T.C

MANİSA CELAL BAYAR UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

DATA STRUCTURES PROJECT REPORT

Baran GEZENOĞLU

170315008

Lecturer

Dr. Didem ABİDİN

Laboratory Assistant

Emre ŞATIR

MANİSA , 2019-2020

CSE 2105 – DATA STRUCTURES 2019 – 2020 FALL SEMESTER

PROJECT REPORT THE BAG ADT

We know that ; there are many way to create a bag class.In this report i will explain this ways and i'll explain which i choose .I choosed which is more efficient to me . It is Binary Search Tree. Also i create a bag class with linked list . I'll add my codes for binary search tree and linked list in this report.

SOME WAYS

1)Arrays

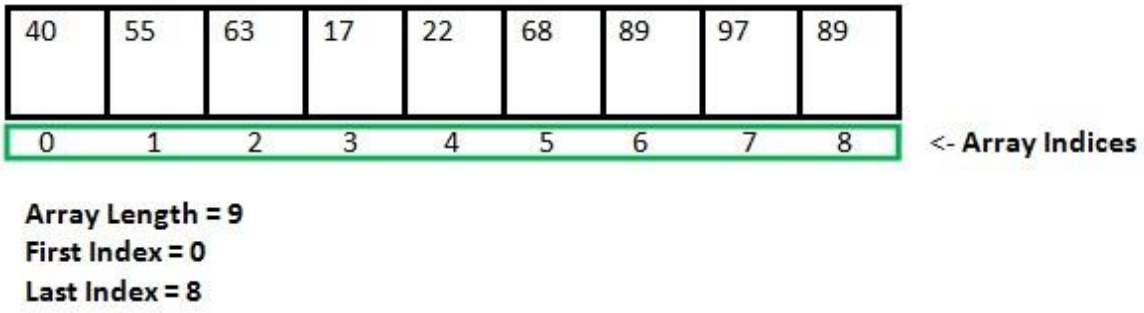

Figure 1. Array Scheme [1].

We can implement the bag by using 2 arrays . First array can hold the Data.This data should be every type . Because as you can see it is 'E' . So it is generic . Second array can hold count . We need count because we should use in methods such as ; size , distictsize , elementsize .

Are arrays efficient?

- We can't know array's upper limit . So we exceed the limit we should create and copy all stuck to new array .
- If we want delete or insert new element we must change locations in array .

So arrays are not efficient .

2)Linked List

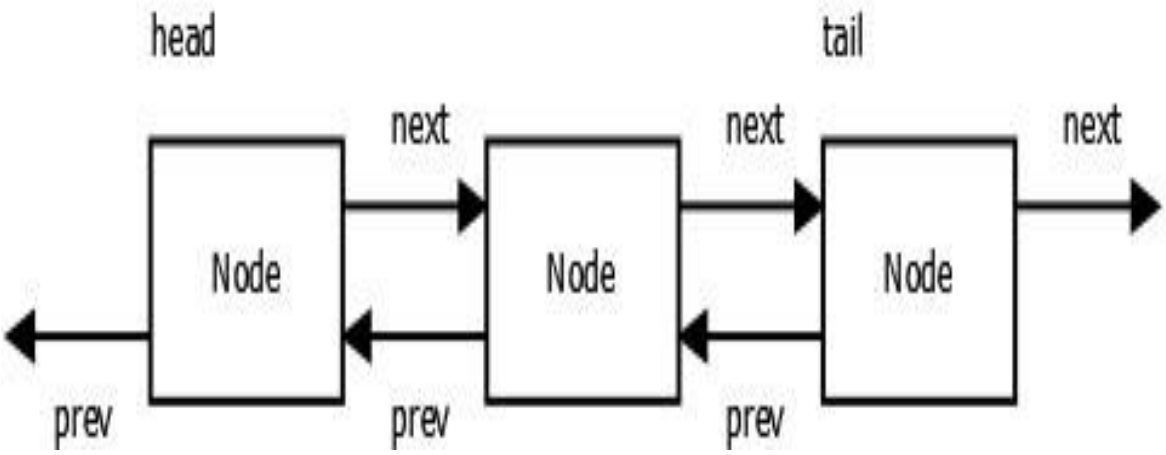

Figure 2 . Linked List Scheme [2].

