

**Gebze Technical University  
Computer Engineering**

**CSE 222 - 2018 Spring**

**HOMEWORK 4 REPORT**

**Burak Özdemir  
141044027**

# 1 INTRODUCTION

## 1.1 Problem Definition

**Part\_1:** Genel agac yapısını binary agac yapısı seklinde implement etmemiz istendi .Ek olarak levelOrderSearch ve postOrderSearch metodları yazmamız istendi.

**Part\_2:** Multidimensional Search Tree yapısı kuruldu. Multidimensional özelliği vector ile sağlandı .Ayrıca sınıf yapısı SearchTree interface inden implement edildi.

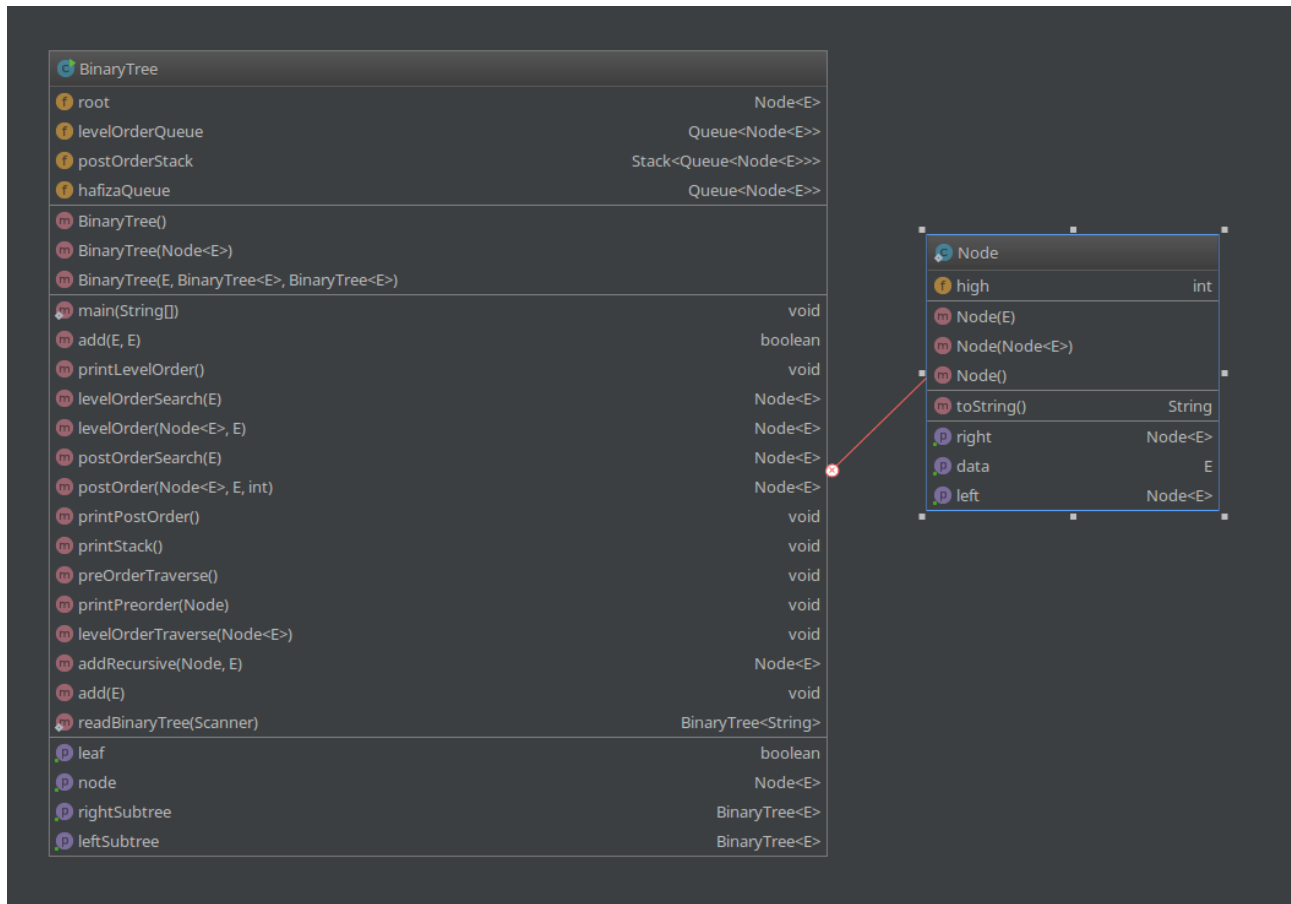
## 1.2 System Requirements

→Java JDK(1.8)






# 2 METHOD

















## 2.1 Class Diagrams

PART 1:



