GTU Department of Computer Engineering
CSE 222/505 - Spring 2021

Homework 4 Report

BAHADIR ETKA KILINÇ
1901042701

1. SYSTEM REQUIREMENTS

Part 1

1. Search for an element

```
public E search(E data)
```

This method takes an item and compares items in priority queue if item is contained in the queue returns item if not returns null.

2. Merge with another heap

```
public void merge(BKPriorityQueue<E> other)
```

This merge method takes an other queue and merges it to itself.

3. Removing ith largest element from the Heap

```
public E removeLargestOf(int index)
```

This methods takes an integer value that is value'th largest element in the queue.

Then deletes from the queue.

4. Iterator's set

```
public E set(E data)
```

This method takes an item to be set that last element returned by the next methods.

Part 2

1. int add (E item)

```
public int add(E data)
```

This method takes an generic type for adding to tree. If element is contained already it increments its occurrence then returns it.

2. int remove (E item)

```
public int remove(E data)
```

This method takes an generic type for removing from tree. If element is contained and it's occurences is bigger than one just decrements its occurrence and returns.

3. int find (E)

```
public int find(E item)
```

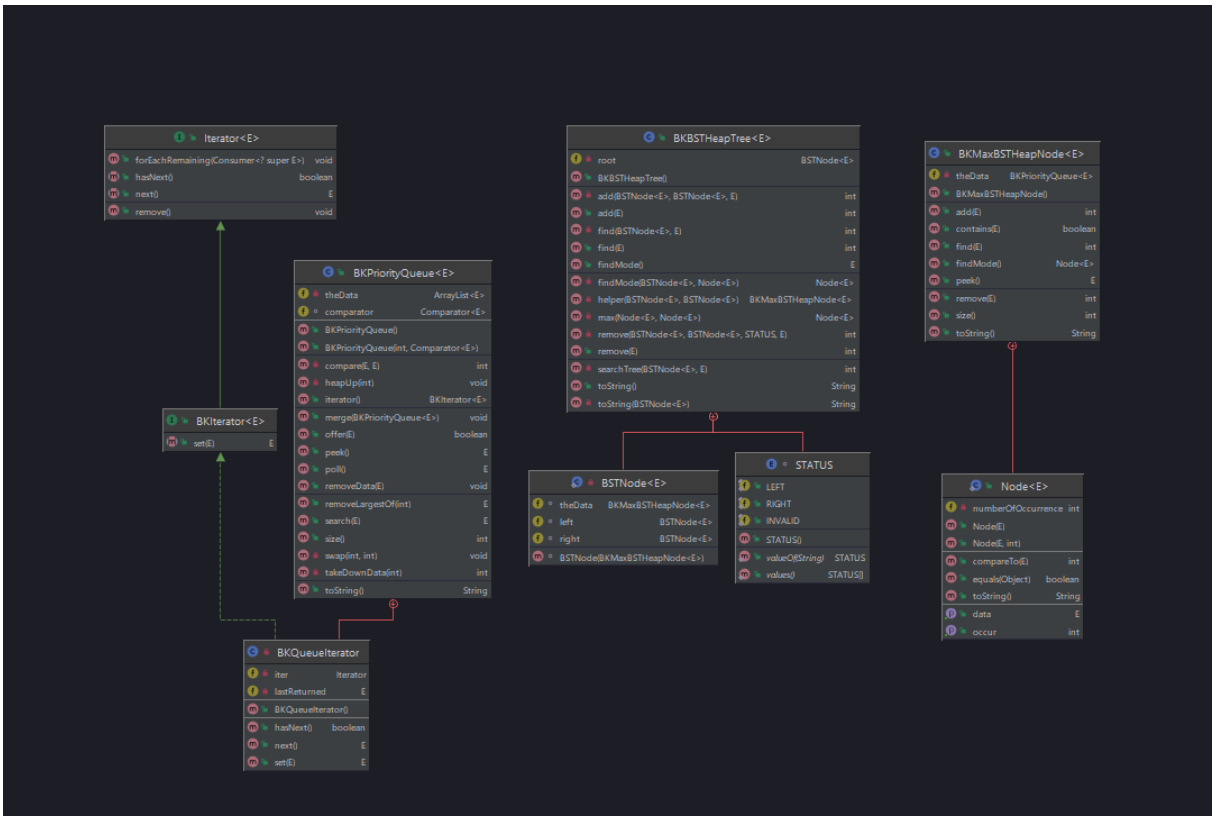
This method takes a generic type. It searches item in tree then returns its occurrence.