Why it can be used ?

- Its has dynamic size unlike arrays.
- We have a pointer . In this way insertion and deletion are easly .

We have three classes . Ther are ;

-Node

-Bag

-Test

And of course i wrote with anytype (Generic , E).

Coding

1)First class is Node .

```
public class Node<E> {
    private int data ;
    private E key ;
    private Node<E> next;

    public Node(E k) {
        data = 1;
        key=k;
        next=null;
    }
```

private int data : is working for count .

private E key : key is just a variable name.

Private Node<E> next : its working for get and set next . After this i wrote getters and setters in Node class .

2)Second class is Bag.

My methods are here .

isEmpty();

```
1
2  public class Bag<E> {
3      private Node<E> head;
4      public Bag() {
5          head=null;
6      }
7
8      public boolean isEmpty() {
9          if(head==null) {
10             return true;
11         }
12         else
13             return false;
```

If first node (head) is empty thats mean bag is empty .In this situation program returns true . If head node is not empty program returns false .

Add();

```
public void add(E key) {
    if(isEmpty()) {
        Node<E> nNode = new Node<E>(key);
        head=nNode;
    }
    else {
        Node<E> current=head;
        Node<E>last=head;
        boolean found= false;
        while(current!=null) {
            if(current.getKey().equals(key)) {
                current.incrementData();
                found=true;
                break;
            }
            last =current;
            current=current.getNext();
        }
        if(!found) {
            Node<E> nNode = new Node<E>(key);
            last.setNext(nNode);
        }
    }
```

if bag is empty , create a new node and assing that is head. Control that , does new element is exist in bag ? if yes then increment data and break.

If no create a new node which has new element and add .

Contains();

```
public boolean contains(E key) {
    Node<E> current = head;
    while(current!=null) {
        if(current.getKey().equals(key)) {
            return true ;
        }
        current=current.getNext();
    }
    return false;
```

Traverse in bag if the element is found ,return true .Else return false .Contains type is boolean because our result is true or false .

Size();

```
82    public int size() {
83        Node<E> current= head;
84        int size=0;
85        while(current !=null) {
86            size=size + current.getData();
87            current=current.getNext();
88
89        }
90        return size;
91    }
```

Data holds the item which refers to how many instances of element is located in the Bag. Return size so total of instances are located in the Bag.

Size's type is integer because we want see integer result Distictsize();

```
92    public int distictSize() {
93        Node<E> current = head;
94        int counter =0;
95        while(current!=null) {
96            counter ++;
97            current=current.getNext();
98        }
99        return counter;
100   }
```

Distict size is counting the number of in Bag . For example ; Bag=[1,2,3,4,4,] so distict size is 4 . Distict size's type is integer because we want see integer result.

Elementsize();

```
101    public int elementSize(E key) {
102        Node<E> current = head;
103        int elementsize=0;
104        while(current!=null) {
105            if(current.getKey()==key) {
106                elementsize=current.getData();
107                break;
108            }
109            current=current.getNext();
110        }
111        return elementsize;
112    }
```

Traverse in bag . If the key(or data or item whatelse) found assain the key elementsize and break. Return the elementsize. Elementsize's type is integer because we want see integer result.

ToString();

```
116    public String toString() {
117        if(isEmpty()) {
118            return "Bag is empty";
119        }
120        Node<E> current = head;
121        String string = "Bag = ";
122        while(current != null) {
123            for(int i =0;i<current.getData();i++) {
124                string +="["+current.getKey()+ "]";
125            }
126            current=current.getNext();
127        }
```

If bag is empty returns "Bag is empty".Else create a new string and loop executes as many as instances of the same item . toString type is string because it is string.