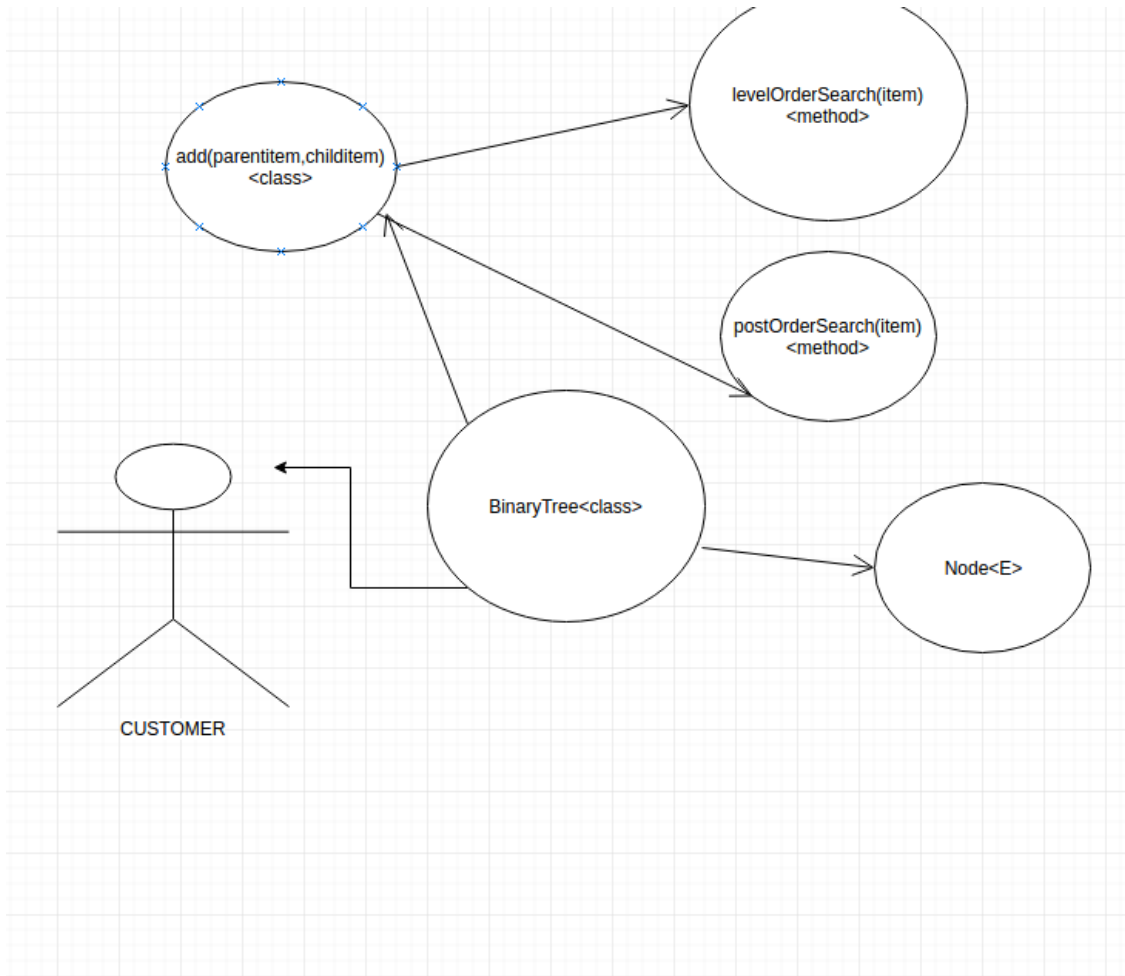
## PART 2:

SearchTree		
	add(Vector<E>)	boolean
	contains(Vector<E>)	boolean
	find(Vector<E>)	Vector<E>
	delete(Vector<E>)	Vector<E>
	remove(Vector<E>)	boolean

