GTU Department of Computer Engineering

CSE 222/505 - Spring 2021

Homework 4 – Part 3

205008003010

Murat SARIBAŞ

-Part1

```
public boolean add(E item) {
    if (theData.contains(item)) O(n)
        return true;
    theData.add(item); O(1)
    int child = theData.size() - 1; O(1)
    up(child); O(logn)
    return true;
}
```

Total: O(n)

```
public E remove() {
    if (theData.isEmpty())   O(1)
        return null;
    E result = theData.get(0); O(1)
    if (theData.size() == 1){
        theData.remove( index: 0); O(1)
        return result;
    }
    theData.set(0, theData.remove( index: theData.size()-1)); O(1)
    down( i: 0); O(logn)
    return result;
}
```

Total: O(logn)

```
public void remove(E item) {
    if (theData.isEmpty())
        return;                          O(1)

    int index = theData.indexOf(item);  O(n)
    if (index == -1)
        return;
    if (size() == 1 && index==0){
        theData.remove( index: 0);       O(1)
        return;
    }
    if (index == size()-1){
        swap( a: 0, index);
        remove(); O(logn)
        return;}
    theData.set(index, theData.remove( index: theData.size()-1)); O(1)
    fixHeap(index); O(logn)
}
```

Total: O(n)

```java
public int size() {
    return theData.size();   O(1)
}
```

```java
public int search(E item) {
    return theData.indexOf(item); O(n)
}
```

```java
public void merge(HeapStruct heap) {
    List<E> otherData = heap.getTheData(); O(1)
    for (int i=0; i<otherData.size(); i++) n times
        add(otherData.get(i)); O(n)
}
                                        Total: O(n^2)
```

```java
public E removeLargestElement(int ith) {
    if (ith <= 0)
        return null;
    if (ith > theData.size())
        return null;                    O(1)

    List<E> arr = getTheData();
    boolean sorted = false;
    E temp;
    while(!sorted) {
        sorted = true;
        for (int i = 0; i < arr.size() - 1; i++) {
            if (compare(arr.get(i),arr.get(i+1))>0) {   O(n^2)
                temp = arr.get(i);
                arr.set(i, arr.get(i+1));
                arr.set(i+1, temp);
                sorted = false;
            }
        }
    }
    temp = arr.get(size()-ith);  O(1)
    remove(temp); O(n)
    return temp;
}
                                        Total: O(n^2)
```

-Part2

```java
private Node<E> add(Node<E> localRoot, E item){
    //if item is not in the tree -- insert it
    if (localRoot == null && !flag){
        addOccurrences = 1;
        return new Node<E>(item);
    }
    //Keep searching because the target item is in the tree
    if (localRoot == null && flag)
        return null;
    //If the heap is not full and the target item is not in the tree
    boolean bool = (localRoot.data.size() < maxHeapSize) && (!flag);
    if ( bool || localRoot.data.indexOf(item)>-1){
        flag = false;
        addOccurrences = localRoot.data.add(item);
        return localRoot;
    }
    // item is equal to localRoot.data
    else if (item.compareTo(localRoot.data.theData.get(0).getData()) == 0){
        return localRoot;
    }
    // item is less than localRoot.data
    else if(item.compareTo(localRoot.data.theData.get(0).getData()) < 0){
        localRoot.left = add(localRoot.left, item);
        /**if flag == true that means the target item is not in the
         * left subtree, search item in the right subtree
         */
        if (flag)
            localRoot.right = add(localRoot.right, item);
        return localRoot;
    }
    // item is greater than localRoot.data
    else {
        localRoot.right = add(localRoot.right, item);
        return localRoot;
    }
```

O(1)

O(1) because heap size is fixed

O(1)

Total : O(logn)

```java
private Node<E> add(Node<E> localRoot, E item, MaxHeap<E> heap){
    if (localRoot == null){
        // insert the heap in the tree with all items
        E value;
        Node<E> returnNode = new Node<E>(item);
        int count = heap.theData.get(0).getCount();
        if (count>1)
            for (int i=1; i<count; i++)
                returnNode.data.add(item);
        for (int i=1; i<heap.theData.size(); i++){
            value = heap.theData.get(i).getData();
            count = heap.theData.get(i).getCount();
            for (int j=0; j<count; j++)
                returnNode.data.add(value);
        }
        return returnNode;
    }
    else {
        int compResult = item.compareTo(localRoot.data.theData.get(0).getData());
        // item is less than localRoot.data
        if (compResult < 0)
            localRoot.left = add(localRoot.left, item, heap);
        // item is greater than localRoot.data
        else
            localRoot.right = add(localRoot.right, item, heap);
        return localRoot;
    }
}
```

O(1) because heap size is fixed

Total: O(logn)

```java
public int add(E item) {
    find(item); O(logn)
    root = add(root, item); O(logn)
    return addOccurrences;
}
```

Total: O(logn)

```java
private int find(Node<E> localRoot, E target){
    if (localRoot == null)
        return -1;
    int index = localRoot.data.indexOf(target);
    //if target item is in the tree
    if (index > -1){
        flag = true;
        return localRoot.data.theData.get(index).getCount();
    }
    else {
        int compResult = target.compareTo(localRoot.data.theData.get(0).getData());
        // item is equal to localRoot.data
        if (compResult == 0)
            return localRoot.data.theData.get(0).getCount();
        // item is greater than localRoot.data
        else if(compResult > 0)
            return find(localRoot.right, target);
        // item is less than localRoot.data
        else{
            int count = find(localRoot.left, target);
            // if item is not in the left subtree, search the item in
            // the right subtree
            if (!flag)
                count = find(localRoot.right, target);
            return count;
        }
    }
}
```

O(1) because heap size is fixed

Total: O(logn)

```java
public int find(E target) {
    flag = false;
    return find(root, target);
}
```

O(logn)

```java
private void find_mode(Node<E> localRoot){
    if (localRoot == null)
        return;
    for (int i=0; i<localRoot.data.theData.size();i++){
        int count = localRoot.data.theData.get(i).getCount();
        if (count > modeCount){
            modeItem = localRoot.data.theData.get(i).getData();
            modeCount = count;
        }
    }
    find_mode(localRoot.left);
    find_mode(localRoot.right);
}
```

O(1) because heap size is fixed

Total: O(n)

```java
public E find_mode() {
    find_mode(root);     O(n)
    System.out.println("Mode Count: " + modeCount + " Mode: " + modeItem.toString());
    return modeItem;
}
```