Clear();

```
113    public void clear() {
114        head=null;
115    }
```

So if we assing null to head bag will be destroyed.

Equals();

```
130    public boolean equals(Bag<E> b) {
131
132           return equals(head, b.head);
133
134
135       }
136    private boolean equals(Node<E> a, Node<E> b) { // i took help in here from my class mate
137           if(a==null || b==null) {
138               return a==b;
139           }
140           if(!a.getKey().equals(b.getKey()))
141               return false;
142           if(a.getData()!=b.getData())
143               return false;
144           return equals(a.getNext(),b.getNext()) ;
145       }
```

So there is overload . public for acces to user . private for devoloper . İf nodes are equals to null , they will be equal each other because they have same value . it is null. İf their keys not equal each other returns false And if they have different value in bags returns false again. Equals type is boolean beucase we are questioning true or false.

Remove();

```
49     public boolean remove(E key) {
50         if(!contains(key)) {
51             System.out.println("The data " + key + " cannot be found here");
52         }
53         else {
54             Node<E> current = head;
55             Node<E> previous= head;
56             while(current!=null) {
57                 if(current.getKey().equals(key)) {
58                     if(current==head) {
59                         current.decreaseData();
60                         if(current.getData()<1) {
61                             head=head.getNext();
62                             return false;
63                         }
64                         return true;
65                     }
66                     else {
67                         current.decreaseData();
68                         if(current.getData()<1) {
69                             previous.setNext(current.getNext());
70                             current.setNext(null);
71                             return false;
72                         }
73                         return true;
74                     }
75                 }
76                 previous=current;
77                 current=current.getNext();
78             }
79         }
80         return false;
```

I left it to the end because it is longer than others .

First of all i want to say that i didnt write remove method by myself . I searched many sites and codes on internet . It is most compatible form for my Bag.So i analyzed it , i understood it and i wrote it with my variables .

First condition is controling is key in here or not ? . if it is not program prints cannot be found . If we have the item we traverse in bag and find it . After we decleare head to the item and ve decrease it . In the end returns true . Of course we decleare next to previous for link connection . If we don't do this we cant have link connection and we can lose the rest of bag.

3)Binary Search Tree

Definition: A binary search tree is a binary tree in which the key values in the left node is less then the root (root <0) and the key values in the right node is greater than the root (root >0).