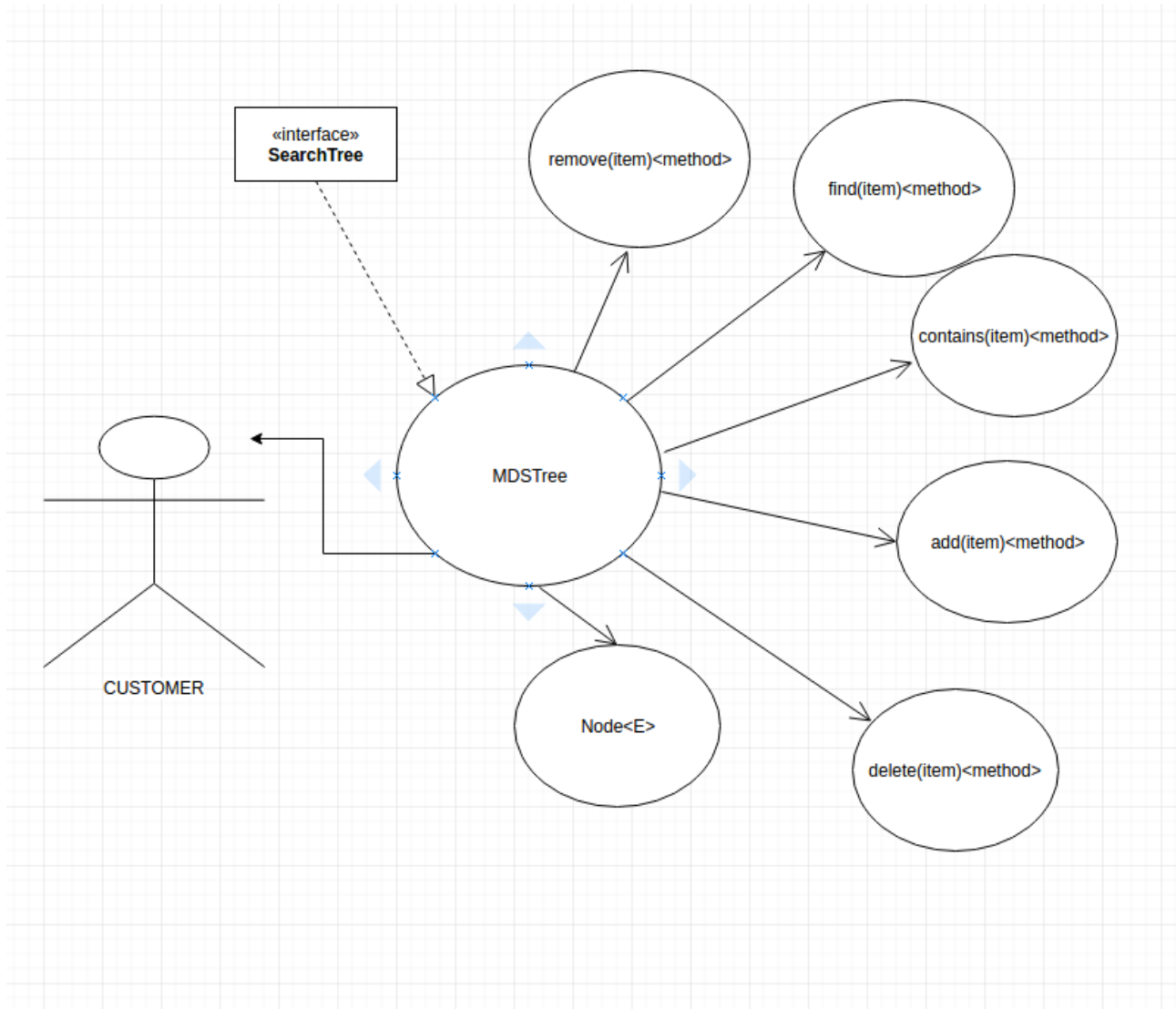
MDSTree		
	indis	Integer
	root	Node<E>
	deleteReturn	Vector<E>
	main(String[])	void
	add(Vector<E>)	boolean
	addDimensional(Node<E>, Vector<E>)	Node<E>
	contains(Vector<E>)	boolean
	contains(Node<E>, Vector<E>)	boolean
	find(Vector<E>)	Vector<E>
	findDimensional(Node<E>, Vector<E>)	Vector<E>
	findNode(Node<E>, Vector<E>)	Node<E>
	delete(Vector<E>)	Vector<E>
	deleteDimensional(Node<E>, Vector<E>)	Node<E>
	levelOrder(Node<E>)	Queue<Vector<E>>
	findParent(Node<E>, Vector<E>)	Node<E>
	remove(Vector<E>)	boolean

## 2.2 Use Case Diagrams

PART1:



## PART2:



### 2.3 Other Diagrams (optional)

Add other diagrams if required.

### 2.4 Problem Solution Approach

#### 2.4.1→Genel Çözüm

**Part\_1:** Genel agac yapısı binary olarak yazılmasındaki çözüm ilk cocugun sol nod ile ve diger cocukların ilk cocuktan itibaren çocukların sag nodu ile ulaşılması ile olmuştur . Yani kardes olan noda sag nod ile child olan noda sol nod ile ulaşılmıştır.level ve post order search kısmında ise Queue yapısı kullanılmıştır.

**Part\_2:** Binary Search Tree sınıfı SearchTree interface inden implement edilmiştir.İçerdeki nod sınıfının datası vector tıpında tutulmuştur.Yani cok boyutluluk vectorun içeriğiyle alakalıdır.

## 2.4.2→Classes and Their Skills

Time Complexity=TC

### 2.4.2.1→Part1

**BinaryTree:**Sınıf içerisinde node sınıfı tutmaktadır.Genel agac yapısı üsttede belirtildiği gibi binary olarak represent edilmiştir . Sol nod ile alt cocuklara sag ile aynı seviyedeki kardes nodlara gıdılmektedir.Genel agac metodları ıplement edilmistir.

+add(parentitem,childitem):Bu metod parametredeki parent itemin en snundakı cocugunun yanına parametredeki cocugu new node olarak ekler.icerisinde levelOrderSearch kullanulduğı için asıl zamanı bu metod ile alır . Yani  $TC=O(n^2)$  dir. Zaman karmasıklığı  $n + n-1 + n-2 \dots 1$  e kadar gttiği için bu degelerin toplamı  $n(n+1)/2$  seklinde gelir . Ordanda  $n^2$  karmasıklığı gelir

+levelOrderSearch():Search yaparken yapıda Queue veriyapısı kullanılmıstır. Her sol cocuk Queue yapısına yeni Node eklenir ve devam isleminde sag taraftakiler ekrana bastırılır .Metodun sonunda Queue dan eleman cıkartılarak recursive isleme devam edilirı . Her recursive ile  $n^2$  karmasıklığı elde edilir . Worstcase durumunda  $O(n^2)$  karmasıklığı gelir.

+postOrderSearch():Burdada stack yapısı kullanılmıstır . Kardes nodlar Queue seklinde stacka konur.Son child noda kadar bu seklıde ilerlenir . Normalde  $O(n)$  karmasıklıgnda olan metod ek bır metod ile(baska seklıde cozemedim) stacktan Queue yapılarının cıkarılıp tekrardan Queue yapılarından nodların cıkartılması ile metod  $O(n^2)$  karmasıklıgında olmustur .

