GTU Department of Computer Engineering

CSE 222/505 - Spring 2021 Homework 4

Report

Muhammet Fikret ATAR

1801042693

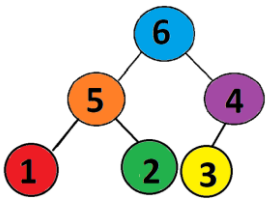
Part 1:

SYSTEM REQUIREMENTS

A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Mapping the elements of a heap into an array is trivial: if a node is stored an index k, then its left child is stored at index 2k+1 and its right child at index 2k+2.

.



Operations on Max Heap Requirement

i. Search for an element

ii. Merge with another heap

iii. Removing ith largest element from the Heap

iv. Extend the Iterator class by adding a method to set the value (value passed as parameter) of the last element returned by the next methods.

Class diagrams Part1;

Part 1:

PROBLEM SOLUTION APPROACH:

My idea is to heapify the complete binary tree formed from the arraylist in reverse level order following a top-down approach.

That is first heapify, the last node in level order traversal of the tree, then heapify the second last node and so on.

Part 2:

PROBLEM SOLUTION APPROACH:

After creating the heap theme above, I integrated this structure into the bst structure

Part 1:

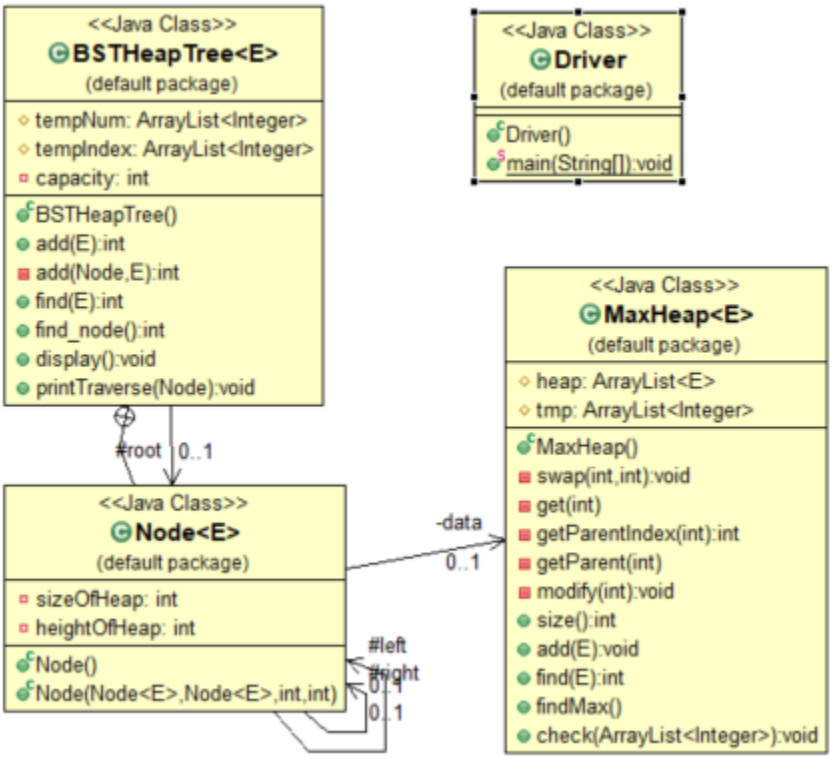
SYSTEM REQUIREMENTS

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree

As a result heap structures are built on this structure.

Part 2 Class diagrams

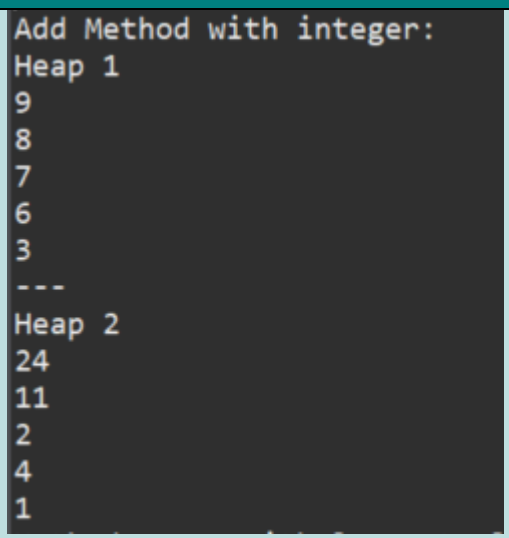
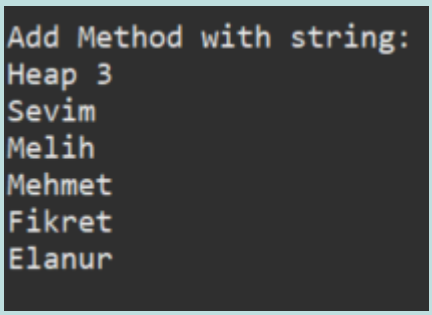


Test Cases both of them :