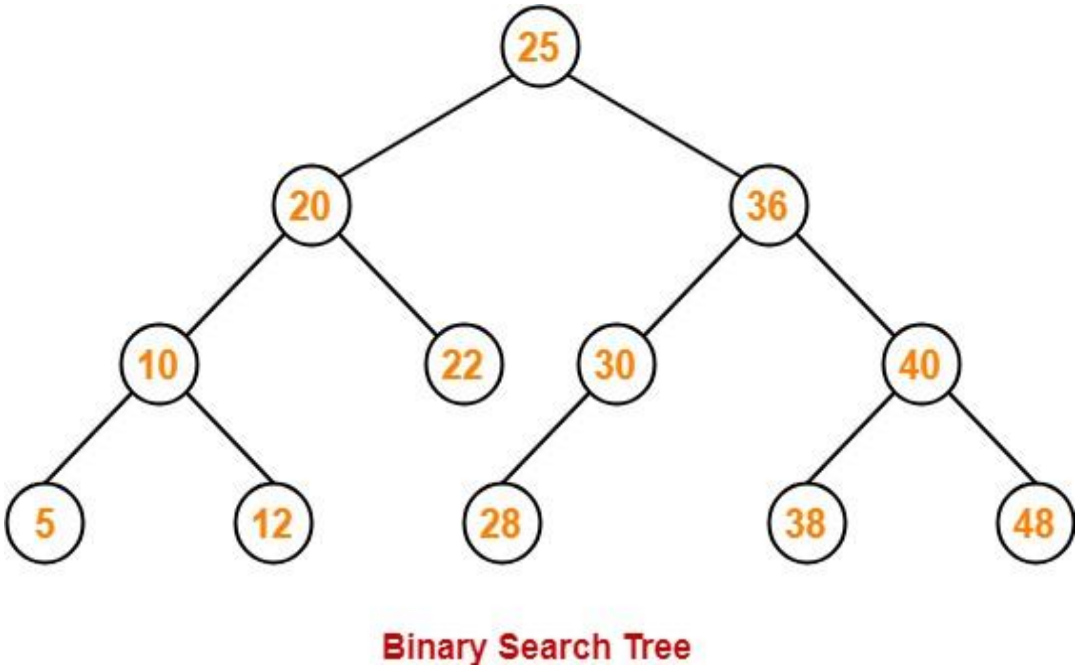

Figure 3. Binar Search Tree Scheme [3].

Why can be used?

- Dynamic size
- Faster

We have three classes like linked lists . But they diffirent .

The classes ;

-Node

-Bag(E extends Comparable) because i used compareTo for removing and traverse .

-Test

Coding ;

1)First class is Node;

```
1 
2 public class Node<E>
3 {
4     public E data;
5     public int count;
6     public Node<E> left;
7     public Node<E> right;
8 
9     public Node(E d)
10    {
11        this.data=d;
12        this.count=1;
13        this.right=null;
14        this.left=null;
15    }
```

Created a Node class .

-Pubic E data→For generic Data . Data is could be anytype .

-Public int count → Count for size , distict size , elementsize . count's type is int because we return integer value .

-Public Node<E> left and right → we'll create a bag with binary search tree and we need left branches and right branches .

After this i created getters and setter in this class .

```
    public Node<E> getRight() {
        return right;
    }

    public int getCount() {
        return count;
    }

    public Node<E> getLeft() {
        return left;
    }

    public void setData(E data) {
        this.data = data;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public void setLeft(Node<E> left) {
        this.left = left;
    }

    public void setRight(Node<E> right) {
        this.right = right;
    }
```

2)Second class is Bag;

This class contains methods for test classes .

```
public class Bag<E extends Comparable<E>> {

    Node<E> root;

    public Bag() {

        root=null;
    }

    public Bag(Node<E> newNode)
    {
        root=newNode;
    }
```

Bag<E extends Comparable<E>> → Its extends comparable because as i see in moodle forum if we use bst for bag we should extends comparable to using compareTo. And created a method for creating new node . Root equals to this node.

IsEmpty();

```java
public boolean isEmpty()
{
    if (root == null)
    {
        return true;
    } else
    {
        return false;
    }
}
```

IsEmpty type is boolean . Because the method is questioning a situation to is it true or false .

Add();

```java
public  void add(E data)
{

    if (isEmpty()) {
        Node<E> newNode = new Node<E>(data);
        root = newNode;

    }

    else{
        Node<E> current=root;
        while (true)
        {
            if (data.compareTo(current.getData()) <0 )
            {
                if (current.getLeft()!=null)
                {
                    current=current.getLeft();
                }
                else
                {
                    Node<E> newNode=new Node<E>(data);
                    current.setLeft(newNode);
                    break;
                }
            }
                else if(data.compareTo(current.getData()) >0)
            {
                if (current.getRight()!=null)
                {
                    current=current.getRight();
                }
                else
                {
                    Node<E> newNode=new Node<E>(data);
                    current.setRight(newNode);
                    break;
                }
            }
            else if(data.compareTo(current.getData()) ==0)
            {

                current.incrementCount();
                break;
            }
        }
    }
}
```

-If the bag is empty create a new node and assing to root .

-If comparison <0 its mean that the new node must add to left . If left side is not empty go left sub tree . Set left child of root and break .