4. find_mode ()

```
public E findMode(){
```

This methods brings returns the data with the highest number of occurrences in the tree.

1. Class Diagram



3.Problem Solution Approach

In this project, a queue that holds the max heap structure requested from me and a tree containing them as nodes is requested. First I implemented a BKPriorityQueue class that holds the max heap structure and adds elements according to the order. Later, I implemented a BKMaxHeapNode class that keeps the number of occurrences of each element for the max heap that I will use in the tree. Then I kept this class as data for the tree in the nodes of the tree.

4.Running and Results

Part1

Creating a BKPriorityQueue and adding some integers in it.

```
BKPriorityQueue<Integer> priorityQueue = new BKPriorityQueue<>();

priorityQueue.offer( item: 10);
priorityQueue.offer( item: 20);
priorityQueue.offer( item: 30);
priorityQueue.offer( item: 50);
priorityQueue.offer( item: 15);
priorityQueue.offer( item: 35);
priorityQueue.offer( item: 25);

System.out.println("After adding some Integers");
System.out.println(priorityQueue);
```

```
After adding some Integers
[50, 30, 35, 10, 15, 20, 25]
```

Searching items first one is contained other one is not contained.

```
System.out.println("Searching an element from Priority Queue which is contained");
System.out.println(priorityQueue.search( data: 10));
```

```
Searching an element from Priority Queue which is contained
10
```

```
System.out.println("Searching an element from Priority Queue which is NOT contained");
System.out.println(priorityQueue.search( data: 12));
```

```
Searching an element from Priority Queue which is NOT contained
null
```

Other queue created for merge to first queue.

```java
BKPriorityQueue<Integer> priorityQueue1 = new BKPriorityQueue<>();

priorityQueue1.offer( item: 60);
priorityQueue1.offer( item: 70);
priorityQueue1.offer( item: 80);
priorityQueue1.offer( item: 110);
priorityQueue1.offer( item: 100);
priorityQueue1.offer( item: 90);
priorityQueue1.offer( item: 85);

System.out.println("Other priority queue created for merging");
System.out.println(priorityQueue1);
```

```
Other priority queue created for merging
[110, 100, 90, 60, 80, 70, 85]
```

Second created queue merges to first created

```java
System.out.println("\nBefore merging queue2 to queue1");
System.out.println("\nqueue 1\n"+priorityQueue);
System.out.println("queue 2\n"+priorityQueue1);

priorityQueue.merge(priorityQueue1);
System.out.println("\nAfter merging\n");
System.out.println("queue 1\n"+priorityQueue);
System.out.println("queue 2\n"+priorityQueue1);
```

```
Before merging queue2 to queue1

queue 1
[50, 30, 35, 10, 15, 20, 25]
queue 2
[110, 100, 90, 60, 80, 70, 85]

After merging

queue 1
[110, 100, 85, 50, 90, 70, 80, 10, 30, 15, 60, 20, 35, 25]
queue 2
[110, 100, 90, 60, 80, 70, 85]
```

Removing fifth largest of element from queue

```java
System.out.println("\nRemoving fifth of largest element from queue 1");
priorityQueue.removeLargestOf( index: 5);
System.out.println("queue 1\n"+priorityQueue);
```

```
Removing fifth of largest element from queue 1
queue 1
[110, 100, 85, 50, 90, 70, 80, 10, 30, 15, 60, 20, 35]
```