### 2.4.2.2→Part 2

**SearchTree:**Bu interface te add,contains,find,delete ve remove metodları vardır. MDSTree sınıfı ile implement edilecektır bu interface

**MDSTree:**İçerisindeki node data yapısı olarak Vector ile çalışır. Yani multidimensional özelliğı vector ile saglanmıstır.Ayrıca generic veri tipi comparable sınıfından extend edilmiştir.SearchTree interface indeki metodlarıda implement eder. Yapı Binary Search yapısıdır .Node eklenırken mesela sürekli seviyedeki Nodların boyutlarına bakarak ilerler ve uygun yere yerlesitirilir .

+add(Vector<E> ):Metod recursive olarak çalışır.Her yeni seviyede dimensonal degerlerinin bır sonrası ile karsılastırma yapar . Egerkı data apısındaki vectorun sonuna gelirse dimensional degeri 0 degeri atanır . Ve böylece yerlesiceğı yere kadar devam eder . metodun karmasıklığı yukseklığı ile orantılıdır . Worst case durumunda ise  $O(n)$  karmasıklığı gelir . Mesela hep sag Node e eklenmesi durumudur .

+contains(Vector<E>):Metod parametre degerine gore arama yapar egerkı node ıceride varsa boolean deger dondurur.Surekli bır yon sectğı için karmasıklığı  $O(\log n)$  degerindedir

+find(Vecotor<E>): Metod parametre degerine gore arama yapar egerkı node ıceride varsa Node un data degerını yanı vector degeri dondurur. .Surekli bır yon sectğı için karmasıklığı  $O(\log n)$  degerindedir .

+delete(Vector<E>):en karmasık metod bu metoddur . Oncelikle Queue veri tıpi ile çalışır metod . findNode metod yardımı ile silinecek olan metodun referansı elde edilir . Dha sonra findParent motodu yaridımı ile parent node bulunur ve aradı bag kopartılır . alt agac yapısı kaybolmasın diye alttakı node lar Queue yapısına atılır . Daha sonra tek tek add metodu ile dogru yerlerine

yerleştirilir. Bu metodun karmasıklığı alt nodların eklenmesi durumunda ortaya çıkar. Çünkü içinde add metodu çağırıldığı için  $O(n^2)$  worst case de diyebiliriz.

`+remove(Vector<E>):` Metod içerisinde delete metodu çağırılmıştır. Değer silinmiş ise true silinemediyse false değerini return eder.

## 3 RESULT

### 3.1.1 Test Cases

#### PART1:

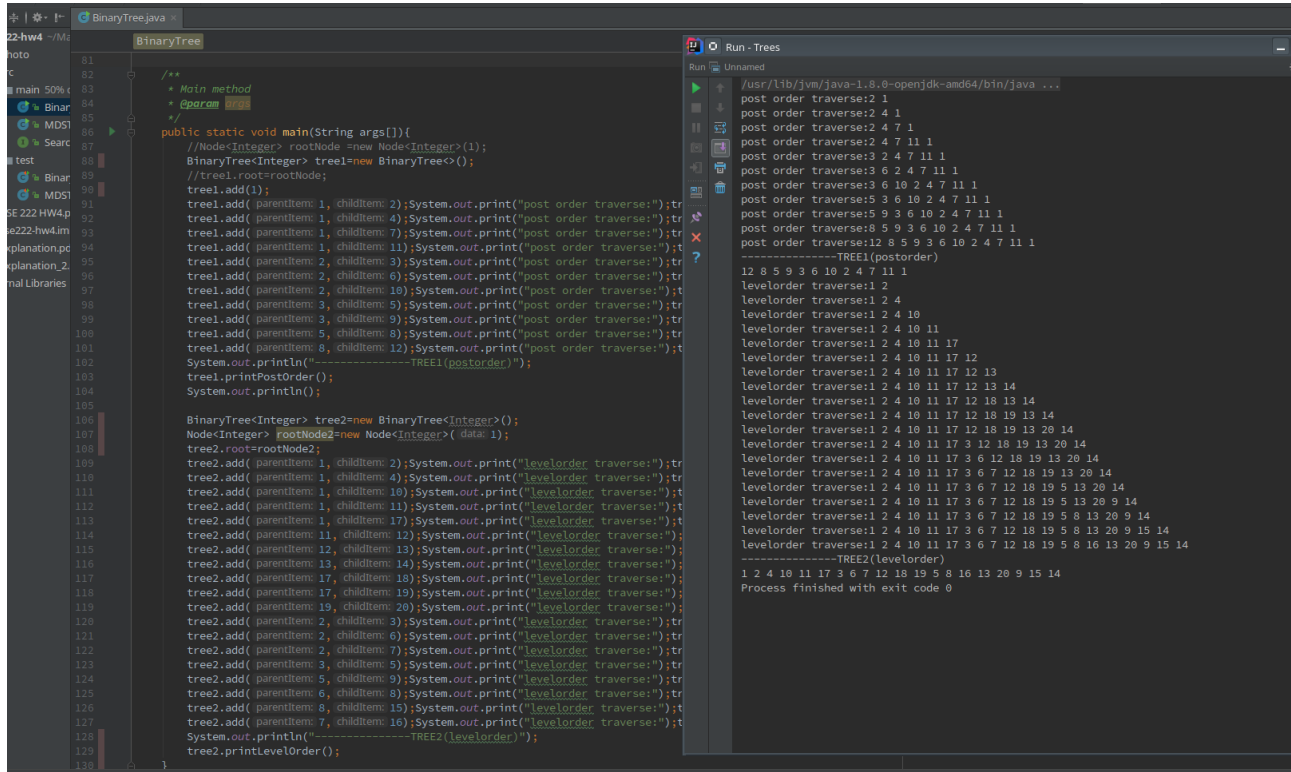
İki ağacın yapısı oluşturulmuştur. İlk ağacın postOrderSearch metodu yardımı ile postOrder bir şekilde test edilmiştir (Basarılı). 2. Ağacın yapısı levelOrderSearch metod yardımı ile levelOrder bir şekilde test edilmiştir (Basarılı) Sonuçlar 3.1.2 bölümündedir.