-If comparison >0 its mean that the new node must add to right . If right side is not empty go right sub tree . Set right child of root and break.

If the node already in tree just increase the count .

Contains();

```
74    public boolean contains(E data)
75    {
76        Node<E> current=root;
77        boolean found=true;
78        while (found) {
79            if (data.compareTo(current.getData()) < 0) {
80                if (current.getLeft() != null) {
81                    current = current.getLeft();
82                }
83                else break;
84
85
86            }  if (data.compareTo(current.getData()) > 0) {
87                if (current.getRight() != null) {
88                    current = current.getRight();
89                }
90                else break;
91
92            }  if (data.compareTo(current.getData()) == 0) {
93
94                found=false;
95            }
96        }
97        return !found;
```

If data less than root go left branches and if it does not have left break .

If data greater than root go right branches and if it does not have right break .

If data is equal to root node , assing false to found .

Its working like while true .

FindMinFromRight();

```
    private Node<E> findMinFromRight(Node<E> node) {
        while(node.getLeft() !=null)
        {
            node=node.getLeft();
        }
        return node;
    }
```

Returns the minimum node in tree.

Remove();

```
    public boolean remove(E data) {
        if (!contains(data)) {
            System.out.println("The item '" + data + "' is not located in the Bag");
            return false;
        }
        else {
            Node<E> temp = remove(root, data);
            if (contains(temp.getData())) {
                return true;
            } else
                return false;
        }
    }
```

```
    private Node<E> remove(Node<E> root,E data) {


        Node<E> current = root;
        if (current == null) {
            return current;
        }
        if (data.compareTo(current.getData()) < 0) {
             current.setLeft(remove(current.getLeft(), data));
        }
        else if (data.compareTo(current.getData()) > 0) {
            current.setRight( remove(current.getRight(), data));
        }
        else {
            if (current.getCount() > 1) {
                current.decreaseCount();
                return current;
            }
            else {
                if (current.getLeft() != null && current.getRight() != null ) {
                    Node<E> MinFromRightSubTree=findMinFromRight(current.getRight());
                    current.setData(MinFromRightSubTree.getData());
                    remove(current.getRight(),MinFromRightSubTree.getData());
                }
                else if (current.getLeft() != null ) {
                    current=current.getLeft();
                }
                  else if (current.getRight() != null ) {
                      current=current.getRight();
                 }
                 else
                {
                    current=null;
                }
            }
        }
        return current;
    }
```

There is an overload. Thanks to public method we can take node and root .In the private method we make comparison . If data less than current node , recall the method by giving a left child as a parameter .

If data greater than current node , recall the method by giving a right child as a parameter .

If the count of item is greater than 1 which means that if there are more than 1 instances of the same item, decrease the count of item.

If the node to be deleted has left and right children, find the minimum item in the right sub tree and copy its value and recall method to delete minimum node from right sub tree.

If the node to be deleted has only left child .

If the node to be deleted has only right child.

If the node to be deleted does not have a child . (Leaf)

Note : I had a hard time writing this method and searched on internet . I'll add this sites in "reseaerches" part .

Clear();

```
159     public void clear()
160     {
161         root=null;
162     }
```

We assing null to root and we destroy bag .

Distictsize();

```
164     public int distictSize() {
165         return (distictSize(root));
166     }
167
168     private int distictSize(Node<E> node)
169     {
170         if (node==null) {
171             return(0);
172         }
173         else {
174         return (distictSize(node.getLeft()) + 1 + distictSize(node.getRight()));
175         }
176     }
```

There is an overload . I create a recursion in the method to traverse . Sum the number of node in tree and return it . They must be unique in the tree.

Size();

```
205    public int size()
206    {
207        return size(root);
208    }
209
210    private int size(Node<E> node) {
211        if (node==null) {
212            return(0);
213        }
214        else {
215        return node.getCount()+ size(node.getLeft()) + size(node.getRight());
216        }
217    }
218
```

And again there is an overload . Public method is using recursion to traverse .