Iterator testing.

```
System.out.println("\nTesting queue's iterator with the set method");
System.out.println("100 will be set with 14");

BKIterator iter = priorityQueue.iterator();

while(iter.hasNext()){
    if(iter.next().equals(100)){
        iter.set(14);
    }
}
```

```
Testing queue's iterator with the set method
100 will be set with 14
queue 1
[110, 90, 85, 50, 60, 70, 80, 10, 30, 15, 14, 20, 35]
```

Part 2

First creating an array list which will be contained with random numbers.

Before sorting the array list, its items will be added to BSTHeapTree.

```
ArrayList<Integer> randomNumbers = new ArrayList<>();

for(int i=0; i<50; ++i){
    int num = randomGenerator.nextInt( bound: 50);
    randomNumbers.add(num);
}
BKBSTHeapTree<Integer> heapTree = new BKBSTHeapTree<>();

for(int i=0; i<50; ++i){
    heapTree.add(randomNumbers.get(i));
}

System.out.println("\nMax Heap Binary Search Tree\n");
System.out.println(heapTree);
IntegerComparator<Integer> comparator = new IntegerComparator<>();
randomNumbers.sort(comparator);
System.out.println("\nSorted ArrayList\n");
System.out.println(randomNumbers);
```

```
Max Heap Binary Search Tree

[[43 ,3], [35 ,2], [23 ,2], [7 ,1], [9 ,2], [21 ,1], [2 ,2]]
[[27 ,3], [20 ,2], [17 ,1], [10 ,2], [18 ,1], [0 ,1], [14 ,1]]
[[25 ,1], [22 ,2], [15 ,1], [3 ,1], [11 ,1], [5 ,4], [6 ,1]]
[[24 ,1], [19 ,1]]
[[39 ,1], [37 ,1], [38 ,1], [31 ,1], [33 ,2], [32 ,1]]
[[49 ,2], [45 ,2], [46 ,2]]


Sorted ArrayList

[49, 49, 46, 46, 45, 45, 43, 43, 43, 39, 38, 37, 35, 35, 33, 33, 32, 31, 27, 27, 27, 25, 24, 23, 23, 22, 22, 21, 20, 20, 19, 18, 17, 15, 14, 11, 10, 10, 9, 9, 7, 6, 5, 5, 5, 5, 3, 2, 2, 0]
```

Finding mode of tree and arraylist.

```
int arrayMode = findRandomsMode(randomNumbers);
int heapMode = heapTree.findMode();

System.out.println("\nTree's Mode is " + heapMode);
System.out.println("Random Number's Mode is " + arrayMode);
```

```
Tree's Mode is 5
Random Number's Mode is 5
```

Searching random 10 elements which are contained in the arraylist.

```
System.out.println("Searching 10 numbers in the tree\n");
int randIndex;
for(int i=0; i<10; ++i){
    randIndex = randomGenerator.nextInt( bound: 50);
    System.out.printf("Searching %d in the tree\n",randomNumbers.get(randIndex));
    System.out.printf("%d times found\n",heapTree.find(randomNumbers.get(randIndex)));
}
```

```
Searching 10 numbers in the tree

Searching 33 in the tree
2 times found
Searching 0 in the tree
1 times found
Searching 22 in the tree
2 times found
Searching 49 in the tree
2 times found
Searching 14 in the tree
1 times found
Searching 23 in the tree
2 times found
Searching 5 in the tree
4 times found
Searching 32 in the tree
1 times found
Searching 31 in the tree
1 times found
Searching 32 in the tree
1 times found
```