#### PART2:

Sınıfa data tipi vektör olan nodlar teker teker eklenmiştir ve interfaceden implement edilen metodlar test edilmiştir (Basarılı) Sonuçlar 3.1.2 bölümündedir.

### 3.1.2 Test Result

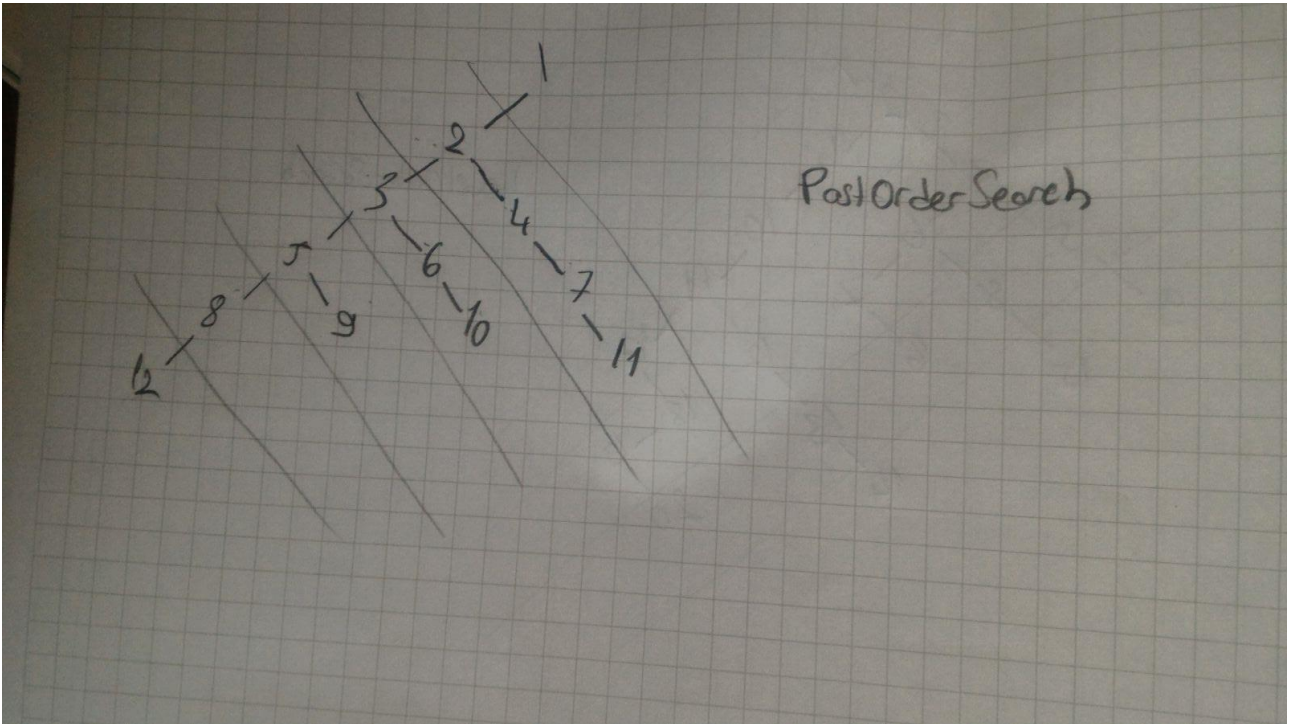
#### PART1:



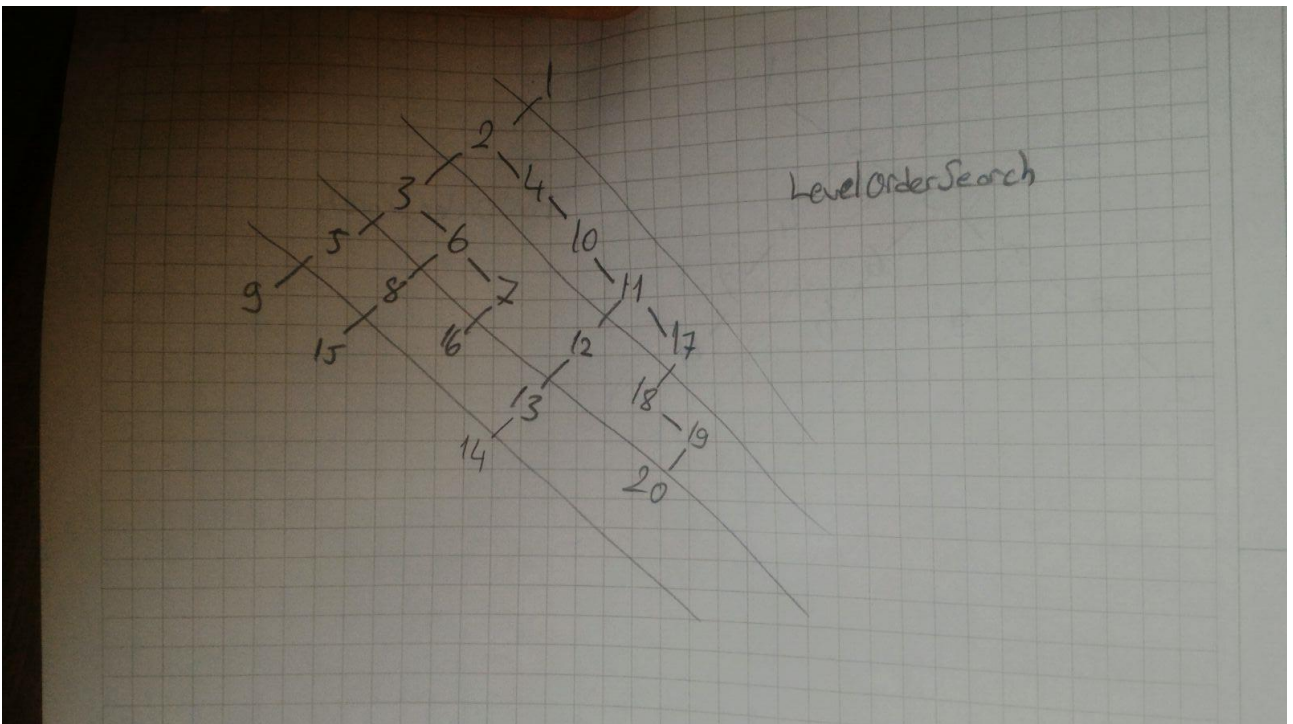
```
BinaryTree.java
1  // Main method
2  // @param tree
3  //
4  public static void main(String args[]) {
5      // Node<Integer> rootNode = new Node<Integer>(1);
6      BinaryTree<Integer> tree1 = new BinaryTree<>();
7      // tree1.root = rootNode;
8      tree1.add(1);
9      tree1.add( parentItem: 1, childItem: 2 ); System.out.print("post order traverse:"); tr
10     tree1.add( parentItem: 1, childItem: 4 ); System.out.print("post order traverse:"); tr
11     tree1.add( parentItem: 1, childItem: 7 ); System.out.print("post order traverse:"); tr
12     tree1.add( parentItem: 1, childItem: 11 ); System.out.print("post order traverse:"); t
13     tree1.add( parentItem: 2, childItem: 3 ); System.out.print("post order traverse:"); tr
14     tree1.add( parentItem: 2, childItem: 6 ); System.out.print("post order traverse:"); tr
15     tree1.add( parentItem: 2, childItem: 10 ); System.out.print("post order traverse:"); t
16     tree1.add( parentItem: 3, childItem: 5 ); System.out.print("post order traverse:"); tr
17     tree1.add( parentItem: 3, childItem: 9 ); System.out.print("post order traverse:"); tr
18     tree1.add( parentItem: 5, childItem: 8 ); System.out.print("post order traverse:"); tr
19     tree1.add( parentItem: 8, childItem: 12 ); System.out.print("post order traverse:"); t
20     System.out.println("-----TREE1(postorder)");
21     tree1.printPostOrder();
22     System.out.println();
23
24     BinaryTree<Integer> tree2 = new BinaryTree<Integer>();
25     Node<Integer> rootNode2 = new Node<Integer>( data: 1 );
26     tree2.root = rootNode2;
27     tree2.add( parentItem: 1, childItem: 2 ); System.out.print("levelorder traverse:"); tr
28     tree2.add( parentItem: 1, childItem: 4 ); System.out.print("levelorder traverse:"); tr
29     tree2.add( parentItem: 1, childItem: 10 ); System.out.print("levelorder traverse:"); t
30     tree2.add( parentItem: 1, childItem: 11 ); System.out.print("levelorder traverse:"); t
31     tree2.add( parentItem: 1, childItem: 17 ); System.out.print("levelorder traverse:"); t
32     tree2.add( parentItem: 11, childItem: 12 ); System.out.print("levelorder traverse:");
33     tree2.add( parentItem: 12, childItem: 13 ); System.out.print("levelorder traverse:");
34     tree2.add( parentItem: 13, childItem: 14 ); System.out.print("levelorder traverse:");