After that sum the count of each node and return it .

Elementsize();

```
178    public int elementSize(E data) {
179        Node<E> current = root;
180
181        int elementsize = 0;
182        if (!contains(data)) {
183            System.out.print(data+" is not found in the Bag, "+data+":");
184             return 0;
185        }
186           else {
187            while (true) {
188                if (data.compareTo(current.getData()) < 0) {
189                    if (current.getLeft() != null) {
190                        current = current.getLeft();
191                    }
192                } else if (data.compareTo(current.getData()) > 0) {
193                    if (current.getRight() != null) {
194                        current = current.getRight();
195                    }
196                } else if (data.compareTo(current.getData()) == 0) {
197                     elementsize = current.getCount();
198                    break;
199                }
200            }
201            return elementsize;
202        }
203    }
```

If node is not located in Bag,

If node is located in Bag ,

If data is less than root node , go left side,

If data is greater than root node , go right side ,

Found the wanted node , assing the count of node to elementsize and break ,

Return elementsize.

ToString();

```
220    public String toString()
221    {
222       return toString(root);
223    }
224
225    private String toString(Node<E> root) {
226        if(isEmpty())
227        {
228            return "Bag is empty";
229        }
230       else {
231            Node<E> current = root;
232            String result = "";
233            if (current == null) {
234                return "";
235            }
236            result += toString(current.getLeft());
237            result += "[" + current.getData().toString() + " : " + current.getCount() + "]";
238            result += toString(current.getRight());
239
240            return result;
241
242        }
```

As before , i used recursion to traverse with recursion .

If the bag is empty we'll see that "Bag is empty ",

Else we'll see a string for example;
[1 : 1] [2 : 1] [3 : 1].

Equals();

```
245    public boolean equals(Bag<E> b) {
246
247        return equals(root, b.root);
248
249
250    }
251    private boolean equals(Node<E> a, Node<E> b) { // i helped here from my class mate
252        if(a==null || b==null) {
253            return a==b;
254        }
255        if(!a.getData().equals(b.getData()))
256            return false;
257        if(a.getCount()!=b.getCount())
258            return false;
259        return equals(a.getLeft(),b.getLeft()) && equals(a.getRight(), b.getRight());
260    }
261
262
263
264 }
```

Again there is an overlaod for recursion .

Equal's type is boolean because we are questioning condition and we should return true or false .

If two root node is equals each other returns a==b (true);

If the nodes are not equal for two bag returns false ,

If the size of bags are different returns false ,

If branches of bags are different returns false ,

There is a no seperate condition for size because we already questioned on second condition ;

If(!a.getData().equals(b.getData()).

REFERENCES

- Figure 1 → https://www.geeksforgeeks.org/arrays-in-java/
- Figure 2 → https://www.laurentluce.com/posts/least-frequently-used-cacheeviction-scheme-with-complexity-o1-in-python/
- Figure 3 → https://www.gatevidyalay.com/binary-search-trees-datastructures/
- https://emresatir.net/dersler/veri-yapilari-ve-algoritmalar/agac-dolasmaalgoritmalari/
- https://www.d.umn.edu/~jallert/cs1/projects/TheBagADT.pdf
- https://stackoverflow.com/questions/14651105/building-a-bag-class-in-java
- https://stackoverflow.com/questions/14744056/bag-class-implemented-injava
- https://algs4.cs.princeton.edu/13stacks/LinkedBag.java.html
- https://algs4.cs.princeton.edu/13stacks/Bag.java.html
- https://stackoverflow.com/questions/14651105/building-a-bag-class-in-java
- https://stackoverflow.com/questions/9597188/the-most-efficient-way-totest-two-binary-trees-for-equality
- https://www.youtube.com/watch?v=6ha35PJCSfo&feature=youtu.be
- https://github.com/calingribble/BagProject
- https://github.com/viperior/java-bag