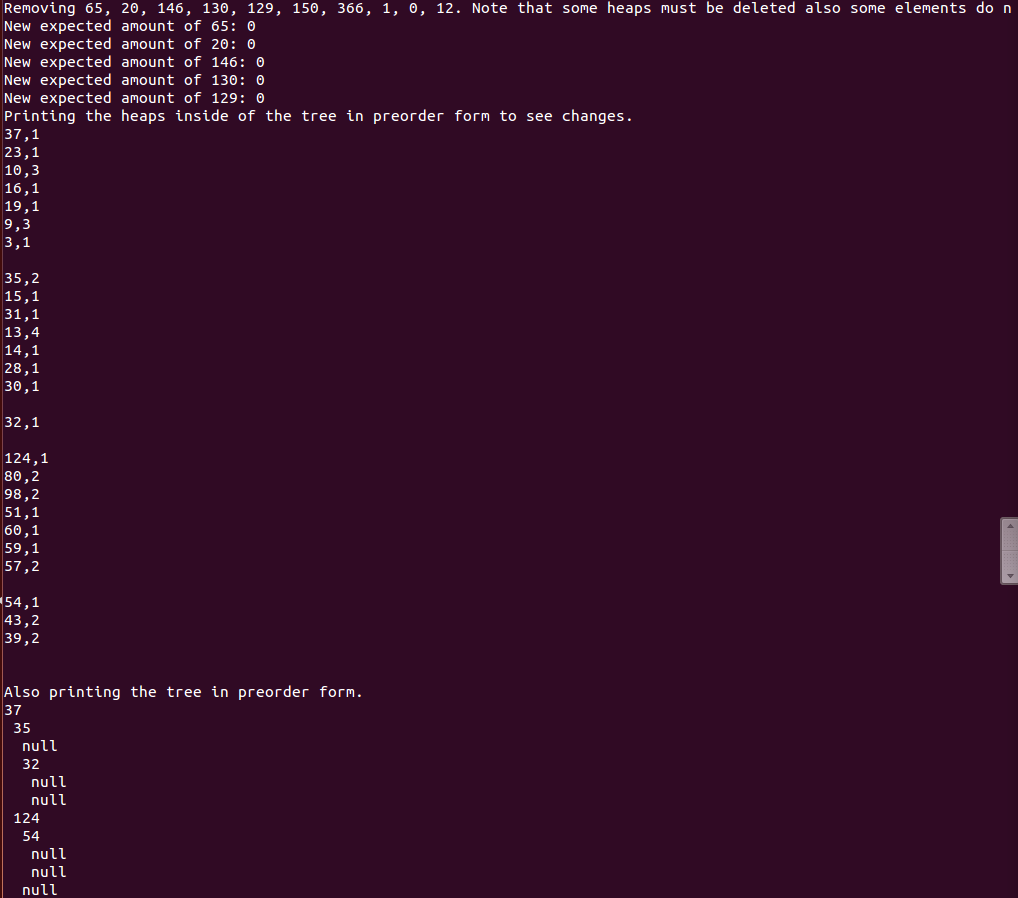




11-) Removing some elements from the tree (only amounts are decreased).



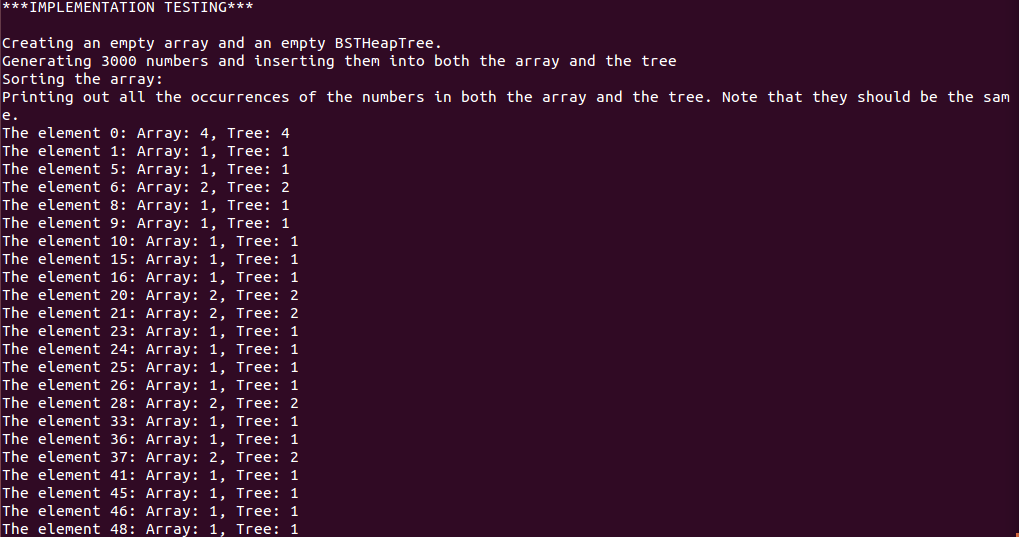
12-) Removing some elements from the tree (some nodes and heaps must be deleted).