Test ID	Test Cases	Test Steps	Expected Result	Actual Results	Pass /Fail
T1	**Test add method**	**Call the method with proper parameter**	**Add element to heap**	**As expected**	**PASS**
T2	Search for an element	**Call the method with proper parameter**	**Find Element and return true or false**	**As expected**	**PASS**
T3	Merge with another heap	**Call the method with proper parameter**	**Merge two heap str**	**As expected**	**PASS**
T4	Removing ith largest element	**Call the method with proper parameter**	**Removing ith largest element from the Heap**	**As expected**	**PASS**
T5	Adding a method	**Call the method with proper parameter**	Extend the Iterator class by adding a method to set the value	**Not as expected**	**FAIL**
T6	Int add (E item) method	**Call the method with proper parameter**	Returns the number of occurrences of the item after insertion	**As expected**	**PASS**
T7	Int remove method(E item)	**Call the method with proper**	Returns the number of occurrences	**Not as expected**	**FAIL**

		parameter	of the item after removal		
T8	int find (E)	**Call the method with proper parameter**	returns the number of occurrences of the item in the BSTHeapTree	**As expected**	**PASS**
T9	find_mode ()	**Call the method with proper parameter**	**Find mode bst heap**	**As expected**	**PASS**

Test And Result:

Tests	Result
T1	 

T2	Search Method: true false true false
T3	Merge Method: Merge Heap 1 And Heap 2: 24 11 9 7 3 2 4 6 1
T4	Method remove ith largest element from the Heap: Heap 1 after remove 2th largerst element 9 7 6 3
T5	**fail**

T6	Insert the 3000 numbers that are randomly generated in the range 0-5000 into the BSTHeapTree Display Method: (2212,1) (2204,1) (1522,2) (1658,2) (1034,2) (1437,1) (804,2) ------- (2541,1) (2514,1) (2302,2) (2475,2) (2220,2) (2289,1) (2375,2) ------- (2571,1) (2563,1) (2561,2) ------- (2575,1) (2570,1) (2564,2) (2567,2) (2549,2) (2551,1) (2550,2) ------- (2579,1) (2461,1) (2355,2)
T7	FAİL

T8

```
Find Method:
Search for 100 numbers in the array and 10 numbers not in the array and make sure that the number of occurrences is correct
Find: (4987,1)
Find: (1346,1)
Find: (2239,2)
Find: (1714,2)
Find: (1426,2)
Find: (176,1)
Find: (2512,2)
Find: (1440,2)
Find: (2466,1)
Find: (4541,1)
Find: (254,3)
Find: (3184,2)
Find: (4793,1)
Find: (2012,2)
Find: (4385,3)
Find: (3284,1)
Find: (2808,1)
Find: (3630,1)
Find: (2678,2)
Find: (2806,1)
Not in heap
Find: (606,2)
Find: (3767,2)
Find: (2045,3)
Find: (2656,1)
Find: (2945,1)
Find: (1157,3)
Find: (496,2)
Find: (3695,2)
```

T9

```
Find_Mode Method:
Max Occurence Value: 254
```

Part3: ANALYZE THE TIME COMPLEXITY:

```
private void swap(int x, int y) {
    /* swaps reference */
    T temp = heap.get(x);
    heap.set(x, heap.get(y));
    heap.set(y, temp);
}
```

T(N)=O(1)

```java
public MyHeap<T> Merge_with_another_heap(MyHeap<T> obj1, MyHeap<T> obj2) {
    // firstly compare obj and merge according to size
    if (obj1.get_size() > obj2.get_size()) {
        try {
            for (int i = 0; i < obj2.get_size(); i++) {

                obj1.add(obj2.get(i));
            }
            return obj1;
        } catch (Exception e) {
            System.out.println("Something went wrong MERGE.");
        }
        // firstly compare obj and merge according to size
    } else if (obj1.get_size() < obj2.get_size()) {
        try {
            for (int i = 0; i < obj1.get_size(); i++) {

                obj2.add(obj1.get(i));
            }
            return obj2;
        } catch (Exception e) {
            System.out.println("Something went wrong.MERGE");
        }
        // firstly compare obj and merge according to size
    } else {
        try {
            for (int i = 0; i < obj2.get_size(); i++) {

                obj1.add(obj2.get(i));
            }
            return obj1;
        } catch (Exception e) {
            System.out.println("Something went wrong.MERGE");
        }
```

T(N)=O(N'2)

```java
public boolean search_heap(T node) {
    int flag = 1;
    for (int i = 0; i < heap.size(); i++) {
        if (node == heap.get(i)) {
            flag = 0;
            return true;
        }

    }
    if (flag == 1) {
        return false;
    }
    return false;
}
```

T(N)=O(N)

```java
public void display() {
    for(int i=0;i<tempNum.size();i++) {
        System.out.println("(" + tempNum.get(i) + "," + tempIndex.get(i) + ")");
    }
}
```

T(N)=O(N)

```java
public int find_node(){
    int res = Collections.max(tempIndex);
    int index = tempIndex.indexOf(res);
    int result = tempNum.get(index);
    return result;
}
```

T(N)=O(1)