35     tree2.add( parentItem: 17, childItem: 18 ); System.out.print("levelorder traverse:");
36     tree2.add( parentItem: 17, childItem: 19 ); System.out.print("levelorder traverse:");
37     tree2.add( parentItem: 19, childItem: 20 ); System.out.print("levelorder traverse:");
38     tree2.add( parentItem: 2, childItem: 3 ); System.out.print("levelorder traverse:"); tr
39     tree2.add( parentItem: 2, childItem: 6 ); System.out.print("levelorder traverse:"); tr
40     tree2.add( parentItem: 2, childItem: 7 ); System.out.print("levelorder traverse:"); tr
41     tree2.add( parentItem: 3, childItem: 5 ); System.out.print("levelorder traverse:"); tr
42     tree2.add( parentItem: 5, childItem: 9 ); System.out.print("levelorder traverse:"); tr
43     tree2.add( parentItem: 6, childItem: 8 ); System.out.print("levelorder traverse:"); tr
44     tree2.add( parentItem: 8, childItem: 13 ); System.out.print("levelorder traverse:"); t
45     tree2.add( parentItem: 7, childItem: 16 ); System.out.print("levelorder traverse:"); t
46     System.out.println("-----TREE2(levelorder)");
47     tree2.printLevelOrder();
48 }
```

```
Run - Trees
Run Unnamed
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
post order traverse:1 2 4 1
post order traverse:2 4 7 1
post order traverse:2 4 7 11 1
post order traverse:3 2 4 7 11 1
post order traverse:3 6 2 4 7 11 1
post order traverse:3 6 10 2 4 7 11 1
post order traverse:5 3 6 10 2 4 7 11 1
post order traverse:5 9 3 6 10 2 4 7 11 1
post order traverse:8 5 9 3 6 10 2 4 7 11 1
post order traverse:12 8 5 9 3 6 10 2 4 7 11 1
-----TREE1(postorder)
12 8 5 9 3 6 10 2 4 7 11 1
levelorder traverse:1 2
levelorder traverse:1 2 4
levelorder traverse:1 2 4 10
levelorder traverse:1 2 4 10 11
levelorder traverse:1 2 4 10 11 17
levelorder traverse:1 2 4 10 11 17 12
levelorder traverse:1 2 4 10 11 17 12 13
levelorder traverse:1 2 4 10 11 17 12 13 14
levelorder traverse:1 2 4 10 11 17 12 18 13 14
levelorder traverse:1 2 4 10 11 17 12 18 19 13 14
levelorder traverse:1 2 4 10 11 17 3 6 12 18 19 13 20 14
levelorder traverse:1 2 4 10 11 17 3 6 7 12 18 19 13 20 14
levelorder traverse:1 2 4 10 11 17 3 6 7 12 18 19 5 8 13 20 9 14
levelorder traverse:1 2 4 10 11 17 3 6 7 12 18 19 5 8 13 20 9 15 14
levelorder traverse:1 2 4 10 11 17 3 6 7 12 18 19 5 8 16 13 20 9 15 14
-----TREE2(levelorder)
1 2 4 10 11 17 3 6 7 12 18 19 5 8 16 13 20 9 15 14
Process finished with exit code 0
```

