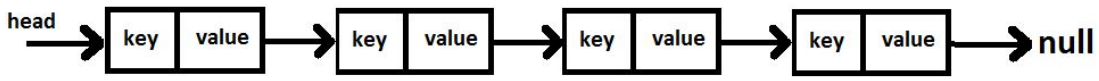
DATA STRUCTURE
Homework : The Bag ADT

Teacher: Dr. Öğr. Üyesi Didem ABİDİN

Students : Şaban Koparal 160316013

Anıl Sağlam 160316039

Linked List Structure



key : N key
value : int value

Introduction

We have created a node object of type N for the linked list structure. We have given key, value, next attributes to this node object. And we added our standard getter,setter and getNext methods. In order to control the number of same nodes in our additional data structure, we have added the increaseValue and decreaseValue methods.

Methods

Add (N key) :

```
/**
 * @param N key
 * @return void
 */
public void add(N key)
{
    if (isEmpty()){
        Node<N> newNode = new Node<N>(key);
        head = newNode;
    }else{
        boolean is_there = false;
        Node<N> currentNode = head;
        Node<N> lastNode = head;

        while (currentNode != null) {
            if (currentNode.getKey().equals(key)){
                currentNode.increaseValue();
                is_there = true;
                break;
            }
            lastNode = currentNode;
            currentNode = currentNode.getNext();
        }

        if (!is_there){
            Node<N> newNode = new Node<N>(key);
            lastNode.setNext(newNode);
        }

    }

}
```

The add method first starts by checking if the list is empty. If it's empty, add it to the top of the list. If not, we check whether the incoming value is in the list and increase the value by one. And we're adding the new value to the end of the list.

Clear() :

```
/**
 * @return void
 */
public void clear() {
    head = null;
}
```

We change the head value to null.

Contains(N key) :

```
/**
 * @param N key
 * @return boolean
 */
public boolean contains(N key)
{
    Node<N> currentNode = head;
    while (currentNode != null){
        if (currentNode.getKey().equals(key)){
            return true;
        }
        currentNode = currentNode.getNext();
    }

    return false;
}
```

We loop the list until next value is null. and if the parameter value that comes with the key value is equal, we are making return true. If there is no match, we return false.

distictSize() :

```
/**
 * @return int
 */
public int distictSize()
{
    Node<N> currentNode = head;
    int counter = 0;
    while (currentNode != null){
        counter++;
        currentNode = currentNode.getNext();
    }

    return counter;
}
```

We return the list elements until the next value is null, and we increment the counter.

equals(Object obj) :

```
/**
 * @param Object obj
 * @return boolean
 */
public boolean equals(Object obj)
{
    if (obj == this){
        return true;
    }
    if (obj == null){
        return false;
    }
    if (obj instanceof Bag){
            return true;
    }
    else{
        return false;
    }
}
```

if the incoming object is an instance of the main object, we return true. Otherwise we return false.

elementSize(N key) :

```
/**
 * @param N key
 * @return int
 */
public int elementSize(N key)
{
    Node<N> currentNode = head;
    int elementsize = 0;
    while (currentNode != null){
        if (currentNode.getKey() == key){
            elementsize = currentNode.getValue();
            break;
        }
        currentNode = currentNode.getNext();
    }

    return elementsize;
}
```

We search the list according to the key value and return the node value. If there is no such record, we return 0.

remove(N key) :

```
/**
 * @param N key
 * @return boolean
 */
public boolean remove(N key)
{

    if (!contains(key)){
        System.out.println("The item '" + key + "' does not exists");
    } else{
        Node<N> currentNode = head;
        Node<N> previousNode = head;
        while (currentNode != null){
            if (currentNode.getKey().equals(key) ){
                if (currentNode == head){
                    currentNode.decreaseValue();
                    if (currentNode.getValue() < 1){
                        head = head.getNext();
                        return false;
                    }
                    return true;
                }
                else{
                    currentNode.decreaseValue();
                    if (currentNode.getValue() < 1){
                        previousNode.setNext(currentNode.getNext());
                        currentNode.setNext(null);
                        return false;
                    }
                    return true;
                }
            }
            previousNode = currentNode;
            currentNode = currentNode.getNext();
        }
    }
    return false;
}
```

We create two variables with the value of head as current node and previous node. We also check if there is an incoming value list. If not, we return a message. We loop through the list until the node equals null. if the value is equal to the node, we reduce the value by one. However, if the value of the key to be deleted is less than 1 and head is set to head, we assign next value. If it is not head but the value is less than 1, we set the previous node with the active node.

size() :

```
/**
 * @return int
 */
public int size()
{
    int size = 0;
    Node<N> currentNode = head;
    while (currentNode != null){
        size = size + currentNode.getValue();
        currentNode = currentNode.getNext();
    }
    return size;
}
```

We return until the node equals null. and we add the value of each node to you variable.

isEmpty() :

```
/**
 * @return boolean
 */
public boolean isEmpty()
{
    if (head == null){
        return true;
    } else{
        return false;
    }
}
```

We return true if head is null, otherwise we return false.

toString() :

```
/**
 * @return String
 */
public String toString()
{
    if(isEmpty()){
        return "The Bag is empty";
    }else {

        Node<N> currentNode = head;

        String text = "Bag Object : ";
        while (currentNode != null){
            for (int i = 0; i < currentNode.getValue(); i++){
                text += "[" + currentNode.getKey() + "]";
            }
            currentNode = currentNode.getNext();
        }
        return text;
    }
}
```

We return until the node value is null, and we combine the value value of each node with the variable named text in the value of key. and return the text variable.