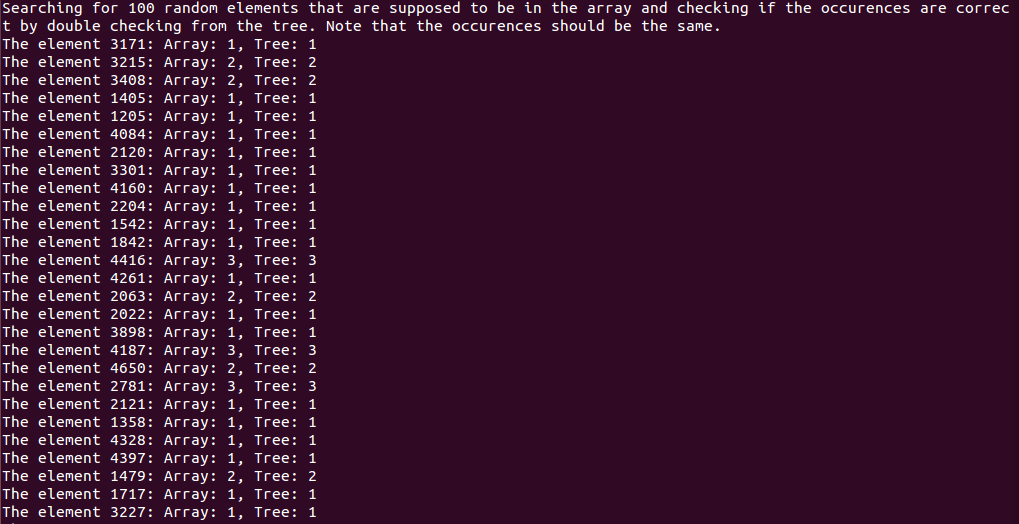
13-) Searching for some elements in the tree.



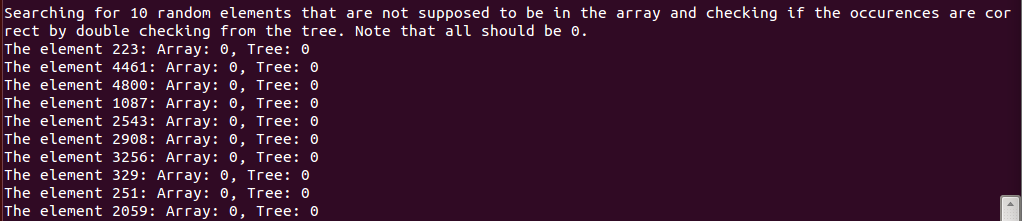
14-) Creating an empty array and a BSTHeapTree and inserting random 3000 elements into them.



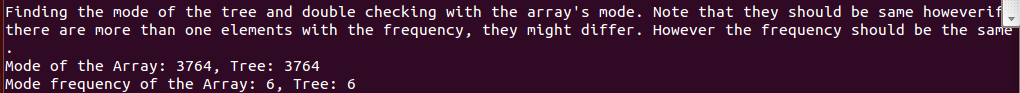
15-) Searching for 100 elements that are supposed to be inside of the array and the tree.



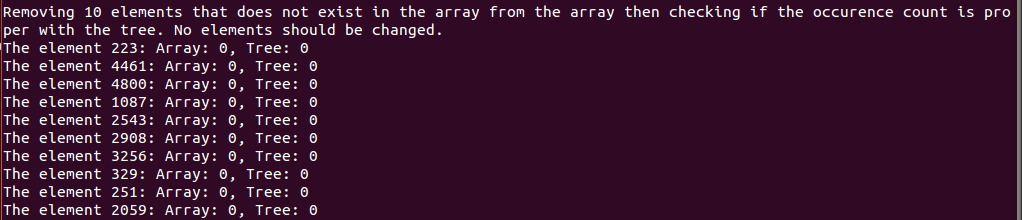
16-) Searching for 10 elements that are not supposed to be inside of the array and the tree.



17-) Finding the mode of the tree and the array.



18-) Trying to remove 10 elements that do not exist in the array or the tree.



19-) Removing 100 elements.