Searching 3 element which are not contained. If item is not contained find method returns -1.

```
System.out.println("\nSearching three elements in tree which are not contained\n");
int randNum;
for(int i=0; i<3; ++i){
    randNum = randomGenerator.nextInt( bound: 50) + 50;
    System.out.printf("Searching %d in the tree\n",randNum);
    int occ = heapTree.find(randNum);
    if(occ == -1)
        System.out.println("Item is not contained.");
    else
        System.out.printf("%d times found\n",heapTree.find(randNum));
}
```

```
Searching three elements in tree which are not contained

Searching 84 in the tree
Item is not contained.
Searching 74 in the tree
Item is not contained.
Searching 83 in the tree
Item is not contained.
```

Deletes randomly 10 elements from tree.

```
System.out.println("\nBefore deleting elements from tree\n");
System.out.println(heapTree);
System.out.println("\nRemoving 10 elements from tree which are contained\n");
for(int i=0; i<10; ++i){
    randIndex = randomGenerator.nextInt( bound: 50);
    System.out.printf("%d will be deleted from tree\n",randomNumbers.get(randIndex));
    System.out.printf("There are %d occurrences left of %d\n",heapTree.remove(randomNumbers.get(randIndex)),randomNumbers.get(randIndex));
}
```

```
Before deleting elements from tree

[[43 ,3], [35 ,2], [23 ,2], [7 ,1], [9 ,2], [21 ,1], [2 ,2]]
[[27 ,3], [20 ,2], [17 ,1], [10 ,2], [18 ,1], [0 ,1], [14 ,1]]
[[25 ,1], [22 ,2], [15 ,1], [3 ,1], [11 ,1], [5 ,4], [6 ,1]]
[[24 ,1], [19 ,1]]
[[39 ,1], [37 ,1], [38 ,1], [31 ,1], [33 ,2], [32 ,1]]
[[49 ,2], [45 ,2], [46 ,2]]


Removing 10 elements from tree which are contained

5 will be deleted from tree
There are 3 occurrences left of 5
43 will be deleted from tree
There are 2 occurrences left of 43
43 will be deleted from tree
There are 1 occurrences left of 43
45 will be deleted from tree
There are 1 occurrences left of 45
31 will be deleted from tree
There are 0 occurrences left of 31
3 will be deleted from tree
There are 0 occurrences left of 3
11 will be deleted from tree
There are 0 occurrences left of 11
27 will be deleted from tree
There are 2 occurrences left of 27
5 will be deleted from tree
There are 2 occurrences left of 5
46 will be deleted from tree
There are 1 occurrences left of 46
```

Trying to delete 3 elements which are not contained in the tree.

```java
System.out.println("\nTry to delete three elements in tree which are not contained\n");

for(int i=0; i<3; ++i){
    randNum = randomGenerator.nextInt( bound: 50) + 50;
    System.out.printf("removing %d in the tree\n",randNum);
    int occ = heapTree.remove(randNum);
    if(occ == -1)
        System.out.printf("Item could not delete from tree because %d is not contained.\n",randNum);
    else
        System.out.printf("There are %d occurrences left of %d\n",heapTree.remove(randNum),randNum);
}
```

```
Try to delete three elements in tree which are not contained

removing 96 in the tree
Item could not delete from tree because 96 is not contained.
removing 52 in the tree
Item could not delete from tree because 52 is not contained.
removing 76 in the tree
Item could not delete from tree because 76 is not contained.
```

```java
System.out.println("\nAfter deletion items from Tree\n");
System.out.println(heapTree);
```

```
After deletion items from Tree

[[44 ,2], [20 ,3], [34 ,2], [14 ,1], [17 ,2], [18 ,1]]
[[42 ,3], [31 ,1], [36 ,1], [15 ,1], [16 ,2], [4 ,1]]
[[26 ,1], [25 ,1], [21 ,1], [7 ,2], [11 ,2], [0 ,3]]
[[22 ,1], [10 ,1], [3 ,2], [9 ,1]]
[[40 ,1], [33 ,1], [28 ,1], [29 ,1]]
[[43 ,1]]
[[46 ,1]]
```