Part 1 Main method sonucu



Post Order Tree Test Agac yapısı



Level Order Tree Test Yapısı



## PART2:

```

MDSTree.java
public static void main(String args[]){
    Vector<Integer> tempA=new Vector<>();Vector<Integer> tempB=new Vector<>();
    Vector<Integer> tempC=new Vector<>();Vector<Integer> tempD=new Vector<>();
    Vector<Integer> tempE=new Vector<>();Vector<Integer> tempF=new Vector<>();
    Vector<Integer> tempG=new Vector<>();Vector<Integer> tempH=new Vector<>();
    Vector<Integer> tempI=new Vector<>();Vector<Integer> tempJ=new Vector<>();
    Vector<Integer> tempK=new Vector<>();MDSTree<Integer> tree=new MDSTree<>();
    tempA.add(40);tempA.add(45);tempA.add(50);
    tempB.add(15);tempB.add(70);tempB.add(80);
    tempC.add(70);tempC.add(10);tempC.add(90);
    tempD.add(69);tempD.add(50);tempD.add(10);
    tempE.add(66);tempE.add(85);tempE.add(40);
    tempF.add(85);tempF.add(90);tempF.add(5);
    tempG.add(10);tempG.add(65);tempG.add(30);
    tempH.add(90);tempH.add(80);tempH.add(8);
    tempI.add(95);tempI.add(100);tempI.add(5);
    tempJ.add(90);tempJ.add(75);tempJ.add(3);
    tempK.add(90);tempK.add(70);tempK.add(3);
    tree.add(tempA);tree.add(tempB);tree.add(tempC);tree.add(tempD);
    tree.add(tempE);tree.add(tempF);tree.add(tempG);tree.add(tempH);
    tree.add(tempI);tree.add(tempJ);tree.add(tempK);
    System.out.println("::::::::::add method::::::::::");
    System.out.println("tree.add(40,45,50)");System.out.println("tree.add(15,70,80)");
    System.out.println("tree.add(70,10,90)");System.out.println("tree.add(69,50,10)");
    System.out.println("tree.add(66,85,40)");System.out.println("tree.add(85,90,5)");
    System.out.println("tree.add(10,65,30)");System.out.println("tree.add(90,80,8)");
    System.out.println("tree.add(95,100,5)");System.out.println("tree.add(90,75,3)");
    System.out.println("tree.add(90,70,3)");
    System.out.println("::::::::::contain method::::::::::");
    Vector<Integer> asd=new Vector<Integer>();
    asd.add(85);asd.add(90);asd.add(6);
    System.out.println("tree.contains(Vector<Integer>(85,90,5)): "+tree.contains(
    System.out.println("tree.contains(Vector<Integer>(85,90,6)): "+tree.contains(
    System.out.println("::::::::::find method::::::::::");
    System.out.println("tree.find(Vector<Integer>(90,75,3)): "+tree.find(tempJ).get(2));
    System.out.println("tree.find(tempJ).get(2));");
    System.out.println("::::::::::delete method::::::::::");
    Queue<Vector<Integer>> sıra=new LinkedList<Vector<Integer>>();
    System.out.println("levelOrder");
    sıra=tree.levelOrder(tree.root);
    while (!sıra.isEmpty()){
        System.out.println(sıra.poll());
    }
    System.out.println("tree.delete(Vector<Integer>(90,80,8)) :");
    tree.delete(tempH);
    sıra=tree.levelOrder(tree.root);
    while (!sıra.isEmpty()){
        System.out.println(sıra.poll());
    }
}

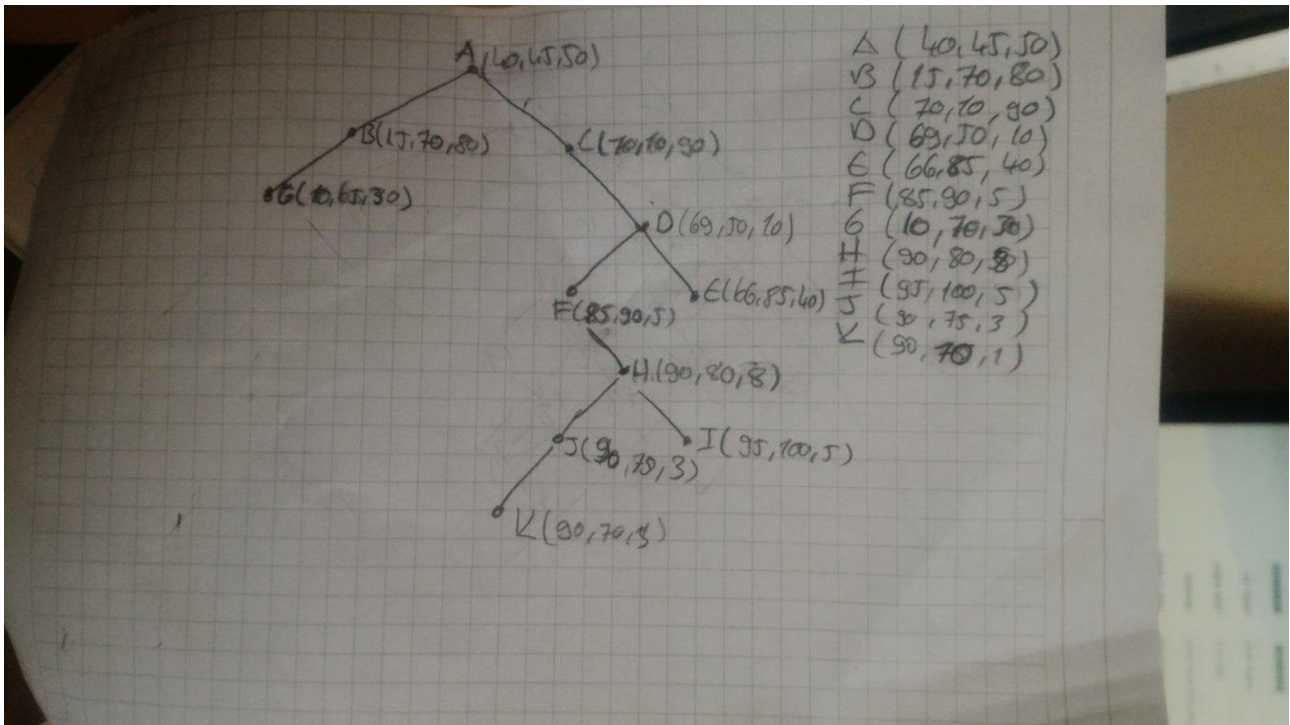
```

```

Run - MDSTree
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
::::::::::add method::::::::::
tree.add(40,45,50)
tree.add(15,70,80)
tree.add(70,10,90)
tree.add(69,50,10)
tree.add(66,85,40)
tree.add(85,90,5)
tree.add(10,65,30)
tree.add(90,80,8)
tree.add(95,100,5)
tree.add(90,75,3)
tree.add(90,70,3)
::::::::::contain method::::::::::
tree.contains(Vector<Integer>(85,90,5)): true
tree.contains(Vector<Integer>(85,90,6)): false
::::::::::find method::::::::::
tree.find(Vector<Integer>(90,75,3)): 90-75-3
::::::::::delete method::::::::::
levelOrder:
[40, 45, 50]
[15, 70, 80]
[70, 10, 90]
[10, 65, 30]
[69, 50, 10]
[85, 90, 5]
[66, 85, 40]
[90, 80, 8]
[90, 75, 3]
[95, 100, 5]
[90, 70, 3]
tree.delete(Vector<Integer>(90,80,8)) :
[40, 45, 50]
[15, 70, 80]
[70, 10, 90]
[10, 65, 30]
[69, 50, 10]
[85, 90, 5]
[66, 85, 40]
[90, 75, 3]
[90, 70, 3]
[95, 100, 5]
Process finished with exit code 0

```

Part 2 Main Method çıktı sonucu



Part 2 agac yapısı.Silinen node H nodudur.